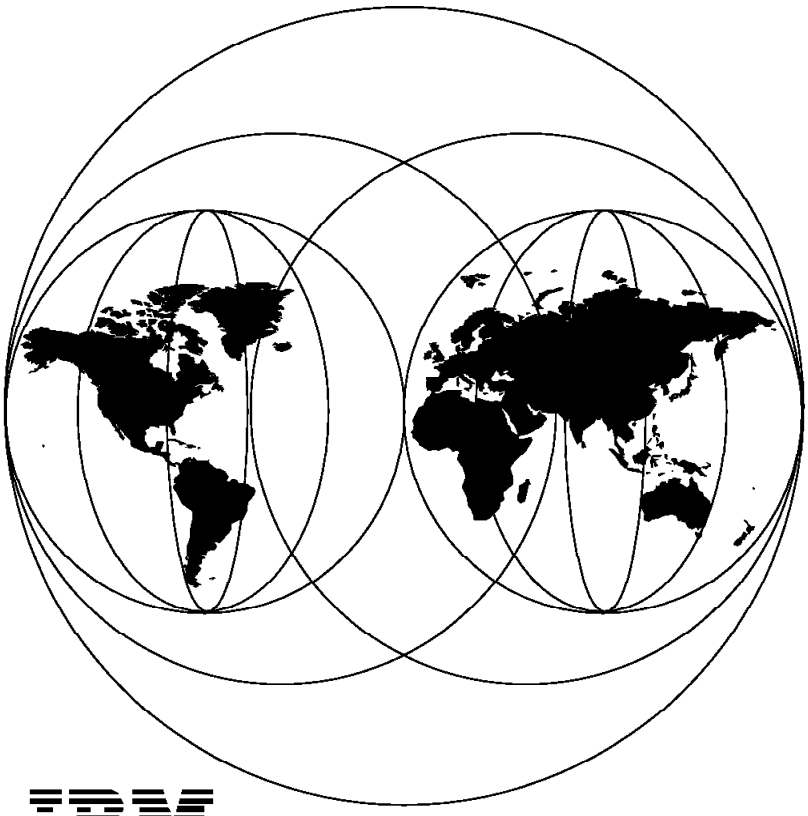
SG24-4989-00

RS/6000 Performance Tools in Focus

May 1997



IBM

International Technical Support Organization

Austin Center

International Technical Support Organization

SG24-4989-00

RS/6000 Performance Tools in Focus

May 1997

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix D, “Special Notices” on page 295.

First Edition (May 1997)

This edition applies to IBM RS/6000 for use with the AIX Operating System Version 4.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JN9B Building 045 Internal Zip 2834
11400 Burnet Road
Austin, Texas 78758-3493

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures ix

Tables xi

Preface xiii
The Team That Wrote This Redbook xiii
Comments Welcome xiv

Chapter 1. Introduction 1
1.1 AIX Performance Tools 3
1.2 Baselines 4
1.3 Permanent Light Monitoring 4

Chapter 2. Standard (UNIX) Performance Tools 7
2.1 The vmstat Command 7
2.1.1 CPU Bound 7
2.1.2 Memory Bound 10
2.1.3 I/O Bound 15
2.1.4 Summary Option 16
2.1.5 Conclusion 17
2.2 The iostat Command 17
2.2.1 TTY Report 19
2.2.2 CPU Report 20
2.2.3 Drive Report 20
2.3 The sar Command 23
2.3.1 Real-Time Sampling and Display 23
2.3.2 Display of Previously Captured Data 24
2.3.3 System Activity Accounting via cron 24
2.3.4 Useful Options 26
2.3.5 Correlation Between vmstat, iostat, and sar 29
2.3.6 The timex Command 30
2.4 The ps Command 30
2.4.1 CPU Information 30
2.4.2 Memory Information 31
2.4.3 New ps Options 32
2.4.4 Useful Shell Scripts 32
2.5 The pstat Command 33
2.6 The netstat Command 35
2.6.1 Using the netstat Command 35
2.6.2 Additional Rules of Thumb 41
2.6.3 AIX Version 4.2.1 Improvements 42
2.7 The nfsstat Command 42
2.7.1 NFS Server Information 43
2.7.2 NFS Client Information 44
2.7.3 AIX Version 4.2.1 Improvements 45
2.8 The no Command 46
2.8.1 no Command Parameters 47
2.8.2 Stream Parameters 48
2.8.3 AIX Version 4.2.1 Improvements 49
2.9 The nfso Command 50
2.9.1 nfso Command Parameters 51

2.9.2 AIX Version 4.2.1 Improvements . . . 53
2.10 The nice Command . . . 54
2.10.1 Using the nice Command . . . 55
2.11 The renice Command . . . 56
2.12 The prof Command . . . 58
2.12.1 The prof Implementation . . . 58
2.13 The gprof Command . . . 59
2.13.1 The gprof Implementation . . . 59

Chapter 3. Legacy (AIX) Performance Tools . . . 63
3.1 The tprof Command . . . 63
3.1.1 The tprof Implementation . . . 63
3.1.2 The Advantages of tprof . . . 64
3.1.3 The Limitations of tprof . . . 64
3.1.4 A Systemwide Example . . . 65
3.1.5 A Source-Line Example . . . 67
3.2 The svmon Command . . . 71
3.2.1 How Much Memory is in Use . . . 72
3.2.2 Who is Using Memory? . . . 73
3.2.3 Detailed Information on a Specific Segment ID . . . 74
3.2.4 List of Top Memory Usage of Segments . . . 75
3.2.5 Correlating svmon and vmstat Outputs . . . 75
3.2.6 Correlating svmon and ps Outputs . . . 76
3.2.7 Finding Memory-Leaking Programs . . . 77
3.2.8 Calculating the Minimum Memory Requirement of a Program . . . 78
3.3 The rmss Command . . . 79
3.3.1 Using rmss . . . 80
3.3.2 Simulating Different Memory Sizes . . . 80
3.4 The filemon Command . . . 83
3.4.1 Using filemon . . . 83
3.4.2 The Global Reports of filemon . . . 87
3.4.3 The Detailed Reports of filemon . . . 89
3.4.4 Comparing filemon and vmstat Outputs . . . 92
3.4.5 Things to Keep in Mind . . . 93
3.5 The fileplace Command . . . 94
3.5.1 Using fileplace . . . 95
3.5.2 Space Efficiency and Sequentiality . . . 96
3.5.3 AIX File System Organization . . . 97
3.6 The lslv Command . . . 98
3.6.1 LVM Policies . . . 99
3.6.2 Logical Volume Fragmentation . . . 102
3.6.3 Relationship Between Policies and Performance . . . 104
3.7 The netpmon Command . . . 106
3.7.1 The netpmon Implementation and Functions . . . 106
3.7.2 Using netpmon . . . 107
3.7.3 The Global Reports of netpmon . . . 110
3.7.4 The Detailed Reports of netpmon . . . 111
3.7.5 Limitations of netpmon . . . 113
3.8 The genld Command . . . 113
3.9 The genkld Command . . . 114
3.10 The genkex Command . . . 115
3.11 The stripnm Command . . . 116
3.12 The trace and trcrpt Commands . . . 120
3.12.1 The trace Implementation . . . 121
3.12.2 Starting and Controlling trace . . . 121

3.12.3 Examples of trace . . . 122
3.12.4 Using trcrpt to View Trace Data . . . 124
3.12.5 trcrpt Examples . . . 124
3.12.6 How to Spot Thrashing . . . 126
3.12.7 Using trace to Identify Other Resource Constraints . . . 127

Chapter 4. Advanced AIX V4 Performance Tools . . . 129
4.1 PDT . . . 129
4.1.1 Enabling and Configuring PDT . . . 130
4.1.2 PDT Report . . . 133
4.1.3 PDT Error Reporting . . . 139
4.2 The perfpmr Package . . . 140
4.2.1 Using the perfpmr Command . . . 140
4.2.2 Output Files . . . 141
4.3 The bf and bfrpt Commands . . . 144
4.3.1 How Does BigFoot Work? . . . 144
4.3.2 Using bf . . . 144
4.3.3 Generating Reports with bfrpt . . . 146
4.4 The stem Command . . . 152
4.4.1 Using stem . . . 153
4.4.2 Shared-Memory Callgraphs . . . 154
4.4.3 Stem Map File . . . 156
4.5 The syscalls Command . . . 158
4.5.1 Using syscalls . . . 159
4.5.2 Examples . . . 160
4.6 The fdpr Command . . . 162
4.6.1 Using fdpr . . . 163
4.6.2 Other fdpr Options . . . 164
4.6.3 Considerations . . . 164
4.7 The lockstat Command . . . 164
4.7.1 Locks on SMP Systems . . . 164
4.7.2 Using lockstat . . . 166
4.7.3 Improving Lock Performance . . . 169
4.8 The cpu_state Command . . . 170
4.8.1 Using cpu_state . . . 171
4.8.2 Differences in AIX V4.1 . . . 173
4.9 The bindprocessor Command . . . 173
4.9.1 Processor Affinity . . . 173
4.9.2 Using bindprocessor . . . 175
4.9.3 Considerations . . . 176
4.10 The schedtune Command . . . 177
4.10.1 Memory Load Control Parameters . . . 178
4.10.2 fork() Retry Interval Parameter . . . 182
4.10.3 Priority Calculation Parameters . . . 183
4.10.4 Time-Slice Increment Parameter . . . 184
4.10.5 Processor Affinity Parameters . . . 185
4.11 The vmtune Command . . . 189
4.11.1 Tuning VMM Page Replacement . . . 190
4.11.2 Tuning Sequential Read-Ahead . . . 194
4.11.3 Tuning Write-Behind . . . 198
4.11.4 Tuning Paging-Space Thresholds . . . 199
4.11.5 Miscellaneous I/O Tuning Parameters . . . 199

Chapter 5. Performance Toolbox . . . 201
5.1 Introduction . . . 201

5.2 Performance Toolbox Concepts . . . 201
5.3 Benefits of Using Performance Toolbox . . . 202
5.4 Manager . . . 203
5.5 Agent . . . 204
5.6 Useful Information . . . 205
5.7 Using Performance Toolbox . . . 207
5.7.1 Manager Main Window . . . 208
5.7.2 Creating a New Console . . . 209
5.7.3 Monitoring a Process . . . 217
5.7.4 Monitoring an SMP with 3dmon . . . 220
5.7.5 Monitoring Multiple Hosts with 3dmon . . . 222
5.8 Investigating Performance Problems . . . 225
5.9 Customizing Performance Toolbox . . . 230
5.9.1 Data Reduction and Alarms with filtd . . . 230
5.10 Monitoring Exceptions with exmon . . . 236
5.11 Analyzing Recordings with the azizo Program . . . 240
5.11.1 The azizo Main Window . . . 243
5.11.2 The azizo Graph Window . . . 244
5.11.3 Zooming-In on Main Graphs . . . 247
5.11.4 Other Options . . . 250
5.12 Conclusion . . . 250

Chapter 6. Additional Performance Tools . . . 251
6.1 xgprof . . . 251
6.2 Program Visualizer . . . 253
6.2.1 pvtrace . . . 254
6.2.2 Starting PV . . . 255
6.2.3 Packaging the Trace . . . 259
6.2.4 Making a Textual Trace Report Using pvreport . . . 260
6.2.5 PV Tutorial . . . 261
6.3 utld . . . 261
6.3.1 Generating a Trace for utld . . . 262
6.3.2 Using utld . . . 263
6.4 Sources for Additional Tools . . . 266

Appendix A. Summary of Rules of Thumb . . . 267

Appendix B. Summary of Tunable AIX Parameters . . . 269

Appendix C. Performance Tools Paths and Filesets . . . 293

Appendix D. Special Notices . . . 295

Appendix E. Related Publications . . . 297
E.1 International Technical Support Organization Publications . . . 297
E.2 Redbooks on CD-ROMs . . . 297
E.3 Other Publications . . . 297

How to Get ITSO Redbooks . . . 299
How IBM Employees Can Get ITSO Redbooks . . . 299
How Customers Can Get ITSO Redbooks . . . 300
IBM Redbook Order Form . . . 301

List of Abbreviations . . . 303

Index 305

ITSO Redbook Evaluation 307

Figures

1. Performance-Tuning Flowchart . . . 2
2. AIX File System Organization . . . 98
3. AIX LVM Policies . . . 99
4. The Inter-Disk Policy . . . 101
5. Logical Volume Striping . . . 105
6. The trace Implementation . . . 121
7. Output File __global.ps . . . 150
8. Shared-Memory Buffer Structure . . . 154
9. Map File Format . . . 156
10. Map File Example . . . 157
11. Example of Conditional Branch Re-Coding . . . 162
12. Data Serialization . . . 165
13. Relationship Between Throughput and Granularity . . . 170
14. Sequential Read Ahead . . . 195
15. The Performance Toolbox Environment . . . 201
16. AIX Performance Toolbox Initial Screen . . . 208
17. Creating a New Console (Step 1) . . . 209
18. Creating a New Console (Step 2) . . . 210
19. Creating a New Console (Step 3) . . . 211
20. Creating a New Console (Step 4) . . . 212
21. Creating a New Console (Step 5) . . . 213
22. Creating a New Console (Step 6) . . . 214
23. Changing the Properties of a Value . . . 215
24. Created New Console . . . 216
25. Monitoring a Process (Step 1) . . . 217
26. Monitoring a Process (Step 2) . . . 218
27. Monitoring a Process (Step 3) . . . 219
28. Monitoring a Process (Step 4) . . . 220
29. 3-D Monitor Selection Menu . . . 221
30. 3dmon Output on a Four-Way SMP . . . 222
31. Configuration and Tiling . . . 223
32. Host Selection . . . 224
33. Multiple Hosts 3dmon Graph . . . 225
34. Local System Monitor Console . . . 226
35. Select Host Processes . . . 227
36. Initial 3dmon Graph of CPU-Intensive Processes . . . 228
37. vmstat Output . . . 229
38. Final 3dmon Graph of CPU-Intensive Processes . . . 230
39. Partial Listing of Statistics . . . 231
40. Output From Alarm . . . 233
41. An Instrument Using Filters . . . 235
42. Pie Chart Using Filters . . . 236
43. The exmon Main Window . . . 237
44. Modified exmon Main Window . . . 238
45. Command Execution Pop-Up . . . 239
46. Analyzing Recordings Pop-Up . . . 241
47. The azizo Main Window . . . 242
48. Recording Files Pop-Up Menu . . . 242
49. The azizo Main Window . . . 243
50. Main Window of azizo with Memory-Related Metrics . . . 244
51. The azizo Graph Window . . . 245

52. Filtered azizo Main Graph Window 246
53. Filter Dialog Box 247
54. Zoom-In Dialog Box 248
55. Zoomed-In Main Graph 248
56. Information Window for a Zoomed-In Main Graph 249
57. The xgprof Start Window 252
58. Zoomed-In Window of the Main Routine 253
59. PV Control Panel 255
60. Trace Generation Pop-Up Menu 257
61. Question Pop-Up Menu 257
62. Partial PV Views (One) 258
63. Partial PV Views (Two) 259
64. Trace Report View 261

Tables

1. List of the Most Important Performance Tools 3
2. List of Performance Tools by Resource 4
3. Send and Receive Spaces for TCP and UDP 48
4. Table of Perfagents for Sun SPARCstations 206
5. Rules of Thumb 268

Preface

This redbook describes and explores the standard, legacy, and advanced performance tools available on RS/6000 with the AIX operating system.

This redbook was written for RS/6000 users, system administrators, and system engineers who need to use the performance monitoring and tuning tools to evaluate system performance and tune the system.

Performance monitoring and system tuning is a vast and complex topic. Fortunately, due to its UNIX heritage, AIX provides a powerful set of performance-monitoring and tuning tools. The standard UNIX performance tools allow you to check the hardware resources of a system. The legacy and advanced tools go beyond the hardware and investigate logical resources. The Performance Toolbox for AIX combines all these tools in one Motif-based toolbox.

All the tools described in this book will help detect and identify a performance bottleneck. General guidelines are given to narrow down the culprit. Interpretation of data and rules of thumb are provided on a tools by tools basis.

Several practical examples are presented to demonstrate the use of the tools and the relevant numbers to look out for in deciding where the performance bottleneck is.

Some knowledge of UNIX and/or the AIX operating system is assumed.

The Team That Wrote This Redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization Austin, Center.

Andreas Hoetzel is an International Technical Support Specialist for RS/6000 and AIX Performance at the International Technical Support Organization, Austin Center. He writes extensively and teaches IBM classes worldwide on all areas of AIX internals, performance, and tuning. Before joining the ITSO, Andreas Hoetzel worked in the AIX Competence Center in Munich Germany as an AIX Technical Support Specialist.

Cintia Scovine Barcelos is an AIX Support Professional in Brazil. She has three years of experience in the field. She holds a degree in Electronic Engineering. Her areas of expertise include HACMP/6000 performance and tuning and capacity planning.

Xiong Guan is an AIX Technical Sales Specialist in Beijing, China. He has three years of experience in RS/6000 and AIX. His areas of expertise focus on AIX system management and HACMP.

Michael Spornhauer is an AIX Support Professional in Germany. He has three years of experience in the second-level support for the AIX base operating system. He holds a degree in Computer Science and his areas of expertise include AIX internals, TCP/IP, performance and tuning.

Gavin Steer is an AIX Support Professional in South Africa. He has been with IBM for 16 years and has 10 years experience in system performance, with the last three years specializing in AIX.

Thanks to the following people for their invaluable contributions to this project:

Yves Bex
International Technical Support Organization, Austin Center

Marcus Brewer
International Technical Support Organization, Austin Center

Rebeca Rodriguez
International Technical Support Organization, Austin Center

Steve Gardner
International Technical Support Organization, Austin Center

Heidemarie Hoetzel
IBM Germany

Mathew Accapadi
IBM Austin

Jim Chen
IBM Austin

Rudy Chukran
IBM Austin

Ryan France
IBM Austin

Stephen Nasypany
IBM Austin

Ann Ruth
IBM Austin

Barbara Wang
IBM Austin

Erik Jensen
IBM T.J. Watson Research Center

Comments Welcome

We want our redbooks to be as helpful as possible. Should you have any comments about this or other redbooks, please send us a note at the following address:

redbook@vnet.ibm.com

Your comments are important to us!

Chapter 1. Introduction

Due to its UNIX heritage, AIX provides a powerful set of performance-monitoring and tuning tools. The available tools give performance information for different components of the system and on various parameters that affect performance.

Since resources are not infinitely fast, tasks (work being performed on the system) are always resource-constrained. The trick is knowing which resource is pacing the system. This is not always as apparent as it seems.

Systems have both real and logical resources. Real resources are physical devices, such as the CPU, memory, disk drives and other I/O components. Logical resources are software abstractions for managing storage or I/O, such as the Logical Volume Manager (LVM), queues, memory buffers, file systems, and the Virtual Memory Manager (VMM).

Traditional UNIX tools exist to collect and report usage data for real resources. However, since many logical resources are implementation-specific, traditional tools do not provide the level of detail required to fully analyze a system, and it is harder to find tools to measure their utilization. AIX has a suite of performance tools, the so-called *Legacy Tools,* that provide fine-grained reports concerning resource usage. These AIX-specific performance tools are introduced later in this book to help you analyze logical resource activity.

Before using these tools, a few concepts have to be clarified.

What is a Performance Bottleneck?: A performance bottleneck is the slowest component in a computer environment. This can either be a system resource like CPU, memory, or disk, or it could be the network. There is always a bottleneck because some resource will always be the slowest. The question is whether this bottleneck is a problem on a daily business.

How to Determine a Performance Bottleneck?: Although there is no cookbook for performance monitoring, there are some general guidelines that should be followed. The sequence of measuring system performance is extremely important. One should always follow the specified path, which is: CPU, Memory, I/O, Network.

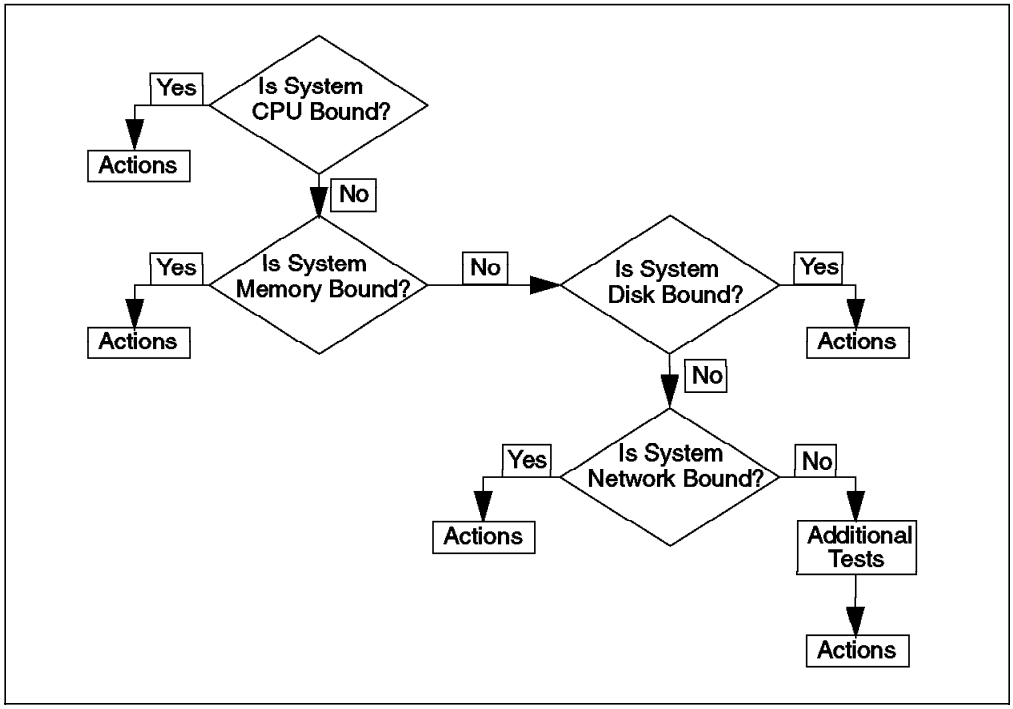


Figure 1. Performance-Tuning Flowchart

Without following this path, you might overlook a bottleneck. For instance, if a system is paging a lot, you might see a heavy usage of the disks. Not looking at the memory statistics before looking at the I/O statistics could mislead you to the assumption that there is an I/O bottleneck.

Important

Always use the sequence as shown in the flowchart to determine a performance bottleneck:

- CPU
- Memory
- Disk I/O
- Network

Always consider the whole system and the interdependencies of its resources.

- Memory depends on many different factors including:
  - How many users will the system be supporting?
  - How many and what type of applications will be running?
  - Will the applications be large, computing-intensive or more I/O oriented?
- The CPU power is dependent on the same factors as the memory, but there are some additional ones to consider:
  - Does the system have to meet some performance criteria such as minimum response time for all users?
  - Do batch applications have to complete within a certain time?
  - Will the CPU workload grow over time?

- The I/O subsystem is one of the most important areas of system design. Improperly configured disks, file systems, or networks can negate any performance gains that may have been realized by an otherwise properly configured system.

You should always keep in mind:

- Performance analysis is not always deterministic.
- Performance problems can appear quickly and with little notice.
- Probably the toughest part of performance management is understanding how all components of the operating system and the various applications running on it interact.
- Subtle resources may be logical rather than physical.

1.1 AIX Performance Tools

As mentioned above, AIX provides several monitoring tools to determine a performance bottleneck. Some of them may also be used to tune the system. Not all of these tools come with the AIX Basic Operating System. Some of them are part of the AIX Performance Agent or AIX Performance Manager LPP software as you can see in Table 1.

	AIX V3.2	AIX V4
Base AIX	vmstat, iostat, sar, ps, pstat netstat, nfsstat, no, nfso, trace, prof, gprof, tprof, svmon, filemon, fileplace, netpmon, rmss, rmap, lvedit, genld, genkld, genkex, stripnm, schedtune, vmtune	vmstat, iostat, sar, ps, pstat, netstat, nfsstat, no, nfso, trace, prof, gprof, pdt, perfpmr, schedtune, vmtune
Performance AIDE (Performance Agent)	xmservd, filtd	xmservd, filtd, tprof, svmon, filemon, fileplace, netpmon, rmss, bf, bfrpt, stem, syscalls, fdpr, lockstat, genld, genkld, genkex, stripnm
Performance Toolbox (Performance Manager)	xmperf, 3dmon, azizo, exmon, chmon	xmperf, 3dmon, 3dplay, azizo, exmon, chmon

Table 1. List of the Most Important Performance Tools

Often, specific performance tools can be used to monitor various system resources. Thus, it is hard to classify tools on the basis of a single system resource. Table 2 on page 4 offers an overview of the performance tools.

	CPU	Memory	Disk	Network
Standard UNIX	vmstat, iostat, sar, ps, pstat	vmstat, sar, ps	iostat, sar, vmstat	netstat, nfsstat, no, nfso
AIX unique	Performance Toolbox, pdt, schedtune	Performance Toolbox, pdt, schedtune, vmtune	Performance Toolbox, lslv, pdt	Performance Toolbox
AIX unique detailed	tprof	svmon, rmss	filemon, fileplace	netpmon
AIX unique very detailed	stem, syscalls, fdpr, trace	bf, bfrpt, fdpr, trace	trace	trace

Table 2. List of Performance Tools by Resource

1.2 Baselines

Having a high number reported from any performance-monitoring tool in any domain (CPU, memory, I/O or network) is not enough to say that the problem is coming from there. Didn't you get that same number before, while everything ran OK? The baseline is here to help you find what has changed since the good-old-times when everybody was happy and now, when the machine seems so slow.

To get a baseline as a reference point from which to work, collect data on your system when things are running smoothly. There are different ways to establish baselines, and different levels at which you can have baselines.

- General-level baseline

  This is the usual processes running on the machine, the usual disk activity, or the file's fragmentation on disk. This can be done both by standard accounting processes which keep day-by-day process activity on the machine and by the new Performance Diagnostic Tool (PDT), discussed later.

- Specific-level baseline

  This allows you to check a really specific domain, like the normal (system + user) CPU usage, the normal pagination rate, the disk activity, and so on. You can have a `sar` command running in the background and keep the data in a file for further analysis, enable system accounting, or use the perfPMR tool. Another possibility is to have the autonomous logging by the `xmservd` daemon coming with the Perfagent LPP, which is the agent part of the IBM Performance Toolbox for AIX.

1.3 Permanent Light Monitoring

Permanent light monitoring, rather than a fixed baseline snapshot of one day, can help you determine if the problem has just appeared that day or if it has been present for days or weeks. It also saves you from having to reproduce the problem; that is not always as easy as it sounds. This king of monitoring should, of course, be as transparent and as unintrusive for the machine as possible.

The tool that you may want to run this permanent light monitoring is the `sar` command, run in background with a large increment and with its output redirected to a file (for minimizing the impact of this monitoring). You can also

enable system accounting via the `cron` daemon, or use the `xmservd` daemon to collect the data, which is probably the best, least-intrusive way.

Manage Exceptions: You may want to check the files created from this monitoring and try to detect abnormal figures to prevent the appearance of any problems. For the `sar` output file, try to find any abnormally high activity on CPU or disks. With the `xmservd` daemon, it is even easier because you can run the `filtd` daemon to catch these figures, and start any alert you may want to specify. This should lead directly to the domain to investigate (CPU, memory, I/O, or network).

Chapter 2. Standard (UNIX) Performance Tools

The standard UNIX performance tools allow you to check the hardware resources of a system: CPU utilization, memory, I/O throughput, and disks. The logical resources, such as VMM, LVM, file systems, queues and buffers, are implementation specific; therefore AIX-specific tools exist, and these are introduced in Chapter 3, “Legacy (AIX) Performance Tools” on page 63, and in Chapter 4, “Advanced AIX V4 Performance Tools” on page 129 **.**

All ″Rules of Thumb″ given in this chapter are only usable as references because of the variations of hardware, like CPU architecture and speed, disk subsystems and controllers, real and virtual memory, and so on. Therefore, statements about good values should be used as a general rule and not as a law. If the system crosses these rules and, at the same time, there is no performance problem, then further investigation is at your discretion. If the given number appears repeatedly within a specific time interval, then you may have a performance problem. The specific time interval also depends on the system environment.

2.1 The vmstat Command

The first tool to use is vmstat, which provides very quick and compact information about various system resources and their related performance problems. The vmstat command reports statistics about kernel threads in the run and wait queue, memory, paging, disks, interrupts, system calls, context switches, and CPU activity. The reported CPU activity is a percentage breakdown of user mode, system mode, idle time, and waits for disk I/O.

The vmstat command uses between 20 and 30 milliseconds of CPU time for each periodic report it generates.

Note

If the vmstat command is used without any options or only with the interval and optionally, the count parameter, like vmstat 2; then the first line of numbers is an average since system reboot.

This line should be ignored.

The vmstat command allows to look for three probable performance problems on the system: CPU bound, memory bound, and I/O bound.

2.1.1 CPU Bound

In the following vmstat output, a CPU-bound program was started.

```
# vmstat 2 10

kthr     memory             page              faults        cpu
----- ----------- ------------------------ ------------ -----------
r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs  us sy id wa
0  0  3269 10342   0   0   0   0    0   0 130   36  20   0  0 99  0
0  0  3275 10336   0   0   0   0    0   0 132   40  26   0  1 99  0
0  0  3282 10326   0   0   0   0    0   0 128 6996  27   3  8 90  0
```

2	**0**	3282	10326	0	0	0	0	0	0	121	80064	20	**15**	**85**	**0**	**0**
2	**0**	3282	10326	0	0	0	0	0	0	126	79579	21	**22**	**78**	**0**	**0**
2	**0**	3282	10326	0	0	0	0	0	0	124	79662	19	**20**	**80**	**0**	**0**
2	**0**	3282	10326	0	0	0	0	0	0	121	79686	20	**16**	**84**	**0**	**0**
0	**0**	3275	10336	0	0	0	0	0	0	128	30305	24	**8**	**30**	**62**	**0**
0	**0**	3275	10336	0	0	0	0	0	0	123	36	20	**0**	**0**	**99**	**0**
0	**0**	3275	10336	0	0	0	0	0	0	122	36	20	**0**	**0**	**99**	**0**

In this example the CPU was idle for 99 percent of the interval until, as can be seen in the third row, a CPU-bound program was started and absorbed the CPU idle time with no I/O wait time consumed. After the program was stopped the CPU utilization went back to the initial values.

To check if the CPU is the bottleneck, consider the four **cpu** columns and the two **kthr** (kernel threads) columns:

- **cpu**

  Percentage breakdown of CPU time usage during the interval.

  - **us**

    The **us** column shows the percent of CPU time spent in user mode. A UNIX process can execute in either user mode or system (kernel) mode. When in user mode, a process executes within its application code and does not require kernel resources to perform computations, manage memory or set variables.

  - **sy**

    The **sy** column details the percentage of time the CPU was executing a process in system mode. This includes CPU resource consumed by kernel processes (kprocs) and others that need access to kernel resources. If a process needs kernel resources, it must execute a system call and is thereby switched to system mode to make that resource available. For example, reading and/or writing of a file requires kernel resources to open the file, seek a specific location, and read and/or write data.

    Optimum use would have the CPU working 100 percent of the time. This holds true in the case of a single-user system with no need to share the CPU. Generally, if **us** + **sy** time is below 90 percent, a single-user system is not considered CPU constrained. However, if **us** + **sy** time on a multiuser system exceeds 80 percent, the processes may spend time waiting in the run queue. Response time and throughput might suffer.

  - **id**

    The **id** column shows the percentage of time which the CPU is idle, or waiting, without pending local disk I/O. If there are no processes available for execution (the run queue is empty), the system dispatches a process called `wait`. The `ps` report (with the `-k` or `-g 0` option) identifies this as `kproc` with a process ID (PID) usually of 516 (for AIX V3, it was PID 514). If the `ps` report shows a high aggregate time for this process, it means there were significant periods of time when no other process was ready to run or waiting to be executed on the CPU. So the system was mostly *idle* and *waiting* for new tasks.

SMP Note

On SMP systems a `wait` process is bound to every CPU. They can be seen with the `pstat -P` command. The command displays the PID/TID in hexadecimal digits. On a four-way system the PIDs for the `wait` processes should be 516, 774, 1032, and 1290.

If there are no I/Os pending to a local disk, all time charged to `wait` is classified as *idle time*. An access to remote disks (NFS-mounted disks) is treated as *idle time* (with a small amount of sy time to execute the NFS requests) because there is no pending I/O request to a local disk.

– **wa**

The **wa** column details the percentage of time the CPU was *idle* with pending local disk I/O. If there is at least one outstanding I/O to a local disk when `wait` is running, the time is classified as waiting for I/O. Unless asynchronous I/O is being used by the process, an I/O request to disk causes the calling process to block (or sleep) until the request has been completed. Once a process's I/O request completes, it is placed on the run queue.

A `wa` value over 25 percent could indicate that the disk subsystem may not be balanced properly, or it may be the result of a disk-intensive workload.

Important for SMP

For SMP systems the **us**, **sy**, **id** and **wa** columns are only averages over the processors. But keep in mind, that the I/O wait statistic per processor is not really a processor-specific statistic; it is a global statistic. An I/O wait is distinguished from *idle time* only by the state of a pending I/O. If there is any pending disk I/O, and the processor is not busy, then it is an I/O wait time. Disk I/O is not tracked by processors; so when there is any I/O wait, all the processors get charged (assuming they are all equally idle). More details about the SMP I/O wait time in 2.3, "The sar Command" on page 23.

• **kthr**

Kernel threads placed on various queues per second over the sampling interval (state changes).

– **r**

Average number of kernel threads placed on the run queue per second; this means the average number of kernel threads which are in the run queue per second. This field indicates the number of runable threads. This value should be less than five for non-SMP systems. For SMP systems, this value should be less than:

$$5 \times (N_{total} - N_{bind})$$

Where N_{total} stands for total number of processors and N_{bind} for the number of processors which have been bound to processes, for example, with the `bindprocessor` command.

If this number increases rapidly, you should probably look at the application(s). But systems may be also running fine with 10 to 15 threads on their run queue, depending on the thread tasks and the amount of time they run.

- **b**

  Average number of kernel threads placed on the wait queue per second. These threads are waiting for resources or I/O. Threads are also located in the wait queue when they are scheduled for execution but waiting for one of their thread pages to be paged in. This value is usually near zero. But, if the run-queue value increases, the wait-queue normally also increases.

- **faults**

  Information about process control, like trap and interrupt rate.

  - **in**

    Explained in 2.1.3, “I/O Bound” on page 15.

  - **sy**

    The number of system calls per second observed in the interval. Resources are available to user processes through well-defined system calls. These calls instruct the kernel to perform operations for the calling process and exchange data between the kernel and the process. Since workloads and applications vary widely, and different calls perform different functions, it is impossible to say how many system calls per-second are too many. But typically, when the sy column raises over 10000 calls per second, there should be some further investigations. For this column, it is advisable to have a baseline measurement that gives a count for a normal sy value.

  - **cs**

    Number of kernel thread context switches per second observed in the interval. The physical CPU resource is subdivided into logical time slices of 10 milliseconds each. Assuming a thread is scheduled for execution, it will run until its time-slice expires, until it is preempted, or until it voluntarily gives up control of the CPU. When another thread is given control of the CPU, the context or working environment of the previous thread must be saved and the context of the current thread must be loaded. AIX has a very efficient context switching procedure, so each switch is inexpensive in terms of resources. Any significant increase in context switches should be cause for further investigation.

2.1.2 Memory Bound

The following vmstat output is an example for a memory-bound system.

```
# vmstat 1 8
kthr     memory             page                  faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 1  1 17264   127  0   0  14 296  818   0 233  172 120 90 10  0  0
 2  0 17264   121  0   0  10 384  871   0 282  486 282 78 14  0  9
 3  0 17264   117  0   0  38 328 1128   1 290  257 156 83 17  0  0
 2  0 17264   119  0   0  67 136  512   0 272  209  94 84 14  0  2
 1  2 17264   115  0   0  69 176 1476   0 300  232  99 85 15  0  0
 2  0 17264   121  0   0  18  24   88   0 186  340 132 84  7  2  7
 2  0 17264   125  0   0  71 160  682   0 282  137  69 94  6  0  0
 2  0 17264   124  0   0  74 160  412   0 262  112  64 85 15  0  0
```

In this example, the size of the free list (**fre**) is already low, and pages are being scanned (**sr**), freed (**fr**) and paged out (**po**). The CPU utilization is around 85

percent in user mode (**us**) and the user application continues to allocate memory. In the third row of numbers, the **fre** number drops under the minimum value for this count; therefore the system starts to free up more pages, until the high water mark for fre is reached. This is the reason for the high numbers of page outs (**po**), freed pages (**fr**), and scanned pages (**sr**). In the sixth row a process finished and its memory is freed, requiring fewer page outs.

To see if the system has performance problems with its VMM, look at the columns of **memory** and **page**.

- **memory**

  Provides information about the real and virtual memory.

  – **avm**

    There is a lot of misconception about this value. It stands for *Active Virtual Memory*. The column gives the average number of 4-K pages that are allocated to paging space. When a process executes, space for working storage is allocated on the paging device. The **avm** value can be used to calculate the amount of paging space assigned to executing processes. The number in the **avm** field divided by 256 will roughly yield the number of megabytes (MB) allocated to paging space systemwide. The same information is reflected in the Percent Used column of the lsps -s command output or by svmon -G under the pg space inuse field.

  – **fre**

    The **fre** column shows the average number of free memory pages. A page is a 4-KB area of real memory. The system maintains a buffer of memory pages, called the free list, that will be readily accessible when the VMM needs space. The minimum number of pages that the VMM keeps on the free list is determined by the minfree parameter of vmtune (for details see 4.11, “The vmtune Command” on page 189). By default, the nominal size of the free list varies depending on the amount of real memory installed. On systems with 64 MB of memory or more, the minimum value minfree is 120 pages. For systems with less than 64 MB, the value is two times the number of MB of real memory, minus eight. For example, a system with 32 MB would have a minfree value of 56 free pages.

    If the number of pages on the free list drops below minfree, the VMM will steal pages until the free list has been restored to the maxfree value, which is defined as minfree plus eight.

    When an application terminates, all of its working pages are immediately returned to the free list. Its persistent pages (files), however, remain in RAM and are not added back to the free list until they are stolen by the VMM for other programs. Persistent pages are also freed if the corresponding file is deleted.

    For this reason, the **fre** value may not indicate all the real memory that can be readily available for use by processes. If a page frame is needed, then persistent pages related to terminated applications are among the first to be handed over to another program.

    If the **fre** value is substantially above the maxfree value, then it is unlikely that the system is *thrashing*. Thrashing means that the system is continuously paging in and out. However, if the system is experiencing thrashing, you can be assured that the **fre** value will be small.

Thrashing

When memory is severely over-committed, the system spends most of its time dealing with page faults. It becomes difficult to choose pages for page out because they will be referenced again by currently running processes. The result is that pages that will soon be referenced get paged out anyway and are then paged in again. This condition is called *thrashing*. The system spends most of its time paging in and paging out instead of executing useful instructions. None of the active processes make any significant progress. The VMM has a *memory load control algorithm* (for details see 4.10, “The schedtune Command” on page 177) that detects when the system is thrashing and then attempts to correct the condition.

- **page**

  Information about page faults and paging activity. These are averaged over the interval and given in units per second.

  – **re**

    The number of page reclaims per second observed in the sample interval. If a page fault occurs and this page is currently on the free list and has not yet been reassigned, this is considered a reclaim since no new I/O request has to be initiated (the page is still in memory). It also includes pages previously requested by VMM for which I/O has not yet been completed or those pre-fetched by VMM’s *read-ahead mechanism* but hidden from the faulting segment. This is not to be confused with the term *repage* which refers to a page that has already incurred a page-fault (the page could be currently on the free list, filesystem, or in paging space).

    Note

    In AIX V4 the reclaiming is no longer supported. The algorithm for reclaims costs performance, and the value delivered does not give very useful information about the performance of the system.

  – **pi**

    The **pi** column details the number (rate) of pages paged in from paging space. Paging space is the part of virtual memory that resides on disk. It is used as an overflow when memory is over-committed. Paging space consists of logical volumes dedicated to the storage of working set pages that have been stolen from real memory. When a stolen page is referenced by the process, a page fault occurs, and the page must be read into memory from paging space.

    Due to the variety of configurations of hardware, software and applications, there is no absolute number to look out for. But five page-ins per second should be the upper limit. This theoretical maximum should not be rigidly adhered to, but used as a reference. This field is important as a key indicator of paging-space activity. Look at it this way: If a page-in occurs, then there must have been a previous page-out for that page. It is also likely in a memory-constrained environment that each page-in will force a different page to be stolen and, therefore, paged out. But systems could also work fine when they have near to 10 **pi**/s for 1 min and then work without any page-ins.

Note

The system slows down when **pi** and **po** are consistently non-zero.

– **po**

The po column shows the number (rate) of pages paged out to paging space. Whenever a page of working storage is stolen, it is written to paging space. If not referenced again, it will remain on the paging device until the process terminates or disclaims the space. Subsequent references to addresses contained within the faulted-out pages results in page faults, and the pages are paged in individually by the system. When a process terminates normally, any paging space allocated to that process is freed. If the system is reading in a significant number of persistent pages (files), you may see an increase in **po** without corresponding increases in **pi**. This does not necessarily indicate thrashing, but may warrant investigation into data access patterns of the applications.

Note

The system considers itself to be thrashing when,

po / fr > 1 / h

The *h* is a system parameter from the schedtune command (see 4.10, “The schedtune Command” on page 177 for details). The default value for *h* in AIX V4 is 6, if less than 128 MB of RAM are in the system. And it is 0, if the memory is greater than or equal to 128 MB. For AIX V3, the default value is 6.

– **fr**

Number of pages that were freed per second by the page-replacement algorithm during the interval. As the VMM page-replacement routine scans the Page Frame Table (PFT), it uses criteria to select which pages are to be stolen to replenish the free list of available memory frames. The criteria include both kinds of pages, working (computational) and file (persistent) pages. Just because a page has been freed, it does not mean that any I/O has taken place. For example, if a persistent storage (file) page has not been modified, it will not be written back to the disk. If I/O is not necessary, minimal system resources are required to free a page.

– **sr**

Number of pages that were examined per second by the page-replacement algorithm during the interval. The VMM page-replacement code scans the PFT and steals pages until the number of frames on the free list is at least the maxfree value. The page-replacement code may have to scan many entries in the PFT before it can steal enough to satisfy the free list requirements. With stable, unfragmented memory, the scan rate and free rate may be nearly equal. On systems with multiple processes using many different pages, the pages are more volatile and disjointed. In this scenario, the scan rate may greatly exceed the free rate.

Note

Memory is over-committed when the ratio of **fr** to **sr** (**fr:sr**) is high.

An **fr:sr** ratio of 1:4 means that for every page freed, four pages had to be examined. It is difficult to determine a memory constraint based on this ratio alone, and what constitutes a high ratio is workload/application dependent. In this case having taken a baseline measurement of the system, when all is fine, may help a lot.

– **cy**

Number of cycles per second of the clock algorithm. The VMM uses a technique known as the clock algorithm to select pages to be replaced. This technique takes advantage of a referenced bit for each page as an indication of what pages have been recently used (referenced). When the page-stealer routine is called, it cycles through the PFT, examining each page's referenced bit.

The **cy** column shows how many times per second the page-replacement code has scanned the PFT. Since the free list can be replenished without a complete scan of the PFT and because all of the vmstat fields are reported as integers, this field is usually zero. If not, it indicates a complete scan of the PFT, and the stealer has to scan the PFT again, because **fre** is still under the `maxfree` value.

The Page-Replacement Algorithm: The PFT includes flags to signal which pages have been referenced and which have been modified. If the page stealer encounters a page that has been referenced, it does not steal that page, but instead, resets the reference flag for that page. The next time the clock hand passes that page and the reference bit is still off, that page is stolen. A page that was not referenced in the first pass is immediately stolen.

The modify flag indicates that the data on that page has been changed since it was brought into memory. When a page is to be stolen, with the modify flag set, a page-out call is made before stealing the page. Pages that are part of working segments are written to paging space; persistent segments are written to disk.

In addition, the page-replacement algorithm keeps track of both new page faults (referenced for the first time) and repage faults (referencing pages that have been paged out) by using a history buffer that contains the IDs of the most recent page faults. It then tries to balance file (persistent data) page outs with computational (working storage or program text) page outs. Which kind of pages are paged out is defined by the `vmtune` parameters `maxperm` and `minperm`. For details, refer to 4.11, "The vmtune Command" on page 189.

A page fault is considered to be either a new page fault or a repage fault.

A new page fault occurs when there is no record of the page having been referenced recently.

Considerations: It is recommended to configure enough paging space on the system so that the paging space used does not approach 100 percent. For systems up to 256 MB of memory, the paging space should be twice the size of real memory. For memories larger than 256 MB, the following is recommended:

total paging space = 512 MB + (memory size - 256 MB) * 1.25

The paging space size can not be less than 16 MB and not greater than 20 percent of total disk space.

When fewer unallocated pages than the npskill value of the vmtune command (more details in 4.11, “The vmtune Command” on page 189) remain on the paging device, the system will begin to kill processes to free some paging space.

By default, the npskill value is calculated via:

MAX (64, number of paging space pages/128)

(for AIX V3 the value was 128). The ″Number of paging space pages″ value is shown in the svmon -G output for the parameter pg space size.

2.1.3 I/O Bound

To prove that the system is I/O bound, it is better to use the iostat command (for details see 2.2, “The iostat Command” on page 17). However vmstat could point to that direction by looking at the wa column, as discussed earlier in this chapter. Other indicators for I/O bound are:

- The **disk xfer** part of vmstat

  To display a statistic about the logical disks (a maximum of four disks is allowed), use the following command:

```
# vmstat hdisk0 hdisk1 1 8
kthr     memory             page              faults        cpu     disk xfer
----  ----------  ----------------------- ------------ ----------- ------
r  b   avm   fre  re  pi  po  fr  sr  cy  in   sy  cs  us sy id wa  1 2 3 4
0  0  3456 27743   0   0   0   0   0   0 131  149  28  0  1 99  0  0 0
0  0  3456 27743   0   0   0   0   0   0 131   77  30  0  1 99  0  0 0
1  0  3498 27152   0   0   0   0   0   0 153 1088  35  1 10 87  2  0 11
0  1  3499 26543   0   0   0   0   0   0 199 1530  38  1 19  0 80  0 59
0  1  3499 25406   0   0   0   0   0   0 187 2472  38  2 26  0 72  0 53
0  0  3456 24329   0   0   0   0   0   0 178 1301  37  2 12 20 66  0 42
0  0  3456 24329   0   0   0   0   0   0 124   58  19  0  0 99  0  0 0
0  0  3456 24329   0   0   0   0   0   0 123   58  23  0  0 99  0  0 0
```

  The **disk xfer** part provides the number of transfers per second to the specified physical volumes that occurred in the sample interval. One to four physical volume names can be specified. Transfer statistics are given for each specified drive in the order specified. This count represents requests to the physical device. It does not imply an amount of data that was read or written. Several logical requests can be combined into one physical request.

- The **in** column of the vmstat output

  Number of device interrupts per second observed in the interval.

```
# vmstat 1 12
kthr     memory             page              faults        cpu
----- -----------  ----------------------  ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 0  0  3544 28065   0   0   0   0    1   0 120   90  21  0  0 99  0
 0  0  3544 28065   0   0   0   0    0   0 125   64  26  0  0 99  0
 0  1  3586 26866   0   0   0   0    0   0 184 2529  51  5 22 11 62
 0  1  3586 26297   0   0   0   0    0   0 186 1257  28  0 16  0 84
 0  1  3586 25158   0   0   0   0    0   0 190 2464  35  5 21  0 74
 0  1  3586 24590   0   0   0   0    0   0 188 1257  27  0 15  0 85
 0  1  3586 23452   0   0   0   0    0   0 195 2480  39  1 29  0 70
 0  1  3586 23088   0   0   0   0    0   0 188 1380 105  4 22  0 74
```

```
 0  1  3586 23087   0   0   0   0    0   0 168 2593 276  9 57  0 34
 1  0  3544 23148   0   0   0   0    0   0 165 2004 269  6 44 25 25
 0  0  3544 23148   0   0   0   0    0   0 127   50  23  0  0 99  0
 0  0  3544 23148   0   0   0   0    0   0 129   66  22  0  0 99  0
```

This column shows the number of hardware or device interrupts (per second) observed over the measurement interval. Examples of interrupts are disk request completions and the 10 millisecond clock interrupt. Since the latter occurs 100 times per second, the **in** field is always greater than 100. But vmstat also provides a more detailed output about the system interrupts.

- The vmstat -i output

  The -i parameter displays the number of interrupts taken by each device since system startup. But, by adding the interval and optional the count parameter, the statistic since startup is only displayed in the first stanza; every trailing stanza is a statistic about the scanned interval.

```
# vmstat -i 1 2
priority level    type   count module(handler)
    0       0   hardware     0 i_misc_pwr(a868c)
    0       1   hardware     0 i_scu(a8680)
    0       2   hardware     0 i_epow(954e0)
    0       2   hardware     0 /etc/drivers/ascsiddpin(189acd4)
    1       2   hardware   194 /etc/drivers/rsdd(1941354)
    3      10   hardware 10589024 /etc/drivers/mpsdd(1977a88)
    3      14   hardware 101947 /etc/drivers/ascsiddpin(189ab8c)
    5      62   hardware 61336129 clock(952c4)
   10      63   hardware 13769 i_softoff(9527c)
priority level    type   count module(handler)
    0       0   hardware     0 i_misc_pwr(a868c)
    0       1   hardware     0 i_scu(a8680)
    0       2   hardware     0 i_epow(954e0)
    0       2   hardware     0 /etc/drivers/ascsiddpin(189acd4)
    1       2   hardware     0 /etc/drivers/rsdd(1941354)
    3      10   hardware    25 /etc/drivers/mpsdd(1977a88)
    3      14   hardware     0 /etc/drivers/ascsiddpin(189ab8c)
    5      62   hardware   105 clock(952c4)
   10      63   hardware     0 i_softoff(9527c)
```

Note

The output will differ from system to system, depending on hardware and software configurations. Check for high numbers in the count column and investigate why this module has to execute so many interrupts.

2.1.4 Summary Option

The summary (-s) option reports the absolute counts of various events since the system was booted. The vmstat -s command requires about 90 milliseconds of CPU time.

```
# vmstat -s
  2895207 total address trans. faults
   145740 page ins
    83203 page outs
     3436 paging space page ins
     7844 paging space page outs
        0 total reclaims
  1260991 zero filled pages faults
     5360 executable filled pages faults
```

```
 1171187 pages examined by clock
      35 revolutions of the clock hand
   72712 pages freed by the clock
   16984 backtracks
       0 lock misses
    1904 free frame waits
       0 extend XPT waits
   58771 pending I/O waits
  111921 start I/Os
  111921 iodones
12935949 cpu context switches
74569755 device interrupts
       0 software interrupts
       0 traps
54273766 syscalls
```

The recommended way of using these statistics is to run this command before and after a workload. Then determine the delta or difference between the two outputs. An awk script called vmstatit that does this automatically is provided in the *Performance Tuning Guide* publication, SC23-2365.

The **page ins** and **page outs** numbers in the summary represent virtual memory activity to page-in or page-out pages from page space and file space. The **paging space page ins** and **paging space page outs** are representative of only paging space. The four fields above could be used to indicate how much of the system′s I/O is for persistent storage. If the value for **paging space page ins** is subtracted from the (system-wide) value for **page ins**, the result will be the number of pages that were read from persistent storage. Likewise, if the value for **paging space page outs** is subtracted from the (system-wide) value for **page outs**, the result will be the number of persistent pages that were written to disk.

If the system is paging too much, using vmtune may help. Creating separate paging spaces on separate volumes may provide some benefit, but increasing the memory would definitely help.

2.1.5 Conclusion

The vmstat command is only the first step to look for performance problems. It gives an indication where the performance problem could be located. With this in mind, choose a resource-specific command and take a deeper look into the system behavior.

2.2 The iostat Command

The iostat command is the fastest way to get a first impression, whether the system has an I/O-bound performance problem or not. The tool reports CPU statistics and I/O statistics for TTY devices, disks, and CD-ROMs. It is used for monitoring system I/O device utilization by observing the time the physical disks are active in relation to their average transfer rates. The iostat command is useful to determine whether a physical volume is becoming a performance bottleneck and if there is a way to improve the situation. The generated reports can be used to change system configurations to better balance the I/O load between physical disks.

The following example shows a part of an iostat output. The first stanza shows the summary statistic since system startup. During the report, there was a copy command started.

```
# iostat 2 2

tty:      tin         tout   avg-cpu:  % user    % sys     % idle    % iowait
          0.0         13.3              0.2       0.2       99.4       0.2

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk0           0.2       2.4       0.2    1192730    706531
hdisk1           0.0       0.3       0.0      60479    190316
cd0              0.0       0.0       0.0          0         0

tty:      tin         tout   avg-cpu:  % user    % sys     % idle    % iowait
          0.0        208.0              0.0      11.5        0.0      88.5

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk0           35.5     848.0      76.0       1696         0
hdisk1           74.5     790.0      58.0          4      1576
cd0              0.0       0.0       0.0          0         0
```

The iostat command works by sampling the kernel's address space and extracting data from various counters that are updated every clock tick (1 clock tick = 10 milliseconds). The results (covering TTY, CPU, and I/O subsystem activity) are reported as per-second rates or as absolute values for the specified interval.

Since the CPU utilization statistics are also available with the iostat report, the percentage of time the CPU is in I/O wait can be determined at the same time.

The system maintains a history of disk activity by default. Note that if the history is disabled, you see this message:

```
# iostat 1 2

tty:      tin         tout   avg-cpu:  % user    % sys     % idle    % iowait
          0.0          8.3              0.4       0.9       98.6       0.1
               " Disk history since boot not available. "


tty:      tin         tout   avg-cpu:  % user    % sys     % idle    % iowait
          0.0        411.9              3.0      22.8        0.0      74.3

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk0           25.7      15.8       3.0         16         0
hdisk1           72.3     2356.4      46.5          4      2376
cd0               0.0       0.0       0.0          0         0
```

This message is displayed instead of the first stanza, which is normally a summary since system startup. Disk I/O history must be enabled to get useful disk I/O information since startup. The interval disk I/O statistics are unaffected by this.

Disk I/O history should be enabled since the CPU resource used in maintaining it is insignificant. History keeping can be disabled or enabled in SMIT under the following path, or use the fastpath smit chgsys:

```
-> System Environments
-> Change/Show Characteristics of Operating System
-> Continuously maintain DISK I/O history
```

Choose **true** to enable history keeping or **false** to disable it.

The iostat command adds little overhead to the system. It uses about 20 milliseconds of CPU time for each report generated.

Note

If the iostat command is used without interval and, optionally, count, then the output is a summary since startup. If the command is used with an interval and count parameter, like iostat 2 5, then only the first stanza is an average since system reboot; the trailing stanzas are interval statistics. The first stanza should be ignored when doing real-time measurements because the history is usually not of interest in this case.

2.2.1 TTY Report

The two columns of TTY information (**tin** and **tout**) in the iostat output show the number of characters read and written by all TTY devices. This includes both real and pseudo TTY devices. Real TTY devices are those connected to an asynchronous port. Some pseudo TTY devices are shells, telnet sessions, and aixterms.

During the following example, the lptest command was started, which causes a higher value for **tout**.

```
# iostat -t 2 6

tty:      tin         tout   avg-cpu:  % user    % sys     % idle    % iowait
          0.0         13.2              0.2       0.2       99.4       0.2
          0.0         81.0              0.0       0.5       98.0       1.5
          0.5       8153.0               0.5       3.5       95.5       0.5
          1.5       8159.5               2.0       2.5       95.5       0.0
          0.0         41.0              0.0       0.0      100.0       0.0
          0.5         40.5              0.0       2.5       93.5       4.0
```

- **tin**

  Shows the total characters per second read by all TTY devices.

- **tout**

  Indicates the total characters per second written to all TTY devices.

Since the processing of input and output characters consumes CPU resources, look for a correlation between increased TTY activity and CPU utilization. If such a relationship exists, evaluate ways to improve the performance of the TTY subsystem. Steps that could be taken include changing the application program, modifying TTY port parameters during file transfer, or perhaps upgrading to a faster or more efficient asynchronous communications adapter.

In InfoExplorer or in the *Performance Tuning Guide*, SC23-2365, you can find the fastport.s script which is intended to condition a TTY port for fast file transfers in raw mode; for example, when a FAX machine is to be connected. Using the script may improve CPU performance by a factor of 3 at 38400 baud.

2.2.2 CPU Report

The CPU statistics columns (% user, % sys, % idle, and % iowait) provide a breakdown of CPU usage. This information is also reported in the vmstat command output in the columns labeled us, sy, id, and wa. For a detailed explanation for the values, see 2.1, “The vmstat Command” on page 7.

In general, a high % iowait indicates that the system has a memory shortage or an inefficient I/O subsystem configuration. Understanding the I/O bottleneck and improving the efficiency of the I/O subsystem requires more data than iostat can provide. Some typical solutions might include:

- Limiting number of active logical volumes and file systems placed on a particular physical disk. The idea is to balance file I/O evenly across all physical disk drives.
- Spreading a logical volume across multiple physical disks. This is particularly useful when a number of different files are being accessed.
- Creating multiple Journaled File Systems (JFS) logs for a volume group and assigning them to specific file systems. This is beneficial for applications that create, delete, or modify a large number of files, particularly temporary files.
- Backing up and restoring file systems to reduce fragmentation. Fragmentation causes the drive to seek excessively and can be a large portion of overall response time.
- Adding additional drives and rebalancing the existing I/O subsystem.

On systems running a primary application, high I/O wait percentage may be related to the workload. On systems with many processes, some will be running while others wait for I/O. In this case, the % iowait can be small or zero because running processes ″hide″ some wait time. Although % iowait is low, a bottleneck may still limit application performance.

If iostat indicates that a CPU-bound situation does not exist, and % iowait time is greater than 25 percent, you have an I/O or disk-bound situation. Now this may be caused by excessive paging due to a lack of real memory. It could also be due to unbalanced disk load, fragmented data or usage patterns. For an unbalanced disk load, the same iostat report provides the necessary information. But for information about file systems or logical volumes, which are logical resources, you have to use an AIX-specific tool like filemon or fileplace.

2.2.3 Drive Report

When suspecting a disk I/O performance problem, the iostat command should be used. To avoid the information about the TTY and CPU statistics, the option -d can be used. In addition, the disk statistics can be limited to the important ones.

A customized, disk-specific iostat output looks like this:

```
# iostat -d hdisk0 hdisk1 2 9

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk0           0.1       1.4       0.1     767706     240078
hdisk1           0.0       0.3       0.0      58282     137712
hdisk0           3.5      10.0       2.0         12          8
hdisk1           0.0       0.0       0.0          0          0
```

```
hdisk0          0.0       0.0       0.0          0         0
hdisk1         39.8    1277.6      24.4          0      2568
hdisk0          0.0       0.0       0.0          0         0
hdisk1        100.0    3424.0      55.0          0      6848
hdisk0          0.0       0.0       0.0          0         0
hdisk1        100.0    3422.0      53.5          0      6844
hdisk0          0.0       0.0       0.0          0         0
hdisk1         65.0    1704.0      42.5          0      3408
hdisk0          0.0       0.0       0.0          0         0
hdisk1         19.0      72.0      17.5          0       144
hdisk0          0.0       0.0       0.0          0         0
hdisk1          0.0       0.0       0.0          0         0
hdisk0          0.0       0.0       0.0          0         0
hdisk1          0.0       0.0       0.0          0         0
```

Starting in the third interval set, there was some heavy I/O activity on `hdisk1`. In the seventh interval set, the I/O activity goes down and then back to the initial values.

Remember that the first set of data represents all activity since system startup.

- **Disks:**

  Shows the names of the physical volumes. They are either `hdisk` or `cd` followed by a number. If physical volume names are specified with the `iostat` command, only those names specified are displayed.

- **% tm_act**

  Indicates the percentage of time the physical disk was active. A drive is active during data transfer and command processing, such as seeking to a new location. The ″disk active time″ percentage is directly proportional to resource contention and inversely proportional to performance. As disk use increases, performance decreases and response time increases. In general, when the utilization exceeds 40 percent, processes are waiting longer than necessary for I/O to complete because most UNIX processes block (or sleep) while waiting for their I/O requests to complete.

  Look for busy vs. idle drives. Moving data from busy to idle drives may help alleviate a disk bottleneck. Paging to and from disk will contribute to the I/O load.

- **Kbps**

  Indicates the amount of data transferred (read or written) to the drive in KB per second. This is the sum of `Kb_read` plus `Kb_wrtn`, divided by the seconds in the reporting interval.

- **tps**

  Indicates the number of transfers per second that were issued to the physical disk. A transfer is an I/O request via the device driver level to the physical disk. Multiple logical requests can be combined into a single I/O request to the disk. A transfer is of indeterminate size.

- **Kb_read**

  Reports the total data (in KB) read from the physical volume during the measured interval.

- **Kb_wrtn**

Shows the amount of data (in KB) written to the physical volume during the measured interval.

Taken alone, there is no unacceptable value for any of the above fields because statistics are too closely related to application characteristics, system configuration, and type of physical disk drives and adapters. Therefore, when evaluating data, look for patterns, and relationships. The most common relationship is between disk utilization (`%tm_act`) and data transfer rate (`tps`).

To draw any valid conclusions from this data, you have to understand the application's disk data access patterns such as sequential, random, or combination, and the type of physical disk drives and adapters on the system.

For example, if an application reads/writes sequentially, you should expect a high disk transfer rate (`tps`) when you have a high disk busy rate (`%tm_act`). `Kb_read` and `Kb_wrtn` can confirm an understanding of an application's read/write behavior. However, they provide no information on the data access patterns.

Generally you do not need to be concerned about a high disk busy rate (`%tm_act`) as long as the disk transfer rate (`tps`) is also high. However, if you get a high disk busy rate and a low disk transfer rate, you may have a fragmented logical volume, file system, or individual file.

An average physical volume utilization greater than 25 percent across all disks indicates an I/O bottleneck.

Discussions of disk, logical volume and file system performance sometimes lead to the conclusion that the more drives you have on your system, the better the disk I/O performance. This is not always true since there is a limit to the amount of data that can be handled by the SCSI adapter. The SCSI adapter can also become a bottleneck.

If all your disk drives are on one SCSI, and your hot file systems are on separate physical volumes, you may benefit from using multiple SCSI adapters. Performance improvement will depend on the type of access.

To see if a particular adapter is saturated, use the `iostat` command and add up all the `Kbps` amounts for the disks attached to a particular SCSI adapter. For maximum aggregate performance, the total of the transfer rates (`Kbps`) must be below the SCSI adapter throughput rating. In most cases, use 70 percent of the throughput rate. This results in:

- SCSI-1 rate of 3.5 MB/s (70 percent of 5 MB/s)
- SCSI-2 rate of 7 MB/s (70 percent of 10 MB/s)

The *Performance Tuning Guide*, SC23-2365, provides in *Chapter 8: Monitoring and Tuning Disk I/O*, additional and detailed information, like:

- Sequential Read Ahead
- Disk-I/O Pacing
- Striping
- Asynchronous Disk I/O
- Raw Disk I/O
- Setting SCSI-Adapter and Disk-Device Queue Limits

Note

Like vmstat, iostat can only give a first indication about a performance bottleneck. The system administrator will have to use more complex tools like filemon to identify the source of the slowdown.

2.3 The sar Command

The sar command is a standard UNIX command used to gather statistical data about the system. Though it can be used to gather some useful data regarding system performance, the sar command is somewhat intrusive and can increase the system load which will exacerbate a pre-existing performance problem.

With its numerous options, sar provides queuing, paging, TTY, and many other statistics. One important new feature of the sar command on RS/6000 is that it reports either systemwide (global among all processors) CPU statistics (which are calculated as averages for values expressed as percentages, and as sums otherwise), or it reports statistics for each individual processor. Therefore, this command is particularly important on SMP systems.

Note

The sar command reports only on local activities; no statistics about network activities are included. The execution is limited to the root user. All data types that sar supports are collected; however, only the data types specified by options are reported.

There are three possibilities to use the sar command:

- Real-time sampling and display
- Display previously captured data
- System activity accounting via cron

Compared to the accounting package, the general use of sar is less intrusive. The system maintains a series of system activity counters that record various activities and provide the data that sar reports. The sar command does not cause these counters to be updated or used; this is done automatically regardless of whether sar runs or not. The command merely extracts the data in the counters and saves them based on the sampling rate and number of samples specified to sar.

The sadc command is intended to be used as a back-end to the sar command. This data collector samples system data a specified number of times at a specified interval measured in seconds. It writes in binary format to the specified outfile or to standard output. The binary file holds all the data sar could show.

2.3.1 Real-Time Sampling and Display

To collect and display system statistic reports immediately, use:

```
# sar -u 2 5

AIX ah6000d 2 4 000002583800    02/21/97

17:58:15    %usr    %sys    %wio   %idle
17:58:17      43       9       1      46
17:58:19      35      17       3      45
17:58:21      36      22      20      23
17:58:23      21      17       0      63
17:58:25      85      12       3       0

Average       44      15       5      35
```

This example is from a single user workstation and shows the CPU utilization.

2.3.2 Display of Previously Captured Data

The -o and -f options (write and read to/from user given data files) allow you to visualize the behavior of your machine in two independent steps. This is less resource consuming during the problem-reproduction period. The analysis of the data can be done on a separate machine by transferring the file because the collected binary file keeps all data sar needs.

```
# sar -o /tmp/sar.out 2 5 > /dev/null
```

The redirection of standard output is used to avoid a screen output.

```
# sar -f/tmp/sar.out

AIX ah6000d 2 4 000002583800    02/21/97

18:10:18    %usr    %sys    %wio   %idle
18:10:20       9       2       0      88
18:10:22      13      10       0      76
18:10:24      37       4       0      59
18:10:26       8       2       0      90
18:10:28      20       3       0      77

Average       18       4       0      78
```

The captured binary data file keeps all information needed for the reports. So every possible sar report could be investigated. This allows also to display the processor-specific information of an SMP system on a single processor system.

2.3.3 System Activity Accounting via cron

The sar command calls a process named sadc to access system data. Two shell scripts (/usr/lib/sa/sa1 and /usr/lib/sa/sa2) are structured to be run by the cron daemon and provide daily statistics and reports. Sample stanzas are included (but commented out) in the /var/spool/cron/crontabs/adm crontab file to specify when the cron daemon should run the shell scripts.

The following lines show a modified crontab for the adm user. Only the comment characters for the data collections were removed:

```
#==================================================================
#      SYSTEM ACTIVITY REPORTS
#  8am-5pm activity reports every 20 mins during weekdays.
#  activity reports every an hour on Saturday and Sunday.
#  6pm-7am activity reports every an hour during weekdays.
#  Daily summary prepared at 18:05.
#==================================================================
0 8-17 * * 1-5 /usr/lib/sa/sa1 1200 3 &
0 * * * 0,6 /usr/lib/sa/sa1 &
0 18-7 * * 1-5 /usr/lib/sa/sa1 &
5 18 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:01 -i 3600 -ubcwyaqvm &
#==================================================================
```

Collection of data in this manner is useful to characterize system usage over a period of time and to determine peak usage hours.

The following example shows how sar tries to access the pre-collected data, which wasn't available (sar searches for the /var/adm/sa/sa<nn> file, where nn is the number of the current day). After executing the sa1 script, sar could display the accounted data from the script that was previously run:

```
# sar
Can't open /var/adm/sa/sa19
Try running /usr/lib/sa/sa1 <inc> <num>
# date
Wed Feb 19 17:51:14 CST 1997
# usr/lib/sa/sa1 1 5
# sleep 600
# date
Wed Feb 19 18:01:44 CST 1997
# sar

AIX ah6000e 2 4 000252195700    02/19/97

17:51:23    %usr    %sys    %wio   %idle
17:51:24       2       4       0      94
17:51:25       0       3       0      97
17:51:26       0       3       0      97
17:51:27       0       3       0      97

Average        0       3       0      96
```

In this example the first collected interval is missing because it is used as the start-up dummy record. Instead of the requested five counts, only four samples are available. But all following sa1 collections will be completely entered into the /var/adm/sa/sa<nn> files.

The /var/adm/sa/sar<nn> files, which are generated by the sa2 script, are not automatically printed, but they keep the ASCII version of the daily sa1 statistic, which is collected in the binary file /var/adm/sa/sa<nn>. Be aware of the small difference in the collection file names; it's only an ″r″ which makes the difference.

You will probably want to modify the sampling rate of the sa1 script (in the crontab of user adm) for prime-time usage. The default rate is every 1200 seconds, that is, every 20 minutes. This is a very small sampling rate to use for evaluating system activity. Of course, a higher sampling rate will generate larger data files. An extremely high sampling rate (more than once a minute) is

probably only useful in special situations and not appropriate for everyday use. Additional information about the sadc, sa1 and sa2 commands is available in the *AIX Version 4 Commands Reference*, SBOF-1851.

Note

If sar is used without an interval, number, or the -f option, it always tries to display the pre-collected data of the current day. Any user can display the pre-collected data, no special user/group permission is needed.

2.3.4 Useful Options

Some of the most useful options for sar are:

sar -P: The -P option reports per-processor statistics for the specified processor or processors. By specifying the ALL keyword, statistics for each individual processor and an average for all processors is reported. Of the flags which specify the statistics to be reported, only the -a, -c, -m, -u, and -w flags are meaningful with the -P flag.

The following example shows the per-processor statistic while a CPU-bound program was running on processor number 0:

```
# sar -P ALL 2 3

AIX itsosmp 2 4 00045067A000    02/23/97

17:30:50 cpu    %usr    %sys    %wio   %idle
17:30:52 0        8      92       0       0
         1        0       4       0      96
         2        0       1       0      99
         3        0       0       0     100
         -      2     24       0     74
17:30:54 0       12      88       0       0
         1        0       3       0      97
         2        0       1       0      99
         3        0       0       0     100
         -      3     23       0     74
17:30:56 0       11      89       0       0
         1        0       3       0      97
         2        0       0       0     100
         3        0       0       0     100
         -      3     23       0     74

Average  0       10      90       0       0
          1        0       4       0      96
          2        0       1       0      99
          3        0       0       0     100
          -        3      24       0      74
```

The last line of every stanza, which starts with a dash (-) in the cpu column, is the average for all processors. There is only an average (-) line if the -p ALL option is used. It is removed if processors are specified. The last stanza, labeled with the word Average instead of a time stamp, keeps the averages for the processor specific rows over all stanzas. If the ALL keyword is used, its last row, starting also with a dash (-), represents the average over all processors for the investigated period.

The following example shows the vmstat output during this time:

```
# vmstat 2
kthr     memory             page              faults        cpu
-----  ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs  us sy  id wa
 0  0  5636 16054   0   0   0   0    0   0 116  266   5   0  1  99 0
 1  1  5733 15931   0   0   0   0    0   0 476 50781  35  2 27  70 0
 1  1  5733 15930   0   0   0   0    0   0 476 49437  27  2 24 74 0
 1  1  5733 15930   0   0   0   0    0   0 473 48923  31  3 23 74 0
 1  1  5733 15930   0   0   0   0    0   0 466 49383  27  3 23 74 0
```

The first numbered line is the summary since startup of the system. The second line reflects the start of the sar command, and with the third row, the reports are comparable. The sar command was used to report the CPU utilization; this information is also displayed in the cpu section of the vmstat. As explained in 2.1, “The vmstat Command” on page 7, the vmstat can only display the average CPU utilization over all processors. This is comparable with the dashed (-) rows from the CPU utilization output from the sar command.

sar -u: This displays the CPU utilization. It is the default if no other flag is specified.

During the following example, a copy command was started:

```
# sar -u -P ALL 1 5

AIX itsosmp 2 4 00045067A000    02/20/97

18:24:57 cpu    %usr    %sys    %wio   %idle
18:24:58  0        0       9       0      91
          1        0       0       0     100
          2        0       0       0     100
          3        0       0       0     100
          -        0       2       0      98
18:24:59  0        0       7      64      28
          1        2      65       1      32
          2        0       6      58      36
          3        0       0      64      36
          -        0      20      47      33
18:25:00  0        1       8      92       0
          1        3      78      20       0
          2        0       1      99       0
          3        0       3      97       0
          -        1      22      77       0
18:25:02  0        0       7      93       0
          1        0       8      92       0
          2        0       1      99       0
          3        0       0     100       0
          -        0       4      96       0
18:25:03  0        0       6      36      58
          1        0       0      42      58
          2        0       1      41      58
          3        0       0      42      58
          -        0       2      40      58

Average   0        0       7      57      35
          1        1      30      31      38
          2        0       2      59      39
          3        0       1      61      39
```

```
-          0       10       52       38
```

As stated in 2.1.1, “CPU Bound” on page 7, the system has no possibility to specify which processor has a pending local I/O request and which does not. The system looks for the `wait` process and if a global variable for local disk activity is set or not. This results in the following queries for every processor:

- If the `wait` process is running and there is no local disk I/O, the time is charged as *idle time*.
- If the `wait` process is running and there is a local disk I/O (regardless for which processor), the time is charged as *wait time*.

The `cp` command is working on processor number 1, and the three other processors should be idle. But since processor number 1 has a pending local I/O request, all idle processors get charged for `wait` time.

sar -A: To display all possible statistics, use the `-A` option. Without the `-P` flag, this is equivalent to specifying `-a`, `-b`, `-c`, `-k`, `-m`, `-q`, `-r`, `-u`, `-v`, `-w` and `-y`. With the `-P` flag, this is equivalent to specifying `-a`, `-c`, `-m`, `-u` and `-w` (which are the useful options for SMP systems).

sar -b: Reports buffer activity for transfers, accesses, and cache (kernel block buffer cache) hit ratios per second. Access to most files in AIX V3 and AIX V4 bypasses kernel block buffering and therefore does not generate these statistics. However, if a program opens a block device or a raw character device for I/O, traditional access mechanisms are used, making the generated statistics meaningful.

These values are usually zero. If they are not zero, then the ″Maximum number of pages in block I/O BUFFER CACHE″ (`maxbuf`) should be tuned. This can be done under the following path in SMIT or with the fast path `smit chgsys`:

```
-> System Environments
-> Change/Show Characteristics of Operating System
-> Maximum number of pages in block I/O BUFFER CACHE
```

sar -c: A report about system calls is displayed by the `-c` option. When used with the `-P` flag, the information is provided for each specified processor; otherwise, it provides only systemwide information.

```
# sar -c 1 5

AIX ah6000e 2 4 000252195700    02/20/97

20:45:02 scall/s sread/s swrit/s  fork/s  exec/s rchar/s wchar/s
20:45:03     203      16       7    0.00    0.00  358749    1810
20:45:04     300      32      26    9.90    1.98  360289    2269
20:45:05      28      13       3    0.00    0.00  362274    1739
20:45:06      28      13       3    0.00    0.00  358687    1722

Average      140      18      10    2.48    0.50  359994    1885
```

If an application is not well programmed, it may create many forks and/or execs per second, which slows down the system. So look out for high numbers in these columns.

Timing Intervals: Usually an interval and a numeric argument are specified on the command line when using sar. They specify the time between the readings and how many readings to take. The interval specified at extraction time can differ from the one specified at collection time. As a general rule, there is no harm in collecting more data than you extract. Conversely, if data is collected at a 20-minute interval (1200 seconds), it would be difficult to extract information for a 10-minute period; not even one reading may coincide with the desired period in the collection data file. Also, it is possible to extract data for a shorter time period than that over which it was collected. This is done using the -s and -e options:

-s hh:mm	start time
-e hh:mm	end time

These options also work for the precaptured data (from the sa1 script). The following example extracts a part of the /var/adm/sa/sa<nn> file that the sa1 script had created by collecting data every 20 minutes:

```
# sar -s 14:30 -e 16:30

AIX ah6000d 2 4 000002583800    02/24/97

14:40:01    %usr    %sys    %wio   %idle
15:00:01      14       3       0      83
15:20:01      15       5       1      80
15:40:01      10       1       0      88
16:00:01      17       2       0      80
16:20:01      16       3       0      81

Average       14       3       0      83
```

For more detailed information about all the sar options and the meaning of their output, please refer to the *AIX Version 4 Commands Reference*, SBOF-1851.

2.3.5 Correlation Between vmstat, iostat, and sar

A lot of the data obtained with sar can also be obtained with vmstat and iostat without any additional strain on the system. Another advantage of vmstat and iostat over sar is that root permission is not required.

sar -q: Similar info as in the kthr section of the vmstat command.

runq-sz same as r column in vmstat.

swpq-sz same as b column in vmstat.

sar -r: Similar info as parts of the memory and page sections of the vmstat command, where:

slots stands for size of paging space minus the avm value (or minus the pg space inuse value of the svmon -G command).

cycle/s is same as the cy column.

odio/s reports the number of non-paging disk I/Os per second. This could be calculated by the vmstat -s output.

sar -u: Same info as the cpu section of the vmstat and iostat commands.

sar -w: Same info as the faults section of the vmstat command.

pswch/s same as cs column in vmstat (AIX V3).

cswch/s same as cs column in vmstat (AIX V4).

sar -y: Similar info as that in the tty section of the iostat command.

outch/s is a per-second rate of the tout column.

rawch/s is a per-second rate of the tin column.

tin and tout are averaged over the interval used with iostat.

2.3.6 The timex Command

The timex -s cmd command reports total system activity during the execution of the cmd command. All the data types listed in the sar command are reported.

While sar collects aggregate data for all the processes running on the system at a given instant, the timex -s command will limit the data reported to an individual command, providing most but not all possible sar output.

2.4 The ps Command

The ps command is a very flexible tool for identifying the programs that are running on the system and the resources they are using. It displays statistics and status information about processes on the system, such as process or thread ID, I/O activity, CPU and memory utilization. In this chapter, we only discuss the options and output fields that are relevant for CPU and memory consumption. For all other options and columns, see the *AIX Version 4 Commands Reference*, SBOF-1851.

The CPU time consumed by the ps command varies with the number of processes to be displayed, but usually does not exceed 0.3 seconds.

2.4.1 CPU Information

The ps command, run periodically, will display the CPU time under the TIME column and the ratio of CPU time to real time under the %CPU column. Look for the processes that dominate usage. The au and v options give similar information on user processes. The options aux and vg display both user and system processes.

The following example is taken from a four-way SMP system:

```
# ps au
USER       PID %CPU %MEM   SZ  RSS    TTY STAT    STIME TIME COMMAND
root     19048 24.6  0.0   28   44  pts/1 A    13:53:00 2:16 /tmp/cpubound
root     19388  0.0  0.0  372  460  pts/1 A      Feb 20 0:02 -ksh
root     15348  0.0  0.0  372  460  pts/4 A      Feb 20 0:01 -ksh
root     20418  0.0  0.0  368  452  pts/3 A      Feb 20 0:01 -ksh
root     16178  0.0  0.0  292  364      0 A      Feb 19 0:00 /usr/sbin/getty
root     16780  0.0  0.0  364  392  pts/2 A      Feb 19 0:00 -ksh
root     18516  0.0  0.0  360  412  pts/0 A      Feb 20 0:00 -ksh
root     15746  0.0  0.0  212  268  pts/1 A    13:55:18 0:00 ps au
```

The %CPU is the percentage of CPU time that has been allocated to that process since the process was started. It is calculated as follows:

(process CPU time / process duration) * 100

Just imagine two processes: The first starts and runs five seconds, but does not finish; then the second starts and runs five seconds but does not finish. The ps command would now show 50 percent %CPU for the first process (five seconds CPU for 10 seconds of elapsed time) and 100 percent for the second (five seconds CPU for five seconds of elapsed time).

On an SMP, this value is divided by the number of available CPUs on the system. Looking back at the previous example, this is the reason why the %CPU value for the cpubound process will never cross 25. It is a four-processor system. The cpubound process uses 100 percent of one processor, but the %CPU value is divided by the number of available CPUs.

Note

It is normal to see a process named kproc (PID of 514 in AIX V3 or PID of 516 in AIX V4) using CPU time. When there are no threads that are runable during a time slice, the scheduler assigns the CPU time for that time slice to this kernel process (kproc), which is known as the *idle* or *wait* kproc. SMP systems will have an *idle* kproc for each processor.

2.4.2 Memory Information

The ps command can also be used to monitor memory usage of individual processes. The ps v <pid> command provides the most comprehensive report on memory-related statistics for an individual process, such as page faults, size of working segment that has been touched, size of working segment and code segment in memory, size of text segment, size of resident set, and percentage of real memory used by this process.

```
# ps v 12740
   PID    TTY STAT  TIME PGIN  SIZE   RSS   LIM  TSIZ  TRS %CPU %MEM COMMAND
 12740      - A     2:54  873  2052  2144 32768   483   344  0.0  3.0 dtwm
```

The following information is provided:

PID	The process ID of the process.
TTY	The controlling workstation for the process.
STAT	Contains the state of the process. In AIX V3 the STAT column showed R for runable; V4 now displays A for active instead.
TIME	The total execution time for the process.
PGIN	Number of page-ins caused by page faults. Since all AIX I/O is classified as page faults, this is basically a measure of I/O volume.
SIZE	Size of working segment that has been touched. The virtual size of the data section of the process (in 1-KB units).
RSS	Sum of the size of working segment and code-segment in memory. The real memory (resident set) size of the process (in 1-KB units).
LIM	The soft limit on memory. If no limit has been specified, then shown as xx. If the limit is set to the system limit (unlimited), a value of UNLIM is displayed.
TSIZ	Size of the text (shared-program) image.
TRS	The size of resident-set (real memory) of text.

%CPU The percentage of time the process has used the CPU since the process started.

%MEM The percentage of real memory used by this process. The RSS value divided by the memory size in KB, times 100, rounded to the nearest full percentage point.

COMMAND Contains the command name.

2.4.3 New ps Options

With AIX V4, new options and suboptions have been added to ps. The -m option allows you to see information related to threads, and -o THREAD provides extra columns with thread-specific information. With both options (-mo THREAD), you can see the thread or threads that belong(s) to each process.

```
# ps -mo THREAD
USER  PID   PPID    TID ST   CP PRI SC  WCHAN     F     TT BND  COMMAND
root 17856  19388     -  A   13  66   1   -   200001  pts/1  -   ps -mo THREAD
 -     -      -   19657 R   13  66   1   -       0     -    -   -
root 19048  19388     -  A  120 120   0   -  8200011  pts/1  3   /tmp/cpubound
 -     -      -   18801 R  120 120   0   -       0     -    3   -
root 19388   4538     -  A    1  60   1   -   240001  pts/1  -   -ksh
 -     -      -   19141 S    1  60   1   -      400     -   -   -
root 19628  19388     -  A  120 124   1   -   200001  pts/1  2   /tmp/cpubound
 -     -      -   20661 R  120 124   1   -       0     -   2   -
```

The **TID** column shows the thread ID, and the **BND** column shows the processes and threads bound to a processor.

On a uniprocessor system, the **BND** column will always list 0 since all threads are of course bound to just that one processor. On an SMP system, the value under **BND** will be - (dash) if the thread is not explicitly bound. If the value is an integer, then that indicates the processor number to which the thread is bound. You can add the options -e and -k to see all the threads on the system. The -e option lists all processes, except kernel processes, and -k lists the kernel processes.

Note

Remember, while the -m flag reports threads associated with processes using extra lines, you have to use the -o flag with the THREAD keyword to display these extra thread-related columns.

To display a thread or threads of a particular process, use:

```
# ps -mo THREAD -p 19048
USER   PID  PPID    TID ST  CP PRI SC   WCHAN      F     TT BND COMMAND
root 19048 19388     -  A  99 109  1      -  8200011  pts/1   3  /tmp/cpubound
  -     -     -   18801 R  99 109  1      -        0      -   3  -
```

2.4.4 Useful Shell Scripts

The top 10 processes that have accumulated the most CPU time:

```
# ps -e|head -n 1;ps -e|egrep -v "TIME|0:"|sort +2b -3 -n -r|head -n 10
```

The top 10 processes that have the most recent CPU usage:

```
# ps -ef|head -n 1;ps -ef|egrep -v "C|0:00| 0 "|sort +3b -4 -n -r|head -n 10
```

The top 10 processes that have the most CPU usage:

```
# ps gu|head -n 1;ps gu|egrep -v "CPU|kproc"|sort +2b -3 -n -r|head -n 10
```

The ps command only takes a snapshot. To gather data over a time period, use the tprof command.

2.5 The pstat Command

The pstat command is a non-interactive form of the crash command. It interprets the contents of the various system tables and writes it to standard output. You must have root user or system group authority to run the pstat command. Because it has a number of new options, pstat is useful for looking at threads, especially on SMP systems.

pstat -S: Shows processor status, which thread is running on which processor at the time of the command. A repetition of this command would allow you to determine the relative affinity of the threads with each processor.

Processor affinity is the dispatching of a thread to a processor that was previously executing it. The degree of emphasis on processor affinity should vary directly with the size of the thread's cache working set and inversely with the length of time since it was last dispatched.

The AIX V4 dispatcher has been modified to enforce affinity with the processors; so affinity is done implicitly by the operating system. More is said about processor affinity in 4.9, "The bindprocessor Command" on page 173.

The information displayed from pstat -S includes the processor number, kernel thread identifier, kernel thread table slot, process identifier, process table slot, and process name.

```
# pstat -S
STATUS OF PROCESSORS:

CPU     TID  TSLOT     PID  PSLOT  PROC_NAME
  0      46fd      70     40f4      64   eatmem
  1      4701      71     49f8      73   crash
  2      4973      73     4b6a      75   cpubound
  3      4df5      77     4eec      78   telnetd
```

Note

The thread ID and process ID are displayed in hexadecimal.

It is pretty hard to see processor affinity with the pstat -S command, but it could be done by picking a thread ID of a running thread and checking every second on which process this thread is running. This script would do it:

```
while true
do
    pstat -S | grep <TID>
    sleep 1
done
```

pstat -P: Shows runable kernel threads only.

```
# pstat -P
THREAD TABLE:


SLT ST   TID    PID  CPUID POLICY PRI CPU EVENT  PROCNAME     FLAGS
  2 r     205     204      0    FIFO  7f  78           wait
       t_flags: sigslih kthread
  3 r     307     306      1    FIFO  7f  78           wait
       t_flags: sigslih kthread
  4 r     409     408      2    FIFO  7f  78           wait
       t_flags: sigslih kthread
  5 r     50b     50a      3    FIFO  7f  78           wait
       t_flags: sigslih kthread
 69 r    4579     4a70     2    other 62  44           cpubound
       t_flags:
 72 r    4837    492e unbound  other  61  4a           crash
       t_flags:
 73 r    4973    4b6a     3    other  62  44           cpubound
       t_flags:
 76 r    4cd7    4dce unbound  other  62  45           cpubound
       t_flags:
```

This example is taken from a four-way SMP system. The first four threads are the *wait* kproc, which is scheduled on a processor, if no other process wants to run. And the last four threads are the running threads. Explanation about the columns:

SLT Shows the slot number in the thread table.

ST Shows the status of the thread as to whether it is runable, sleeping or otherwise.

TID The thread ID (in hexadecimal) of the kernel thread.

PID The process ID (in hexadecimal) to which the thread belongs.

CPUID The ID of the processor on which the process is bound. If the thread is not bound, then the word unbound is displayed in this field.

POLICY The thread's scheduling policy.

PRI The priority in hexadecimal.

CPU Shows the short-term CPU usage of the thread (in hexadecimal). The maximum value for this field is 78 ticks (which is 120 in decimal).

EVENT The type of system event, if one is happening.

PROCNAME Process name.

FLAGS/t_flags Displays the signal that the process is currently waiting on if the thread is waiting. An additional line (t_flags) is added. The t_flags line displays in English what signal the process is currently waiting on. If the thread is not waiting, the field is empty.

In AIX V4, there are three possible values for thread-scheduling policy:

- FIFO: Once a thread with this policy is scheduled, it runs to completion unless it is blocked. It voluntarily yields control of the CPU when a higher-priority thread becomes dispatchable. Only fixed-priority threads can have an FIFO scheduling policy.
- RR: This is similar to the AIX V3 scheduler Round-Robin scheme based on 10 ms time slices. When an RR thread has control at the end of its time slice, it

moves to the tail of the queue of dispatchable threads of its priority. Only fixed-priority threads can have an RR scheduling policy.

- OTHER: This policy is defined by POSIX1003.4a as implementation-defined. In AIX V4, this policy is defined to be equivalent to RR, except that it applies to threads with non-fixed priority. The recalculation of the running thread's priority value at each clock interrupt means that a thread may lose control because its priority value has risen above that of another dispatchable thread. In AIX V3, this is the only scheduling policy, and it is the default scheduling policy in AIX V4.

pstat -A: Shows all entries in the kernel thread table. The output is similar to `pstat -P`.

2.6 The netstat Command

The `netstat` command is used to show network status. It gives a good indication of the reliability of the local network interface. Traditionally, it is used more for problem determination than for performance measurement. However, it is useful in determining the amount of traffic on the network to ascertain whether performance problems are due to network congestion.

The `netstat` command displays information regarding traffic on the configured network interfaces, such as:

- The address of any protocol control blocks associated with the sockets and the state of all sockets
- The number of packets received, transmitted, and dropped in the communications subsystem
- Cumulative statistics for Error Collisions Packets transferred
- Routes and their status

Most of the variations of this command use less than 0.2 seconds of CPU time.

2.6.1 Using the netstat Command

The `netstat` command displays the contents of various network-related data structures for active connections. In this chapter we will only discuss the options and output fields that are relevant for network performance determinations. For all other options and columns, see the *AIX Version 4 Commands Reference*, SBOF-1851.

netstat -i: Shows the state of all configured interfaces.

The following example shows the statistics for a workstation with an integrated Ethernet and a token-ring adapter:

```
# netstat -i
Name  Mtu   Network     Address            Ipkts Ierrs Opkts  Oerrs  Coll
lo0   16896 <Link>                         144834   0   144946    0    0
lo0   16896 127         localhost          144834   0   144946    0    0
tr0   1492  <Link>10.0.5a.4f.3f.61         658339   0   247355    0    0
tr0   1492  9.3.1       ah6000d            658339   0   247355    0    0
en0   1500  <Link>8.0.5a.d.a2.d5                0   0      112    0    0
en0   1500  1.2.3       1.2.3.4                 0   0      112    0    0
```

The count values are summarized since system startup.

Name	Interface name.
Mtu	Maximum transmission unit. The maximum size of packets in bytes that are transmitted using the interface. In special cases, it could be changed to increase performance.
Ipkts	Total number of packets received.
Ierrs	Total number of input errors. For example, packets without header, checksum errors.
Opkts	Total number of packets transmitted.
Oerrs	Total number of output errors.
Coll	Number of packet collisions detected.

Note

The `netstat -i` command does not support the collision count for Ethernet interfaces.

netstat -i -Z: This is an undocumented function of the `netstat` command. It clears all the statistic counters for the `netstat -i` command to zero.

netstat -I <interface> <interval>: Displays the statistics for the specified interface. It offers information similar to `netstat -i` for the specified interface and reports it for a given time interval.

```
# netstat -I en0 1
    input   (en0)      output           input   (Total)    output
   packets  errs  packets  errs colls  packets  errs  packets   errs colls
       0     0       27     0     0   799655     0   390669     0     0
       0     0        0     0     0        2     0        0     0     0
       0     0        0     0     0        1     0        0     0     0
       0     0        0     0     0       78     0      254     0     0
       0     0        0     0     0      200     0       62     0     0
       0     0        1     0     0        0     0        2     0     0
```

The previous example shows the `netstat -I` output for the `ent0` interface. Two reports are generated side by side, one for the specified interface and one for all available interfaces (`Total`). The fields are similar to the ones in the `netstat -i` example, `input packets = Ipkts`, `input errs = Ierrs` and so on.

netstat -m: Displays the statistics recorded by the ″mbuf″ memory-management routines. The following example shows the first part of the `netstat -m` output:

```
# netstat -m
32 mbufs in use:
16 mbuf cluster pages in use
71 Kbytes allocated to mbufs
297 requests for mbufs denied
0 calls to protocol drain routines
...
...
```

If the `netstat -m` command indicates that requests for mbufs or clusters have failed or been denied, then you may want to increase the value of `thewall` by using `no -o thewall=<newvalue>`. (More information about the `no` command in 2.8, “The no Command” on page 46.) The new value should be in KB. If buffers are not available when a request is received, the request is lost.

netstat -v: The netstat -v command displays the statistics for each Common Data Link Interface (CDLI)-based device driver that is up. Interface-specific reports could be requested via the commands tokstat, entstat or fddistat (for details, see the *AIX Version 4 Commands Reference*, SBOF-1851).

Every interface has its own specific information and some general information. The following example shows the token-ring and Ethernet part of the netstat -v command; other interface parts are similar. It shows the statistics for the kind of adapter we had. With a different adapter, the statistics will differ somewhat. (The most important output fields are highlighted.):

```
TOKEN-RING STATISTICS (tok0) :
Device Type: Token-Ring High-Performance Adapter (8fc8)
Hardware Address: 10:00:5a:4f:3f:61
Elapsed Time: 0 days 2 hours 21 minutes 0 seconds

Transmit Statistics:                    Receive Statistics:
--------------------                    -------------------
Packets: 31568                          Packets: 138066
Bytes: 3017565                          Bytes: 23156284
Interrupts: 30965                       Interrupts: 139310
Transmit Errors: 0                      Receive Errors: 0
Packets Dropped: 0                      Packets Dropped: 1384
Max Packets on S/W Transmit Queue: 13   Bad Packets: 0
S/W Transmit Queue Overflow: 0

Current S/W+H/W Transmit Queue Length: 0

Broadcast Packets: 50                   Broadcast Packets: 101305
Multicast Packets: 0                    Multicast Packets: 0
Timeout Errors: 0                       Receive Congestion Errors: 0

Current SW Transmit Queue Length: 0
Current HW Transmit Queue Length: 0

General Statistics:
-------------------

No mbuf Errors: 1384                    Lobe Wire Faults: 0
Abort Errors: 0                         AC Errors: 0
Burst Errors: 3                         Frame Copy Errors: 0
Frequency Errors: 0                     Hard Errors: 0
Internal Errors: 0                      Line Errors: 0
Lost Frame Errors: 0                    Only Station: 0
Token Errors: 0                         Remove Received: 0
Ring Recovered: 0                       Signal Loss Errors: 0
Soft Errors: 0                          Transmit Beacon Errors: 0

Driver Flags: Up Broadcast Running
         AlternateAddress ReceiveFunctionalAddr 16 Mbps

Token-Ring High-Performance Adapter (8fc8) Specific Statistics
------------------------------------------------------------
DMA Bus Errors: 0                       DMA Parity Errors: 0
ARI/FCI Errors: 0
------------------------------------------------------------
ETHERNET STATISTICS (ent1) :
Device Type: Ethernet High Performance LAN Adapter
```

```
Hardware Address: 02:60:8c:2d:a7:fc
Elapsed Time: 0 days 1 hours 49 minutes 10 seconds

Transmit Statistics:                          Receive Statistics:
--------------------                          -------------------
Packets: 295741                               Packets: 295048
Bytes: 23051459                               Bytes: 22541375
Interrupts: 295736                            Interrupts: 294744
Transmit Errors: 0                            Receive Errors: 0
Packets Dropped: 0                            Packets Dropped: 2260
Max Packets on S/W Transmit Queue: 64         Bad Packets: 0
S/W Transmit Queue Overflow: 690

Current S/W+H/W Transmit Queue Length: 0

Broadcast Packets: 5                          Broadcast Packets: 2
Multicast Packets: 0                          Multicast Packets: 0
No Carrier Sense: 0                           CRC Errors: 0
DMA Underrun: 0                               DMA Overrun: 0
Lost CTS Errors: 0                            Alignment Errors: 0
Max Collision Errors: 0                       No Resource Errors: 0
Late Collision Errors: 0                      Receive Collision Errors: 0
Deferred: 0                                   Packet Too Short Errors: 0
SQE Test: 0                                   Packet Too Long Errors: 0
Timeout Errors: 0                             Packets Discarded by Adapter:
                                              0
Single Collision Count: 9                     Receiver Start Count: 1
Multiple Collision Count: 0

Current HW Transmit Queue Length: 0

General Statistics:
-------------------

No mbuf Errors: 2260

Adapter Reset Count: 0
Driver Flags: Up Broadcast Running
          Simplex AlternateAddress

Ethernet High Performance LAN Adapter Specific Statistics:
----------------------------------------------------------
Receive Buffer Pool Size: 37
Transmit Buffer Pool Size: 37
In Promiscuous Mode for IP Multicast: No
Packets Uploaded from Adapter: 295048
Host End-of-List Encountered: 0
82586 End-of-List Encountered: 0
Receive DMA Timeouts: 0
Adapter Internal Data: 0x0 0x0 0x0 0x1 0x10
```

The highlighted fields are the ones to look for, and their meaning is:

- **Transmit and Receive Errors**

  The number of output/input errors encountered on this device. This is a counter for unsuccessful transmissions due to hardware/network errors.

  This could also slow down the performance of the system.

- **Max Packets on S/W Transmit Queue**

  The maximum number of outgoing packets ever queued to the software transmit queue.

  An indication of an inadequate queue size is if the maximal transmits queued equals the current queue size (`xmt_que_size`). This says that the queue was full at some point.

  To check the current size of the queue, use `lsattr -El <adapter>` (where adapter is `tok0` or `ent0`). Since the queue is associated with the device driver and adapter for the interface, we must use the adapter name, not the interface name. The queue size can be changed via SMIT or with the `chdev` command.

- **S/W Transmit Queue Overflow**

  The number of outgoing packets that have overflowed the software transmit queue.

  A value other than zero requires the same actions as would be needed if the ″Max Packets on S/W Transmit Queue″ reaches the `xmt_que_size`. The transmit queue size has to be increased.

- **Received Broadcast Packets**

  The number of broadcast packets received without any error.

  If the value for broadcast packets is high, compare it with the total received packets. The received broadcast packets should be less than 20 percent of the total received packets. If it is high, this could be an indication for a high network load; then multi-casting should be used.

- **Max Collision Errors**

  The number of unsuccessful transmissions due to too many collisions. The number of collisions encountered exceeded the number of retries on the adapter.

- **Late Collision Errors**

  The number of unsuccessful transmissions due to the late collision error.

- **Timeout Errors**

  The number of unsuccessful transmissions due to adapter reported timeout errors.

- **Single Collision Count**

  The number of outgoing packets with single (only one) collision encountered during transmission.

- **Multiple Collision Count**

  The number of outgoing packets with multiple (2 - 15) collisions encountered during transmission.

- **Receive Collision Errors**

  The number of incoming packets with collision errors during reception.

- **No mbuf Errors**

  The number of times that mbufs were not available to the device driver. This usually occurs during receive operations when the driver must obtain memory buffers to process inbound packets. If the mbuf pool for the requested size is empty, the packet will be discarded. The `netstat -m`

command could be used to confirm this, and the parameter thewall should be increased.

This value is interface specific and not identical with the "requests for mbufs denied" from the netstat -m output. Compare the values of the example for netstat -m and netstat -v (Ethernet and token-ring part).

To determine network performance problems check for any *Error* counts in the netstat -v output.

netstat -p <protocol>: Shows statistics about the value specified for the protocol variable (udp, tcp, ip, icmp), which is either a well-known name for a protocol or an alias for it. Some protocol names and aliases are listed in the /etc/protocols file. A null response means that there are no numbers to report. If there is no statistics routine for it, then the program report of the value specified for the protocol variable is unknown.

The following example shows the output for the ip protocol:

```
# netstat -p ip
ip:
:
        491351 total packets received
        0 bad header checksums
        0 with size smaller than minimum
        0 with data size < data length
        0 with header length < data size
        0 with data length < header length
        0 with bad options
        0 with incorrect version number
        25930 fragments received
        0 fragments dropped (dup or out of space)
       0 fragments dropped after timeout
        12965 packets reassembled ok
        475054 packets for this host
        0 packets for unknown/unsupported protocol
        0 packets forwarded
        3332 packets not forwardable
        0 redirects sent
        405650 packets sent from this host
        0 packets sent with fabricated ip header
        0 output packets dropped due to no bufs, etc.
        0 output packets discarded due to no route
        5498 output datagrams fragmented
        10996 fragments created
        0 datagrams that can't be fragmented
        0 IP Multicast packets dropped due to no receiver
        0 ipintrq overflows
```

The highlighted fields are the ones to look for, and their meaning is:

- **Bad Header Checksum or Fragments Dropped**

  If the output shows bad header checksum or fragments dropped due to dup or out of space, then this indicates either a network that is corrupting packets or device driver receive queues that are not large enough.

- **Dropped after Timeout**

  If the fragments dropped after timeout is other than zero, then the "time to life counter" of the ip fragments expired due to a busy network before all

fragments of the datagram arrived. To avoid this, increase the value of the ipfragttl network parameter with the no command. Another reason could be a lack of mbufs; so increase thewall.

The following example shows the output for the udp protocol:

```
# netstat -p udp
udp:
        11521194 datagrams received
        0 incomplete headers
        0 bad data length fields
       0 bad checksums
        16532 dropped due to no socket
        232850 broadcast/multicast datagrams dropped due to no socket
       77 dropped due to full socket buffers
        11271735 delivered
        796547 datagrams output
```

Statistics of interest are:

- **Bad Checksums**

  This could happen due to hardware card or cable failure.

- **Dropped due to full Socket Buffers**

  This could be due to insufficient transmit and receive UDP sockets, too few nfsd daemons, and/or too small nfs_socketsize, udp_recvspace and sb_max values.

If the netstat -p udp command indicates socket overflows, then you may need to increase the number of the nfsd daemons on the server. But first, check the affected system for CPU or I/O saturation and verify the recommended setting for the other communication layers by using no -a. If the system is saturated, you need to reduce its load or increase its resources.

netstat -s -s: The netstat -s command shows statistics for each protocol (while netstat -p shows the statistics for the specified protocol). The undocumented -s -s option shows only those lines of the netstat -s output that are not zero. This makes it easier to look for error counts.

2.6.2 Additional Rules of Thumb

In general there is no reason not to increase the transmit and receive queues. This requires a few bytes in memory, but avoids some problems.

Host problems

- If the number of errors during input packets is greater than 1 percent of the total number of input packets (from netstat -i):

  Ierrs > 0.01 x Ipkts

  Then execute netstat -m to check for a lack of memory.

- If the number of errors during output packets is greater than 1 percent of the total number of output packets (from netstat -i):

  Oerrs > 0.01 x Opkts

  Then the send queue size (xmt_que_size) for that interface should be increased. The size of the xmt_que_size could be checked with the command:

```
# lsattr -El <adapter>
```

Overloaded Network

- To check for an overloaded Ethernet network, calculate (from `netstat -v`):

  (Max Collision Errors + Timeouts Errors) / Transmit Packets

  If the answer is greater than 5 percent, then the network should be reorganized to balance the load.

- Another indication for a high network load is (from `netstat -v`):

  If the total number of collisions from `netstat -v` (for Ethernet) is greater than 10 percent of the total transmitted packets.

  Number of collisions / Number of Transmit Packets > 0.1

2.6.3 AIX Version 4.2.1 Improvements

There will be some changes to the `netstat` command. The one relevant to performance is the display of the discovered Path Maximum Transmission Unit (PMTU).

For two hosts communicating across a path of multiple networks, a transmitted packet will become fragmented if its size is greater than the smallest MTU of any network in the path. Since packet fragmentation can result in reduced network performance, it is desirable to avoid fragmentation by transmitting packets with a size no greater than the smallest MTU in the network path. This size is called the path MTU.

This value could be displayed via the `netstat -r` command. In this example the `netstat -r -f inet` command is used to display only the routing tables:

```
# netstat -r -f inet
Routing tables
Destination      Gateway           Flags    Refs     Use   PMTU If  Exp  Groups

Route Tree for Protocol Family 2:
default          itsorusi          UGc       1      348   -   tr0   -
9.3.1            sv2019e           Uc       25    12504   -   tr0   -
itsonv           sv2019e           UHW       0      235   -   tr0   -
itsorusi         sv2019e           UHW       1      883  1492 tr0   -
ah6000d          sv2019e           UHW       1      184  1492 tr0   -
ah6000e          sv2019e           UHW       0      209   -   tr0   -
sv2019e          sv2019e           UHW       4    11718  1492 tr0   -
coyote.ncs.mainz itsorusi          UGHW      1       45  1492 tr0   -
kresna.id.ibm.co itsorusi          UGHW      0       14  1492 tr0   -
9.184.104.111    kresna.id.ibm.com UGc       0        5   -    tr0   -
127              localhost         U         3       96  -    lo0   -
```

2.7 The nfsstat Command

The `nfsstat` command displays statistical information about the Network File System (NFS) and the Remote Procedure Calls (RPC) interface to the kernel for clients and servers. This command could also be used to re-initialize the counters for these statistics. For performance issues, the RPC statistics (`-r` option) are the first place to look. The NFS statistics show you how the applications use NFS.

RPC Statistics: The nfsstat command displays statistical information pertaining to the ability of a client or server to receive calls like:

- Total number of RPC calls received or rejected
- Number of times no RPC packet was available when trying to receive
- Number of packets that were too short or had malformed headers
- Total number of RPC calls sent or rejected by a server
- Number of times a call had to be transmitted again
- Number of times a reply did not match the call
- Number of times a call timed out
- Number of times a call had to wait on a busy client handle
- Number of times authentication information had to be refreshed

2.7.1 NFS Server Information

The NFS server displays the number of NFS calls received (calls) and rejected badcalls) due to authentication, as well as the counts and percentages for the various kinds of calls made.

The following example shows the server part of the nfsstat command, specified by the -s option:

```
# nfsstat -s

Server rpc:
calls    badcalls  nullrecv  badlen    xdrcall
44124083   0          9902       0          0

Server nfs:
calls      badcalls
34841553   733
null       getattr    setattr    root       lookup     readlink   read
73697  0%  11221849 32% 57209  0%  0  0%     11892420 34% 122697  0% 7609895 21%
wrcache    write      create     remove     rename     link       symlink
0  0%      1296536  3% 86924  0%  84555  0%  2328  0%   470  0%    169  0%
mkdir      rmdir      readdir    fsstat
387  0%    374  0%    1607749  4% 784294  2%
```

RPC output for server (-s):

calls The total number of RPC calls received from clients.

badcalls The total number of calls rejected by the RPC layer.

nullrecv The number of times an RPC call was not available when it was thought to be received.

badlen Packets truncated or damaged. The number of RPC calls with a length shorter than a minimum-sized RPC call.

xdrcall The number of RPC calls whose header could not be External Data Representation (XDR) decoded.

It also displays a count of the various kinds of calls and their respective percentages.

The NFS performance could be increased by adding additional nfsd daemons. But watch the nullrecv column in the nfsstat -s output. If the number starts to grow, it may mean there are too many nfsd daemons. However, this is usually not the case on AIX NFS servers as much as it could be the case on other platforms. The reason for that is that all nfsd daemons are not awakened at the same time when an NFS request comes into the server. Instead, the first nfsd

wakes up, and if there is more work to do, this daemon will wake up the second nfsd, and so on.

2.7.2 NFS Client Information

The NFS client displays the number of calls sent and rejected, as well as the number of times a client handle was received (nclget), the number of times a call had to sleep while awaiting a handle (nclsleep), and a count of the various kinds of calls and their respective percentages.

The following example shows the nfsstat output specified for clients via the -c option:

```
# nfsstat -c

Client rpc:
calls    badcalls  retrans  badxid    timeout  wait      newcred
73365      424       1         10        425      0         0

Client nfs:
calls      badcalls   nclget     nclsleep
72952      0          72952      0
null       getattr    setattr    root       lookup     readlink   read
0  0%      8043 11%   1507  2%   0  0%      24696 33%  0  0%      11330 15%
wrcache    write      create     remove     rename     link       symlink
0  0%      21292 29%  1637  2%   1216  1%   791  1%    0  0%      0  0%
mkdir      rmdir      readdir    fsstat
0  0%      0  0%      1387  1%   1053  1%
```

RPC output for the client (-c):

calls The total number of RPC calls made to NFS.
badcalls The total number of calls rejected by the RPC layer.
retrans The number of times a call had to be retransmitted due to a timeout while waiting for a reply from the server. This is applicable only to RPC over connection-less transports.
badxid The number of times a reply from a server was received that did not correspond to any outstanding call. This means the server is taking too long to reply.
timeout The number of times a call timed-out while waiting for a reply from the server.
wait The number of calls waiting on busy client.
newcred The number of times authentication information had to be refreshed.

It also displays a count of the various kinds of calls and their respective percentages.

For performance monitoring, nfsstat -c will give information on whether the network is dropping packets. A network may drop a packet if it cannot handle it. Dropped packets may be the result of the response time of the network hardware or software or an overloaded CPU on the server. Dropped packets are not actually lost since a replacement request is issued for them.

The retrans column in the RPC section displays the number of times requests were retransmitted due to a timeout in waiting for a response. This is related to dropped packets. If the retrans number consistently exceeds five percent of the total calls in column one, then it indicates a problem with the server keeping up

with demand. Use vmstat, netpmon, and iostat on the server machine to check the load.

The high badxid count implies that requests are reaching the various NFS servers, but the servers are too loaded to send replies before the client's RPC calls time out and are retransmitted. The badxid value is incremented each time a duplicate reply is received for a transmitted request (an RPC request retains its XID through all transmission cycles). Excessive retransmissions place an additional strain on the server, further degrading response time. If badxid and timeout are greater then five percent of the total calls, increase the timeo parameter of the NFS-Mount options via smit chnfsmnt.

If the server is CPU-bound, it will affect NFS and its daemons. To improve the situation, the server must be tuned or upgraded, or the user can localize the application files. If the server is I/O-bound, the server file systems can be reorganized, or localized files can be used.

If the number of retransmits and timeouts are close to the same value, then it's almost definite that packets are being dropped. Packets are rarely dropped on the client.

Usually, packets are dropped on either the network or on the server. The server could drop packets if it overflows its interface driver's transmit queue or if the server's User Datagram Protocol (UDP) socket buffer was overflown (nfs_socketsize). If there are no socket buffer overflows or Oerrs on the server, and the client is getting lots of retransmits and timeouts, then the chances are that packets are being dropped on the network. This could include such things as media and network devices such as routers, bridges, concentrators.

Network sniffers and other tools can be used to debug such problems.

2.7.3 AIX Version 4.2.1 Improvements

AIX V4.2.1 will include the NFS Version 3 networking commands. Following is a short description of the performance-related changes in the nfsstat command.

The NFS part of the nfsstat command is divided into Version 2 and Version 3 statistics of NFS, and the RPC part is divided into Connection oriented and Connectionless statistics.

- The following two fields are added to the nfsstat -s command:

  dupchecks The number of RPC calls that looked up in the duplicate request cache.

  dupreqs The number of duplicate RPC calls found.

- The following fields are added to the nfsstat -c command:

  badverfs The number of times a call failed due to a bad verifier in the response.

  timers The number of times the calculated timeout value was greater than or equal to the minimum specified timeout value for a call.

  cantconn The number of times a call failed due to a failure to make a connection to the server.

  nomem The number of times a call failed due to a failure to allocate memory.

  interrupts The number of times a call was interrupted by a signal before completing.

cantsend The number of times a send failed due to a failure to make a connection to the client.

The `wait` field has been removed.

In addition, the `nfsstat -m` command now displays the mount options of the mounted filesystems, which could also be changed to increase the performance.

2.8 The no Command

The `no` command can be used to configure network attributes. It sets or displays current network attributes in the currently running kernel. Therefore the command must be run again after each startup or after the network has been configured. For more information on how the network attributes interact with each other, refer to the *AIX Version 4 System Management Guide: Communications and Networks*, SC23-2526.

Note

The `no` command performs no range checking. If used incorrectly, the `no` command can cause your system to become inoperable.

The `no` parameters and their values can be displayed by using `no -a`:

```
thewall              = 8192
sb_max               = 65536
somaxconn            = 1024
net_malloc_police    = 0
rto_low              = 1
rto_high             = 64
rto_limit            = 7
rto_length           = 13
arptab_bsiz          = 7
arptab_nb            = 25
tcp_ndebug           = 100
ifsize               = 8
strmsgsz             = 0
strctlsz             = 1024
nstrpush             = 8
strthresh            = 85
psetimers            = 20
psebufcalls          = 20
strturncnt           = 15
pseintrstack         = 12288
lowthresh            = 90
medthresh            = 95
subnetsarelocal      = 1
maxttl               = 255
ipfragttl            = 60
ipsendredirects      = 1
ipforwarding         = 0
udp_ttl              = 30
tcp_ttl              = 60
arpt_killc           = 20
tcp_sendspace        = 16384
```

```
tcp_recvspace          = 16384
udp_sendspace          = 9216
udp_recvspace          = 41600
rfc1122addrchk         = 0
nonlocsrcroute         = 0
tcp_keepintvl          = 150
tcp_keepidle           = 14400
bcastping              = 0
udpcksum               = 1
tcp_mssdflt            = 512
icmpaddressmask        = 0
tcp_keepinit           = 150
ie5_old_multicast_mapping  = 0
rfc1323                = 0
ipqmaxlen              = 100
directed_broadcast     = 1
```

Some network attributes are runtime attributes that can be changed at any time; others are load-time attributes that must be set before the `netinet` kernel extension is loaded. They need to be placed near the top of the /etc/rc.net file. If your system uses Berkeley-style network configuration, set the attributes near the top of the /etc/rc.bsdnet file.

To display or change a specific parameter, use the `no -o` command:

```
# no -o thewall
thewall = 8192
# no -o thewall=16384
```

The parameters can be reset to their default value by using the `-d` option:

```
# no -d thewall
# no -o thewall
thewall = 8192
```

2.8.1 no Command Parameters

You can choose to tune either for maximum throughput or for minimum memory use. Some recommendations apply to one or the other; some apply to both. All `no` parameters are listed in Appendix B, “Summary of Tunable AIX Parameters” on page 269. In this section only some performance relevant parameters are described:

thewall: This is one of the most important parameters because it limits the memory that is assigned to the networking subsystem.

Note

Before AIX V4.2, the `thewall` value was usually set in the /etc/rc.net file, and any value specified by the Object Database Manager (ODM) was overwritten. But in AIX V4.2, the highest instance for setting the `thewall` value after system start-up is the ODM. The value set in the /etc/rc.net file is overwritten. So if there are any problems with the value of the `thewall` parameter after start-up, check the /etc/rc.net file, and also check the ODM by using `lsattr -El sys0` and look for the `maxmbuf` parameter.

After expanding the mbuf pools, verify using the vmstat command that paging rates have not increased. If you cannot expand the pools to necessary levels without adversely affecting the paging rates, additional physical memory may be required.

sb_max: The sb_max value sets the absolute upper bound on the size of TCP and UDP socket buffers, like setsockopt(), udp_sendspace, udp_recvspace, tcp_sendspace, and tcp_recvspace. The recommended size for this parameter is twice the size of the largest socket buffer. Remember to set rfc1323 to one, if the buffer size crosses 64 KB.

TCP and UDP Buffer Sizes: The TCP/UDP send space buffers and TCP/UDP receive space buffers have to be set to less or equal the sb_max size. Remember to set rfc1323 to one, if the buffer size crosses 64 KB.

The following table shows the default sizes for TCP/UDP send and receive spaces along with their recommended values.

Device Name	Device	Default sendspace	Default recvspace	Recommended sendspace	Recommended recvspace
Token Ring (TCP)	tr	16384	16384	16384	16384
Token Ring (UDP)	tr	9216	41600	9216	41600
Ethernet (TCP)	en or et	16384	16384	16384	16384
Ethernet (UDP)	en or et	9216	41600	9216	41600
FDDI (TCP)	fi	16384	16384	57344	57344
FDDI (UDP)	fi	9216	41600	57344	57344
SOCC (TCP)	so	16384	16384	57344	57344
SOCC (UDP)	so	9216	41600	57344	57344
ATM	at	9216	41600	57344	57344

Table 3. Send and Receive Spaces for TCP and UDP

rfc1323: Enables Transmission Control Protocol (TCP) enhancements as specified by Request for Comments (RFC) 1323, *TCP Extensions for High Performance*. If the TCP socket buffers sizes should cross the 64-KB limit, this parameter has to be set to 1 to allow the TCP window to slide or pan.

ipfragttl: This value has to be tuned, if the netstat -s shows fragments dropped after timeout.

2.8.2 Stream Parameters

Since AIX V4 stream parameters can also be set via the no command. For more details about the streams parameters refer to Appendix B, “Summary of Tunable AIX Parameters” on page 269. Following is a list of the no streams parameters:

- lowthresh
- medthresh
- nstrpush
- psebufcalls
- pseintrstack

- psetimers
- strctlsz
- strmsgsz
- strthresh
- strturncnt

2.8.3 AIX Version 4.2.1 Improvements

The following parameters are added to the no command, but not all are really performance related:

- clean_partial_conns

  Allows the system to keep on working if another system is sending SYN Flood Attack.

- arpqsize

  Specifies the maximum number of packets to queue while waiting for address resolution protocol (ARP) responses. The default value is 1, and it is increased to a minimum value of 5 when path MTU discovery is enabled. The value will not automatically decrease if path MTU discovery is subsequently disabled. This attribute is supported by Ethernet, 802.3, token-ring, and FDDI interfaces. This attribute applies to AIX V4.1.5, V4.2.1, and later.

- route_expire

  Specifies whether the route expires. A value of zero allows no route expiration, which is the default. Negative values are not allowed for this option.

- psecache

  This is a streams-related parameter used for debugging.

- pmtu_default_age

  Specifies the default amount of time (in minutes) before the path MTU value for UDP paths is checked for a lower value. A value of 0 allows no aging. The default value is 10 minutes. The pmtu_default_age value can be overridden by UDP applications.

  Since routes can change dynamically, the path MTU value for a path may also change over time. Decreases in the path MTU value will result in packet fragmentation. Path MTU values that are discovered in this way are periodically checked for decreases. By default, decreases are checked for every 10 minutes, and this value can be changed by modifying the value of the pmtu_default_age option of the no command.

- pmtu_rediscover_interval

  Specifies the default amount of time (in minutes) before the path MTU value for UDP and TCP paths are checked for a higher value. A value of 0 allows no path MTU rediscovery.

  Increases in the path MTU value can result in a potential increase in network performance; so discovered path MTU values are periodically checked for increases. By default, increases are checked for every 30 minutes, and this value can be changed by modifying the value of the pmtu_rediscover_interval option of the no command.

- udp_pmtu_discover

Enables or disables path MTU discovery for UDP applications. UDP applications must be specifically written to utilize path MTU discovery. A value of 0 disables the feature, while a value of 1 enables it. The default value is 0.

- `tcp_pmtu_discover`

  Enables or disables path MTU discovery for TCP applications. A value of 0disables path MTU discovery for TCP applications, while a value of 1 enables it. The default value is 0.

- `ipignoreredirects`

  Specifies whether or not to process redirects that are received. The default value of 0 processes redirects as usual. A value of 1 ignores redirects.

- `ipsrcroutesend`

  Specifies whether applications can send source-routed packets. The default value of 1 allows source-routed packets to be sent. A value of 0 causes `setsockopt()` to return an error if an application attempts to set the source-routing option, and removes any source-routing options from outgoing packets.

- `ipsrcrouterecv`

  Specifies whether the system accepts source-routed packets. The default value of 0 causes all source-routed packets destined for this system to be discarded. A value of 1 allows source-routed packets to be received.

- `ipsrcrouteforward`

  Specifies whether the system forwards source-routed packets. The default value of 1 allows the forwarding of source-routed packets. A value of 0 causes all source-routed packets that are not at their destinations to be discarded.

2.9 The nfso Command

The `nfso` command can be used to configure Network File System attributes. It sets or displays network options in the currently running kernel. Therefore the command must run after each system startup or network configuration.

Note

The `nfso` command performs no range checking. If used incorrectly, the `nfso` command can make your system inoperable.

The `nfso` parameters and their values can be displayed by using `nfso -a`:

```
# nfso -a
portcheck= 0
udpchecksum= 1
nfs_socketsize= 60000
nfs_setattr_error= 0
nfs_gather_threshold= 4096
nfs_repeat_messages= 0
nfs_duplicate_cache_size= 0
nfs_server_base_priority= 0
nfs_dynamic_retrans= 1
nfs_iopace_pages= 32
```

Most NFS attributes are runtime attributes that can be changed at any time, with the exception of nfs_socketsize, which needs NFS to be stopped first and restarted afterwards.

To display or change a specific parameter, use the nfso -o command:

```
# nfso -o portcheck
portcheck= 0
# nfso -o portcheck=1
```

The parameters can be reset to their default value by using the -d option:

```
# nfso -d portcheck
# nfso -o portcheck
portcheck= 0
```

2.9.1 nfso Command Parameters

portcheck: Checks whether an NFS request originated from a privileged port. The default value of 0 disables the port checking that is done by the NFS server. A value of 1 directs the NFS server to do port checking on the incoming NFS requests.

This is a configuration decision with minimal performance consequences.

udpchecksum: Performs the checksum of NFS UDP packets. The default value of 1 directs the NFS server or client to build UDP checksums for the packets that it sends to the NFS clients or servers. A value of 0 disables the checksum on UDP packets from the NFS server or client.

Turning this off is not recommended. Make sure this value is set to 1 in any network where packet corruption may occur. Slight performance gains can be realized by turning it off, but this increases the chance of data corruption.

nfs_socketsize: Sets the queue size of the NFS server UDP socket. This socket is used for receiving the NFS client requests and can be adjusted so that the NFS server is less likely to drop packets under heavy load. The value of the nfs_socketsize variable must be less than the sb_max option, which can be manipulated by the no command.

Increase the size of the nfs_socketsize variable when netstat reports packets dropped due to full socket buffers for UDP and increasing the number of nfsd daemons has not helped.

nfs_setattr_error: When set to a value of 1, the NFS server ignores invalid setattr requests. This is provided for certain Personal Computer applications. The default value is 0.

Tuning this parameter should not increase the performance.

nfs_gather_threshold: Determines when to attempt to gather write requests to a file. If the size of the NFS write request is less than the value of the nfs_gather_threshold option, the NFS server writes the data and immediately responds to the NFS client. If the size of the NFS write request is equal to or greater than the value of this option, the NFS server writes the data and waits for a small amount of time before responding to the NFS client.

The write gathering can be a performance advantage for sequential writes, but can produce slight performance decreases for random writes. Look at the following two scenarios:

1. Delays are observed in responding to RPC requests, particularly those where the client is exclusively doing non-sequential writes or the files being written are being written with file locks held on the client.
2. Clients are writing with write sizes smaller than 4096 KB and write gather is not working.

If write gather is to be disabled, change the `nfs_gather_threshold` to a value greater than the largest possible write. For AIX V4 running NFS Version 2, that would be 8192. So changing the value to 8193 disables write gather. Use this for the situation described above in the first scenario. If write gather is being bypassed due to a small write size, say 1024 KB (described in second scenario), change the write gather parameter to gather smaller writes. For example, set it to 1024.

nfs_repeat_messages: Checks for duplicate NFS messages. This option is used to avoid displaying duplicate NFS messages. When set to a value of 1, all NFS messages are printed to the screen. If set to a value of 0, duplicate messages appearing one after the other are not printed to the screen. Only the first message of such a sequence is displayed. When a different message appears, a message similar to the following will be displayed:

```
Last NFS message repeated n times
```

Tuning this parameter does not increase the performance.

nfs_duplicate_cache_size: The `nfso` command cannot be used to decrease the `nfs_duplicate_cache_size` value. Any attempt will fail; the system has to be rebooted. The duplicate cache size should be increased for very fast or busy servers that have a high throughput capability. The duplicate cache is used to allow the server to correctly respond to NFS client retransmissions. If the server flushes this cache before the client is able to retransmit, then the server may respond incorrectly, and the client can observe anomalous NFS behavior. Therefore, if the server can process 1000 operations before a client retransmits, then the duplicate cache size will need to be increased.

Calculate the number of NFS operations that are being received per second at the NFS server and multiply this by 4. This will produce a duplicate cache size that should be sufficient to allow correct response from the NFS server. The operations that are affected by the duplicate cache are the following: `setattr`, `write`, `create`, `remove`, `rename`, `link`, `symlink`, `mkdir`, and `rmdir`.

nfs_server_base_priority: If this value is set, the `nfsd` processes will use this as their base priority. Acceptable values are from 31 to 126. The default value is 0, which means that the `nfsd` processes will have a regular, floating priority. Therefore as they increase their cumulative CPU time, their priority will change. This parameter can be used to set a static priority for the `nfsd` daemons. Other values within the acceptable range will be used to set the priority of the `nfsd` daemon when an NFS request is received at the server. This option can be used if the NFS server is overloading the system (lowering or making the `nfsd` daemon less favored). It can also be used if it is desired that the `nfsd` daemons be one of the most favored processes on the server. Care must be taken when setting the parameter in that it may render the system almost unusable by other processes.

This can occur if the NFS server is very busy and will essentially lock out other processes from having run time on the server.

nfs_dynamic_retrans: With this parameter set to 1, the NFS client will attempt to adjust its timeout behavior based on past NFS server response. It allows the NFS client to automatically decrease the size of NFS read/write packets to attempt to respond to network or server load problems. This also allows the NFS client the ability to vary the timeout value used for the retransmission. All of this is done based on an accumulative history of the NFS server's response time. In most cases this parameter does not need to be adjusted. There are some instances where the straightforward timeout behavior is desired for the NFS client. In these cases, the value should be set to 1 before mounting file systems.

nfs_iopace_pages: This is the maximum number of dirty pages that the NFS client will flush to the NFS server at one time. This is often useful when, for instance, large compilations of images are flushed by the binder and interactive performance suffers or when an application writes a large file to an NFS-mounted filesystem. That file data is written to the NFS server when the file is closed. In some cases, the resource it takes to write that file to the server may prevent other NFS file I/O from occurring. The default value for this parameter limits the number of 4-K pages written to the server to 32. The NFS client will schedule 32 pages for writing to the server and then will wait for these to complete before scheduling the next 32. The default value will usually be sufficient for most environments. The value should be decreased if there are large amounts of contention for NFS client resources. If there is low contention, then the value can be increased. As a practical matter, the value should not be set above a value of 128 and should not be lower than two.

2.9.2 AIX Version 4.2.1 Improvements

With AIX V4.2.1, the NFS subsystem will be updated to Version 3. Because of this, the `nfso` command has some changes:

- `nfs_duplicate_cache_size`

  The old parameter `nfs_duplicate_cache_size` is now divided into a TCP and a UDP parameter:

  - `nfs_tcp_duplicate_cache_size` specifies the number of entries to store in the NFS server's duplicate cache for the TCP network transport.
  - `nfs_udp_duplicate_cache_size` specifies the number of entries to store in the NFS server's duplicate cache for the UDP network transport.

The following parameters are added to the `nfso` command:

- `nfs_max_connections`

  Specifies the maximum number of TCP connections the NFS server allows to be opened at any one time. If new TCP connections are requested from NFS clients and the new connection increases the total beyond this amount, the existing TCP connection will close because of the new connection.

- `nfs_max_threads`

  Specifies the maximum number of NFS server threads that are created to service incoming NFS requests. The maximum number may also be specified as a parameter to the `nfsd` daemon.

- `nfs_use_reserved_ports`

Specifies using non reserved IP port number. The default value of 0 will use a non-reserved IP port number when the NFS client communicates with the NFS server.

- nfs_tcp_socketsize

  Sets the queue size of the NFS server TCP socket. This parameter is similar to the nfs_socketsize parameter.

- nfs_device_specific_bufs

  This option allows the NFS server to use memory allocations from network devices if the network device supports such a feature. The NFS server's use of these special memory allocations can positively affect the overall performance of the NFS server. Valid settings for this option are 0 or 1. The default is 1, which means the NFS server is allowed to use the special network device memory allocations. If the value of 0 is used, then the NFS server will use the traditional memory allocations for its processing of NFS client requests.

- nfs_server_clread

  This option allows the NFS server to be very aggressive about the reading of a file. The NFS server can only respond to the specific NFS read request from the NFS client. However, the NFS server can read data in the file that exists immediately after the current read request. This is normally referred to as read-ahead. The NFS server does read-ahead by default. However, when the nfs_server_clread option is enabled, it tells the NFS server to be very aggressive about doing read-ahead for the NFS client. Valid settings for this option are 0 or 1. The default value is 1 or enabled. If the value of 0 is used, then the normal system default read-ahead methods are used.

2.10 The nice Command

With the nice command, the user can specify a value to be added to or subtracted from the current priority of a process. By using nice, the priority of the process does not get fixed; only its nice value is changed. The priority value is still recalculated periodically based on the CPU usage, the nice value, and the default user-process-priority value. The nice command can only set the nice value for a process at the creation time of the process.

Note

Any user can run a command at a lower priority than normal by using nice. Only the root user can use nice to run commands at higher than normal priority.

Priority Calculation: The priority of most user processes varies with the amount of CPU time the process has used recently. The formula for calculating the priority of a process or thread is:

```
priority value =base value + nice value + (CPU penalty based on recent CPU usage)
```

The base value is also called the user-process-priority value and is 40. The default nice value of a foreground process is 20, of a background process 24, if started from the ksh shell.

Note

Some shells (such as ksh) will automatically add a nice value of 4 to the default nice value if a process is started in the background (using &). For example, if you execute program & from a ksh, this program will automatically be started with a nice value of 24. In this case, if this program was preceded by a nice command (nice -10 program &), then this program would be started with a nice value of 34.

The scheduler's CPU penalty calculations are based on two parameters that can be set with the schedtune command: -r and -d. For more details about the schedtune command, please refer to 4.10, "The schedtune Command" on page 177.

The formula used by the scheduler to calculate the amount to be added to the priority of a process or thread as a penalty for recent CPU use is:

CPU penalty = (recent CPU use value) * (r value / 32)

The default value for r is 16.

The once-per-second recalculation of the recently used CPU value of each process/thread is:

(old recent CPU use value) * (d value / 32)

The default value for d is 16.

The recent CPU usage value is displayed as the "C" column in the ps command output. The maximum value of recent CPU usage is 120. You can view the current nice value (NI), priority (PRI) and CPU usage value (C) for user processes by running the ps -el command.

2.10.1 Using the nice Command

There are several ways to change the nice value, when creating a thread:

Favoring Processes: The following two commands execute a program with higher priority:

```
# nice --15 program
```

or better:

```
# nice -n -15 program
```

The nice command subtracts 15 from the default nice value (20) and the program will start with a priority of 45 because the user-priority-level is 40.

Disfavoring Threads: The following two commands execute a program with lower priority:

```
# nice -10 program
```

or better:

```
# nice -n 10 program
```

The nice command adds 10 to the default nice value (20) and the program will start with a priority of 70 because the user-priority-level is 40.

Note

The csh command contains a built-in command named nice. The /usr/bin/nice command and the csh command's nice command do not necessarily work the same way. For information on the built-in csh command nice, see the csh command in the *AIX Version 4 Commands Reference*, SBOF-1851.

The following example shows the execution of four sleep commands, with different nice values:

```
# nice -10 sleep 100000 &      # Adds 10 to the nice value of 24 (& in ksh shell)
# nice -n -15 sleep 100000 &   # Subtracts 15 from the nice value 24 (& in ksh)
# nice -n -10 sleep 100000 &   # Subtracts 10 from the nice value 24 (& in ksh)
# nice -n -10 sleep 100000     # Subtracts 10 from the default (20) nice value
```

The header line is added to the ps command output:

```
# ps -el |grep sleep
      F S UID   PID  PPID   C PRI NI ADDR     SZ    WCHAN    TTY  TIME CMD
 200001 A   0 18112  2492   0  74  34 35fa     44  9ef72d8  pts/3  0:00 sleep
 200001 A   0  9666  2492   0  49   9 2bf5     44  9ef7658  pts/3  0:00 sleep
 200001 A   0 16068  2492   0  54  14 21f0     44  9ef7698  pts/3  0:00 sleep
 200001 A   0 10182  2492   0  50  10  fe7     44  9ef76d8  pts/3  0:00 sleep
```

The **NI** column shows the modified nice value for each process. And the **PRI** column is the modified process priority.

2.11 The renice Command

With the renice command, the user can specify a value to be added to or subtracted from the priority of one or more running processes. The modification is done to the nice values of the processes. The priority of these processes is still non-fixed. For more information about processes and their priority, refer to 2.10, "The nice Command" on page 54.

Note

If you do not have root user authority, you can only change the priority of processes you own and can only increase their priority value within the range of 0 to 20, with 20 giving the worst priority. If you have root user authority, you can alter the priority of any process and change the priority value within the range of -20 to 20.

The renice command modifies the nice value of one or more processes that are already running. The processes are identified either by process ID, process group ID, or the name of the user who owns the processes. It cannot be used on fixed-priority processes.

For AIX V4, the syntax of renice has been changed to complement the alternative syntax of nice, which uses the -n flag to identify the nice-value increment.

The following four examples show the different usages of the renice command. The xclock command is started by a staff user in background:

```
# ps -lu mike
      F S UID   PID  PPID   C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD
 8200011 A 201  7746 20540   0  64 24 6898   192          pts/5  0:00 xclock
  200001 A 201  20540 22352   0  60 20  5a0   360  5a9b844  pts/5  0:00 ksh
# renice -8 7746
# ps -lu mike
      F S UID   PID  PPID   C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD
 8200011 A 201  7746 20540   0  52 12 6898   192          pts/5  0:00 xclock
  200001 A 201  20540 22352   0  60 20  5a0   360  5a9b844  pts/5  0:00 ksh
```

In the previous example, the renice command is used without the -n option. This alters the nice value from the default value of 20. The renice -<Increment> command always uses the default nice value (20) for the calculation, not the current value. If you want to use the current value for the calculation, you need to use the following commands:

```
# ps -lu mike
      F S UID   PID  PPID   C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD
  200001 A 201  7778 20540   0  64 24 6898   192          pts/5  0:00 xclock
  200001 A 201  20540 22352   0  60 20  5a0   360  5a9b844  pts/5  0:00 ksh
# renice -n -8 7778
# ps -lu mike
      F S UID   PID  PPID   C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD
  200001 A 201  7778 20540   0  56 16 6898   192          pts/5  0:00 xclock
  200001 A 201  20540 22352   0  60 20  5a0   360  5a9b844  pts/5  0:00 ksh
```

In the previous example the renice command subtracts eight from the current nice value.

```
# ps -lu mike
      F S UID   PID  PPID   C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD
  200001 A 201  7786 20540   0  64 24 6898   192          pts/5  0:00 xclock
  200001 A 201  20540 22352   0  60 20  5a0   360  5a9b844  pts/5  0:00 ksh
# renice -n 8 7786
# ps -lu mike
      F S UID   PID  PPID   C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD
  200001 A 201  7786 20540   0  72 32 6898   192          pts/5  0:00 xclock
  200001 A 201  20540 22352   0  60 20  5a0   360  5a9b844  pts/5  0:00 ksh
```

In the previous example the renice command adds eight to the current nice value.

```
# ps -lu mike
      F S UID   PID  PPID   C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD
  200001 A 201  7798 20540   0  64 24 5874   192          pts/5  0:00 xclock
  200001 A 201  20540 22352   0  60 20  5a0   360  5a9b844  pts/5  0:00 ksh
# renice -n +8 7798
# ps -lu mike
      F S UID   PID  PPID   C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD
  200001 A 201  7798 20540   0  72 32 5874   192          pts/5  0:00 xclock
  200001 A 201  20540 22352   0  60 20  5a0   360  5a9b844  pts/5  0:00 ksh
```

The last example is similar to the third one, the renice command uses the current nice value for its modification and adds eight to it.

2.12 The prof Command

The prof command displays a profile of CPU usage for each external symbol (routine) of a specified program. In detail, it displays the percentage of execution time spent between the address of that symbol and the address of the next, the number of times that function was called, and the average number of milliseconds per call. It interprets the profile data collected by the monitor subroutine for the object file (a.out by default), reads the symbol table in the object file, and correlates it with the profile file (mon.out by default) generated by monitor. A usage report is sent to the terminal, or may be redirected to a file.

To use the prof command, a source program in C, FORTRAN, PASCAL, or COBOL must be compiled with the -p option to put a special profiling startup function that calls the monitor subroutine to track function calls, into the object file. When the program is executed, the monitor subroutine creates a mon.out file to track execution time. Therefore, only programs that explicitly exit or return from the main program cause the mon.out file to be produced. Also the -p flag causes the compiler to insert a call to the mcount subroutine into the object code generated for each recompiled function of your program. While the program runs, each time a parent calls a child function, the child calls the mcount subroutine to increment a distinct counter for that parent-child pair. This counts the number of calls to a function.

Note

Currently, you cannot use the prof command for profiling optimized code.

By default, the displayed report is sorted by decreasing percentage of CPU time. This is the same as when specifying the -t option. The options -c and -n sort the output differently. If the -s option is used, a summary file mon.sum is produced. This is useful when more than one profile file is specified with the -m option. The -m option specifies files containing monitor data. The -z option will include all symbols, even if there are zero calls and time associated.

2.12.1 The prof Implementation

The monitor routine calls the moncontrol routine to start the data gathering. The moncontrol routine calls a profile subroutine to start the address sampling that is used to determine what routine is executing at a particular time. The mcount routine provides the counter for the number of calls to each routine. All of these routines are needed, and are inserted for profiling when you compile with the -p flag.

The following example shows the first part of the prof output for a modified version of the Whetstone benchmark (Double Precision) program. It is well described in *A Synthetic Benchmark by H.J. Curnow and B.A. Wichman in Computer Journal, Vol. 19 #1,* February 1976:

```
# cc -o cwhet -p -lm cwhet.c
# cwhet > cwhet.out
# prof
Name                %Time     Seconds     Cumsecs  #Calls   msec/call
.main                33.0       21.54       21.54       1  21540.
.__mcount            27.2       17.74       39.28
.mod8                16.6       10.83       50.11 8990000      0.0012
.mod9                10.3        6.74       56.85 6160000      0.0011
```

.cos	3.0	1.96	58.81	**1920000**	0.0010
.log	2.4	1.57	60.38	**930000**	0.0017
.exp	2.3	1.50	61.88	**930000**	0.0016
.mod3	1.9	1.22	63.10	**140000**	0.0087
.sqrt	1.4	0.90	64.00		
.atan	1.1	0.74	64.74	**640000**	0.0012
.sin	0.9	0.59	65.33	**640000**	0.0009
._doprnt	0.0	0.01	65.34	**10**	1.0
.pout	0.0	0.00	65.34	**10**	0.0
.exit	0.0	0.00	65.34	**1**	0.
.free	0.0	0.00	65.34	**2**	0.

In this example, we see a lot of calls to the mod8 and mod9 routines. As a starting point, check in the source code why they are used so much. Another starting point could be to investigate why a routine needs so much time.

Note

If the program you want to monitor uses a fork, you have to be careful because the parent and the child will create the same file (mon.out). To avoid this, you have to change the current directory of the child process.

2.13 The gprof Command

The gprof command produces an execution profile of C, PASCAL, FORTRAN, or COBOL programs. The statistics of called subroutines are included in the profile of the calling program. The gprof command is useful in identifying how a program consumes CPU resources. It is roughly a superset of the prof command, giving additional information and providing more visibility to active sections of code.

2.13.1 The gprof Implementation

The source code must be compiled with the -pg option. This links in versions of library routines compiled for profiling and reads the symbol table in the named object file (a.out by default), correlating it with the call graph profile file (gmon.out by default). This means that the compiler inserts a call to the mcount function into the object code generated for each recompiled function of your program. The mcount function maintains counters for each time a parent calls a child function. Also the monitor function is enabled to estimate the time spent in each routine.

The gprof command generates two useful reports:

- The call-graph profile, which shows the routines in descending order by CPU time plus their descendants. It allows you to understand which parent routines called a particular routine most frequently and which child routines were called by a particular routine most frequently.
- The flat profile of CPU usage, which shows the usage by routine and number of calls, similar to the prof output.

Each part starts with explanatory pages describing the output columns. These pages could be suppressed by using the -b option.

The following example shows the profiling for the cwhet benchmark program. It is also used in 2.12.1, “The prof Implementation” on page 58:

```
# cc -o cwhet -pg -lm cwhet.c
# cwhet > cwhet.out
# gprof cwhet > cwhet.gprof
```

The Call-Graph Profile: The call-graph profile is the first part of the cwhet.gprof file.

```
granularity: Each sample hit covers 4 bytes. Time: 75.92 seconds

                                     called/total       parents
index  %time    self descendents  called+self    name           index
                                     called/total       children

                21.70       25.24       1/1           .__start [2]
[1]     61.8    21.70       25.24       1         .main [1]
                10.50        0.00 8990000/8990000     .mod8 [4]
                 7.01        0.00 6160000/6160000     .mod9 [5]
                 1.70        0.00  930000/930000      .log [7]
                 1.70        0.00  930000/930000      .exp [6]
                 1.54        0.00 1920000/1920000     .cos [8]
                 1.24        0.00  140000/140000      .mod3 [10]
                 0.93        0.00  640000/640000      .atan [12]
                 0.62        0.00  640000/640000      .sin [15]
                 0.00        0.00      10/10          .pout [28]

-----------------------------------------------

6.6s                                                  <spontaneous>
[2]     61.8    0.00       46.94                  .__start [2]
                21.70       25.24       1/1           .main [1]
                 0.00        0.00       1/1           .exit [38]

-----------------------------------------------
```

To read this report, look at the first index [1] in the left-hand column. The .main function is the current function. It was started by .__start (the parent function is on top of the current function), and it, in turn, calls .mod8 and .mod9 (the child functions are beneath the current function). All time of .main is propagated to .__start. The self and descendents columns of the children of the current function should add up to the descendents entry for the current function.

The Flat Profile: The flat profile sample is the second part of the cwhet.gprof file.

```
granularity: Each sample hit covers 4 bytes. Time: 75.92 seconds

  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 32.8      24.88    24.88                             .__mcount [3]
 28.6      46.58    21.70        1 21700.00 46940.00  .main [1]
 13.8      57.08    10.50  8990000     0.00     0.00  .mod8 [4]
  9.2      64.09     7.01  6160000     0.00     0.00  .mod9 [5]
  2.2      65.79     1.70   930000     0.00     0.00  .exp [6]
  2.2      67.49     1.70   930000     0.00     0.00  .log [7]
  2.0      69.03     1.54  1920000     0.00     0.00  .cos [8]
  2.0      70.52     1.49                             .qincrement [9]
  1.6      71.76     1.24   140000     0.01     0.01  .mod3 [10]
```

```
 1.3      72.78     1.02                              .__stack_pointer [11]
 1.2      73.71     0.93   640000     0.00     0.00  .atan [12]
 1.1      74.51     0.80                              .qincrement1 [13]
 1.0      75.29     0.78                              .sqrt [14]
 0.8      75.91     0.62   640000     0.00     0.00  .sin [15]
 0.0      75.92     0.01                              .__mcount [16]
 0.0      75.92     0.00      180     0.00     0.00  .fwrite [17]
 0.0      75.92     0.00      180     0.00     0.00  .memchr [18]
 0.0      75.92     0.00       90     0.00     0.00  .__flsbuf [19]
 0.0      75.92     0.00       90     0.00     0.00  ._flsbuf [20]
```

The flat profile is much less complex than the call-graph profile and very similar to the output of prof. The primary columns of interest are the self seconds and the calls columns. These reflect the CPU seconds spent in each function and the number of times each function was called. The next columns to look at are self ms/call, meaning CPU time used by the body of the function itself, and total ms/call, meaning time in the body of the function plus any descendent functions called.

Normally, the top functions on the list are targets for optimization, but you should also consider how many calls are made to the function. Sometimes it may be easier to make slight improvements to a frequently called function than to make extensive changes to a piece of code called once. A cross reference index is the last item produced:

```
Index by function name

 [19] .__flsbuf           [38] .exit               [5] .mod9
 [34] .__ioctl             [6] .exp               [43] .moncontrol
 [16] .__mcount           [39] .expand_catname    [44] .monitor
  [3] .__mcount           [32] .free              [22] .myecvt
 [23] .__nl_langinfo_std  [33] .free_y            [27] .nl_langinfo
 [11] .__stack_pointer    [17] .fwrite            [28] .pout
 [24] ._doprnt            [40] .getenv            [29] .printf
 [35] ._findbuf           [41] .ioctl              [9] .qincrement
 [20] ._flsbuf            [42] .isatty            [13] .qincrement1
 [36] ._wrtchk             [7] .log               [45] .saved_category_nam
 [25] ._xflsbuf            [1] .main              [46] .setlocale
 [26] ._xwrite            [18] .memchr            [15] .sin
 [12] .atan               [21] .mf2x2             [31] .splay
 [37] .catopen            [10] .mod3              [14] .sqrt
  [8] .cos                 [4] .mod8              [30] .write
```

Note

If the program you want to monitor uses a fork, you have to be careful because the parent and the child will create the same file (gmon.out). To avoid this, you have to change the current directory of the child process.

Chapter 3. Legacy (AIX) Performance Tools

This chapter describes some advanced, detailed commands AIX provides that usually are used to check system-performance issues. Some of them come with the AIX Basic Operating System; some of them are part of the AIX Performance Toolbox LPP software.

These tools were originally created as development tools with the intention to gather more detailed data than the standard UNIX tools are able to provide. The data gathered is unique to AIX and helps to solve more complex problems. These tools were the original advanced performance tools for AIX, but have now been expanded with even more tools. Some people have adopted the name legacy tools for this original set.

The purpose of this chapter is to provide information on when to use the legacy tools, what their output means, and how to interpret it. For detailed information on the syntax of the commands, refer to the *AIX Version 4 Command Reference*, SBOF-1851 and to InfoExplorer.

3.1 The tprof Command

Profilers are tools that allow you to get information about processes running on a system. There are two different types of profilers. The first type includes prof and gprof. These are is standard UNIX tools and need a program compiled with specific options to get the information. The second type includes tprof. It is an AIX-specific profiler and is based on the trace facility. By using these profilers, one can identify the heaviest processes on the machine, the heaviest subroutines in your programs, and the time spent in each subroutine.

The tprof command is a very versatile AIX profiler that provides a detailed profile of CPU usage for every AIX process ID and name. The tprof command is a trace-driven profiler and provides a global and a detailed view. It profiles at the application level, routine level, and even at the source-statement level. It can profile any program produced by one of the following compilers: C for AIX, C Set++ for AIX, and XL FORTRAN, but does not work with COBOL or PASCAL programs at the statement level.

3.1.1 The tprof Implementation

The raw data for tprof is obtained via the trace facility. When a program is profiled, the trace facility is activated and instructed to collect data from the trace hook (hook ID 234) that records the contents of the instruction address register when a system-clock interrupt occurs (100 times per second). Several other trace hooks are also activated to allow tprof to track process and dispatch activity. The trace records are not written to a disk file. They are written to a pipe that is read by a program that builds a table of the unique program addresses that have been encountered and the number of times each one occurred. When the trace is stopped, the table of addresses and their occurrence counts are written to disk. The data-reduction component of tprof then correlates the instruction addresses that were encountered with the ranges of addresses occupied by the various programs and reports the distribution of address occurrences (ticks) across the programs involved in the workload.

The distribution of ticks is roughly proportional to the CPU time spent in each program (10 milliseconds per tick). Once the programs that use a lot of CPU time have been identified, the programmer can take action to restructure the hot spots in the program or minimize their use.

The tprof command has been enhanced to provide the capability to profile kernel extensions, stripped executables, and stripped libraries. If the user program being profiled was compiled with the -g option, profiling is possible down to the source-line level. This is known as micro-profiling.

3.1.2 The Advantages of tprof

- It is a sophisticated tool. Standard UNIX performance tools often do not capture enough information to fully describe the enhanced performance and functionality of AIX. They cannot profile optimized code, need to have source code for recompilation, and there is sometimes an excessive overhead.
- There is little or no overhead. Unlike most profilers, CPU time is not increased as with prof and gprof. The operation of keeping track of the running process and the active instruction is really not CPU consuming, so that the process can run at normal speed, and the results are close to reality.
- It accepts optimized code. Since this profiler is AIX specific, it handles optimized code. However, compiling without the -0 option makes the line number tprof uses more precise. The results will also be more complete if the program has been compiled with the -g option (debug). When the C compiler is optimizing, it sometimes does enough rearrangement of code to make tprof output harder to interpret.
- The trace tool allows the user to see a broader spectrum of CPU usage, from the global view of all processes down to the individual source-code level.
- It can use executable programs for profiling. The tprof command can be run using the executable program for routine-level profiling. There is no need to recompile with special flags.
- It allows subroutine-level analysis. If your program is not stripped, tprof will give you the time spent in each subroutine.

3.1.3 The Limitations of tprof

- There is some inaccuracy when profiling short-running programs. Since activity is recorded at 100 samples per second, estimates for short-running programs may not be sufficiently accurate. Programs of less than 1 second duration are not shown in the report. Averaging multiple runs will help generate a more accurate picture.
- Only one user at a time can run tprof. The tprof command uses the system trace facility. Only one user can execute trace at a time; therefore, only one tprof can be running, and it cannot execute in parallel with other trace-based tools like filemon or netpmon.
- It only profiles CPU activity. There is no profiling of other system resources like memory, disks, and network.
- It accepts only C, C++, and FORTRAN for statement-level profiling. The tprof command will do subroutine-level profiling for any executable on which the stripnm command will produce a symbol table.

- You cannot profile routines when interrupts are disabled. The tprof command cannot determine the address of a routine when interrupts are disabled. Therefore, it charges any ticks that occur while interrupts are disabled to the i_enable (for AIX V3.2) or unlock_enable (AIX V4) routines.

Note

Since the tprof command uses the trace facility, it causes little system overhead. The tprof command enables only a few trace hooks; so its overhead is less than that of a full trace. For example, tprof degraded the performance of a large compile by less than two percent.

3.1.4 A Systemwide Example

The tprof command can be used to profile the entire system by specifying the sleep command as the command to execute. So if you want to profile the system for 10 minutes, run **"**tprof -x sleep 600**"**. This will gather tprof statistics for all processes running on the system for the duration of the next 600 seconds. After the data is collected, more post processing can be done with the -s, -k, and -p flags.

```
# tprof -x sleep 600
Starting Trace  now
Starting  sleep  600
Tue Feb 25 13:20:48 1997
System: AIX itsosmp Node: 4 Machine: 00045067A000
Trace is done now
601.939 secs in measured interval
 * Samples from __trc_rpt2
 * Reached second section of __trc_rpt2

# cat __prof.all
          Process      PID      TID    Total   Kernel     User   Shared    Other
          =======      ===      ===    =====   ======     ====   ======    =====
             wait      516      517    86486    86486        0        0        0
         cpubound    19918    19671    61766    59996      347     1423        0
         cpubound    19306    18803    59915    58250      301     1364        0
         cpubound    19056    17785    59905    58111      263     1531        0
             wait      774      775     3267     3267        0        0        0
            tprof    19576    19329      274       96      178        0        0
            trace    21106    20859      132      132        0        0        0
              gil     2064     2581      120      120        0        0        0
              gil     2064     2323      117      117        0        0        0
              gil     2064     2839      115      115        0        0        0
              gil     2064     3097      115      115        0        0        0
            syncd     3854     4631       98       98        0        0        0
            pstat    21628    21381       94       63        0       31        0
            pstat    21630    21383       83       52        3       28        0
             wait     1032     1033       80       80        0        0        0
         swapper        0        3       66       66        0        0        0
            init        1      459       47        5       42        0        0
         telnetd     4610    15115        6        5        0        1        0
     pciconsvr.ip    15080    16369        6        5        1        0        0
            netm     1806     1807        5        5        0        0        0
             ksh     6916     7181        5        4        1        0        0
          bootpd     8520     8785        5        4        0        1        0
            afsd     9296     9561        4        4        0        0        0
              sh    18810    18563        4        4        0        0        0
```

```
          wait     1290     1291        3        3        0        0        0
 pcimapsvr.ip    16870    14831        3        2        1        0        0
          cron    13988     3501        2        1        1        0        0
      sendmail     5610     5107        2        2        0        0        0
         tprof    15466    20595        2        2        0        0        0
      sendmail    21632    21385        2        2        0        0        0
         sleep    18810    18563        2        2        0        0        0
       syslogd     5348     5613        1        1        0        0        0
       portmap     5878     6143        1        1        0        0        0
         tprof    18796    18549        1        1        0        0        0
           ksh    21628    21381        1        1        0        0
           ksh    21630    21383        1        1        0        0        0
       =======      ===      ===    =====   ======     ====   ======    =====
         Total                     272736   267219     1138     4379        0

       Process     FREQ    Total   Kernel     User   Shared    Other
       =======      ===    =====   ======     ====   ======    =====
      cpubound        3   181586   176357      911     4318        0
          wait        4    89836    89836        0        0        0
           gil        4      467      467        0        0        0
         tprof        3      277       99      178        0        0
         pstat        2      177      115        3       59        0
         trace        1      132      132        0        0        0
         syncd        1       98       98        0        0        0
       swapper        1       66       66        0        0        0
          init        1       47        5       42        0        0
           ksh        3        7        6        1        0        0
       telnetd        1        6        5        0        1        0
  pciconsvr.ip        1        6        5        1        0        0
          netm        1        5        5        0        0        0
        bootpd        1        5        4        0        1        0
      sendmail        2        4        4        0        0        0
          afsd        1        4        4        0        0        0
            sh        1        4        4        0        0        0
  pcimapsvr.ip        1        3        2        1        0        0
          cron        1        2        1        1        0        0
         sleep        1        2        2        0        0        0
       syslogd        1        1        1        0        0        0
       portmap        1        1        1        0        0        0
       =======      ===    =====   ======     ====   ======    =====
         Total       36   272736   267219     1138     4379        0
```

The first part of the report shows the number of ticks consumed by, or on behalf of, each process. The PID is the process ID. The TID is thread ID. The Total column is the total ticks consumed by the process. The Kernel is ticks consumed in kernel mode. The User is ticks consumed in user mode, and the Shared is ticks consumed in shared library. The Other column is a catch-all category that is normally 0. A tick is 1/100 of a second. The Total at the bottom is the total ticks consumed by all processes. The processes are listed in descending order on Total ticks consumed.

The second part of the report summarizes the results by program, regardless of process ID. It shows the number (FREQ) of different occurrences for each program during the measured interval.

3.1.5 A Source-Line Example

The source-line example shows the profiling of the cwhet benchmark program. This program was also used as an example for the prof and gprof commands in Chapter 2, “Standard (UNIX) Performance Tools” on page 7.

To get a source-statement-level profiling, you need to compile with the -g option:

```
# cc -g -lm cwhet.c -o cwhet
```

Profile the program with the following command:

```
# tprof -k -p cwhet -x cwhet
Starting Trace  now
Starting  cwhet
Tue Mar 11 16:51:52 1997
System: AIX ah6000d Node: 4 Machine: 000002583800

     0      0      0 1.000000e+00 -1.000000e+00 -1.000000e+00 -1.000000e+00
120000 140000 120000 -7.139009e-05 7.538388e-04 -2.949938e-04 5.302094e-04
140000 120000 120000 -9.301742e-11 1.171939e-10 1.476543e-10 3.578717e-10
3450000      1      1 1.000000e+00 -1.000000e+00 -1.000000e+00 -1.000000e+00
2100000      1      2 6.000000e+00 6.000000e+00 1.476543e-10 3.578717e-10
320000       1      2 5.059279e-08 5.059279e-08 5.059152e-08 5.059152e-08
8990000      1      2 9.999000e-01 1.000000e+00 9.999000e-01 9.999000e-01
6160000      1      2 6.000000e+00 3.000000e+00 2.000000e+00 3.000000e+00
     0       2      3 1.000000e+00 -1.000000e+00 -1.000000e+00 -1.000000e+00
930000       2      3 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00

Trace is done now
62.755 secs in measured interval
* Samples from __trc_rpt2
* Reached second section of __trc_rpt2
```

The output shows tprof statements giving information about the status and the program execution time and the output of the compiled program. The cwhet output starts after the System: line and ends before the Trace is done now line. The line 62.755 secs in measured interval indicates that the profiling was active during this amount of time.

If the -x Command flag is used, tprof allows the execution of an arbitrary command. Subprograms of the program specified by the -p flag are profiled if this flag is specified. In our example the program cwhet will be profiled like this. The -k option will profile the kernel.

Additional options for tprof are: -e to profile kernel extensions, -v to get additional information and to keep intermediate files, -s to profile shared libraries, and -t Process_Id to profile a specific process. At least one of -s, -k, -e or -p needs to be specified to get a detailed profile, instead of just a summary.

If the -x Command flag is omitted, the tprof command uses the __trc_rpt2 trace report file in the current directory to produce its output.

For the previous example, a summary report named __cwhet.all is produced. This report contains an estimate of the amount of CPU time spent in each process that was executing while the tprof program was monitoring the system. It also contains an estimate of the amount of CPU time spent in each subprogram of the sample program. The summary report shows the amount of

time the CPU spends in kernel and user mode. The tprof command reports CPU time in ticks, where 100 ticks equals one second.

If source-statement-level profiling is indicated, additional files are created and placed in the current directory. For example, the __t.<routine>_cwhet.c files have a profiled source listing for each routine. This means they show each source line with line numbers and tick counts, in order by line number. The __h.cwhet.c shows the hottest source lines in the source file, and it is ordered by tick count. This makes it easy to find hot spots in the program.

Note

When the -p program flag is used, you should make sure that the PATH environment variable includes the current directory; otherwise the files __h.xxxx.c and __t.xxx_yyy.c will not be created.

The following example of the file __cwhet.all shows the reports of how many CPU ticks each routine involved in the execution consumed. All values are in ticks (1/100) of a second (the section lines are inserted for readability):

```
---------------------- First section -----------------------------------

   Process      PID      TID    Total  Kernel     User  Shared   Other
   =======      ===      ===    =====  ======    =====  ======   =====
    cwhet    48144    57113     5037       3     5034       0       0
   netscape     3318    51967      407      40      367       0       0
          X     2756     3533       45      38        7       0       0
       wait      516      517       11      11        0       0       0
       init        1      441        8       0        8       0       0
    tprof    17678    46615        8       1        7       0       0
         sh    48144    57113        8       5        3       0       0
      syncd     2074     2595        5       5        0       0       0
 xdaliclock    33426    29339        5       5        0       0       0
        gil     1032     1291        2       2        0       0       0
        gil     1032     1549        2       2        0       0       0
        gil     1032     2065        2       2        0       0       0
       afsd    11602    12123        2       2        0       0       0
        gil     1032     1807        1       1        0       0       0
       xant    35934     3943        1       0        1       0       0
    trace    17668    46605        1       1        0       0       0
    tprof    54272    54793        1       1        0       0       0
    =======      ===      ===    =====   ======     ====   ======    =====
      Total                       5546     119     5427       0       0

---------------------- Second section ----------------------------------

    Process     FREQ    Total   Kernel     User   Shared    Other
    =======     ====    =====   ======     ====   ======    =====
    cwhet       1        5037        3     5034        0        0
   netscape        1      407       40      367        0        0
          X        1       45       38        7        0        0
       wait        1       11       11        0        0        0
    tprof       2          9        2        7        0        0
       init        1        8        0        8        0        0
         sh        1        8        5        3        0        0
        gil        4        7        7        0        0        0
      syncd        1        5        5        0        0        0
 xdaliclock        1        5        5        0        0        0
```

```
       afsd        1        2        2        0        0        0
       xant        1        1        0        1        0        0
    trace       1        1        1        0        0        0
    =======      ===    =====   ======     ====   ======    =====
      Total       17     5546      119     5427        0        0

--------------------------- Third section ---------------------------------

Total Ticks For cwhet(    USER) =   5034

   Subroutine   Ticks     %           Source   Address  Bytes
=============  ======  ======         =======   =======  =====
        .main    2811    50.7         cwhet.c  268436748   3428
        .mod8     933    16.8         cwhet.c  268436088    140
        .mod9     417     7.5         cwhet.c  268435968    120
        .mod3     197     3.6         cwhet.c  268436228    344
         .exp     157     2.8
../../../../../../../src/bos/usr/ccs/lib/libm/POWER/exp.c  268442908 592
         .cos     150     2.7
../../../../../../../src/bos/usr/ccs/lib/libm/POWER/cos.c  268440884    608
         .log     140     2.5
../../../../../../../src/bos/usr/ccs/lib/libm/POWER/log.c  268442116    792
        .atan     101     1.8
../../../../../../../src/bos/usr/ccs/lib/libm/POWER/atan.c  268441492    624
        .sqrt      85     1.5          sqrt.s  268443500    356
         .sin      43     0.8
../../../../../../../src/bos/usr/ccs/lib/libm/POWER/sin.c  268440324    560

--------------------------- Fourth section --------------------------------

Total Ticks For cwhet(  KERNEL) =      3

    Subroutine   Ticks     %           Source   Address  Bytes
 =============  ======  ======         =======   =======  =====
.unlock_enable       2     0.0           low.s     37636    508
    .ld_findfp       1     0.0 ../../../../../src/bos/kernel/ldr/ld_libld.c
1120300    232
```

The first part of the report shows the number of ticks consumed by, or on behalf of, each process. The columns give tick counts for kernel code, user code, and shared code. The 0ther column is a catch-all category that is normally 0.

The second part of the report summarizes the results by program, regardless of process ID. It shows the number (FREQ) of different occurrences for each program during the measured interval.

The third part breaks down the user ticks associated with the executable program (user mode) being profiled. It reports the number of ticks used by each function in the executable (Ticks column) and the percentage of the total CPU ticks (5034) that each function's ticks represent (% column). The Source column shows where the corresponding subroutine comes from. The Address column is the virtual address where the subroutine resides. The Byte column contains the size of the subroutine. From an application-tuning perspective, the developer has control over the code that executes in user mode, but has no control over the code contained in system calls (system mode). The third part is omitted if there are no user-code ticks observed.

The fourth part breaks down the kernel ticks associated with the kernel functions (system mode) being profiled. There will be another part showing the shared library consumption if shared libraries were used and the -s flag was specified. System mode time can sometimes be reduced by using a more efficient subroutine or by reducing the number of times the system call is requested.

From the report above, we can see that 50 percent of the total CPU consumption in user mode is in the main subroutine. This is the majority of the user mode CPU time. From the second part, we can see that 9 ticks are spent for the tprof command and 1 tick for the trace command, which is started by the tprof command. This is a small usage compared to the total CPU time (5546); it is less than one percent.

Note

Remember that process names and identification numbers are not necessarily unique. The program associated with a given numerical process ID changes with each exec call. If one application program execs another, both program names will appear in the tprof output associated with the same process ID. When a process forks another process, the forked process inherits the process name of the original process, but it has a different process ID.

If you have access to the source code and want to know what lines of codes are being executed most frequently, you can look at the hottest lines file produced by tprof. The first column indicates the line number in the source file, and the second column is the number of ticks charged to that line. From top to bottom, it is listed in descending order. So the top one is the busiest line that has a significant effect on CPU performance of the application. One point to note is that no call counts are displayed; so you do not know how many times that line is called. The gprof output can give you this information. Following is an abbreviated example of the __h.cwhet.c file:

```
Hot Line Profile for cwhet.c

   Line Ticks

   125    670
   124    629
   193    261
   194    230
   192    212
   201    188
   143    180
   142    159
   100    150
    85    144
   202    123
   103    120
   104    105
   200     91
   101     84
```

From the annotated version of the routine main in the source file cwhet.c, called __t.main_cwhet.c, we can see the following: (The example shows the parts, which are marked as hot spots)

```
Ticks Profile for main in cwhet.c

   Line   Ticks   Source

... skip ...
   100    150          j = j * (k - j) * (l - k);
```

```
   101     84           k = l * k - (l - j ) * k;
   102     66           l = (l - k) * (k + j);
   103    120           e1[l - 2] = j + k + l;
   104    105           e1[k - 2] = j * k * l;
   105      -         }
   106      -     #ifdef POUT
   107      -         pout(n6, j, k, e1[0], e1[1], e1[2], e1[3]);
   108      -     #endif
   109      -
   110      -         /**** Module 7 : Trigonometric functions ****/
   111      -
   112      -         x = y = 0.5;
   113      8         for (i = 1; i <= n7; i++) {
   114     36            x = t * atan(t2 * sin(x) * cos(x) / (cos(x + y) + cos(x
   115     14            y = t * atan(t2 * sin(y) * cos(y) / (cos(x + y) + cos(x
   116      -         }
   117      -     #ifdef POUT
   118      -         pout(n7, j, k, x, x, y, y);
   119      -     #endif
   120      -
   121      -         /**** Module 8 Procedure Call ****/
   122      -
   123      -         x = y = z = 1.0;
   124    629         for (i = 1; i <= n8; i++) {
   125    670           mod8(x, y, &z);
   126      -     #ifdef HARD
   127     18           x = z;
   128      -     #endif
   129      -         }
... skip...
   173      -     } /* End of Main */

 2811 Total Ticks for main in cwhet.c
```

This shows that the largest numbers of ticks are associated with calling the subroutine mod8 and the **for** loop into which the subroutine is imbedded.

3.2 The svmon Command

The svmon command is an AIX-specific tool that provides global, process-level, and segment-level reporting of memory use. The svmon command gives a more in-depth analysis of memory usage. It is more informative but also more intrusive than the vmstat and ps commands. The svmon command captures a snapshot of the current state of memory. The displayed information can be analyzed using four different reports:

Global (-G)	Displays statistics describing the real memory and paging space in use for the whole system
Process (-P)	Displays memory usage statistics for active processes
Segment (-S)	Displays memory usage for a specified number of segments or the top ten
Detailed Segment (-D)	Displays detailed information on specified segments

The svmon command can be run in intervals with the -i option. The -G and -P options are the most useful.

The svmon command is not a true snapshot because it runs at the user level with interrupts enabled. If no command line argument is present, the -G is taken as the default. If an interval is used, statistics will be displayed until the command is killed or until the number of intervals, which can be specified right after the interval, is reached.The overhead to run svmon is substantial, but the information is very valuable.

Note

The svmon -G command uses about 3.2 seconds of CPU time. A svmon command for a single process (svmon -P <PID>), takes about .7 seconds of CPU time.

The svmon command can only be executed by the root user.

3.2.1 How Much Memory is in Use

To print out global statistics, use the -G flag. In this example, we will repeat it five times at two-second intervals.

```
# svmon -G -i 2 5
       m e m o r y           i n  u s e           p i n       p g  s p a c e
  size inuse  free   pin    work  pers  clnt   work  pers  clnt     size   inuse
 16384 16250   134  2006   10675  2939  2636   2006     0     0    40960   12674
 16384 16254   130  2006   10679  2939  2636   2006     0     0    40960   12676
 16384 16254   130  2006   10679  2939  2636   2006     0     0    40960   12676
 16384 16254   130  2006   10679  2939  2636   2006     0     0    40960   12676
 16384 16254   130  2006   10679  2939  2636   2006     0     0    40960   12676
```

Explanation of the svmon output

memory

- **size** Total size of memory in 4K pages.
- **inuse** Number of pages in RAM that are in use by a process plus the number of persistent pages that belonged to a terminated process and are still resident in RAM. This value is the total size of memory minus the number of pages on the free list.
- **free** Number of pages on the free list.
- **pin** Number of pages pinned in RAM (a pinned page is a page that is always resident in RAM and cannot be paged out).

in use

- **work** Number of working pages in RAM.
- **pers** Number of persistent pages in RAM.
- **clnt** Number of client pages in RAM (client page is a remote file page).

pin

- **work** Number of working pages pinned in RAM.
- **pers** Number of persistent pages pinned in RAM.
- **clnt** Number of client pages pinned in RAM.

pg space

- **size** Total size of paging space in 4 K pages.
- **inuse** Total number of allocated pages.

In our example, there are 16384 pages of total size of memory. One has to multiply this number by four to see the total real memory size (64 MB). While 16250 pages are in use, there are 134 pages on the free list and 2006 pages are pinned in RAM. Of the total pages in use, there are 10675 working pages in RAM, 2939 persistent pages in RAM, and 2636 client pages in RAM. The sum of these three parts is equal to the inuse column of the memory part. The pin part

divides the pinned memory size into working, persistent and client categories. The sum of them is equal to the pin column of the memory part. There are 40960 pages (160 MB) of total paging space, and 12676 pages are in use. The inuse column of memory is usually greater than the inuse column of pg spage because memory for file pages is not freed when a program completes while paging space allocation is.

If the -r flag is set, it reports real memory frames that have been recently referenced during the interval instead of the paging-space statistics.

```
# svmon -G -i 2 5 -r
       m e m o r y           i n  u s e           p i n            r e f
  size inuse  free   pin   work  pers  clnt   work  pers  clnt    inuse     pin
 16384 16010   374  2007  10701  2785  2524   2007     0     0     9532    1336
 16384 16014   370  2007  10704  2786  2524   2007     0     0      724     510
 16384 16014   370  2007  10704  2786  2524   2007     0     0      675     481
 16384 16216   168  2007  10704  2890  2622   2007     0     0     2129     957
 16384 16216   168  2007  10704  2890  2622   2007     0     0      756     485
```

This example shows that there are 9532 inuse memory frames and 1336 pinned memory frames that are referenced during the first two seconds.

3.2.2 Who is Using Memory?

The following command displays the memory usage statistics for the top 10 processes. If you do not specify a number, it will display all the processes currently running in this system.

```
# svmon -Pau 10

  Pid                     Command        Inuse        Pin      Pgspace
15012                  maker4X.exe         4783       1174         4781
 2750                            X         4353       1178         5544
15706                        dtwm          3257       1174         4003
17172                    dtsession         2986       1174         3827
21150                       dtterm         2941       1174         3697
17764                      aixterm         2862       1174         3644
 2910                       dtterm         2813       1174         3705
19334                       dtterm         2813       1174         3704
13664                       dtterm         2804       1174         3706
17520                      aixterm         2801       1174         3619

Pid:  15012
Command:  maker4X.exe

Segid  Type  Description          Inuse    Pin  Pgspace  Address Range
 1572  pers  /dev/hd3:62             0      0        0  0..-1
  142  pers  /dev/hd3:51             0      0        0  0..-1
 1bde  pers  /dev/hd3:50             0      0        0  0..-1
  2c1  pers  /dev/hd3:49             1      0        0  0..7
  9ab  pers  /dev/hd2:53289          1      0        0  0..0
  404  work  kernel extension       27     27        0  0..24580
 1d9b  work  lib data               39      0       23  0..607
  909  work  shared library text    864      0        7  0..65535
  5a3  work  sreg[4]                 9      0       12  0..32768
 1096  work  sreg[3]                32      0       32  0..32783
 1b9d  work  private              1057      1     1219  0..1306 : 65307..65535
 1af8  clnt                        961      0        0  0..1716
    0  work  kernel               1792   1146     3488  0..32767 : 32768..65535
...
```

The output is divided into summary and detail sections. The summary section lists the top 10 highest memory-usage processes in descending order.

Pid 15012 is the process ID that has the highest memory usage. The Command indicates the command name, in this case maker4X.exe. The Inuse column (total number of pages in real memory from segments that are used by the process) shows 4783 pages (each page is 4 KB). The Pin column (total number of pages pinned from segments that are used by the process) shows 1174 pages. The Pgspace column (total number of paging space pages that are used by the process) shows 4781 pages.

The detailed section displays information about each segment for each process that is shown in the summary section. This includes the segment ID, the type of the segment, description (a textual description of the segment, including the volume name and inode of the file for persistent segments), number of pages in RAM, number of pinned pages in RAM, number of pages in paging space, and address range.

The Address Range specifies one range for a persistent or client segment and two ranges for a working segment. The range for a persistent or a client segment takes the form '0..x,' where x is the maximum number of virtual pages that have been used. The range field for a working segment can be '0..x : y..65535', where 0..x contains global data and grows upward, and y..65535 contains stack area and grows downward. For the address range, in a working segment, space is allocated starting from both ends and working towards the middle. If the working segment is non-private (kernel or shared library), space is allocated differently. In this example, the segment ID 1b9d is a private working segment; its address range is 0..1306 : 65307..65535. The segment ID 909 is a shared library text working segment; its address range is 0..65535.

A segment may be used by multiple processes. Each page in real memory from such a segment is accounted for in the Inuse field for each process using that segment. Thus, the total for Inuse may exceed the total number of pages in real memory. The same is true for the Pgspace and Pin fields. The sum of Inuse, Pin, and Pgspace of all segments of a process is equal to the numbers in the summary section.

3.2.3 Detailed Information on a Specific Segment ID

The -D option displays detailed memory-usage statistics for segments.

```
# svmon -D 404
Segid:  404
Type:  working
Description:  kernel extension
Address Range:  0..24580
Size of page space allocation:  0 pages ( 0.0 Mb)
Inuse:  28 frames ( 0.1 Mb)
    Page   Frame      Pin     Ref     Mod
   12294    3320      pin     ref     mod
   24580    1052      pin     ref     mod
   12293   52774      pin     ref     mod
   24579   20109      pin     ref     mod
   12292   19494      pin     ref     mod
   12291   52108      pin     ref     mod
   24578   50685      pin     ref     mod
   12290   51024      pin     ref     mod
   24577    1598      pin     ref     mod
   12289   35007      pin     ref     mod
   24576     204      pin     ref     mod
   12288     206      pin     ref     mod
    4112   53007      pin             mod
```

```
   4111   53006      pin             mod
   4110   53005      pin             mod
   4109   53004      pin             mod
   4108   53003      pin             mod
   4107   53002      pin             mod
   4106   53001      pin             mod
   4105   53000      pin             mod
   4104   52999      pin             mod
   4103   52998      pin             mod
   4102   52997      pin             mod
   4101   52996      pin             mod
   4100   52995      pin             mod
   4099   52994      pin             mod
   4098   52993      pin             mod
   4097   52992      pin     ref     mod
```

We already explained the Segid, Type, Description, Address Range, Size of page space allocation, and Inuse fields. Let's look at the detailed part:

Page Specifies the index of the page within the segment
Frame Specifies the index of the real memory frame that the page resides in
Pin Specifies a flag indicating whether the page is pinned
Ref Specifies a flag indicating whether the page's reference bit is on
Mod Specifies a flag indicating whether the page is modified

The size of page space allocation is 0 because all the pages are pinned in real memory.

3.2.4 List of Top Memory Usage of Segments

The -S option is used to sort segments by memory usage and to display the memory-usage statistics for the top memory-usage segments. If count is not specified, then a count of 10 is implicit. The following command sorts system and non-system segments by the number of pages in real memory and prints out the top 10 segments of the resulting list.

```
# svmon -Sau

Segid  Type  Description          Inuse    Pin  Pgspace  Address Range
    0  work  kernel                1990   1408     3722  0..32767 : 32768..65535
    1  work  private, pid=4042     1553      1     1497  0..1907 : 65307..65535
 1435  work  private, pid=3006     1391      3     1800  0..4565 : 65309..65535
 11f5  work  private, pid=14248    1049      1     1081  0..1104 : 65307..65535
 11f3  clnt                        991      0        0  0..1716
  681  clnt                        960      0        0  0..1880
  909  work  shared library text    900      0        8  0..65535
  101  work  vmm data               497    496        1  0..27115 : 43464..65535
  a0a  work  shared library data    247      0      718  0..65535
 1bf9  work  private, pid=21094     221      1      320  0..290 : 65277..65535
```

All output fields have been described in the previous examples.

3.2.5 Correlating svmon and vmstat Outputs

There are some relationships between the svmon and vmstat outputs.

```
# svmon -G
               memory                    in use                    pin       pg space
     size    inuse    free     pin    work   pers   clnt    work   pers   clnt    size   inuse
    16384    16254     130    2016   11198   2537   2519    2016      0      0   40960   13392
```

The vmstat command was run in a separate window while the svmon command was running.

```
# vmstat 5
kthr     memory             page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 0  0 13392   130   0   0   0   0    2   0 125  140  36  2  1 97  0
 0  0 13336   199   0   0   0   0    0   0 145 14028  38 11 22 67  0
 0  0 13336   199   0   0   0   0    0   0 141   49  31  1  1 98  0
 0  0 13336   199   0   0   0   0    0   0 142   49  32  1  1 98  0
 0  0 13336   199   0   0   0   0    0   0 145   49  32  1  1 99  0
 0  0 13336   199   0   0   0   0    0   0 163   49  33  1  1 92  6
 0  0 13336   199   0   0   0   0    0   0 142   49  32  0  1 98  0
```

The global svmon report shows related numbers. The number that vmstat reports as Active Virtual Memory (avm) is reported by svmon as pg space inuse (13392). The number of page frames on the free list (130) is identical in both reports.

3.2.6 Correlating svmon and ps Outputs

There are some relationships between the svmon and ps outputs.

```
# svmon -P 7226

  Pid                      Command        Inuse        Pin      Pgspace
 7226                      telnetd          936          1           69

Pid:   7226
Command:  telnetd

Segid  Type  Description          Inuse    Pin  Pgspace  Address Range
  828  pers  /dev/hd2:15333           0      0        0  0..0
 1d3e  work  lib data                 0      0       28  0..559
  909  work  shared library text    930      0        8  0..65535
 1cbb  work  sreg[3]                  0      0        1  0..0
 1694  work  private                  6      1       32  0..24 : 65310..65535
 12f6  pers  code,/dev/hd2:69914      0      0        0  0..11
```

Compare with:

```
# ps v 7226
   PID    TTY STAT  TIME PGIN  SIZE   RSS   LIM  TSIZ   TRS %CPU %MEM COMMAND
  7226      - A     0:00   51   240    24 32768    33     0  0.0  0.0 telnetd
```

SIZE refers to the virtual size in KB of the data section of the process (in paging space). This number is equal to the number of working-segment pages of the process that have been touched (that is, the number of paging-space pages that have been allocated) times four. It needs to be multiplied with four since pages are in 4 KB units and SIZE is in 1 KB units. If some working-segment pages are currently paged out, this number is larger than the amount of real memory being used. The SIZE value (240) correlates with the Pgspace number from svmon for private (32) plus lib data (28) in 1 KB units. The library data is new in AIX V4 (it was part of the shared library sreg[13] in AIX V3). Hence, the output from AIX V3 is different from that shown here, so is the translation formula.

RSS refers to the real memory (resident set) size in KB of the process. This number is equal to the sum of the number of working-segment and code-segment pages in memory times four. Remember that code-segment pages are shared among all of the currently running instances of the program. If 26 ksh processes are running, only one copy of any given page of the ksh executable would be in memory, but ps would report that code-segment size as

part of the RSS of each instance of ksh. The RSS value (24) correlates with the Inuse numbers from svmon for private (6) working-storage segments, for code (0) segments, and for lib data (0) of the process in 1 KB units.

TRS refers to the size of the resident set (real memory) of text. This is the number of code-segment pages times four. As was noted earlier, this number exaggerates memory use for programs of which multiple instances are running. This does not include the shared text of the process. The TRS value (0) correlates with the number of the svmon pages in the code segment (0) of the Inuse column in 1 KB units. The TRS value can be higher than the TSIZ value because other pages, such as the XCOFF header and the loader section, may be included in the code segment.

```
SIZE = 4 * Pgspace of (work lib data + work private)
RSS = 4 * Inuse of (work lib data + work private + pers code)
TRS = 4 * Inuse of (pers code)
```

TSIZ refers to the size of the text section of the executable file. Pages of the text section of the executable are only brought into memory when they are touched, that is, branched to or loaded from. The TSIZ value does not reflect actual memory usage so it is not shown by svmon. The TSIZ value can also be seen by executing dump -ov against an executable (for example, dump -ov /usr/bin/ls).

3.2.7 Finding Memory-Leaking Programs

A memory leak is a program bug that consists of repeatedly allocating memory, using it, and then neglecting to free it. A memory leak in a long-running program, such as an interactive application, is a serious problem because it can result in memory fragmentation and the accumulation of large numbers of mostly garbage-filled pages in real memory and page space. Systems have been known to run out of page space because of a memory leak in a single program.

A memory leak can be detected with svmon or with the ps v command by looking for processes whose private plus lib data working segments continually grow. Identifying the offending subroutine or line of code is more difficult, especially in AIX Windows applications, which generate large numbers of malloc() and free() calls. Some third-party programs exist for analyzing memory leaks, but they usually require access to the program source code.

Some uses of realloc(), while not actually programming errors, can have the same effect as a memory leak. If a program frequently uses realloc() to increase the size of a data area, the working segment of the process can become increasingly fragmented if the storage released by realloc() cannot be reused for anything else.

In general, memory that is no longer required should be released with the free() and disclaim() system calls. On the other hand, it wastes CPU time to free memory after the last malloc(), if the program will finish soon. When the program terminates, its working segment is destroyed and the real memory page frames that contained working-segment data are added to the free list. Following is a memory-leaking program example; its Inuse, Pgspace, and Address Range of the private working segment are continually growing.

```
# svmon -P 19556 -i 1 10
  Pid                       Command        Inuse        Pin      Pgspace
19556                        pacman         3085          1         1580
Pid:  19556
Command:  pacman
Segid  Type  Description          Inuse    Pin  Pgspace  Address Range
  9c8  pers  /dev/hd2:53289           1      0        0  0..0
  aaf  work  lib data                12      0        6  0..1081
  909  work  shared library text    1502      0        7  0..65535
 1eba  work  private                1568      1     1567  0..1562 : 65313..65535
 16f3  pers  code,/dev/lv01:12302      2      0        0  0..1


  Pid                       Command        Inuse        Pin      Pgspace
19556                        pacman         3114          1         1609
Pid:  19556
Command:  pacman
Segid  Type  Description          Inuse    Pin  Pgspace  Address Range
  9c8  pers  /dev/hd2:53289           1      0        0  0..0
  aaf  work  lib data                12      0        6  0..1081
  909  work  shared library text    1502      0        7  0..65535
 1eba  work  private                1597      1     1596  0..1591 : 65313..65535
 16f3  pers  code,/dev/lv01:12302      2      0        0  0..1


  Pid                       Command        Inuse        Pin      Pgspace
19556                        pacman         3149          1         1644
Pid:  19556
Command:  pacman
Segid  Type  Description          Inuse    Pin  Pgspace  Address Range
  9c8  pers  /dev/hd2:53289           1      0        0  0..0
  aaf  work  lib data                12      0        6  0..1081
  909  work  shared library text    1502      0        7  0..65535
 1eba  work  private                1632      1     1631  0..1626 : 65313..65535
 16f3  pers  code,/dev/lv01:12302      2      0        0  0..1
...
```

3.2.8 Calculating the Minimum Memory Requirement of a Program

To calculate the minimum memory requirement of a program, use:

> Total memory pages (4 KB units) = T + (N * (PD + LD)) + F

where:

T = # of pages for text (shared by all users)
N = # of copies of this program running simultaneously
PD = # of working segment pages in process private segment
LD = # of shared library data pages used by the process
F = # of file pages (shared by all users)

Multiplying the result by four gives the number of KB required. You may want to add the kernel, kernel extension, and shared library text-segment requirements to this as well.

If we estimate the minimum memory requirement for the above program `pacman`, the formula would be:

T = 2 (Inuse of code,/dev/lv01:12302 of pers)
PD = 1632 (Inuse of private of work)
LD = 12 (Inuse of lib data of work)
F = 1 (Inuse of /dev/hd2:53289 of pers.

That is 2 + (N * (1632+ 12)) + 1, equal to 1644 * N + 3 in 4 KB units.

3.3 The rmss Command

The command name rmss is an acronym for Reduced-Memory System Simulator. The rmss command provides you with a means to simulate IBM RS/6000 systems with different sizes of real memory that are smaller than your actual machine, without having to extract and replace memory boards. Moreover, rmss provides a facility to run an application over a range of memory sizes, displaying, for each memory size, performance statistics such as the response time of the application and the amount of paging. In short, rmss is designed to help you answer the question: "How many megabytes of real memory does an RS/6000 need to run AIX and a given application with an acceptable level of performance?" or, in the multiuse context, "How many users can run this application simultaneously in a machine with X MB of real memory?"

It is important to keep in mind that the memory size simulated by rmss is the total size of the machine's real memory size, including the memory used by AIX and any other programs that may be running. It is not the amount of memory used specifically by the application itself. Because of the performance degradation it can cause, rmss can be used only by root or a member of the system group.

Important note

Before using rmss, it is a good idea to use the command schedtune -h 0 to turn off VMM memory-load control. Otherwise, VMM memory-load control may interfere with your measurements at small memory sizes. When your experiments are complete, reset the memory load control parameters to the values that are normally in effect on your system. (If you normally use the default parameters, use schedtune -D.)

The rmss command reduces the effective memory size of an RS/6000 by stealing free page frames from the list of free frames that is maintained by the VMM. The stolen frames are kept in a pool of unusable frames and are returned to the free frame list when the effective memory size is to be increased. Also, rmss dynamically adjusts certain system variables and data structures that must be kept proportional to the effective size of memory.

It is also important to run the application multiple times at each memory size. There are two good reasons for doing so. First, when changing memory size, rmss often clears out a lot of memory. Thus, the first time you run your application after changing memory sizes it is possible that a substantial part of the run time may be due to your application reading files into real memory. But, since the files may remain in memory after your application terminates, subsequent executions of your application may result in substantially shorter elapsed times. Another reason to run multiple executions at each memory size is to get a feel for the average performance of the application at that memory size. The RS/6000 and AIX are complex systems, and it is impossible to duplicate the system state each time your application runs. Because of this, the performance of your application may vary significantly from run to run.

3.3.1 Using rmss

- To display the current memory size, use:

  ```
  # rmss -p
  Simulated memory size is 64 Mb.
  ```

  This command shows that the current memory size is 64 MB. If you have used the rmss command to reduce the memory size, it will display the current memory size setting, not real memory size.

- To restore the memory to actual value, use:

  ```
  # rmss -r
  Simulated memory size changed to 64 Mb.
  ```

  This command restores the memory size to real memory size. The rmss command reports usable real memory. On some machines that contain bad memory or memory that is in use, rmss reports the amount of real memory as the amount of physical real memory minus the memory that is bad or in use by the system.

- To reduce memory size, use:

  ```
  # rmss -c 11.5
  Simulated memory size changed to 11.5 Mb.
  ```

  This command reduces the usable memory size to 11.5 MB. The memory size is an integer or decimal number in units of megabytes (for example, 12.25). It must be between 4 MB and the amount of physical real memory in your machine. Sometimes the rmss command may not be able to change the memory size to less than 8 MB because of the size of inherent system structures such as the kernel.

Note

Never forget to reset the system memory size after you use the rmss command to reduce it.

3.3.2 Simulating Different Memory Sizes

Let's run rmss with a program "eatmem.c" that is a memory-eating program.

```
# rmss -s 64 -f 8 -d 8 -n 1 -o eatmem1.out eatmem
```

This command starts simulating memory sizes from 64 MB (-s 64) and ends at 8 MB (-f 8), with an 8 MB difference (-d 8) each time it runs the eatmem program. The program (eatmem) is executed and measured once at each memory size (-n 1). The output is written to file eatmem1.out (-o).

```
# cat eatmem1.out
Hostname:  itsosmp.itsc.austin.ibm.com
Real memory size:  127.40625 Mb
Time of day:  Mon Mar 10 08:21:54 1997
Command:  eatmem
Simulated memory size initialized to 64 Mb.
Number of iterations per memory size = 1 warmup + 1 measured = 2.
Memory size     Avg. Pageins    Avg. Response Time      Avg. Pagein Rate
(megabytes)                           (sec.)            (pageins / sec.)
--------------------------------------------------------------------------
64                  1.0               4.9                   0.2
56                  0.0               4.7                   0.0
```

```
48              10.0            4.6             4.5
40              0.0             4.6             0.0
32              0.0             4.6             0.0
24              56.0            4.7             10.0
16              116.0           10.8            12.8
Unable to simulate 8 Mb because of excess of pinned pages.
The results are valid, but you may want to try a larger final memory size
next time.
```

The report displays the real memory size to be 127.40625 MB, starting simulated memory size at 64 MB, running program eatmem, and running two times at each simulated memory size.

We can see a 'warmup' word in the output. The rmss always runs the command once at each memory size as a warmup before running and actually measuring the command. The warmup is needed to avoid the I/O that occurs when the application is not already in memory. Although such I/O does affect performance, it is not necessarily due to a lack of real memory. The warmup run is not included in the number of iterations specified by the -n flag. In order to better control the number of iterations, you should specify the -n value; otherwise the system will determine during initialization how many times your application must be run in order to accumulate a total run time of 10 seconds.

The report consists of four columns. The left-most column gives the memory size, while the Avg. Pageins column gives the average number of page-ins that occurred when the application was run at that memory size. The Avg. Response Time column gives the average amount of time it took the application to complete, while the Avg. Pagein Rate column gives the average rate of page-ins per second. It is important to note that the Avg. Pageins and Avg. Pagein Rate columns refer to all page-in operations that occurred while the program was run, not only those initiated by the program.

From 64 MB to 32 MB, the Avg. Pagein Rate is relative small; from 24 MB to 16 MB, the Avg.Pagein Rate grows gradually. While the Avg.Pageins actually decrease when the memory size changes from 48 MB (10.0 pageins) to 40 MB (0.0 pageins), it should not be viewed with alarm. In a real-life system it is impossible to expect the results to be perfectly smooth. The important point is that the Avg.Pagein Rate is relatively low at both 48 MB and 40 MB.

Finally, there are a couple of deductions that we can make from the report. First of all, if the performance of the application is deemed unacceptable at 24 MB (as it probably would be), then adding memory would improve performance significantly. Note that the response time rises from approximately 4.6 seconds at 48 MB to 10.8 seconds at 16 MB, an increase of 135 percent. On the other hand, if the performance is deemed unacceptable at 48 MB, adding memory will not improve performance much because page-ins do not slow the program appreciably at 48 MB.

Now let's look at a test with an interval value of 2 and focus on the range from 32 MB to 8 MB.

```
# rmss -s 32 -f 8 -d 2 -n 1 -o eatmem2.out eatmem
# cat eatmem2.out

Hostname:  itsosmp.itsc.austin.ibm.com
Real memory size:  127.40625 Mb
Time of day:  Mon Mar 10 08:28:14 1997
```

```
Command:  eatmem
Simulated memory size initialized to 32 Mb.
Number of iterations per memory size = 1 warmup + 1 measured = 2.
Memory size     Avg. Pageins    Avg. Response Time      Avg. Pagein Rate
(megabytes)                           (sec.)            (pageins / sec.)
---------------------------------------------------------------------------
32                  1.0               4.7                   0.2
30                  5.0               4.7                   1.6
28                  10.0              4.6                   3.0
26                  0.0               4.7                   0.0
24                  363.0             6.0                   50.9
22                  484.0             8.6                   56.4
20                  481.0             8.2                   58.4
18                  589.0             11.0                  67.3
16                  600.0             13.6                  74.0
Unable to simulate 14 Mb because of excess of pinned pages.
The results are valid, but you may want to try a larger final memory size
next time.
```

By reducing the interval value, you could find out what memory size can accommodate the application without a lot of page-ins. When it is at 28 MB, its Avg. Pagein Rate is relatively high (3.0 page-ins/sec); and decreasing to 0.0 page-ins/sec when going from 28 MB to 26 MB. It finally goes up to 74.0 page-ins/sec when it is 16 MB. But what does it mean? According to the figures in the last example, it could be some kind of daemon running during the test run at 28 MB, or some other outside influences. Take a look at the Avg. Response Time, it grows gradually when memory changes in the 26 MB to 16 MB range. Therefore, you should concentrate on that range.

The two examples tell us the importance of choosing an interval so that the paging-rate measurements can be clearly interpreted. If you choose too large an interval, it will be hard to tell what the precise amount of memory should be; conversely, an interval too small will take a long time to test. From a larger to smaller interval may be a good way to simulate more precisely.

The final smallest memory size that is attainable by using rmss varies with the size of real memory. When rmss is unable to change to a given memory size, it displays an informative error message.

As mentioned earlier, rmss reduces the effective memory size of an RS/6000 by stealing free page frames from the list of free frames that is maintained by the VMM. Please refer to 2.1.2, “Memory Bound” on page 10.

Note

It may take a short while (up to 15 to 20 seconds) to change the memory size. In general, the more you wish to reduce the memory size, the longer time the rmss command takes to complete.

3.4 The filemon Command

The filemon command collects and presents trace data on the various layers of file system utilization, including the logical file system, virtual memory segments, LVM, and physical disk layers. Data can be collected on all the layers, or on specific layers by specifying the -0 layer option. The default is to collect data on the VM segments, LVM, and physical layers. Both summary and detailed reports are generated.

Important Note

The filemon command will only collect data for those files opened after filemon was started unless you specify the -u flag.

The filemon command uses the trace facility to obtain a detailed picture of I/O activity during a time interval. Since it uses the trace facility, filemon can be run only by root or by a member of the system group, and it cannot be executed in parallel with other trace-based commands like tprof and netpmon. Tracing is started by the filemon command, optionally suspended with trcoff, resumed with trcon, and terminated with trcstop. As soon as tracing is terminated, filemon writes its report to stdout.

3.4.1 Using filemon

The following commands give a simple example of filemon usage measuring a copy of a large file from one disk to another disk. The output is customized for easy reading.

```
# filemon -o fmon.out -O all ; cp /large_file/bigfile /temp_dir/big.sav;trcstop

Fri Mar 14 10:25:36 1997
System: AIX itsosmp Node: 4 Machine: 00045067A000

33.431 secs in measured interval
Cpu utilization:  25.8%
```

The 33.431 secs is the elapsed time in seconds while the trace was active. The CPU utilization is 25.8 percent during the 33.431 seconds elapsed time.

```
Most Active Files
------------------------------------------------------------------------
  #MBs  #opns   #rds   #wrs  file                volume:inode
------------------------------------------------------------------------
  98.7      1  25271      0  bigfile             /dev/test_lv:18
  98.7      1      0  25270  big.sav             /dev/templv:17
   0.0      2      4      0  ksh.cat             /dev/hd2:10613
   0.0      1      2      0  cmdtrace.cat        /dev/hd2:10484

Most Active Segments
------------------------------------------------------------------------
  #MBs  #rpgs  #wpgs  segid  segtype             volume:inode
------------------------------------------------------------------------
  98.7  25270      0   1629  ???
  98.7      0  25268   34fb  ???
   0.1     14     18   0241  .inodes             /dev/hd2:3
   0.1     26      0   26f2  .indirect           /dev/test_lv:4
   0.0      1      3   196d  .diskmap            /dev/test_lv:6
   0.0      3      0   0c66  persistent          /dev/hd2:2061
   0.0      0      2   19cd  page table
...

Most Active Logical Volumes
------------------------------------------------------------------------
```

```
  util  #rblk  #wblk   KB/s  volume               description
------------------------------------------------------------------------
  0.99     24 202256 3025.4  /dev/templv          /temp_dir
  0.96 202376     64 3027.8  /dev/test_lv         /large_file
  0.02     88    144    3.5  /dev/hd4             /
  0.02    176    144    4.8  /dev/hd2             /usr
  0.00      0     48    0.7  /dev/hd8             jfslog
  0.00      0      8    0.1  /dev/loglv01         jfslog
  0.00      0      8    0.1  /dev/loglv00         jfslog

Most Active Physical Volumes
------------------------------------------------------------------------
  util  #rblk  #wblk   KB/s  volume               description
------------------------------------------------------------------------
  0.95     24 202264 3025.5  /dev/hdisk2
  0.82 202376     72 3027.9  /dev/hdisk1
  0.04    264    336    9.0  /dev/hdisk0


------------------------------------------------------------------------
Detailed File Stats
------------------------------------------------------------------------

FILE: bigfile  volume: /dev/test_lv (/large_file)  inode: 18
opens:1
total bytes xfrd:103510016
reads:25271(0 errs)
  read sizes (bytes):avg  4096.0 min    4096 max    4096 sdev     0.0
  read times (msec):avg   0.921 min   0.020 max  85.137 sdev   2.578

FILE: /temp_dir/big.sav  volume: /dev/templv  inode: 17
opens:1
total bytes xfrd:103505920
writes:25270(0 errs)
  write sizes (bytes):avg  4096.0 min    4096 max    4096 sdev     0.0
  write times (msec):avg   0.372 min   0.233 max  60.513 sdev   0.521

FILE: /usr/lib/nls/msg/en_US/ksh.cat  volume: /dev/hd2 (/usr)  inode: 10613
opens:2
total bytes xfrd:16384
reads:4(0 errs)
  read sizes (bytes):avg  4096.0 min    4096 max    4096 sdev     0.0
  read times (msec):avg   4.998 min   0.059 max  19.612 sdev   8.438
lseeks:10

FILE: /usr/lib/nls/msg/en_US/cmdtrace.cat  volume: /dev/hd2 (/usr)  inode: 10484
opens:1
total bytes xfrd:8192
reads:2(0 errs)
  read sizes (bytes):avg  4096.0 min    4096 max    4096 sdev     0.0
  read times (msec):avg   4.130 min   0.175 max   8.085 sdev   3.955
lseeks:8

------------------------------------------------------------------------
Detailed VM Segment Stats   (4096 byte pages)
------------------------------------------------------------------------

SEGMENT: 1629  segtype: ???
segment flags:
reads:25270(0 errs)
  read times (msec):avg  15.089 min   2.673 max  92.985 sdev   7.207
  read sequences: 1
  read seq. lengths:avg 25270.0 min   25270 max   25270 sdev     0.0

SEGMENT: 34fb  segtype: ???
segment flags:
writes:25268(0 errs)
  write times (msec):avg  59.685 min   9.728 max 116.388 sdev  17.883
  write sequences: 29
  write seq. lengths:avg   871.3 min       1 max   25240 sdev  4605.2

SEGMENT: 0241  segtype: .inodes  volume: /dev/hd2  inode: 3
segment flags:pers defer
reads:14(0 errs)
  read times (msec):avg  17.543 min   4.416 max  36.813 sdev   8.921
```

```
  read sequences: 13
  read seq. lengths:avg     1.1 min       1 max       2 sdev     0.3
writes:18(0 errs)
  write times (msec):avg 142.172 min  44.010 max 288.454 sdev  70.872
  write sequences: 18
  write seq. lengths:avg     1.0 min       1 max       1 sdev     0.0

SEGMENT: 26f2  segtype: .indirect  volume: /dev/test_lv  inode: 4
segment flags:pers defer
reads:26(0 errs)
  read times (msec):avg   9.553 min   2.351 max  18.087 sdev   4.232
  read sequences: 1
  read seq. lengths:avg    26.0 min      26 max      26 sdev     0.0

SEGMENT: 196d  segtype: .diskmap  volume: /dev/test_lv  inode: 6
segment flags:pers defer
reads:1(0 errs)
  read times (msec):avg  36.281 min  36.281 max  36.281 sdev   0.000
  read sequences: 1
  read seq. lengths:avg     1.0 min       1 max       1 sdev     0.0
writes:3(0 errs)
  write times (msec):avg  47.309 min  29.471 max  56.260 sdev  12.613
  write sequences: 3
  write seq. lengths:avg     1.0 min       1 max       1 sdev     0.0

SEGMENT: 0c66  segtype: persistent  volume: /dev/hd2  inode: 2061
segment flags:pers
reads:3(0 errs)
  read times (msec):avg  13.528 min   9.769 max  17.950 sdev   3.373
  read sequences: 1
  read seq. lengths:avg     3.0 min       3 max       3 sdev     0.0

SEGMENT: 19cd  segtype: page table
segment flags:pgtbl
writes:2(0 errs)
  write times (msec):avg  17.206 min  14.558 max  19.854 sdev   2.648
  write sequences: 2
  write seq. lengths:avg     1.0 min       1 max       1 sdev     0.0

------------------------------------------------------------------------
Detailed Logical Volume Stats   (512 byte blocks)
------------------------------------------------------------------------

VOLUME: /dev/templv  description: /temp_dir
reads:3(0 errs)
  read sizes (blks): avg     8.0 min       8 max       8 sdev     0.0
  read times (msec):avg  37.823 min   3.526 max  59.813 sdev  24.572
  read sequences: 3
  read seq. lengths:avg     8.0 min       8 max       8 sdev     0.0
writes:6548(0 errs)
  write sizes (blks): avg    30.9 min       8 max      32 sdev     4.5
  write times (msec):avg  58.943 min   9.254 max 251.050 sdev  17.853
  write sequences: 54
  write seq. lengths:avg  3745.5 min       8 max   10528 sdev  3668.5
seeks:56(0.9%)
  seek dist (blks):init    120,
avg 96615.0 min       8 max  231320 sdev 81102.0
time to next req(msec): avg   5.085 min   0.015 max  87.129 sdev   4.884
throughput:3025.4 KB/sec
utilization:0.99

VOLUME: /dev/test_lv  description: /large_file
reads:4829(0 errs)
  read sizes (blks): avg    41.9 min       8 max      64 sdev    21.4
  read times (msec):avg  15.698 min   2.302 max  92.701 sdev   7.984
  read sequences: 1384
  read seq. lengths:avg   146.2 min       8 max   11424 sdev   475.7
writes:6(0 errs)
  write sizes (blks): avg    10.7 min       8 max      16 sdev     3.8
  write times (msec):avg  40.992 min  21.197 max  69.583 sdev  16.724
  write sequences: 6
  write seq. lengths:avg    10.7 min       8 max      16 sdev     3.8
seeks:1390(28.7%)
  seek dist (blks):init    112,
avg  1045.6 min       8 max  188496 sdev 12273.7
```

```
time to next req(msec): avg   6.883 min   0.003 max  89.506 sdev   6.925
throughput:3027.8 KB/sec
utilization:0.96

VOLUME: /dev/hd4  description: /
reads:11(0 errs)
  read sizes (blks): avg     8.0 min       8 max       8 sdev     0.0
  read times (msec):avg  19.621 min   5.972 max  52.266 sdev  13.585
  read sequences: 9
  read seq. lengths:avg     9.8 min       8 max      16 sdev     3.3
writes:15(0 errs)
  write sizes (blks): avg     9.6 min       8 max      16 sdev     3.2
  write times (msec):avg 253.726 min  13.597 max 468.711 sdev 190.812
  write sequences: 15
  write seq. lengths:avg     9.6 min       8 max      16 sdev     3.2
seeks:24(92.3%)
  seek dist (blks):init     24,
avg  2099.8 min       8 max    8280 sdev  3202.3
time to next req(msec): avg 1159.166 min   0.004 max 26973.092 sdev 5183.380
throughput:3.5 KB/sec
utilization:0.02

VOLUME: /dev/hd2  description: /usr
reads:22(0 errs)
  read sizes (blks): avg     8.0 min       8 max       8 sdev     0.0
  read times (msec):avg  16.680 min   4.328 max  36.748 sdev   7.826
  read sequences: 21
  read seq. lengths:avg     8.4 min       8 max      16 sdev     1.7
writes:17(0 errs)
  write sizes (blks): avg     8.5 min       8 max      16 sdev     1.9
  write times (msec):avg 141.758 min  43.854 max 288.311 sdev  72.614
  write sequences: 17
  write seq. lengths:avg     8.5 min       8 max      16 sdev     1.9
seeks:38(97.4%)
  seek dist (blks):init 1158600,
avg 84743.1 min       8 max 1158240 sdev 216114.9
time to next req(msec): avg 856.714 min   0.004 max 29161.273 sdev 4618.851
throughput:4.8 KB/sec
utilization:0.02

VOLUME: /dev/hd8  description: jfslog
writes:6(0 errs)
  write sizes (blks): avg     8.0 min       8 max       8 sdev     0.0
  write times (msec):avg  24.956 min  18.328 max  31.755 sdev   4.736
  write sequences: 5
  write seq. lengths:avg     9.6 min       8 max      16 sdev     3.2
seeks:5(83.3%)
  seek dist (blks):init   2120,
avg    12.0 min       8 max      16 sdev     4.0
time to next req(msec): avg 4989.772 min  32.874 max 29395.709 sdev 10915.003
throughput:0.7 KB/sec
utilization:0.00

VOLUME: /dev/loglv01  description: jfslog
writes:1(0 errs)
  write sizes (blks): avg     8.0 min       8 max       8 sdev     0.0
  write times (msec):avg  26.221 min  26.221 max  26.221 sdev   0.000
  write sequences: 1
  write seq. lengths:avg     8.0 min       8 max       8 sdev     0.0
seeks:1(100.0%)
  seek dist (blks):init    792
time to next req(msec): avg 29583.572 min 29583.572 max 29583.572 sdev   0.000
throughput:0.1 KB/sec
utilization:0.00

VOLUME: /dev/loglv00  description: jfslog
writes:1(0 errs)
  write sizes (blks): avg     8.0 min       8 max       8 sdev     0.0
  write times (msec):avg   9.103 min   9.103 max   9.103 sdev   0.000
  write sequences: 1
  write seq. lengths:avg     8.0 min       8 max       8 sdev     0.0
seeks:1(100.0%)
  seek dist (blks):init     32
time to next req(msec): avg  69.104 min  69.104 max  69.104 sdev   0.000
```

```
throughput:0.1 KB/sec
utilization:0.00

------------------------------------------------------------------------
Detailed Physical Volume Stats   (512 byte blocks)
------------------------------------------------------------------------

VOLUME: /dev/hdisk2  description:
reads:3(0 errs)
  read sizes (blks): avg     8.0 min       8 max       8 sdev     0.0
  read times (msec):avg  11.693 min   3.257 max  17.883 sdev   6.179
  read sequences: 3
  read seq. lengths:avg     8.0 min       8 max       8 sdev     0.0
writes:2346(0 errs)
  write sizes (blks): avg    86.2 min       8 max     128 sdev    41.8
  write times (msec):avg  14.798 min   0.190 max 187.811 sdev  10.655
  write sequences: 55
  write seq. lengths:avg  3677.5 min       8 max   10528 sdev  3653.0
seeks:58(2.5%)
  seek dist (blks):init 1028472,
avg 103083.6 min       8 max  278624 sdev 86519.1
time to next req(msec): avg   5.259 min   0.063 max  87.148 sdev   4.855
throughput:3025.5 KB/sec
utilization:0.95

VOLUME: /dev/hdisk1  description:
reads:3982(0 errs)
  read sizes (blks): avg    50.8 min       8 max     112 sdev    19.4
  read times (msec):avg  11.136 min   0.023 max 164.865 sdev  12.216
  read sequences: 1184
  read seq. lengths:avg   170.9 min       8 max   11424 sdev   554.8
writes:7(0 errs)
  write sizes (blks): avg    10.3 min       8 max      16 sdev     3.6
  write times (msec):avg  15.665 min   0.692 max  36.422 sdev  13.568
  write sequences: 7
  write seq. lengths:avg    10.3 min       8 max      16 sdev     3.6
seeks:1191(29.9%)
  seek dist (blks):init 1094000,
avg  2272.3 min       8 max  577504 sdev 27157.6
time to next req(msec): avg  10.415 min   0.137 max  89.506 sdev   5.994
throughput:3027.9 KB/sec
utilization:0.82

VOLUME: /dev/hdisk0  description:
reads:33(0 errs)
  read sizes (blks): avg     8.0 min       8 max       8 sdev     0.0
  read times (msec):avg  16.124 min   4.137 max  31.980 sdev   7.568
  read sequences: 31
  read seq. lengths:avg     8.5 min       8 max      16 sdev     2.0
writes:36(0 errs)
  write sizes (blks): avg     9.3 min       8 max      24 sdev     3.5
  write times (msec):avg  36.717 min   1.945 max 447.324 sdev  72.864
  write sequences: 36
  write seq. lengths:avg     9.3 min       8 max      24 sdev     3.5
seeks:67(97.1%)
  seek dist (blks):init 1498824,
avg 145586.2 min       8 max 1417280 sdev 328426.9
time to next req(msec): avg 726.347 min   0.886 max 26794.065 sdev 3927.114
throughput:9.0 KB/sec
utilization:0.04
```

The output is composed of two different types of reports: global and detailed.

3.4.2 The Global Reports of filemon

The global reports list the most active files, segments, logical volumes, and physical volumes during the measured interval. They are shown at the beginning of the `filemon` report. By default, the logical file and virtual memory reports are limited to the 20 most active files and segments, respectively, as measured by the total amount of data transferred. If the **-v** flag has been

specified, activity for all files and segments is reported. All information in the reports is listed from top to bottom as most active to least active.

Most Active Files

#MBs Total number of MBs transferred over measured interval for this file. The rows are sorted by this field in decreasing order.
#opns Number of opens for files during measurement period.
#rds Number of read calls to file.
#wrs Number of write calls to file.
file File name (full path name is in detailed report).
volume:inode The logical volume that the file resides in and the inode number of the file in the associated file system. This field can be used to associate a file with its corresponding persistent segment shown in the detailed VM segment reports. This field may be blank for temporary files created and deleted during execution.

It is clear that the most active files are bigfile on logical volume test_lv and big.sav on logical volume templv. Both have 98.7 MB transferred during 33.431 seconds, have 25271 read calls, and 25270 write calls, respectively.

The application utilizes the terminfo database for screen management; so the ksh.cat and cmdtrace.cat are also busy. Any time the shell needs to post a message to the screen, it uses the catalogs for the source of the data.

To identify unknown files

Translate the logical volume name to the mount point of the file system, and use the `find` command:

```
# find /filesystem_name -inum <inode number> -print
```

Or use the `ncheck` command:

```
# ncheck -i <inode number> <volume name>
```

Most Active Segments

#MBs Total number of MBs transferred over measured interval for this segment. The rows are sorted by this field in decreasing order.
#rpgs Number of 4-KB pages read into segment from disk.
#wpgs Number of 4-KB pages written from segment to disk (page out).
#segid VMM ID of memory segment.
segtype Type of segment: working segment, persistent segment (local file), client segment (remote file), page table segment, system segment, or special persistent segments containing file system data (log, root directory, .inode, .inodemap, .inodex, .inodexmap, .indirect, .diskmap).
volume:inode For persistent segments, name of logical volume that contains the associated file and the file's inode number. This field can be used to associate a persistent segment with its corresponding file, shown in the detailed file stats reports. This field will be blank for non-persistent segments.

Note

If the `filemon` command is still active, the virtual memory analysis tool `svmon` can be used to display more information about a segment, given its segment ID (segid), as follows: `svmon -D <segid>`.

In our example, the segtype ??? means the system cannot identify the segment type, and you have to use `svmon` to get more information.

Most Active Logical Volumes

util	Utilization of logical volume
#rblk	Number of 512 byte blocks read from logical volume
#wblk	Number of 512 byte blocks written to logical volume
KB/s	Average transfer data rate in KB per second
volume	Logical volume name
description	Either the file system mount point or the LV type (paging, jfslog, boot, or sysdump). For example, the LV /dev/hd2 is /usr; /dev/hd6 is paging, and /dev/hd8 is jfslog. There may also be the word *compressed.* This means all data is compressed automatically using LZ compression before being written to disk, and all data is uncompressed automatically when read from disk.

The utilization is presented in percentage, 0.99 means 99 percent busy during measured interval. The logical volume templv has 202256 blocks written and has an average data transfer rate of 3025.4 KB/s. The logical volume test_lv has 202376 blocks read and has an average data transfer rate of 3027.8 KB/s. The other five LVs have little utilization and transfer rate.

Most Active Physical Volumes

util	Utilization of physical volume. Note: logical volume I/O requests start before and end after physical volume I/O requests. For that reason, total logical volume utilization will appear to be higher than total physical volume utilization.
#rblk	Number of 512-byte blocks read from physical volume.
#wblk	Number of 512-byte blocks written to physical volume.
KB/s	Average transfer data rate in KB per second.
volume	Physical volume name.
description	Simple description of the physical volume type, for example, CD-ROM SCSI, 2.0 GB SCSI disk.

Similar meaning for physical volume as for logical volume. We can see that hdisk2 and hdisk1 are very busy (95 percent and 82 percent), but hdisk0 has a utilization of only four percent. Therefore, spreading the file and logical volume activities over all the three hard disks probably results in better performance.

3.4.3 The Detailed Reports of filemon

The detailed reports give additional information for the global reports. There is one entry for each reported file, segment, or volume in the detailed reports. The fields in each entry are described below for the four detailed reports. Some of the fields report a single value; others report statistics that characterize a distribution of many values. For example, response-time statistics are kept for all read or write requests that were monitored. The average, minimum, and maximum response times are reported as well as the standard deviation of the

response times. The standard deviation is used to show how much the individual response times deviated from the average. Roughly two-thirds of the sampled response times are between average minus standard deviation and average plus standard deviation. If the distribution of response times is scattered over a large range, the standard deviation will be large compared to the average response time.

Detailed File Stats: Detailed file statistics are provided for each file listed in the *Most Active Files* report. These stanzas can be used to determine what access has been made to the file. In addition to the number of total bytes transferred, opens, reads, writes, and lseeks, the user can also determine the read/write size and times.

FILE	Name of the file. The full path name is given, if possible.
volume	Name of the logical volume/file system containing the file.
inode	Inode number for the file within its file system.
opens	Number of times the file was opened while monitored.
total bytes xfrd	Total number of bytes read/written from/to the file.
reads	Number of read calls against the file.
read sizes (bytes)	Read transfer-size statistics (avg/min/max/sdev), in bytes.
read times (msec)	Read response-time statistics (avg/min/max/sdev), in milliseconds.
writes	Number of write calls against the file.
write sizes (bytes)	Write transfer-size statistics.
write times (msec)	Write response-time statistics.
lseeks	Number of lseek subroutine calls.

The file bigfile on LV test_lv has an inode number of 18; it was opened once, and has been read 25271 times with no error. The average read size is 4096.0 bytes, 4096 bytes in minimum, 4096 bytes in maximum, and 0.0 bytes is the standard deviation (SDEV). The average read time is 0.921 milliseconds; 0.020 milliseconds was the fastest read; 85.137 milliseconds was the slowest read, and 2.578 milliseconds was the statistical standard deviation.

Note

The read sizes and write sizes will give you an idea of how efficiently your application is reading and/or writing information. Using a multiple of 4 KB pages is best.

Detailed VM Segment Stats: Each element listed in the *Most Active Segments* report will have a corresponding stanza that shows detailed information about real I/O to and from memory.

SEGMENT	Internal AIX segment ID.
segtype	Type of segment contents.
segment flags	Various segment attributes.
volume	For persistent segments, the name of the logical volume containing the corresponding file.
inode	For persistent segments, the inode number for the corresponding file.
reads	Number of 4096-byte pages read into the segment (that is, paged in).
read times (msec)	Read response-time statistics (avg/min/max/sdev), in milliseconds.

read sequences	Number of read sequences. A sequence is a string of pages that are read (paged in) consecutively. The number of read sequences is an indicator of the amount of sequential access.
read seq. lengths	Statistics describing the lengths of the read sequences, in pages.
writes	Number of pages written from the segment to disk (that is, paged out).
write times (msec)	Write response-time statistics.
write sequences	Number of write sequences. A sequence is a string of pages that are written (paged out) consecutively.
write seq. lengths	Statistics describing the lengths of the write sequences, in pages.

By examining the reads and read-sequence counts, you can determine if the access is sequential or random. For example, if the read-sequence count approaches the reads count, the file access is more random. On the other side, if the read-sequence count is significantly smaller than the read count and the read-sequence length is a high value, the file access is more sequential. The same logic applies for the writes and write sequence. In the example, segment IDs 0241, 196d and 19cd have same write as write-sequence counts; so these corresponding files accesses are more random.

Detailed Logical/Physical Volume Stats: Each element listed in the *Most Active Logical/Physical Volumes* reports will have a corresponding stanza that shows detailed information about the logical/physical volume. In addition to the number of reads and writes, the user can also determine read and write times and sizes and the initial and average seek distances for the logical / physical volume.

VOLUME	Name of the volume.
description	Description of the volume. (Describes contents, if dealing with a logical volume; describes type, if dealing with a physical volume.)
reads	Number of read requests made against the volume.
read sizes (blks)	Read transfer-size statistics (avg/min/max/sdev), in units of 512-byte blocks.
read times (msec)	Read response-time statistics (avg/min/max/sdev), in milliseconds.
read sequences	Number of read sequences. A sequence is a string of 512-byte blocks that are read consecutively. It indicates the amount of sequential access.
read seq. lengths	Statistics describing the lengths of the read sequences, in blocks.
writes	Number of write requests made against the volume.
write sizes (blks)	Write transfer-size statistics.
write times (msec)	Write-response time statistics.
write sequences	Number of write sequences. A sequence is a string of 512-byte blocks that are written consecutively.
write seq. lengths	Statistics describing the lengths of the write sequences, in blocks.
seeks	Number of seeks that preceded a read or write request; also expressed as a percentage of the total reads and writes that required seeks.

seek dist (blks)	Seek-distance statistics in units of 512-byte blocks. In addition to the usual statistics (avg/min/max/sdev), the distance of the initial seek operation (assuming block 0 was the starting position) is reported separately. This seek distance is sometimes very large; so it is reported separately to avoid skewing the other statistics.
seek dist (cyls)	(Physical volume only) Seek-distance statistics in units of disk cylinders.
time to next req	Statistics (avg/min/max/sdev) describing the length of time, in milliseconds, between consecutive read or write requests to the volume. This column indicates the rate at which the volume is being accessed.
throughput	Total volume throughput in KB per second.
utilization	Fraction of time the volume was busy. The entries in this report are sorted by this field in decreasing order.

A long seek time may increase I/O response time and result in decreased application performance. In the example, the logical volume templv has 6548 writes, but there are only 54 write sequences and 56 seeks. This means that the logical volume is sequential; the utilization is 99 percent, and average throughput is 3025.4 KB/s during the measured interval. By examining the reads and read sequence counts, you can determine if the access is sequential or random. The same logic applies to the writes and write sequence. In the example, hdisk2 and hdisk1 have 2.5 (58/2349) percentage seeks and 29.9 (1191/3989) percentage seeks compared to writes and reads, respectively; so the files are more sequential, and there is a higher throughput (3025.5 KB/s and 3027.9 KB/s). The physical volume utilization is 95 percent and 82 percent, and hdisk0 has 97.1 (67/69) percentage seeks compared to reads and writes. Therefore, the file is random, and there is a lower throughput (9.0 KB/s). The physical volume utilization is four percent.

Attention

Although `filemon` reports average, minimum, maximum, and standard deviation in its detailed-statistics sections, the results should not be used to develop confidence intervals or other formal statistical inferences. In general, the distribution of data points is neither random nor symmetrical.

3.4.4 Comparing filemon and vmstat Outputs

During the `filemon` execution, we also ran the `vmstat` command shown below. From the example above, we know the CPU utilization is 25.8 percent; the two disks utilizations are 95 percent and 82 percent, and the throughput is about the maximum data rate (3027.8 KB/s), almost 70 percent of SCSI-1's 5 MB/s. Since there are no paging-space page-ins and page-outs, the workload is not a memory bottleneck. The sum of `us` and `sy` (user and system) CPU-utilization is not higher than 27 percent; so the workload is not approaching the CPU limits of the system during that interval. But the `wa` (I/O wait) percentage is 61 - 90 percent; a significant amount of time is being spent waiting on non-overlapped file I/O, and the workload is I/O-bound.

```
kthr     memory             page              faults        cpu      disk xfer
----- ----------- ------------------------ ------------ ----------- ----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in    sy  cs us sy id wa  1  2  3  4
 0  0  5114 23170   0   0   0   0    6   0 117     2   7  0  0 98  1  0  0  0
 0  3  5228 22940   0  11   0   0    0   0 516  5910 150  2 14 45 38 37  2  0
 0  3  5255 21764   0   0   0   0    0   0 663  1291 269  1 14  2 83  3 133 92
 0  3  5257 20521   0   0   0   0    0   0 672  1402 292  2 16  0 82  0 137 90
 1  2  5260 19244   0   0   0   0    0   0 677  1406 282  2 14  0 84  0 136 91
 0  3  5262 17940   0   0   0   0    0   0 658  1451 273  2 18  0 80  0 131 84
 0  3  5265 16753   0   0   0   0    0   0 661  1342 273  2 16  0 82  0 130 92
 0  3  5267 15647   0   0   0   0    0   0 669  1232 279  2 12  0 86  0 129 102
 2  2  5269 14205   0   0   0   0    0   0 658  1649 252  3 16  0 81  0 122 82
 1  2  5271 12779   0   0   0   0    0   0 633  1575 239  2 17  0 81  0 118 69
 1  2  5273 11735   0   0   0   0    0   0 673  1182 267  1 12  0 87  0 130 101
 0  3  5275 10291   0   0   0   0    0   0 647  1574 232  1 16  0 82  0 118 74
 1  2  5278  9040   0   0   0   0    0   0 667  1377 274  2 14  0 84  0 129 89
 2  2  5280  7404   0   0   0   0    0   0 628  1792 237  2 19  0 79  0 120 63
 1  3  5282  5763   0   0   0   0    0   0 626  1787 236  2 16  0 82  0 112 62
 1  2  5285  4035   0   0   0   0    0   0 628  1846 224  3 19  0 78  0 112 59
 1  2  5287  2439   0   0   0   0    0   0 631  1766 216  1 17  0 81  0 120 67
 1  2  5289   749   0   0   0   0    0   0 633  1845 217  3 16  0 81  0 117 60
 2  1  5291   125   0   0   0 890 2131   0 632  1645 339  2 18  0 80  0 115 70
 1  2  5295   133   0   0   0 1652 3366  0 620  1796 437  1 22  0 77  0 117 61
 2  2  5299   127   0   0   0 1690 4448  0 627  1841 445  2 22  0 76  0 121 62
 1  2  5302   127   0   0   0 1618 3781  0 626  1759 437  3 22  0 76  0 121 64
 1  2  5305   134   0   0   0 1672 4557  0 624  1792 420  2 22  0 76  0 121 64
 1  2  5309   134   0   0   0 1735 5498  0 610  1873 435  3 24  0 73  0 116 57
 1  2  5312   134   0   0   0 1700 5956  1 617  1825 427  3 24  0 73  0 115 62
 0  3  5316   124   0   0   0 1761 3972  0 613  1943 434  3 22  0 75  0 111 57
 1  3  5319   125   0   0   0 1652 2277  0 605  1777 406  1 24  0 75  0 116 58
 1  2  5323   127   0   0   0 1705 2296  0 619  1833 422  2 22  0 76  0 111 60
 1  3  5326   127   0   0   0 1717 2770  0 616  1873 441  2 25  0 73  1 112 57
 0  3  5329   133   0   0   0 1672 2406  0 626  1804 427  2 25  0 72  0 116 59
 0  3  5333   132   0   0   0 1636 2329  0 615  1792 419  3 24  0 73  0 114 64
 3  3  5337   128   0   0   0 1645 2464  0 642  1753 486  2 23  0 75 32 112 62
 0  3  5341   128   0   0   0 1491 2326  0 630  1634 417  2 21  0 76 28 113 68
 1  2  5344   127   0   0   0 1690 3322  0 634  1833 442  1 23  0 76  0 122 66
 1  2  5348   128   0   0   0 1653 3114  0 629  1792 450  2 21  0 77  0 119 68
 0  3  5325   120   0   0   0 432  656   0 570  4750 268  2 13 24 61 62 23 13
 0  3  5296   162   0   0   0 282  430   0 583   795 279  3  6 17 73 108  0  0
 2  2  5303   161   0   0   0   9   15   0 480  2862 109  9 16 68  7  2  1  1
 1  2  5146   329   0   0   0   0    0   0 483  2425  88  7 13 71  8  2  2  1
```

3.4.5 Things to Keep in Mind

- The /etc/inittab file is always very active. Daemons specified in /etc/inittab are checked regularly to determine whether they are required to be respawned.
- The /etc/passwd file is also always very active. Because files and directories access permissions are checked.
- A long seek time increases I/O response time and decreases performance.
- If the majority of the reads and writes require seeks, you may have fragmented files and/or overly active file systems on the same physical disk.
- If the number of reads and writes approaches the number of sequences, physical disk access is more random than sequential. Sequences are strings of pages that are read (paged in) or written (paged out) consecutively. The `seq. lengths` is the length, in pages, of the sequences.

A random file access can also involve many seeks. In this case, you cannot distinguish from the `filemon` output if the file access is random or if the file is fragmented. You have to further investigate with the `fileplace` command.

Note

Remote files show up in the `volume:inode` column with the remote system name.

Because `filemon` can potentially consume some CPU power, use this tool with discretion, and analyze the system performance while taking into consideration the overhead involved in running the tool.

- In a CPU-saturated environment with little I/O, `filemon` slowed a large compile by about one percent.
- In a CPU-saturated environment with a high disk-output rate, `filemon` slowed the writing program by about five percent.

How to Solve Disk-Limited Programs: Disk sensitivity can come in a number of forms, with different resolutions:

- If large, I/O-intensive background jobs are interfering with interactive response time, you may want to activate I/O pacing.
- If it appears that a small number of files are being read over and over again, you should consider whether additional real memory would allow those files to be buffered more effectively.
- If `iostat` indicates that your workload I/O activity is not evenly distributed among the system disk drives, and the utilization of one or more disk drives is often 40-50 percent or more, consider reorganizing file systems.
- If the workload′s access pattern is predominantly random, you may want to consider adding disks and distributing the randomly accessed files across more drives.
- If the workload′s access pattern is predominantly sequential and involves multiple disk drives, you may want to consider adding one or more disk adapters. It may also be appropriate to consider building a striped logical volume to accommodate large, performance-critical sequential files.

3.5 The fileplace Command

Since files can be very dynamic, file system performance tends to follow the theory of progressive chaos. A system may start out perfect, but through time, it progresses to a state of total chaos. Access to fragmented files may result in a large number of seeks and longer I/O response time. At some point, the system administrator may choose to reorganize the placement of files within logical volumes, and the placement of logical volumes within physical volumes, to reduce fragmentation and to more evenly distribute the total I/O load.

There are some factors that affect file system performance:

- Dynamic allocation of resources may cause:
  - Logically contiguous files to be fragmented
  - Logically contiguous LVs to be fragmented
  - File blocks to be scattered
- Effects when files are accessed from disk:

- Sequential access no longer sequential.
- Random access affected.
- Access time dominated by longer seek time.
- Once the file is in memory, these effects diminish.

3.5.1 Using fileplace

The fileplace command displays the placement of a file's blocks within a logical volume or within one or more physical volumes. The fileplace command expects an argument containing the name of the file to examine.

Attention

Most variations of this command use fewer than 0.3 seconds of CPU time.

By default, the fileplace command sends its output to the display, but the output can be redirected to a file via normal shell redirection. Its usage is:

```
fileplace [-l] [-p] [-i] [-v] <filename>
```

-l: display logical blocks

-p: display physical blocks

-i: display indirect blocks

-v: verbose mode (show efficiency and sequentiality)

Here is a short example:

```
# fileplace -pvi /unix

File: /unix  Size: 2326373 bytes  Vol: /dev/hd2
Blk Size: 4096  Frag Size: 4096  Nfrags:  562   Compress: no
Inode: 47570  Mode: -r-xr-xr-x  Owner: root  Group: system

INDIRECT BLOCK: 147794

  Physical Addresses (mirror copy 1)                       Logical Fragment
  ----------------------------------                       ----------------
  0147788-0147793  hdisk0    6 frags    24576 Bytes,  1.1%  0047916-0047921
  unallocated                4 frags    16384 Bytes,  0.0%  unallocated
  0147795-0147930  hdisk0  136 frags   557056 Bytes, 24.2%  0047923-0048058
  unallocated                1 frags     4096 Bytes,  0.0%  unallocated
  0147931-0148131  hdisk0  201 frags   823296 Bytes, 35.8%  0048059-0048259
  unallocated                1 frags     4096 Bytes,  0.0%  unallocated
  0148132-0148350  hdisk0  219 frags   897024 Bytes, 39.0%  0048260-0048478

  562 frags over space of 563 frags:  space efficiency = 99.8%
  4 fragments out of 562 possible:  sequentiality = 99.5%
```

A physical block is a contiguous allocated disk space, and the block size is equal the size of the logical blocks used for the logical division of files and directories. In this example, the Blk Size is 4096 bytes. The Frag Size is a 4096-byte block. File system fragmentation sizes are used for better space utilization by subdividing 4-KB blocks. In AIX V4, it is now possible to create a JFS with an allocation unit or fragment of 512-KB, 1-KB, 2-KB, or 4-KB blocks. Although there is an advantage for space utilization, fragmentation can impact performance due to free space fragmentation. With a smaller fragment size, there may also be an increase in disk I/O operations. The Nfrags is the number

of fragments this file has. The Compress is no, meaning this file is not compressed. If a file system is compressed, all data is compressed automatically using LZ compression before being written to disk, and all data is uncompressed automatically when read from disk. In addition to increased disk I/O activity and free space fragmentation problems, there are the following performance considerations:

- Degradation in file system usability arising as a direct result of the data compression/decompression activity.
- All logical blocks in a compressed file system, when modified for the first time, will be allocated 4096 bytes of disk space, and this space will subsequently be reallocated when the logical block is written to disk.
- In order to perform data compression, approximately 50 CPU cycles per byte are required, and about 10 CPU cycles per byte are required for decompression.

The column under Physical Addresses shows the physical block numbers and physical volume where part of the file resides. The column under Logical Fragment shows the logical block numbers where part of the file resides. The columns in the middle show the number of fragments that are contiguous, the number of bytes in these contiguous fragments, and the percentage of the block range compared to the total size.

The report generated by the -piv options displays the indirect block 147794, which is in use due to the file being larger than 32 KB.

Attention

The fileplace command will not display NFS remote files. If a remote file is specified, the fileplace command returns an error message.

The fileplace command reads the file's list of blocks directly from the logical volume on disk. If the file is newly created, extended, or truncated, the information may not be on disk yet. Use the sync command to flush the information to the logical volume.

3.5.2 Space Efficiency and Sequentiality

Higher space efficiency means files are less fragmented and will probably provide better sequential file access; a higher sequentiality indicates that the files are more contiguously allocated, and this will probably be better for sequential file access.

```
Space efficiency = Total number of fragments used for file storage /
                   (Largest fragment physical address -
                   Smallest fragment physical address + 1).

Sequentiality = (Total number of fragments -
                Number of grouped fragments + 1) /
                Total number of fragments.
```

If you find that your sequentiality or space efficiency values become low, you may want to use the reorgvg command to improve disk utilization and efficiency.

In this example, the Largest fragment physical address - Smallest fragment physical address + 1 is: 148350 - 147788 + 1 = 563 fragments; total used fragments is: 6 + 136 + 201 + 219 = 562; the space efficiency is 562 / 563 (99.8 percent); the sequentiality is (562 - 4 + 1) / 562 = 99.5 percent.

As the total number of fragments used for file storage does not include the indirect blocks location, but the physical address does, the space efficiency can never be 100 percent for files larger than 32 KB, even if the file is located on contiguous fragments. The AIX file system structure is illustrated on Figure 2 on page 98.

3.5.3 AIX File System Organization

If a file is too large to be contained in just eight data blocks (> 32 KB), a 4-KB block is used to store more pointers. This 4-KB block is called an indirect block. It can point to up to 1024 data blocks. This is called single indirection. Once a file grows to the size that it must be referenced via an indirect block, the array of eight addresses within the inode are no longer used. If the file shrinks down to a size that no longer requires indirection, the eight inode pointers are once again used.

If a file is so large that there are not enough pointers in a single indirect block to point to all of the data blocks of that file, double indirection is invoked. In double indirection, a 4-KB block is used to point to 512 indirect blocks which, in turn, each point to 1024 data blocks. This means that in a default JFS, a file can theoretically grow to a size of 512 * 1024 * 4096 bytes (2 GB). In AIX V4, the maximum file system size is 256 GB. AIX V 4.2 added support for files greater than 2 GB. If a JFS is created with the option *bf=true,* indicating that the file system supports files greater than 2 GB, then the size of the data blocks are 128 KB rather than 4 KB. Theoretically, the maximum file size in AIX 4.2 is now 64 GB. The first indirect block will still point to 1024 4-KB blocks; however, the other 511 indirect blocks point to 1024 128 KB blocks. Therefore, the real maximum file size is just under 64 GB.

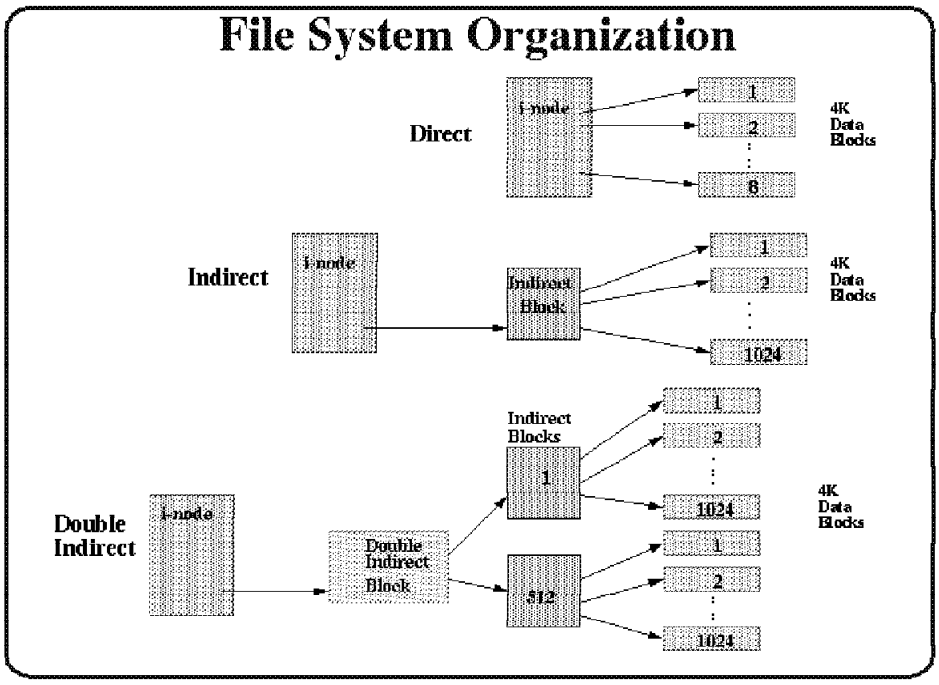


Figure 2. AIX File System Organization

When you have a file system performance issue, try to answer the following questions, and use the proper tool to identify it.

- Look for most active file systems and logical volumes:
  - Can “hot” file systems be better located on physical drive or be spread across multiple physical drives? (`lslv`)
  - Are “hot” files local or remote? (`filemon`)
  - Does paging space dominate disk utilization? (`filemon`)
  - Is there enough memory to cache the file pages being used by running processes? (`svmon`)
  - Does the application perform a lot of synchronous (non-cached) file I/O?
- Look for file fragmentation:
  - Are “hot” files heavily fragmented? (`fileplace`)
- Look for heavy physical volume utilization:
  - Is the “type of drive” (SCSI-1, SCSI-2, tape, and so on) or SCSI adapter causing a bottleneck? (`filemon`).

3.6 The lslv Command

The `lslv` command shows, among other information, the logical volume fragmentation. Before we start to discuss it, let’s look at the logical volume concept. Figure 3 on page 99 shows the logical volume manager (LVM) policies.

There are various factors that affect a logical volume performance:

- Position on PV (Intra-policy)
- Range of PVs (Inter-policy)
- Maximum number of PVs to use
- Number of copies of each LP (Mirroring)

- Mirror Write consistency (for mirrors)
- Allocate each LP copy on separate PV (Strictness)
- Relocate LV during reorganization (reorgvg)
- Scheduling policy (parallel or sequential writes)
- Write verify (read after a write)
- Striping

Note

This command uses mainly CPU time. As an example, the command:

```
#lslv -p hdisk0 hd1
```

consumes about 0.5 seconds of CPU time.

3.6.1 LVM Policies

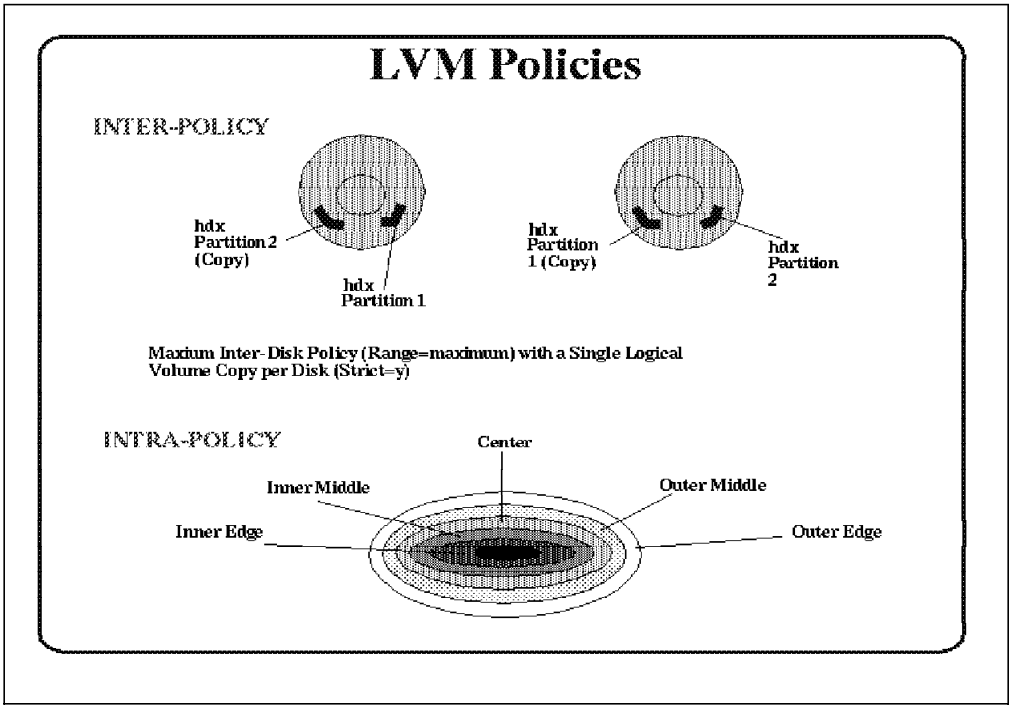


Figure 3. AIX LVM Policies

To view policies, use the `lslv` command or `smitty lslv`.

```
$ lslv hd2
LOGICAL VOLUME:     hd2                    VOLUME GROUP:   rootvg
LV IDENTIFIER:      00012729dd72205e.5     PERMISSION:     read/write
VG STATE:           active/complete        LV STATE:       opened/syncd
TYPE:               jfs                    WRITE VERIFY:   off
MAX LPs:            512                    PP SIZE:        4 megabyte(s)
COPIES:             1                      SCHED POLICY:   parallel
LPs:                114                    PPs:            114
STALE PPs:          0                      BB POLICY:      relocatable
INTER-POLICY:       minimum                RELOCATABLE:    yes
INTRA-POLICY:       center                 UPPER BOUND:    32
MOUNT POINT:        /usr                   LABEL:          /usr
MIRROR WRITE CONSISTENCY: on
EACH LP COPY ON A SEPARATE PV ?: yes
```

Intra-physical Volume Allocation Policy: The intra-physical volume allocation policy specifies what strategy should be used for choosing physical partitions on a physical volume. The five general strategies are *outer edge*, *inner edge*, *outer middle*, *inner middle*, and *center*.

- The outer edge and inner edge strategies have the slowest average seek times.
- The outer middle and inner middle strategies allocate reasonably good locations for partitions with reasonably good average seek times.
- The center strategy has the fastest average seek times.

Inter-Physical Volume Allocation Policy: The inter-physical volume allocation policy specifies which strategy should be used for choosing physical devices to allocate the physical partitions of a logical volume. The choices are the *MINIMUM* and *MAXIMUM* options.

- The *MINIMUM* option indicates that as little as possible physical volumes should be used to allocate the required physical partitions. This option provides the greatest reliability, without having copies, for a logical volume. There are two choices available when using the *MINIMUM* option: without copies and with copies.
  - Without copies: The *MINIMUM* option indicates one physical volume should contain all the physical partitions of this logical volume. If the allocation program must use two or more physical volumes, it uses the minimum number possible, remaining consistent with the other parameters.
  - With copies: The *MINIMUM* option indicates that as many physical volumes as there are copies should be used. Otherwise, the minimum number of physical volumes possible are used to hold all the physical partitions.
- The *MAXIMUM* option intends, considering other constraints, to spread the physical partitions of this logical volume over as many physical volumes as possible. This is a performance-oriented option and should be used with copies to improve availability. If an un-copied logical volume is spread across multiple physical volumes, the loss of any physical volume containing a physical partition from that logical volume is enough to cause the logical volume to be incomplete. Compared to the parallel write copies option of the scheduling policies, this parameter is more important for obtaining better performance. To read and write from/to different physical volumes probably needs shorter time than from/to the same physical volume. Figure 4 on page 101 shows these concepts.

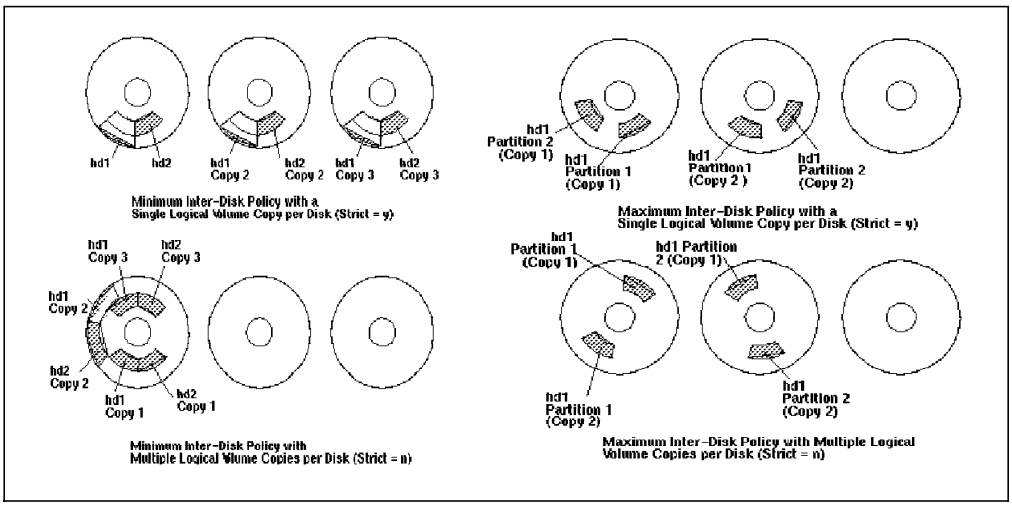


Figure 4. The Inter-Disk Policy

Scheduling Policies: Two different types of scheduling policies can be used for logical volumes with multiple copies:

- Sequential Write Copies: This policy performs copied write procedures in order: primary, secondary, tertiary. This policy waits for the write operation to complete for the previous physical partition before starting the write operation to the next one.
- Parallel Write Copies: This policy starts the write operation for all the physical partitions of a logical partition at the same time. When the write operation to the physical partition that takes the longest to complete finishes, the write operation returns.

Note

Specifying logical volume copies may increase I/O read operation performance. The multiple copies allow the system to direct the read operation to the copy that can be most quickly accessed.

Mirror Write Consistency: The LVM always ensures data consistency among mirrored copies of a logical volume during normal I/O processing. For every write to a logical volume, the LVM generates a write request for every mirror copy. A problem arises if the system crashes in the middle of processing a mirrored write (before all copies are written). If mirror write consistency recovery is requested for a logical volume, the LVM keeps additional information to allow recovery of these inconsistent mirrors. Mirror write consistency recovery should be performed for most mirrored logical volumes. Logical volumes, such as the page space, that do not use the existing data when the volume group is re-varied on do not need this protection.

Other Attributes

WRITE VERIFY Specifies whether to verify all writes to the logical volume with a follow-up read. This option enhances availability, but decreases performance.

LV IDENTIFIER Consists of two parts. The first part (00012729dd72205e) indicates the volume group identifier that this logical volume belongs to, and the second part (5) is the sequential number of the logical volume created on this volume group.

BB POLICY Specifies whether to use Bad Block Relocation. Bad Block Relocation redirects read/write requests from a disk block that can no longer retain data to one that can. The process is transparent; the application does not receive notice that requests directed to a physical block are actually serviced by a different block.

RELOCATABLE Specifies whether to allow the relocation of the logical volume during volume group reorganization.

UPPER BOUND Specifies the maximum number of physical volumes for allocation.

In the example above, the logical volume hd2 has *center* for INTRA-POLICY, *minimum* for INTER-POLICY, *on* for MIRROR WRITE CONSISTENCY, *parallel* for SCHED POLICY, and *off* for WRITE VERIFY. It has one copy; therefore, the *yes* for EACH LP COPY ON A SEPARATE PV has no meaning. Also MIRROR WRITE CONSISTENCY and SCHED POLICY have no meaning with *copies*=1. The /usr file system is frequently used; these policies have the fastest seek time and achieve higher performance.

Striping: There is no item relevant to striping in `lslv` output, but you can specify striping when creating a logical volume. Striping is designed to increase the read/write performance of frequently accessed, large sequential files. However, there are some constraints imposed by implementing striping. Primarily, when a logical volume is created as striped, it must use at least two physical volumes, and mirroring is not possible. Thus, availability is less than optimal, but performance will increase. Stripe size can be any power of 2 from 4 KB to 128 KB, but it is often set to 64 KB to get the highest levels of sequential I/O throughput. The number of logical partitions in a striped logical volume must be a multiple of the number of disk drives used. The logical volume cannot be mirrored; therefore, copies=1.

3.6.2 Logical Volume Fragmentation

To check fragmentation, use the command `lslv -l lvname`.

```
$ lslv -l hd2
hd2:/usr
PV               COPIES        IN BAND       DISTRIBUTION
hdisk0           114:000:000   22%           000:042:026:000:046
```

The output of COPIES shows the logical volume hd2 has only one copy. The IN BAND shows how well the intra-policy is followed. The higher the percentage, the better the allocation efficiency. Each logical volume has its own intra-policy. If the operating system cannot meet this requirement, it chooses a best way to meet the requirements. There are a total of 114 logical partitions (LP); 42 LPs are located on outer middle, 26 LPs on center, and 46 LPs on inner edge. Since the logical volume intra-policy is center, the in-band is 22 percent (26 / (42+26+46). The DISTRIBUTION shows how the physical partitions are placed in each part of the intra-policy. That is:

outer edge : outer middle : center : inner middle : inner edge

To see the placement on the physical volume, use the command:

```
$ lslv -p hdisk0 hd2
hdisk0:hd2:/usr
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE      1-10
FREE   FREE   FREE   FREE   FREE   USED   FREE   FREE   FREE   FREE     11-20
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE     21-30
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE     31-40
FREE   FREE   FREE   FREE   FREE   FREE   FREE   USED   USED   USED     41-50
USED   USED   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE     51-60
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE     61-70
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE     71-80
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE     81-90
FREE   FREE   STALE  STALE  FREE   FREE                               91-96

FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE     97-106
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE    107-116
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE    117-126
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE    127-136
FREE   FREE   FREE   FREE   FREE   FREE   FREE   0047   0048   0049    137-146
0050   0051   0052   0053   0054   0055   0056   0057   0058   0059    147-156
0060   0061   0062   0063   0064   0065   0066   0067   0068   0069    157-166
0070   0071   0072   0073   0074   0075   0076   0077   0078   0079    167-176
0080   0081   0082   0083   0084   USED   USED   0085   0086   0087    177-186
USED   USED   USED   USED   USED   0088                               187-192

0089   0090   0091   0092   0093   0094   0095   0096   0097   USED    193-202
USED   USED   USED   USED   USED   USED   USED   USED   USED   USED    203-212
USED   USED   USED   USED   USED   USED   USED   USED   USED   USED    213-222
USED   USED   USED   USED   USED   USED   USED   USED   USED   USED    223-232
USED   USED   USED   USED   USED   USED   USED   USED   USED   USED    233-242
USED   USED   USED   USED   USED   USED   USED   USED   USED   USED    243-252
USED   USED   0098   0099   0100   0101   USED   USED   USED   USED    253-262
USED   0102   0103   0104   0105   0106   0107   0108   0109   0110    263-272
0111   0112   0113   0114   USED   USED   USED   USED   USED   USED    273-282
USED   USED   USED   USED   USED                                      283-287

USED   USED   USED   USED   USED   USED   USED   USED   USED   USED    288-297
USED   USED   USED   USED   USED   USED   USED   USED   USED   USED    298-307
USED   USED   USED   USED   USED   USED   USED   USED   USED   USED    308-317
USED   USED   USED   USED   USED   USED   USED   USED   USED   USED    318-327
USED   USED   USED   USED   USED   USED   USED   USED   FREE   FREE    328-337
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE    338-347
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE    348-357
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE    358-367
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE    368-377
FREE   FREE   FREE   FREE   FREE   FREE                               378-383

FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE    384-393
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE    394-403
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE    404-413
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE    414-423
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   0001   0002    424-433
0003   0004   0005   USED   0006   0007   0008   0009   0010   0011    434-443
0012   0013   0014   0015   USED   0016   0017   0018   0019   0020    444-453
0021   0022   0023   0024   0025   0026   0027   0028   0029   0030    454-463
0031   0032   0033   0034   0035   0036   0037   0038   0039   0040    464-473
0041   0042   0043   0044   0045   0046                               474-479
```

From top to bottom, there are five blocks, they represent outer edge, outer middle, center, inner middle, and inner edge, respectively. A USED indicates that

the physical partition at this location is used by a logical volume other than the one specified. A number indicates the logical partition number of the logical volume specified with the `lslv -p` command. A FREE indicates that this physical partition is not used by any logical volume. Logical volume fragmentation occurs if logical partitions are not contiguous across the disk. A STALE physical partition is a physical partition that contains data you cannot use. You can also see the STALE physical partitions with the `lspv -m` command. Physical partitions marked as STALE must be updated to contain the same information as valid physical partitions. This process, called resynchronization, can be done at vary-on time, or can be started any time the system is running. Until the STALE partitions have been rewritten with valid data, they are not used to satisfy read requests, nor are they written to on write requests.

This example shows that the logical volume hd2 (mounted on /usr) is fragmented into eight segments. There are 42 LPs placed on outer middle, 26 LPs on center, and 46 LPs locate on inner edge. The same results as the `lslv -l` command above. There are two STALE PPs on the physical partitions 93 and 94. There is no free physical partition on center of this physical volume. If you want to improve the performance of hd2, you can use the `reorgvg` command after you change the policies for hd2.

3.6.3 Relationship Between Policies and Performance

By using `lslv`, `fileplace`, `filemon`, and `iostat`, you can find I/O, volume group, and logical volume problems. There are some ways to prevent these problems from happening by better utilizing the physical and logical volume structure.

- Logical volume reorganization for highest performance:
  - Allocate hot LVs to different PVs to reduce disk contention.
  - Spread hot LV across multiple PVs so that parallel access is possible.
  - Place hottest LVs in center of PVs, moderate LVs in the middle of PVs, and place coldest LVs on edges of PVs so that the hottest logical volumes have the fastest access time.
  - Do not use mirroring if possible since mirroring performs multiple writes that will affect performance in writing, and it only provides marginal improvement in reading. However, if mirroring is needed, set scheduling policy to `parallel` and allocation policy to `strict`. Parallel scheduling policy will enable reading from closest disk, and strict allocation policy allocates each copy on separate PVs.
  - Make LV contiguous to reduce access time.
  - Set inter-policy to maximum. This will spread each logical volume across as many physical volumes as possible, allowing reads and writes to be shared among several physical volumes.
  - Place frequently used logical volumes close together to reduce seek time.
  - Set write verify to `no` so that there is no follow-up read (similar to parity check) performed following a write.
- Implement logical volume striping:
  - Spread the logical volume across as many physical volumes as possible.
  - Use as many adapters as possible for the physical volumes.
  - Create a separate volume group for striped logical volumes.
  - Striping Recommendations:
    - The stripe unit size should be equal to the `max_coalesce`, which is by default 64 KB. The `max_coalesce` value is the largest request size (in terms of data transmitted) that the SCSI device driver will build.

- Use a minpghead value of 2: This will give the number of minimum pages with which sequential read-ahead starts.
- Using a maxpgahead of 16 times the number of disk drives causes the maximum pages to be read ahead to be done in units of the stripe size (64 KB) times the number of disk drives, resulting in the reading of one stripe unit from each disk for each read ahead. If possible, modify applications that use striped logical volumes to perform I/O in units of 64 KB. Figure 5 depicts a striped logical volume.

– Limitations of striping
  - Mirroring is not possible. Disk striping can increase performance, but does not allow mirroring.
  - Disk striping is mostly effective for sequential disk I/Os. With randomly accessed files, it is not as effective.
  - More than one disk should be used. Disk striping only takes effect when using more than one disk.

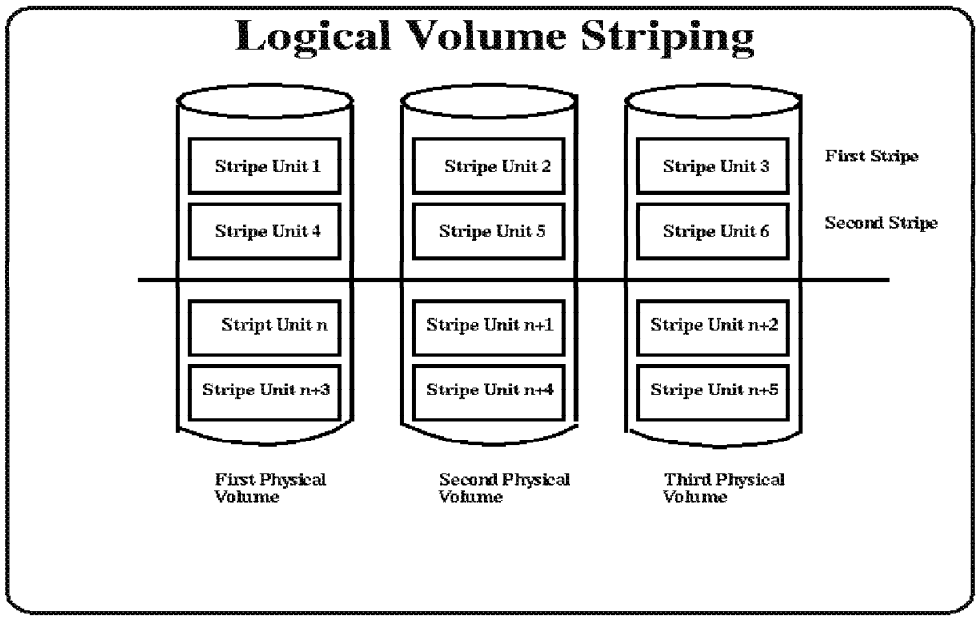


Figure 5. Logical Volume Striping

- Create an additional log logical volume to separate the log of the most active file system from the default log. This will increase parallel resource usage. Try to create a log logical volume for a hot file system on a fast drive. Follow these steps:
  – Backup the file system.
  – Create the log LV: `mklv -t jfslog -y LVname VGname 1 PVname` or `smitty mklv`.
  – Format the log: `/usr/sbin/logform /dev/LVname`.
  – Modify the affected file system and logical volume control block (LVCB): `chfs -a log=/dev/LVname /filesystemname`
  – Unmount and then mount the affected file system.
- Reorganize or add paging space:
  – The general recommendation is that the sum of the size of the paging spaces should be equal to at least twice the size of the real memory of the machine, up to a memory size of 256 MB (512 MB paging space). For memory larger than 256 MB, we recommend:

Total paging space = 512 MB + (memory size - 256 MB) * 1.25

– As paging spaces are being accessed in round-robin fashion, creating unequal paging space sizes on different physical volumes will lead to an unequal use of these paging spaces when one or more are full and therefore eliminate the advantage of having more than one paging space.

3.7 The netpmon Command

Understanding network performance is difficult. An easy way to tell if the network is affecting overall performance is to compare those operations that involve the network with those that do not. If you are running a program that does a considerable amount of remote reads and writes and it is running slowly, but everything else seems to be running normally, then it is probably a network problem. Some of the potential network bottlenecks may be caused by:

- Client-network interface
- Network bandwidth
- Server network interface
- Server CPU load
- Server memory usage
- Server bandwidth
- Inefficient configuration

There are several tools that measure network statistics and give a variety of information. For the standard tools, `netstat` and `nfsstat`, refer to Chapter 2, “Standard (UNIX) Performance Tools” on page 7. The `netpmon` command is an AIX performance tool used to measure network device-driver I/O, CPU usage, Internet socket calls, and NFS I/O. To improve performance, one can use the `no` (network options), `nfso`, `chdev`, and `ifconfig` commands.

3.7.1 The netpmon Implementation and Functions

The `netpmon` command uses the trace facility to obtain a detailed picture of network activity during a time interval. Since it uses the trace facility, the `netpmon` command can only be run by root or by a member of the system group. Also, the `netpmon` command cannot run together with any of the other trace-based performance commands like `tprof` and `filemon`. In its normal mode, `netpmon` runs in the background while one or more application programs or system commands are being executed and monitored. Tracing is started by the `netpmon` command, optionally suspended with `trcoff`, resumed with `trcon`, and terminated with `trcstop`. As soon as tracing is terminated, the `netpmon` writes its report to stdout.

The `netpmon` command focuses on the following system activities:

- CPU usage
  - By processes and interrupt handlers.
  - How much is network-related.
  - What causes idle time.
- Network device driver I/O
  - Monitors I/O operations through all Ethernet, token-ring, and Fiber-Distributed Data Interface (FDDI) network device drivers.

  - In the case of transmission I/O, the command monitors utilizations, queue lengths, and destination hosts. For receive ID, the command also monitors time in the demux layer.

- Internet socket calls
  - Monitors send, recv, sendto, recvfrom, sendmsg, read, and write subroutines on Internet sockets.
  - Reports statistics on a per-process basis for the Internet Control Message Protocol (ICMP), Transmission Control Protocol (TCP), and the User Datagram Protocol (UDP).
- NFS I/O
  - On client: RPC requests, NFS read/write requests.
  - On server: Per-client, per-file, read/write requests.

3.7.2 Using netpmon

The following netpmon command running on an NFS server executes the sleep command and creates a report after 400 seconds. During the measured interval, a copy to an NFS-mounted file system /nfs_mnt is taking place.

```
# netpmon -o netpmon.out -O all; sleep 400; trcstop
```

With the -O option, you can specify the report type you wish to be generated. Valid report type values are:

cpu	CPU usage.
dd	Network device-driver I/O.
so	Internet socket call I/O.
nfs	NFS I/O.
all	All reports are produced. This is the default value.

```
# cat netpmon.out

Thu Feb 27 15:02:45 1997
System: AIX itsosmp Node: 4 Machine: 00045067A000
401.053 secs in measured interval
========================================================================
Process CPU Usage Statistics:
-----------------------------
                                                    Network
Process (top 20)             PID  CPU Time   CPU %   CPU %
------------------------------------------------------------
nfsd                       12370   42.2210   2.632   2.632
nfsd                       12628   42.0056   2.618   2.618
nfsd                       13144   41.9540   2.615   2.615
nfsd                       12886   41.8680   2.610   2.610
nfsd                       12112   41.4114   2.581   2.581
nfsd                       11078   40.9443   2.552   2.552
nfsd                       11854   40.6198   2.532   2.532
nfsd                       13402   40.3445   2.515   2.515
lrud                        1548   16.6294   1.037   0.000
netpmon                    15218    5.2780   0.329   0.000
gil                         2064    2.0766   0.129   0.129
trace                      18284    1.8820   0.117   0.000
syncd                       3602    0.3757   0.023   0.000
swapper                        0    0.2718   0.017   0.000
init                           1    0.2201   0.014   0.000
afsd                        8758    0.0244   0.002   0.000
bootpd                      7128    0.0220   0.001   0.000
ksh                         4322    0.0213   0.001   0.000
pcimapsvr.ip               16844    0.0204   0.001   0.000
netm                        1806    0.0186   0.001   0.001
------------------------------------------------------------
Total (all processes)               358.3152  22.336  20.787
Idle time                          1221.0235  76.114
```

```
========================================================================
First Level Interrupt Handler CPU Usage Statistics:
---------------------------------------------------
                                                 Network
FLIH                              CPU Time   CPU %   CPU %
------------------------------------------------------------
PPC decrementer                     9.9419   0.620   0.000
external device                     4.5849   0.286   0.099
UNKNOWN                             0.1716   0.011   0.000
data page fault                     0.1080   0.007   0.000
floating point                      0.0012   0.000   0.000
instruction page fault              0.0007   0.000   0.000
------------------------------------------------------------
Total (all FLIHs)                  14.8083   0.923   0.099

========================================================================
Second Level Interrupt Handler CPU Usage Statistics:
----------------------------------------------------
                                                 Network
SLIH                              CPU Time   CPU %   CPU %
------------------------------------------------------------
tokdd                              12.4312   0.775   0.775
ascsiddpin                          0.5178   0.032   0.000
------------------------------------------------------------
Total (all SLIHs)                  12.9490   0.807   0.775

========================================================================
Network Device-Driver Statistics (by Device):
---------------------------------------------
                      ----------- Xmit -----------   -------- Recv ---------
Device                 Pkts/s  Bytes/s  Util  QLen   Pkts/s  Bytes/s   Demux
------------------------------------------------------------------------------
token ring 0            31.61     4800  1.7% 0.046   200.93   273994  0.0080

========================================================================

Network Device-Driver Transmit Statistics (by Destination Host):
----------------------------------------------------------------
Host                   Pkts/s  Bytes/s
----------------------------------------
ah6000c                 31.57     4796
9.3.1.255                0.03        4
itsorusi                 0.00        0

========================================================================

TCP Socket Call Statistics (by Process):
----------------------------------------
                                  ------ Read -----   ----- Write -----
Process (top 20)            PID   Calls/s   Bytes/s   Calls/s   Bytes/s
------------------------------------------------------------------------
telnetd                   18144      0.03       123      0.06         0
------------------------------------------------------------------------
Total (all processes)                0.03       123      0.06         0

========================================================================

NFS Server Statistics (by Client):
----------------------------------
                       ------ Read -----   ----- Write -----     Other
Client                 Calls/s   Bytes/s   Calls/s   Bytes/s   Calls/s
------------------------------------------------------------------------
ah6000c                   0.00         0     31.54    258208      0.01
------------------------------------------------------------------------
Total (all clients)       0.00         0     31.54    258208      0.01

========================================================================
Detailed Second Level Interrupt Handler CPU Usage Statistics:
-------------------------------------------------------------
SLIH: tokdd
count:                93039
  cpu time (msec):    avg 0.134   min 0.026   max 0.541   sdev 0.051
SLIH: ascsiddpin
count:                8136
```

```
  cpu time (msec):      avg 0.064   min 0.012   max 0.147   sdev 0.018
COMBINED (All SLIHs)
count:                  101175
  cpu time (msec):      avg 0.128   min 0.012   max 0.541   sdev 0.053

========================================================================
Detailed Network Device-Driver Statistics:
----------------------------------------
DEVICE: token ring 0
recv packets:           80584
  recv sizes (bytes):   avg 1363.6  min 50      max 1520    sdev 356.3
  recv times (msec):    avg 0.081   min 0.010   max 0.166   sdev 0.020
  demux times (msec):   avg 0.040   min 0.008   max 0.375   sdev 0.040
xmit packets:           12678
  xmit sizes (bytes):   avg 151.8   min 52      max 184     sdev 3.3
  xmit times (msec):    avg 1.447   min 0.509   max 4.514   sdev 0.374

========================================================================
Detailed Network Device-Driver Transmit Statistics (by Host):
------------------------------------------------------------
HOST: ah6000c
xmit packets:           12662
  xmit sizes (bytes):   avg 151.9   min 52      max 184     sdev 2.9
  xmit times (msec):    avg 1.448   min 0.509   max 4.514   sdev 0.373

HOST: 9.3.1.255
xmit packets:           14
  xmit sizes (bytes):   avg 117.0   min 117     max 117     sdev 0.0
  xmit times (msec):    avg 1.133   min 0.884   max 1.730   sdev 0.253

HOST: itsorusi
xmit packets:           1
  xmit sizes (bytes):   avg 84.0    min 84      max 84      sdev 0.0
  xmit times (msec):    avg 0.522   min 0.522   max 0.522   sdev 0.000

========================================================================
Detailed TCP Socket Call Statistics (by Process):
------------------------------------------------
PROCESS: telnetd   PID: 18144
reads:                  12
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (msec):    avg 0.085   min 0.053   max 0.164   sdev 0.027
writes:                 23
  write sizes (bytes):  avg 3.5     min 1       max 26      sdev 7.0
  write times (msec):   avg 0.143   min 0.067   max 0.269   sdev 0.064
PROTOCOL: TCP (All Processes)
reads:                  12
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (msec):    avg 0.085   min 0.053   max 0.164   sdev 0.027
writes:                 23
  write sizes (bytes):  avg 3.5     min 1       max 26      sdev 7.0
  write times (msec):   avg 0.143   min 0.067   max 0.269   sdev 0.064

========================================================================
Detailed NFS Server Statistics (by Client):
------------------------------------------
CLIENT: ah6000c
writes:                 12648
  write sizes (bytes):  avg 8187.5  min 4096    max 8192    sdev 136.2
  write times (msec):   avg 138.646 min 0.147   max 1802.067 sdev 58.853
other calls:            5
  other times (msec):   avg 1.928   min 0.371   max 8.065   sdev 3.068
COMBINED (All Clients)
writes:                 12648
  write sizes (bytes):  avg 8187.5  min 4096    max 8192    sdev 136.2
  write times (msec):   avg 138.646 min 0.147   max 1802.067 sdev 58.853
other calls:            5
  other times (msec):   avg 1.928   min 0.371   max 8.065   sdev 3.068
```

Like the filemon command, the output of netpmon is composed of two different types of reports: global and detailed. The global reports list statistics for the most active processes, first-level interrupt handlers, second level interrupt handlers, network device drivers, network device-driver transmits, TCP socket

calls, and NFS server or client statistics during the measured interval. The global reports are shown at the beginning of the `netpmon` output. The detailed reports give additional information for the global reports. By default, the reports are limited to the top 20 most active statistics measured. All information in the reports are listed from top to bottom as most active to least active.

3.7.3 The Global Reports of netpmon

The reports generated by the `netpmon` command begin with a header, which identifies the date, the machine ID, and the length of the monitoring period in seconds. This is followed by a set of global and detailed reports for all specified report types.

Process CPU Usage Statistics: Each row describes the CPU usage associated with a process. Unless the verbose (`-v`) option is specified, only the top 20 most active processes are included in the list. At the bottom of the report, CPU usage for all processes is totaled, and CPU idle time is reported. The `Network CPU %` is the percentage of total time that this process spent executing network-related code.

If the `-t` flag is used, there is also a thread CPU usage statistic present. Each process row described above is immediately followed by rows describing the CPU usage of each thread owned by that process. The fields in these rows are identical to those for the process, except for the name field. (Threads are not named.)

In our example report, the idle time percentage number (76.114 percent) shown in the global CPU usage report is calculated from the idle time (1221.0235) divided by the measured interval times 4 (401.053 X 4), because there are 4 CPUs in this server, therefore the results look somewhat strange. If you want to look at each CPU's activity, you can use `sar`, `ps`, or any other SMP-specific command. Similar calculation applies to the total CPU percentage that is occupied by all processes. The idle time is due to network I/O. The difference between the CPU time totals (1221.0235 + 358.315) and the measured interval is due to interrupt handlers and the multiple CPUs. It appears that in this report, the majority of the CPU usage was network-related: (20.787 / 22.336) = 93.07 percent. About 77.664 percent of CPU usage are either idle or wait time.

Note

If the result of total network CPU percentage divided by total CPU percentage is greater than 0.5 from *Process CPU Usage Statistics* for NFS server, then the majority of CPU usage is network-related.

First Level Interrupt Handler CPU Usage Statistics: Each row describes the CPU usage associated with a first-level interrupt handler (FLIH). At the bottom of the report, CPU usage for all FLIHs is totaled.

CPU Time Total amount of CPU time used by this FLIH.

CPU % CPU usage for this interrupt handler as a percentage of total time.

Network CPU % Percentage of total time that this interrupt handler executed on behalf of network-related events.

Second Level Interrupt Handler CPU Usage Statistics: Each row describes the CPU usage associated with a second-level interrupt handler (SLIH). At the bottom of the report, CPU usage for all SLIHs is totaled.

Network Device-Driver Statistics (by Device): Each row describes the statistics associated with a network device.

Device	Name of special file associated with device.
Xmit Pkts/s	Packets per second transmitted through this device.
Xmit Bytes/s	Bytes per second transmitted through this device.
Xmit Util	Busy time for this device, as a percent of total time.
Xmit Qlen	Number of requests waiting to be transmitted through this device, averaged over time, including any transaction currently being transmitted.
Recv Pkts/s	Packets per second received through this device.
Recv Bytes/s	Bytes per second received through this device.
Recv Demux	Time spent in demux layer as a fraction of total time.

In this example, the `Xmit QLen` is only 0.046, it is very small compared to its default size (30). Its `Recv Bytes/s` is 273994, much smaller than the token ring transmit speed (16 Mb/s). Therefore in this case the network is not saturated.

Network Device-Driver Transmit Statistics (by Destination Host): Each row describes the amount of transmit traffic associated with a particular destination host, at the device-driver level.

Host	Destination host name. An * (asterisk) is used for transmissions for which no host name can be determined.
Pkts/s	Packets per second transmitted to this host.
Bytes/s	Bytes per second transmitted to this host.

TCP Socket Call Statistics for Each Internet Protocol (by Process): These statistics are shown for each used Internet protocol. Each row describes the amount of read/write subroutine activity on sockets of this protocol type associated with a particular process. At the bottom of the report, all socket calls for this protocol are totaled.

NFS Server Statistics (by Client): Each row describes the amount of NFS activity handled by this server on behalf of a particular client. At the bottom of the report, calls for all clients are totaled.

On a client machine, the NFS server statistics are replaced by the NFS client statistics (*NFS Client Statistics for each Server (by File)*, *NFS Client RPC Statistics (by Server)*, *NFS Client Statistics (by Process*)).

3.7.4 The Detailed Reports of netpmon

Detailed reports are generated for all requested (`-0`) report types. For these report types, a detailed report is produced in addition to the global reports. The detailed reports contain an entry for each entry in the global reports with statistics for each type of transaction associated with the entry.

Transaction statistics consist of a count of the number of transactions for that type, followed by response time and size distribution data (where applicable). The distribution data consists of average, minimum, and maximum values, as well as standard deviations. Roughly two-thirds of the values are between

average minus standard deviation and average plus standard deviation. Sizes are reported in bytes. Response times are reported in milliseconds.

Detailed Second-Level Interrupt Handler CPU-Usage Statistics

SLIH	Name of second-level interrupt handler.
count	Number of interrupts of this type.
cpu time (msec)	CPU usage statistics for handling interrupts of this type.

Detailed Network Device-Driver Statistics (by Device)

DEVICE	Path name of special file associated with device.
recv packets	Number of packets received through this device.
recv sizes (bytes)	Size statistics for received packets.
recv times (msec)	Response time statistics for processing received packets.
demux times (msec)	Time statistics for processing received packets in the demux layer.
xmit packets	Number of packets transmitted through this device.
xmit sizes (bytes)	Size statistics for transmitted packets.
xmit times (msec)	Response time statistics for processing transmitted packets.

There are other detailed reports, such as *Detailed Network Device-Driver Transmit Statistics (by Host)* and *Detailed TCP Socket Call Statistics for Each Internet Protocol (by Process)*. For an NFS client, there are the *Detailed NFS Client Statistics for Each Server (by File)*, *Detailed NFS Client RPC Statistics (by Server)* and *Detailed NFS Client Statistics (by Process)* reports. For an NFS server, there is the *Detailed NFS Server Statistics (by Client)* report. They have similar output fields as explained above.

In our example, the results from the *Detailed Network Device-Driver Statistics* lead to:

recv bytes = 80584 packets * 1364 bytes/packet = 109,916,576 bytes

xmit bytes = 12678 packets * 152 bytes/packet = 1,927,056 bytes

total bytes exchanged = 109,916,576 + 1,927,056 = 111,843,632 bytes

total bits exchanged = 111,843,632 * 8 bits/byte = 894,749,056 bits

transmit speed = 894,749,056 / 401.053 = 2.23 Mb/s (assuming that the copy took the whole monitoring period)

As in the global device driver report, we come to the conclusion that this case is not network-saturated. The average receive size is 1363.6 bytes, near to the default MTU (maximum transmission unit) value, which is 1492 when the device is a token-ring card. If this value is larger than the MTU (from `lsattr -E -l interface`, replacing `interface` with Ethernet (en0) or token-ring (tr0)), you could change the MTU or adapter transmit-queue length value to get better performance with `ifconfig tr0 mtu 8500` or `chdev -l 'tok0' -a xmt_que_size='150'`. But if the network is congested already, it will probably be worse if you change the MTU or queue value.

Note

- If transmit and receive packet sizes are small from the device driver statistics report, then increasing the current MTU size will probably result in better network performance.
- If system wait time due to network calls is high from the network wait time statistics for the NFS client report, the poor performance is due to the network.

3.7.5 Limitations of netpmon

The netpmon command uses the trace facility to collect the statistics; therefore it has an impact on the system workload.

- In a moderate, network-oriented workload, the netpmon command increases overall CPU utilization by 3-5 percent.
- In a CPU-saturated environment with little I/O of any kind, netpmon slowed a large compile by about 3.5 percent.
- This command reports at the adapter level.

3.8 The genld Command

The genld command extracts a list of loaded objects for each process currently running on the system.

For each process currently running, the genld command will print a report consisting of the process ID and name, followed by the list of objects loaded for that process. The object′s address and path name are displayed. For members of libraries, the path name of the library is shown as a directory, with the name of the loaded member shown as a file in that directory; for example, /usr/lib/libc.a/shr.o, is the library where shr.o is a loaded member of libc.a. Path names ending with a / are executables or object modules.

Example: To obtain the list of loaded objects for each running process, enter:

```
# genld

Proc_pid:        0      Proc_name: swapper

Proc_pid:        1      Proc_name: init
                                   d00fe0c0  /usr/lib/libs.a/shr.o
                                   d012f350  /usr/lib/libc.a/meth.o
                                   d00004e0  /usr/lib/libc.a/shr.o
                                   10000000  init

Proc_pid:      516      Proc_name: wait

Proc_pid:      774      Proc_name: netm

Proc_pid:     1032      Proc_name: gil

Proc_pid:     1874      Proc_name: lvmb

Proc_pid:     2156      Proc_name: srcmstr
                                   d0131000  /usr/lib/nls/loc/en_US/
                                   d00fe0c0  /usr/lib/libs.a/shr.o
                                   d01370c0  /usr/lib/libodm.a/shr.o
                                   d01980c0  /usr/lib/libsrc.a/shr.o
                                   d012f350  /usr/lib/libc.a/meth.o
                                   d00004e0  /usr/lib/libc.a/shr.o
                                   10000000  srcmstr
```

```
Proc_pid:     2488      Proc_name: dtlogin
                                        d01a7000  /usr/lib/netsvc/libbind/
                                        d00fe0c0  /usr/lib/libs.a/shr.o
                                        d025ac30  /usr/lib/libX11.a/shr4net.o
                                        d02550c0  /usr/lib/libIM.a/shr.o
                                        d02450c0  /usr/lib/libiconv.a/shr4.o
                                        d01a90c0  /usr/lib/libX11.a/shr4.o
                                        d012f350  /usr/lib/libc.a/meth.o
                                        d00004e0  /usr/lib/libc.a/shr.o
                                        10000000  dtlogin

Proc_pid:     2656      Proc_name: dtsession
                                        d01a7000  /usr/lib/netsvc/libbind/
                                        d0131000  /usr/lib/nls/loc/en_US/
                                        d06f90c0  /usr/lib/libDtWidget.a/shr.o
                                        d09080c0  /usr/lib/libDtHelp.a/shr.o
                                        d01900c0  /usr/lib/libbsd.a/shr.o
                                        d00fe0c0  /usr/lib/libs.a/shr.o
                                        d07370c0  /usr/lib/libtt.a/shr.o
                                        d06690c0  /usr/lib/libi18n.a/shr.o
                                        d04b00c0  /usr/lib/libXm.a/shr4.o
                                        d06840c0  /usr/lib/libDtSvc.a/shr.o
                                        d01370c0  /usr/lib/libodm.a/shr.o
                                        d02ad0c0  /usr/lib/libgair4.a/shr.o
                                        d01530c0  /usr/lib/libXext.a/shr.o
                                        d025ac30  /usr/lib/libX11.a/shr4net.o
                                        d02550c0  /usr/lib/libIM.a/shr.o
                                        d02450c0  /usr/lib/libiconv.a/shr4.o
                                        d01a90c0  /usr/lib/libX11.a/shr4.o
                                        d0460d00  /usr/lib/libXt.a/shr4.o
                                        d012f350  /usr/lib/libc.a/meth.o
```

You can also find the virtual address in the genkld output. For example, process dtlogin has a load object /usr/lib/libc.a/meth.o, whose virtual address is d012f350. This object is also used by the process srcmstr.

3.9 The genkld Command

The genkld command extracts the list of shared libraries and shared objects currently loaded onto the system and displays the address, size, and path name for each object on the list.

For shared objects loaded onto the system, the kernel maintains a linked list consisting of data structures called loader entries. A loader entry contains the name of the object, its starting address, and its size. This information is gathered and reported by the genkld command.

Example: To obtain a list of loaded shared objects, enter:

```
# genkld
Virtual Address     Size               File

        d025d000          4f71f        /usr/lpp/gai/60x00004001/loadddx/
        d025c0c0            e4f        /usr/lib/libdbm.a/shr.o
        d025c0c0            e4f        /usr/lib/libdbm.a/shr.o
        d025ac30            6b5        /usr/lib/libX11.a/shr4net.o
        d02550c0           4a83        /usr/lib/libIM.a/shr.o
        d02450c0           f754        /usr/lib/libiconv.a/shr4.o
        d01a90c0          9bb57        /usr/lib/libX11.a/shr4.o
        d025ac30            6b5        /usr/lib/libX11.a/shr4net.o
        d02550c0           4a83        /usr/lib/libIM.a/shr.o
        d02450c0           f754        /usr/lib/libiconv.a/shr4.o
        d01a90c0          9bb57        /usr/lib/libX11.a/shr4.o
        d01a7000           17ec        /usr/lib/netsvc/libbind/
        d00fe0c0           3079c       /usr/lib/libs.a/shr.o
        d01900c0           75e7        /usr/lib/libbsd.a/shr.o
        d01470c0           b296        /usr/lib/libcfg.a/shr.o
```

```
d01370c0            f35e        /usr/lib/libodm.a/shr.o
d01980c0            e0c2        /usr/lib/libsrc.a/shr.o
d012f350            12a5        /usr/lib/libc.a/meth.o
d00004e0           fccc0        /usr/lib/libc.a/shr.o
d01980c0            e0c2        /usr/lib/libsrc.a/shr.o
d01900c0            75e7        /usr/lib/libbsd.a/shr.o
d01470c0            b296        /usr/lib/libcfg.a/shr.o
d01370c0            f35e        /usr/lib/libodm.a/shr.o
d0131000            2c41        /usr/lib/nls/loc/en_US/
d0131000            2c41        /usr/lib/nls/loc/en_US/
d00fe0c0           3079c        /usr/lib/libs.a/shr.o
d012f350            12a5        /usr/lib/libc.a/meth.o
d00004e0           fccc0        /usr/lib/libc.a/shr.o
d012f350            12a5        /usr/lib/libc.a/meth.o
d00fe0c0           3079c        /usr/lib/libs.a/shr.o
d00004e0           fccc0        /usr/lib/libc.a/shr.o
```

If path names end with /, they are shared objects (see the output of the `file` command). If path names end without /, they are modules of shared libraries. For example, shr.o is a module out of the libs.a library.

```
# file /usr/lib/nls/loc/en_US/
/usr/lib/nls/loc/en_US/: executable (RISC System/6000) or object module not
stripped
```

If some shared libraries and shared objects are loaded more than once, there is an entry for each of them. For example, /usr/lib/libdbm.a/shr.o has two entries with the same virtual address (d025c0c0) and same size (e4f).

3.10 The genkex Command

The genkex command extracts the list of kernel extensions and device drivers currently loaded onto the system and displays the address, size, and path name for each kernel extension and device driver in the list.

For kernel extensions loaded onto the system, the kernel maintains a linked list consisting of data structures called loader entries. A loader entry contains the name of the extension, its starting address, and its size. This information is gathered and reported by the genkex command.

Example

To generate the list of loaded kernel extensions and device drivers, enter:

```
# genkex

Virtual Address        Size        File

    1a19598        156c        /usr/lib/drivers/rmss.ext
    8020000        e34         /unix
    7e59000        5bd90       ./dkload/afs.ext
    7dcf000        dbc         /unix
    1a182a0        12ec        ./dkload/export
    7dca000        d38         /unix
    19ff940        18954       /usr/lib/drivers/nfs_clnt.ext
    7dcc000        d14         /unix
    19f1de0        db54        /usr/lib/drivers/nfs_krpc.ext
    7dcd000        cf0         /unix
    19f1a98        32c         /usr/lib/drivers/nfs_kdes.ext
    7dcb000        cf0         /unix
    19edac0        3fd0        /etc/drivers/smt_load
    7db1000        cf0         /unix
    19ed8d0        1e8         /etc/drivers/smt_loadpin
    7d9a000        c30         /unix
    19d8840        15084       /usr/lib/drivers/rcm_loadpin
    19d7ac0        d64         /etc/drivers/rcm_load
    7d83000        c30         /unix
```

```
19d56a0         2418          /etc/drivers/gxmedd
19c7840         de58          /etc/drivers/lft_loadpin
7d5c000         c24           /unix
19bc740         b0dc          /etc/drivers/ptydd
19b8b00         3c1c          /usr/lib/drivers/if_tr
7d37000         c24           /unix
199ffa0         18b50         /usr/lib/drivers/netinet
7cd5000         c24           /unix
198df60         12034         /etc/drivers/ldterm
7d0d000         c24           /unix
198b1c0         2d90          /etc/drivers/tok_demux
5533000         c24           /unix
1979180         12038         /etc/drivers/tokdd
19768e0         2888          /etc/drivers/eth_demux
1964b40         11d88         /etc/drivers/ethdd
1950260         148cc         /etc/drivers/rsdd
1942fa0         d2b8          /etc/drivers/ttydbg
5544000         c24           /unix
1941b00         1480          /etc/drivers/ppddpin
193e360         3790          /etc/drivers/ppdd
1939640         4d18          /etc/drivers/mousedd
1934be0         4a50          /etc/drivers/tabletdd
192db20         70a0          /etc/drivers/kbddd
1921f00         bbfc          /etc/drivers/fd
19131c0         ed1c          /etc/drivers/bblddpin
190b3c0         7de8          /usr/lib/drivers/bbldd
5535000         c24           /unix
190a260         113c          /usr/lib/drivers/pse/stdmod
1908f30         131c          /usr/lib/drivers/pse/sc
1907d60         11c8          /usr/lib/drivers/pse/spx
1906980         13d8          /usr/lib/drivers/pse/stddev
18d90a0         2d8c4         /usr/lib/drivers/pse/pse
5524000         c24           /unix
18be840         1a850         /etc/drivers/hd_pin_bot
18b4460         a3bc          /etc/drivers/hd_pin
54b9000         bdc           /unix
18aaca0         979c          /etc/drivers/scdiskpin
18a4340         693c          /etc/drivers/scdisk
1895b40         e7ec          /etc/drivers/pscsi720ddpin
188e2c0         786c          /etc/drivers/pscsi720dd
188d008         12a0          /etc/drivers/mca_ppc_busdd
15b080          0             /unix
549d000         bdc           /unix
```

If some kernel extensions or device drivers are loaded more than once, there is an entry for each of them. Like /unix, they have many entries, which have different virtual addresses and the same or different sizes.

3.11 The stripnm Command

The `stripnm` command prints the symbol table of a specified object file to standard output. The file can be a single object file or an archive library of object files. If the file is an archive, a listing for each object file in the archive is produced. If the symbol table has been stripped from the object file, the `stripnm` command extracts symbol names from the traceback tables. If the traceback tables do not exist, an error message is displayed.

When run using the `-s` flag, the `stripnm` command extracts routine names first from the traceback tables and then from the symbol table, if it exists. Traceback tables are found at the end of routines, and contain the symbolic names of these routines. Routines defined as static do not appear in the symbol table, but may have traceback tables.

The `stripnm` command can also search for the glue code. The glue code is a set of executable instructions in the object file. In the text section of the object file, the glue code is composed of the following sequence of instructions:

8182xxxx xxxx is offset in the table of contents (TOC) and can be any string.
90410014 /* st r2, 14(r1) */
800c0000 /* l r0, 0(r12) 8*/
804c004 /* l r2, 4(r12) */
7c0903a6 /* mtctr r0 */
4e800420 /* bctr */

The stripnm command searches the text section from beginning to end for this sequence. If the command finds a sequence of instructions that matches, it is reported as glue code.

The stripnm command can also be used to search for symbol information in the /unix file. If the /unix file does not correspond to the currently running kernel, a warning message is displayed.

Examples: We will use a program child.c as example. This program does a while execution until the end condition is met.

```
#include <stdio.h>
void eatcpu()
{
#define limite 10e7*5

        double f;

        f=0.0;
        while ( f < limite )
          f++;
}

main(int argc, char ** argv)
{
        int opt;

        mkdir(argv[1]);
        chdir(argv[1]);
        opt=atoi(argv[1]);

        if ( opt == 3)  {
                eatcpu();
                exit(0);
        }
        sleep(60);
        exit(0);
}
```

1. To list the symbols of the child object file, enter:

```
# stripnm child

Symbols from child

errno                   |         0|extern|          |      |     |
chdir                   |         0|extern|          |      |     |
mkdir                   |         0|extern|          |      |     |
exit                    |         0|extern|          |      |     |
fflush                  |         0|extern|          |      |     |
atoi                    |         0|extern|          |      |     |
sleep                   |         0|extern|          |      |     |
TOC                     | 536872640|unamex|          |      |     |.data
_adata                  | 536872640|unamex|          |      |     |.data
errno                   | 536872644|unamex|          |      |     |.data
mkdir                   | 536872648|unamex|          |      |     |.data
chdir                   | 536872652|unamex|          |      |     |.data
atoi                    | 536872656|unamex|          |      |     |.data
exit                    | 536872660|unamex|          |      |     |.data
sleep                   | 536872664|unamex|          |      |     |.data
** no name **           | 536872668|unamex|          |      |     |.data
fflush                  | 536872672|unamex|          |      |     |.data
```

```
                        |          | file   |          |      |      |
crt0main.s              |          | file   |          |      |      |
.__start                | 268435912|extern  |          |      |      |.text
__start                 | 536872608|extern  |          |      |      |.data
_adata                  | 536872036|unamex  |          |      |      |.data
p_xargc                 | 536872676|extern  |          |      |      |.bss
p_xargv                 | 536872680|extern  |          |      |      |.bss
p_xrcfg                 | 536872684|extern  |          |      |      |.bss
p_xrc                   | 536872688|extern  |          |      |      |.bss
child.c                 |          | file   |          |      |      |
** no name **           | 268436032|unamex  |          |      |      |.text
.main                   | 268436032|extern  |          |      |      |.text
.eatcpu                 | 268436204|extern  |          |      |      |.text
** no name **           | 268436568|unamex  |          |      |      |.text
dbxxx.s                 |          | file   |          |      |      |
.__dbsubc               | 268436492|extern  |          |      |      |.text
.__dbsubg               | 268436504|extern  |          |      |      |.text
.__dbsubn               | 268436512|extern  |          |      |      |.text
dbxxx                   | 536872052|unamex  |          |      |      |.data
__dbargs                | 536872096|extern  |          |      |      |.data
__dbsubc                | 536872616|extern  |          |      |      |.data
__dbsubg                | 536872624|extern  |          |      |      |.data
__dbsubn                | 536872632|extern  |          |      |      |.data
glink.s                 |          | file   |          |      |      |
.exit                   | 268436420|unamex  |          |      |      |.text
.exit                   | 268436420|extern  |          |      |      |.text
glink.s                 |          | file   |          |      |      |
.mkdir                  | 268436312|unamex  |          |      |      |.text
.mkdir                  | 268436312|extern  |          |      |      |.text
glink.s                 |          | file   |          |      |      |
.chdir                  | 268436348|unamex  |          |      |      |.text
.chdir                  | 268436348|extern  |          |      |      |.text
glink.s                 |          | file   |          |      |      |
.atoi                   | 268436384|unamex  |          |      |      |.text
.atoi                   | 268436384|extern  |          |      |      |.text
glink.s                 |          | file   |          |      |      |
.sleep                  | 268436456|unamex  |          |      |      |.text
.sleep                  | 268436456|extern  |          |      |      |.text
glink.s                 |          | file   |          |      |      |
.fflush                 | 268436528|unamex  |          |      |      |.text
.fflush                 | 268436528|extern  |          |      |      |.text
```

Each symbol name, which is **no name** if the system cannot determine the symbol name, is followed by its address (a series of blanks if the address is undefined), the type of class (unamex means unnamed external symbol), and section type. The address field can be displayed as a decimal (the default value) or hexadecimal (if the -x flag is specified).

Note

The stripnm command does not list all symbols from the symbol table. Only file names and named and unnamed external symbols are reported. When used with the -s flag, the stripnm command will also suppress printing of some duplicated symbols in the symbol table.

1. To list the symbols address values of the child object file in hexadecimal mode, enter:

```
# stripnm  -x child

Symbols from child

errno                   |0x00000000|extern|          |      |      |
chdir                   |0x00000000|extern|          |      |      |
mkdir                   |0x00000000|extern|          |      |      |
exit                    |0x00000000|extern|          |      |      |
fflush                  |0x00000000|extern|          |      |      |
atoi                    |0x00000000|extern|          |      |      |
sleep                   |0x00000000|extern|          |      |      |
```

```
TOC                    |0x200006c0|unamex|        |      |      |.data
_adata                 |0x200006c0|unamex|        |      |      |.data
errno                  |0x200006c4|unamex|        |      |      |.data
mkdir                  |0x200006c8|unamex|        |      |      |.data
chdir                  |0x200006cc|unamex|        |      |      |.data
atoi                   |0x200006d0|unamex|        |      |      |.data
exit                   |0x200006d4|unamex|        |      |      |.data
sleep                  |0x200006d8|unamex|        |      |      |.data
** no name **          |0x200006dc|unamex|        |      |      |.data
fflush                 |0x200006e0|unamex|        |      |      |.data
                       |          | file |        |      |      |
crt0main.s             |          | file |        |      |      |
.__start               |0x100001c8|extern|        |      |      |.text
__start                |0x200006a0|extern|        |      |      |.data
_adata                 |0x20000464|unamex|        |      |      |.data
p_xargc                |0x200006e4|extern|        |      |      |.bss
p_xargv                |0x200006e8|extern|        |      |      |.bss
p_xrcfg                |0x200006ec|extern|        |      |      |.bss
p_xrc                  |0x200006f0|extern|        |      |      |.bss
child.c                |          | file |        |      |      |
** no name **          |0x10000240|unamex|        |      |      |.text
.main                  |0x10000240|extern|        |      |      |.text
.eatcpu                |0x100002ec|extern|        |      |      |.text
** no name **          |0x10000458|unamex|        |      |      |.text
dbxxx.s                |          | file |        |      |      |
.__dbsubc              |0x1000040c|extern|        |      |      |.text
.__dbsubg              |0x10000418|extern|        |      |      |.text
.__dbsubn              |0x10000420|extern|        |      |      |.text
dbxxx                  |0x20000474|unamex|        |      |      |.data
__dbargs               |0x200004a0|extern|        |      |      |.data
__dbsubc               |0x200006a8|extern|        |      |      |.data
__dbsubg               |0x200006b0|extern|        |      |      |.data
__dbsubn               |0x200006b8|extern|        |      |      |.data
glink.s                |          | file |        |      |      |
.exit                  |0x100003c4|unamex|        |      |      |.text
.exit                  |0x100003c4|extern|        |      |      |.text
glink.s                |          | file |        |      |      |
.mkdir                 |0x10000358|unamex|        |      |      |.text
.mkdir                 |0x10000358|extern|        |      |      |.text
glink.s                |          | file |        |      |      |
.chdir                 |0x1000037c|unamex|        |      |      |.text
.chdir                 |0x1000037c|extern|        |      |      |.text
glink.s                |          | file |        |      |      |
.atoi                  |0x100003a0|unamex|        |      |      |.text
.atoi                  |0x100003a0|extern|        |      |      |.text
glink.s                |          | file |        |      |      |
.sleep                 |0x100003e8|unamex|        |      |      |.text
.sleep                 |0x100003e8|extern|        |      |      |.text
glink.s                |          | file |        |      |      |
.fflush                |0x10000430|unamex|        |      |      |.text
.fflush                |0x10000430|extern|        |      |      |.text
```

2. To list symbols from the traceback tables and symbol table (if it exists) of the child object file, enter:

```
# stripnm  -s child

Symbols from child

.main                  | 268436032|extern|        |      |      |.text
.eatcpu                | 268436204|extern|        |      |      |.text
glink.s                |          | file |        |      |      |
.mkdir                 | 268436312|extern|        |      |      |.text
.chdir                 | 268436348|extern|        |      |      |.text
.atoi                  | 268436384|extern|        |      |      |.text
.exit                  | 268436420|extern|        |      |      |.text
.sleep                 | 268436456|extern|        |      |      |.text
.fflush                | 268436528|extern|        |      |      |.text
errno                  |         0|extern|        |      |      |
TOC                    | 536872640|unamex|        |      |      |.data
errno                  | 536872644|unamex|        |      |      |.data
mkdir                  | 536872648|unamex|        |      |      |.data
chdir                  | 536872652|unamex|        |      |      |.data
atoi                   | 536872656|unamex|        |      |      |.data
```

```
exit                  | 536872660|unamex|          |      |      |.data
sleep                 | 536872664|unamex|          |      |      |.data
** no name **         | 536872668|unamex|          |      |      |.data
fflush                | 536872672|unamex|          |      |      |.data
                      |          | file |          |      |      |
crt0main.s            |          | file |          |      |      |
.__start              | 268435912|extern|          |      |      |.text
__start               | 536872608|extern|          |      |      |.data
_adata                | 536872036|unamex|          |      |      |.data
p_xargc               | 536872676|extern|          |      |      |.bss
p_xargv               | 536872680|extern|          |      |      |.bss
p_xrcfg               | 536872684|extern|          |      |      |.bss
p_xrc                 | 536872688|extern|          |      |      |.bss
child.c               |          | file |          |      |      |
** no name **         | 268436568|unamex|          |      |      |.text
dbxxx.s               |          | file |          |      |      |
.__dbsubc             | 268436492|extern|          |      |      |.text
.__dbsubg             | 268436504|extern|          |      |      |.text
.__dbsubn             | 268436512|extern|          |      |      |.text
dbxxx                 | 536872052|unamex|          |      |      |.data
__dbargs              | 536872096|extern|          |      |      |.data
__dbsubc              | 536872616|extern|          |      |      |.data
__dbsubg              | 536872624|extern|          |      |      |.data
__dbsubn              | 536872632|extern|          |      |      |.data
glink.s               |          | file |          |      |      |
glink.s               |          | file |          |      |      |
glink.s               |          | file |          |      |      |
glink.s               |          | file |          |      |      |
glink.s               |          | file |          |      |      |
glink.s               |          | file |          |      |      |
```

3.12 The trace and trcrpt Commands

The AIX trace facility is a powerful system observation tool. The trace facility captures a sequential flow of time-stamped system events that can be monitored. This includes entry and exit to selected subroutines, kernel routines, kernel extension routines, and interrupt handlers, thereby providing a fine level of detail on system activity. Events are shown in time sequence and in the context of other events. The trace command is a valuable tool for observing system and application execution. Where other tools provide high-level statistics, such as CPU utilization or I/O-wait time, the trace facility is useful in expanding the information to understand what events are happening, who is responsible, when the events are taking place, how they are affecting the system, and why.

The operating system is instrumented to provide general visibility to system execution. Users can extend visibility into their applications by inserting additional events and providing formatting rules.

Care was taken in the design and implementation of this facility to make the collection of trace data efficient so that system performance and flow would be minimally altered by activating trace. Because of this, the trace facility is extremely useful as a performance-analysis tool and as a problem-determination tool.

The overhead added by trace varies widely, depending on the workload and the number of hook IDs being collected. As an extreme case, a long-running, CPU-intensive job in an otherwise idle system took 3.2 percent longer when trace was running with all hooks enabled. When the trace data fills the buffers and must be written to the log, additional CPU is required for file I/O. Usually this is less than five percent. Since trace claims and pins buffer space, it may be slowing down the execution times of programs if the environment is memory constrained. Be aware that the trace log and report files can become very large.

When using trace, the classic trade-off of time versus resource exists. On a fast, busy system with all trace hooks collected (the default), trace will produce megabytes of trace output for each second of real time. This necessitates zeroing in on the phenomenon you wish to study.

3.12.1 The trace Implementation

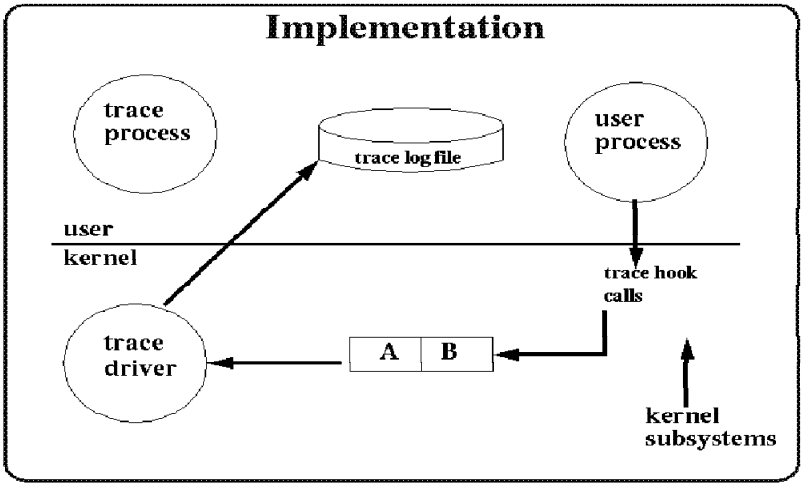


Figure 6. The trace Implementation

The trace command will generate statistics on user processes and kernel subsystems. The binary information is written to two buffers in memory. A is the first, and B is the second; their default size is 131072. The trace process then transfers the information to the trace log file on disk. The default is /var/adm/ras/trcfile, default size is 1310720. This file grows very rapidly. The trace program runs as a process that may be seen by the ps command. The trace acts as a daemon, similar to accounting.

The data recorded for each traced event consists of a word containing the trace hook identifier and the hook type followed by a variable number of words of trace data optionally followed by a time stamp. The word containing the trace hook identifier and the hook type is called the hook word. The remaining two bytes of the hook word are called hook data and are available for recording event data. A trace hook is a specific event that is to be monitored. A unique number is assigned to that event and called a hook ID, which is a three-digit hexadecimal number. Trace monitors these hooks.

3.12.2 Starting and Controlling trace

The trace facility provides three distinct modes of use:

- Subcommand or Interactive Mode:

  Trace is started with a shell command (trace) and carries on a dialog with the user via subcommands (look for the '->' prompt). The subcommands are:

trcon	Start collection of data
trcoff	Stop collection of data
q , quit	Stop collection of data and exit trace
! command	Run 'command'
?	Display summary of trace commands

- Command or Non-Interactive Mode:

  Trace is started with a shell command (trace -a) that includes a flag that specifies that the trace facility is to run asynchronously. The original shell process is free to run ordinary commands interspersed with the trace-control commands trcon to start data collection, trcoff to stop data collection, and trcstop to exit trace.

- Application-Controlled Mode:

  Trace is started (with trcstart()) and controlled by subroutine calls (such as trcon(), trcoff()) from an application program.

3.12.3 Examples of trace

1. Trace system events during execution of anycmd:

   ```
   #trace
   -> !anycmd
   -> q
   #
   ```

2. Trace system events until the trace buffer is full (-f) or until trace is stopped, and include a message in the trace log header. Trace system events during execution of mycmd.

   ```
   #trace -f -m "Trace of events during mycmd"
   -> !mycmd
   -> q
   ```

   Because the -f flag is included, the trace will end if the buffer is full, so the trcoff command may fail with an error message indicating that the trace is not currently running.

3. Trace system events during execution of mycmd1 and later during execution of mycmd2.

   ```
   # trace
   -> !mycmd1
   -> trcoff
   -> trcon
   -> !mycmd2
   -> q
   ```

4. Trace the activity generated by a copy command by starting the trace in asynchronous mode, setting the trace buffer to 1000000 bytes (-T), the log file size to 2000000 (-L), and writing the trace log to trace.out. The default log file is /var/adm/ras/trcfile.

   ```
   # trace -a -L 2000000 -T 1000000 -o trace.out
   # cp /a20kfile /b
   # trcstop
   ```

5. With the -d option, trace is not immediately started but delayed until the first trcon occurs. So a trace collection for the find command might be done as follows:

   ```
   # trace -a -d -f -T 80000000 -L 80000000 -o ./trace.out
   # trcon ; find / -name munga.out -print ; trcoff
   # trcstop
   ```

   The trace command will collect 80 MB of trace data (-T) and stop collecting trace data when the buffer is full (-f). It is important that the system has sufficient free memory (> 80 MB) that can be dedicated to the trace

command, or the command will fail or the workload will be perturbed (perhaps by excessive paging).

6. If the workload is more complex, or is running in the background (for example, a database engine), the same basic technique as in the example above is used. When the workload exhibits interesting behavior, the trace may be started as follows:

   ```
   # trace -a -d -f -T 80000000 -L 80000000 -o ./trace.out
   # trcon ; sleep 60 ; trcoff
   # trcstop
   ```

   It is unlikely, in the example above, that a full 60-second trace will be collected due to the use of the -f flag. You will actually get a trace corresponding to one full trace buffer (that is, 80 MB). Please bear in mind that trace adds considerable path length in system time; so using trace will distort the user/system time mix.

7. The trace data accumulates very rapidly. A technique to reduce the volume and bracket the data collection around an area of interest is to issue several commands on the same line.

   ```
   # trace -a -o trace.out; cp /usr/lib/boot/unix /dev/null; ps; trcstop
   ```

   Another possibility to reduce the trace output is collecting only specific events using -k or -j options.

Note

Do not forget to stop the trace by using trcstop once you have collected your data.

You can also use smit or smitty to start trace:

```
# smitty -> Problem Determination
               -> Trace
                       -> START Trace
```

Or use the smit fastpath (another is smitty trace):

```
# smitty trcstart
```

```
                                START Trace

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                      [Entry Fields]
  EVENT GROUPS to trace (default is kernel trace)     []                      +
  ADDITIONAL event IDs to trace                       []                      +X
  Event IDs to EXCLUDE from trace                     []                      +X
  Trace MODE                                          [alternate]             +
  STOP when log file full?                            [no]                    +
  LOG FILE                                            [/var/adm/ras/trcfile]
  SAVE PREVIOUS log file?                             [no]                    +
  Omit PS/NM/LOCK HEADER to log file?                 [yes]                   +
  Omit DATE-SYSTEM HEADER to log file?                [no]                    +
  Run in INTERACTIVE mode?                            [no]                    +
  Trace BUFFER SIZE in bytes                          [131072]                #
  LOG FILE SIZE in bytes                              [1310720]               #
```

As shown above, you can specify the event groups to trace. For example, group proc contains EXECS, EXITS, and FORKS with trace hook IDs 134, 135, and 139. These hook IDs correspond to the appropriate system calls. You can specify additional event IDs to trace, the trace mode (alternate means trace events are captured in the trace log file), the trace log file, the running mode, the trace buffer size, and the log file size.

3.12.4 Using trcrpt to View Trace Data

A general-purpose trace report facility is provided by the trcrpt command. The report facility provides little data reduction, but converts the binary trace output to a readable ASCII listing. Data can be visually extracted by the reader, or tools can be developed to further reduce the data.

The report facility displays text and data for each event according to rules provided in the trace format file. The default trace format file is /etc/trcfmt. It contains a stanza for each event ID. The stanza for the event provides the report
facility with formatting rules for that event. This technique allows users to add their own events to programs and insert corresponding event stanzas in the format file to specify how the new events should be formatted.

When trace data is formatted, all data for a given event is usually placed on a single line. The header of the trace report tells you when and where the trace was taken, as well as the command that was used to produce it. Additional lines may contain explanatory information. Depending on the fields included, the formatted lines can easily exceed 80 characters. It is best to view the reports on an output device that supports 132 columns.

The trcrpt facility does not produce any summary reports, but simple summaries could be created through further processing of the trcrpt output using awk.

If trcrpt is run on a machine different from the one where the trace output was collected, then the output of the trcnm command is also needed as input to the trcrpt command (-n trcnm_file). It is also preferable to use a copy of /etc/trcfmt from the source machine and use this with the -t trcfmt_file option of trcrpt.

3.12.5 trcrpt Examples

1. To produce a report from the trace.out file generated in example 7 above, including the execution name and PID and writing the output to trcrpt.out, you could use:

   ```
   # trcrpt -O exec=on,pid=on trace.out > trcrpt.out
   ```

2. To get a list of all events with event ID and name:

   ```
   # trcrpt -j | more
   004 TRACEID IS ZERO
   355 DIAGEX
   709 INPUTDD:
   ...
   001 TRACE ON
   002 TRACE OFF
   003 TRACE HEADER
   005 LOGFILE WRAPAROUND
   006 TRACEBUFFER WRAPAROUND
   007 UNDEFINED TRACE ID
   ...
   12E CLOSE SYSTEM CALL
   130 CREAT SYSTEM CALL
   ```

```
134 EXEC SYSTEM CALL
135 EXIT SYSTEM CALL
137 FCNTL SYSTEM CALL
139 FORK SYSTEM CALL
13A FSTAT SYSTEM CALL
13E FULLSTAT SYSTEM CALL
14C IOCTL SYSTEM CALL
14E KILL SYSTEM CALL
152 LOCKF SYSTEM CALL
154 LSEEK SYSTEM CALL
15B OPEN SYSTEM CALL
15F PIPE SYSTEM CALL
...
```

3. To get only the open system calls (event 15b) out of trace.out:

```
# trcrpt -d 15b -O exec=on,pid=on trace.out > trcrpt.out
# more trcrpt.out

ID  PROCESS NAME   PID      I    ELAPSED_SEC     DELTA_MSEC APPL    SYSCALL KERNEL
INTERRUPT

15B cp             9666          0.012020224      12.020224 open /usr/lib/boot/unix
fd=3 RDONLY
15B cp             9666          0.012117504       0.097280 open /dev/null fd=4
WRONLY TRUNC
15B ps             9668          0.676786432     664.668928 open /dev fd=3 RDONLY
15B ps             9668          0.680534016       3.747584 open /dev/pts fd=4
RDONLY
15B ps             9668          0.718295296      37.761280 open /dev/.SRC-unix
fd=4 RDONLY
15B trcstop        9670          0.747842816      29.547520 open /dev/systrctl
fd=3 RDONLY
```

   The ID column shows the hook identifier. The PROCESS NAME column shows the running command or program name. The PID shows the corresponding process ID of the running command or program. The ELAPSED_SEC column shows the elapsed time in seconds after the trace is started. The DELTA_MSEC is the time in milliseconds between the previous event and this event. The APPL SYSCALL KERNEL INTERRUPT are actually indention based. The indention gives the level at which the event occurred (application (APPL), system call (SYSCALL), kernel level (KERNEL), or interrupt level (INTERRUPT)). For example, the system call function open() begins under SYSCALL and is followed by the parameters being passed. In our example, ps, with a PID 9668, was invoked 0.676786432 seconds after the trace was started; the time between the last cp event and this event was 0.664668928 seconds.

4. To get just the opens performed by ‘cp’:

```
# trcrpt -d 15b -p <pid_of_cp> -O exec=on trace.out > cp.out
```

When using trace to determine system problems or CPU idle time, you should monitor the system under a full workload. Review the various calls and interrupts and the time incurred for each. This should give you a better handle on what areas are creating problems.

If it appears that the system is waiting on remote requests, there may be a network problem. Or, if there is idle time waiting on an adapter, that also should be evident. Logical resources may also be a problem. The buffer pool for communications may be in contention. Additional information can be gained through the netstat, nfsstat, or netpmon commands.

3.12.6 How to Spot Thrashing

The operating system determines whether a memory over-commitment is likely. This over-commitment can cause thrashing and severe degradation in system performance. Without the trace facility, it is difficult to determine when the operating system has detected thrashing and subsequently invoked the memory load control mechanism.

The thrash condition is reached when the system spends more time paging rather than performing work. When this occurs, selected processes may be suspended temporarily, and the system can be noticeably slower.

The only reliable way to determine when the system has detected that it is thrashing is to run trace, capturing only trace hook IDs 207 (swapper sched stats) and 208 (swapper process stats).

```
# trace -aj207,208 ; sleep 200 ; trcstop
# trcrpt -O exec=on,pid=on > trcrpt.out
# more trcrpt.out

Wed Mar  5 09:06:05 1997
System: AIX ah6000c Node: 4
Machine: 000127294600
Internet Address: 09030173 9.3.1.115

trace -aj207,208

ID  PROCESS NAME   PID      I    ELAPSED_SEC     DELTA_MSEC   APPL    SYSCALL KERNEL  INTERRUPT

001 trace          21078         0.000000000       0.000000                    TRACE ON channel
0 Wed Mar  5 09:06:06 1997
208 scheduler      0             0.851751296     851.751296                    sched proc stats:
nrun=0 nsusp=0 npageio=
0 nevents=73 nstarting=0
207 scheduler      0             0.851764096       0.012800                    sched sys stats:
thrashing=0 v_repage_hi=
6 sysrepage=0 pgflts=426348 pgstls=181894
208 scheduler      0             1.851817472    1000.053376                    sched proc stats:
nrun=0 nsusp=0 npageio=
0 nevents=73 nstarting=0
207 scheduler      0             1.851832064       0.014592                    sched sys stats:
thrashing=1 v_repage_hi=
6 sysrepage=0 pgflts=426358 pgstls=181894
208 scheduler      0             2.851917056    1000.084992                    sched proc stats:
nrun=2 nsusp=0 npageio=
0 nevents=71 nstarting=0
207 scheduler      0             2.851931520       0.014464                    sched sys stats:
thrashing=2 v_repage_hi=
6 sysrepage=0 pgflts=426367 pgstls=181894
208 scheduler      0             3.851999104    1000.067584                    sched proc stats:
nrun=1 nsusp=0 npageio=
0 nevents=72 nstarting=0
207 scheduler      0             3.852013312       0.014208                    sched sys stats:
thrashing=3 v_repage_hi=
6 sysrepage=0 pgflts=426377 pgstls=181894
208 scheduler      0             4.852001408     999.988096                    sched proc stats:
nrun=0 nsusp=0 npageio=
0 nevents=73 nstarting=0
207 scheduler      0             4.852015360       0.013952                    sched sys stats:
thrashing=4 v_repage_hi=
6 sysrepage=0 pgflts=426386 pgstls=181894
208 scheduler      0             5.852055424    1000.040064                    sched proc stats:
nrun=0 nsusp=0 npageio=
0 nevents=73 nstarting=0
207 scheduler      0             5.852069632       0.014208                    sched sys stats:
thrashing=5 v_repage_hi=
6 sysrepage=0 pgflts=426396 pgstls=181894
208 scheduler      0             6.852113792    1000.044160                    sched proc stats:
nrun=0 nsusp=0 npageio=
0 nevents=73 nstarting=0
207 scheduler      0             6.852127616       0.013824                    sched sys stats:
```

```
thrashing=6 v_repage_hi=
6 sysrepage=0 pgflts=426405 pgstls=181894
208 scheduler      0             7.852273792    1000.146176                  sched proc stats:
nrun=0 nsusp=0 npageio=
0 nevents=73 nstarting=0
207 scheduler      0             7.852287104       0.013312                  sched sys stats:
thrashing=-1 v_repage_hi
=6 sysrepage=0 pgflts=426415 pgstls=181902
208 scheduler      0             8.852322944    1000.035840                  sched proc stats:
nrun=0 nsusp=0 npageio=
0 nevents=73 nstarting=0
```

The output field thrashing indicates the number of seconds the system has been free of thrashing. This value is not incremented beyond 100. A value of -1 indicates the system was currently thrashing.

The output field nsusp indicates the number of processes that were suspended when thrashing was detected.

3.12.7 Using trace to Identify Other Resource Constraints

Following is a trcrpt report showing that the memory allocation is probably constrained. You can see every activity going out to VMM pagefault, VMM disk allocation, and VMM page assign. We go into a page-in/page-out state until we get a VMM zero filled page. After this event, process ID 5572 is eventually resumed.

```
# more trcrpt.out

ID  PROCESS NAME   PID      I    ELAPSED_SEC     DELTA_MSEC   APPL    SYSCALL KERNEL  INTERRUPT
...
104 -5572-         5572          0.203864448       0.003584           return from sbrk [46 usec]
100 -5572-         5572          0.203915520       0.051072                        DATA
ACCESS PAGE FAULT
1B2 -5572-         5572          0.203937280       0.021760                VMM pagefault:
V.S=3A00.183C client_segment
100 -5572-         5572          0.203988352       0.051072                        DATA
ACCESS PAGE FAULT
1B2 -5572-         5572          0.203999232       0.010880                VMM pagefault:
V.S=019E.0202 deferred_update system_segment commit_in_progress
1BD -5572-         5572          0.204056320       0.057088                VMM disk
allocation:  V.S=019E.0202 dbl k=893C deferred_update system_segment commit_in_progress
pdtx/devid=0000
1B0 -5572-         5572          0.204132864       0.076544                VMM page assign:
V.S=019E.0202 ppa ge=09AC deferred_update system_segment commit_in_progress
1B9 -5572-         5572          0.204140032       0.007168                VMM zero filled
page: V.S=019E.0202 ppage=09AC deferred_update system_segment commit_in_progress
200 -5572-         5572          0.204151936       0.011904                resume   -5572-
100 -5572-         5572          0.204176896       0.024960                        DATA
ACCESS PAGE FAULT
1B2 -5572-         5572          0.204183296       0.006400                VMM pagefault:
V.S=3A00.183C client_segment
1BD -5572-         5572          0.204226944       0.043648                VMM disk
allocation:  V.S=3A00.183C dbl k=893D client_segment pdtx/devid=0000
1B0 -5572-         5572          0.204273024       0.046080                VMM page assign:
V.S=3A00.183C ppage=09A1client_segment
1B9 -5572-         5572          0.204276352       0.003328                VMM zero filled
page: V.S=3A00.183C ppage=09A1 client_segment
200 -5572-         5572          0.204281472       0.005120                resume   -5572-
```

Chapter 4. Advanced AIX V4 Performance Tools

This chapter describes some advanced, detailed commands that were introduced with AIX V4. Some of them come with the AIX basic operating system; some of them are part of the AIX Performance Toolbox LPP software.

These advanced, AIX-specific tools exist because logical resources like VMM, LVM, file systems, queues, and buffers are implementation-specific, and standard tools do not provide enough detailed information or do not allow fine tuning.

4.1 PDT

The performance diagnostic tool (PDT) collects information concerning system configuration and tracks changes in the workload and performance. It attempts to identify potential problems that could impact the overall performance of the system. Based on the configuration and the historical record of performance measurements, PDT tries to identify:

- Unbalanced use of the resources or asymmetrical aspects of configuration or device utilization. In general, if there are several resources of the same type, a better performance can be achieved by meeting the following goals:
  - Comparable numbers of physical volumes (disks) on each disk adapter
  - Paging space equally distributed across multiple physical volumes
  - Roughly equal measured load on different physical volumes
- Reaching the limit of resource utilization. Trends that would attempt to exceed those limits should be detected and reported. Performance suffers when a disk or a file system is 100 percent full.
- New consumers of resource-expensive processes that have not been observed before. Trends can indicate a change in the nature of the workload as well as increases in the amount of resources used:
  - Number of users logged on
  - Total number of processes
  - CPU-idle percentage
- Hardware and software errors that can lead to performance problems. PDT checks hardware and software logs and reports bad VMM pages.
- Inappropriate setting of system parameters. AIX allows system administrators to change system parameters in order to enhance the performance. However, if parameters are improperly set, it can greatly degrade performance.

PDT uses normally less than 30 seconds of CPU time. Daily data collection takes several elapsed minutes, but most of that time is spent sleeping.

This tool is only available in AIX V4.

4.1.1 Enabling and Configuring PDT

PDT must be enabled in order to begin data collection and report writing. PDT can be enabled by executing as root the script /usr/sbin/perf/diag_tool/pdt_config. When executed the following output is displayed:

```
_____________________________PDT customization menu_____________________________

1) show current  PDT report recipient and severity level
2) modify/enable PDT reporting
3) disable       PDT reporting
4) modify/enable PDT collection
5) disable       PDT collection
6) de-install    PDT
7) exit pdt_config
Please enter a number:
```

Selecting option 4 enables PDT default data collection and reporting. PDT entries are added to the user adm crontab (/var/spool/cron/crontabs/adm), as shown bellow:

```
0  9 * * 1-5   /usr/sbin/perf/diag_tool/Driver_ daily
0 10 * * 1-5   /usr/sbin/perf/diag_tool/Driver_ daily2
0 21 * * 6     /usr/sbin/perf/diag_tool/Driver_ offweekly
```

Daily and weekly reports are mailed to user adm. The default time for data collection is 9 a.m. and for report generation is 10 a.m. These can be changed by altering the crontab for user adm. It would be interesting to change those times to the peek utilization hours; so the reports would reflect the worst case in the environment. PDT also maintains a copy of the last report in /var/perf/tmp/PDT_REPORT. Before a new report is written, the previous report is renamed /var/perf/tmp/PDT_REPORT.last.

The recipient as well as the severity-level of the reports can be changed with option 2:

```
_____________________________PDT customization menu_____________________________

1) show current  PDT report recipient and severity level
2) modify/enable PDT reporting
3) disable       PDT reporting
4) modify/enable PDT collection
5) disable       PDT collection
6) de-install    PDT
7) exit pdt_config
Please enter a number: 2

enter id@host for recipient of report: root
enter severity level for report (1-3):3

report recipient and severity level
root 3
```

The above configuration changed the recipient of the reports to root and the severity-level to 3. This means that the PDT report will be mailed to the root user, and severity-level 1, 2 and 3 messages will be included.

Hint

Set the severity level of the PDT report to 3 so you always get the most information without having to generate higher-level reports.

Remember that selecting severity-level x results in the reporting of all problems of severity less than or equal to x. Note the use of option 1 to determine the current PDT report recipient and report severity level. In order to disable PDT reporting, use option 3, and to disable collection, use option 5.

Changes in the configuration can be done by altering directly the customization files located in /var/perf/cfg/diag_tool:

- .collection.control

  List of scripts that are executed to generate the report.

- .files

  List of files and directories that are analyzed for systematic growth in size.

  By default, the following files are monitored:

  ```
  /usr/adm/wtmp
  /var/spool/qdaemon
  /var/adm/ras/
  /tmp/
  ```

  It is a good policy to append the user and data areas to .files. In this way the system administrator can forecast possible problems.

- .reporting.list

  Contains the information about the recipient and severity-level of the reports. The default is to mail the report to user adm and severity-level of 1. These settings can be altered by directly editing this file or with the `pdt_config` command as shown in the previous example.

- .retention.list

  Number of days for data retention. PDT periodically reviews the collected data and discards data that is out of date. The reports are based on the current set of historical data. The default is 35 days. In order to generate reports based on a larger interval of time, this number should be increased.

- .nodes

  List of nodes that PDT tries to reach in order to check if they are still up or if the network is working properly. This file does not exist by default and should be created if the system administrator wants to monitor a set of nodes. For example, to monitor nodes ah6000a and itsosmp, the file .nodes would be:

  ```
  ah6000a
  itsosmp
  ```

  Nodes that are normally accessed by the users should be added to this file.

- .thresholds

  Contains the thresholds used in analysis and reporting. These thresholds have an effect on PDT report organization and content.

  - DISK_STORAGE_BALANCE

The SCSI controllers having the largest and the smallest disk storage are identified. If the difference (in MB) between these two exceeds DISK_STORAGE_BALANCE, a message is reported. The default value is 800. Any integer value between 0 and 10000 is valid.

– PAGING_SPACE_BALANCE

  The paging spaces having the largest and the smallest areas are identified. If the difference (in MB) between these two exceeds PAGING_SPACE_BALANCE a message is reported. The default value is 4. Any integer value between 0 and 100 is accepted.

– NUMBER_OF_BALANCE

  The SCSI controllers having the largest and the least number of disks attached are identified. If the difference between these two counts exceeds NUMBER_OF_BALANCE, a message is reported. The default value is 1. It can be set to any integer value in the range of 0 to 10000.

– MIN_UTIL

  Applies to process utilization. Changes in the top three CPU or memory consumers are only reported if the new process had a utilization in excess of MIN_UTIL. The default value is 3. Any integer value from 0 to 100 is valid.

– FS_UTIL_LIMIT

  Applies to journaled file system utilization. If the file system has a percentage use above FS_UTIL_LIMIT, a message is reported. The default value is 90 percent. Any integer value between 0 and 100 is accepted.

  Special attention should be given to /, /var, and /tmp file systems. The operating system uses these areas for normal operation. If there is no space left in one of these, the behavior of the system is unpredictable. Error messages are given when the execution of commands fails, but most of the times they don't lead to the real problem. A convenient procedure would be to decrease FS_UTIL_LIMIT to 70 or 80 percent; so problems involving these file systems would be detected earlier.

– MEMORY_FACTOR

  The objective is to determine if the total amount of memory is adequately backed up by paging space. If real memory is close to the amount of used paging space, then the system is likely paging and would benefit from the addition of memory.

  The formula is based on experience and actually compares MEMORY_FACTOR * memory with the average used paging space.

  The current default is .9; by decreasing this number, a warning will be produced more frequently (and perhaps, unnecessarily). Increasing this number will eliminate the message altogether. It can be set anywhere between .001 and 100.

– TREND_THRESHOLD

  Used in all trending assessments. It is applied after a linear regression is performed on all available historical data. This technique basically draws the best line among the points. The slope of the fitted line must exceed the last_value * TREND_THRESHOLD. The objective is to try to

ensure that a trend, however strong its statistical significance, actually has some practical significance.

For example, if we determine that a file system is growing at X MB a day, and the last_value for the file system size is 100 MB, we require that X exceeds 100 MB * TREND_THRESHOLD to be reported as a trend of practical significance. The default value is 0.01; so a growth rate of 1 MB per day would be required for reporting. The threshold can be set anywhere between 0.00001 and 100000.

This assessment applies to trends associated with:

1. CPU use by a top-three process
2. Memory use by a top-three process
3. Size of files indicated in the .files file
4. Journaled file systems
5. Paging spaces
6. Hardware and software errors
7. Workload indicators
8. Processes per user
9. Ping delay to nodes in the .nodes file
10. Percentage of packet loss to nodes in the .nodes file

– EVENT_HORIZON

Used also in trending assessments. For example, in the case of file systems, if there is a significant (both statistical and practical) trend, the time until the file system is 100 percent full is estimated. If this time is within EVENT_HORIZON, a message is reported. The default value is 30, and it can be any integer value between 0 and 100000.

4.1.2 PDT Report

By default, PDT reports are generated with severity-level 1 and mailed to user adm. The severity level may vary from 1 to 3, and as the severity level increases, more detailed information is added to the report.

Reports based on existing data can also be generated on demand by any user executing the script /usr/sbin/perf/diag_tool/pdt_report [SeverityNum]. If no severity number is given, 1 is assumed. The report is written to stdout; so it does not affect /var/perf/tmp/PDT_REPORT or /var/perf/tmp/PDT_REPORT.last files.

Immediately after PDT is installed and enabled, it is not able to generate a report. If a report is requested, the following error is returned:

```
awk: 0602-533 Cannot find or open file /var/perf/tmp/.SM.
```

This happens because no data has been collected. Data can be collected on demand by executing /usr/sbin/perf/diag_tool/Driver_ daily (as root or adm user) or by changing the collection data time in the user adm crontab. This command generates the file /var/perf/tmp/.SM, which is the database that PDT uses to generate the reports.

A PDT report consists of several sections:

- Header

  Provides information on the time and date of the report, the host name, and the time period for which data was analyzed. The content of this section does not differ with other severity levels.

  Example of a report header:

```
Performance Diagnostic Facility 1.0

 Report printed: Thu Feb 6 17:47:20 1997

 Host name: itsosmp.itsc.austin.ibm.com
 Range of analysis includes measurements
 from: Hour 12 on Tuesday, February 4th, 1997
 to: Hour 17 on Wednesday, February 12th, 1997


 Notice: To disable/modify/enable collection or reporting
         execute the pdt_config script as root
```

- Alerts

  Focuses on identified violations of applied concepts and thresholds. The following subsystems may have problems when they appear in the alerts section: file systems, I/O configuration, paging configuration, I/O balance, paging space, virtual memory, real memory, processes and network.

  For severity 1 levels, alerts focus on file systems, physical volumes, paging and memory. If severity 2 or 3 is selected, information on configuration and processes are added.

  Example of an Alerts section:

```
----------------------- Alerts --------------------

  I/O CONFIGURATION
   -  Note: volume hdisk1 has 872 MB available for allocation
      while volume hdisk0 has 148 MB available
   -  Physical volume hdisk2 is unavailable; (in no volume group)

  PAGING CONFIGURATION
   -  Physical Volume hdisk2 (type: SCSI) has no paging space defined
   -  Paging space paging00 on volume group rootvg is fragmented
   -  Paging space paging01 on volume group uservg is fragmented

  I/O BALANCE
   -  Phys. volume hdisk2 is not busy
      volume hdisk2, mean util. = 0.00 %

  PROCESSES
   -  First appearance of 20642 (cpubound) on top-3 cpu list
      (cpu % = 24.10)
   -  First appearance of 20106 (eatmem) on top-3 memory list
      (memory % = 8.00)

  FILE SYSTEMS
   -  File system hd2 (/usr) is nearly full at 100 %

  NETWORK
   -  Host ah6000e appears to be unreachable.
```

```
      (ping loss % = 100) and has been for the past 4 days
```

The I/O configuration indicates that the data is not well-distributed through the disks and that hdisk2 is not used at all. This disk should be added to a volume group and have a paging space defined on it. The existing paging areas are fragmented and should be reorganized, for example with the command reorgvg. The I/O balance section shows that hdisk2 is not busy. This happens because hdisk2 has not been assigned to a volume group.

In most systems /usr file system use is nearly 100 percent. Usually this is not a problem, but system administrators should check if there is any application writing data to this file system.

- Upward and Downward Trends

  PDT employs a statistical technique to determine whether there is a trend in a series of measurements. If a trend is detected, the slope of the trend is evaluated for its practical significance. For upward trends, the following items are evaluated: files, file systems, hardware and software errors, paging space, processes, and network. For downward trends, the following can be reported: files, file systems and processes.

  Example of an Upward Trends and Downward Trends section:

```
---------------------- Upward Trends ----------------

  FILES
   -  File (or directory) /usr/adm/wtmp SIZE is increasing
      now, 20 KB and increasing an avg. of 2163 bytes/day
   -  File (or directory) /var/adm/ras/ SIZE is increasing
      now, 677 KB and increasing an avg. of 11909 bytes/day
  FILE SYSTEMS
   -  File system hd9var (/var) is growing
      now, 17.00 % full, and growing an avg. of 0.38 %/day
   -  File system lv00 (/usr/vice/cache) is growing
      now, 51.00 % full, and growing an avg. of 4.64 %/day
      At this rate, lv00 will be full in about 9 days
  PAGE SPACE
   -  Page space hd6 USE is growing
      now, 81.60 MB and growing an avg. of 2.69 MB/day
      At this rate, hd6 will be full in about 29 days
  ERRORS
   -  Software ERRORS; time to next error is 0.958 days

---------------------- Downward Trends --------------
  PROCESSES
   -  Process 13906 (maker4X.e) CPU use is declining
      now 1.20 % and declining an avg. of 0.68 % per day
   -  Process 13906 (maker4X.e) MEMORY use is declining
      now 13.00 and declining an avg. of 0.98 % per day
  FILES
   -  File (or directory) /tmp/ SIZE is declining
  FILE SYSTEMS
   -  File system hd3 (/tmp) is shrinking
```

The /usr/adm/wtmp is liable to grow unbounded. If it gets too large, login times can increase. In some cases, the solution is to delete the file. In most cases, it is important to identify the user causing the growth and work with that user to correct the problem.

The errorlog file is located in the directory /var/adm/ras. The ERRORS section shows that the number of software errors is increasing. This is probably the

reason why the directory size increased. You should also check the errorlog, verify which application is in error, and correct the problem.

The increase in the paging space use may be due to a process with a memory leak. That process should be identified and the application fixed. However, the paging space might not be well dimensioned and may need to be enlarged. Later on in this section, a rule of thumb will be given to size the paging space.

- System Health

  Gives an assessment of the average number of processes in each process state on the system. Additionally, workload indicators are noted for any upward trends.

  Example of a System Health section:

```
---------------------- System Health ---------------

  SYSTEM HEALTH
   -  Current process state breakdown:
      75.00 [ 100.0 %] : active
      0.40 [ 0.5 %] : swapped
      75.00 = TOTAL
      [based on 1 measurement consisting of 10 2-second samples]
```

- Summary

  The severity level of the current report is listed. There is also an indication given as to whether more details are available at higher severity levels.

  Example of a Summary section:

```
-------------------- Summary -------------------------
  This is a severity level 3 report
  No further details available at severity levels > 3
```

Severity 1 Problems

- Journaled file system becomes unavailable

  The file system could have been removed. Use the `lsfs` command to verify.

- Journaled file system nearly full

  This problem could be caused by large or core files within the file system. The process or user that generated those files should be identified. The system administrator should also verify if PDT report indicates a long term growth trend for this file system.

- Physical volume not allocated to a volume group

  If a physical volume is not allocated to a volume group, AIX has no access to this disk, and its space is being wasted. The `lspv` command could be used to ensure that the disk is not allocated to any volume group, and if not, the command `extendvg` should be used to add the disk to a volume group.

- All paging spaces defined on one physical volume

  A better I/O throughput could be achieved if the paging space is split equally among all physical volumes. Only one paging space should be defined per physical volume because the system will only have access to one at a time. SMIT can be used to create, modify, activate, or deactivate the paging areas.

- System appears to have too little memory for current workload

The vmstat or svmon commands can give further details about the paging activity. Please refer to 2.1, “The vmstat Command” on page 7, and to 3.2, “The svmon Command” on page 71, for more information on those commands.

- Page space nearly full

  The paging space might not be well dimensioned and may need to be enlarged. For systems up to 256 MB of memory, the paging space should be twice the size of real memory. For memories larger than 256 MB, the following is recommended:

  ```
  total paging space = 512 MB + (memory size - 256 MB) * 1.25
  ```

  The paging space size cannot be less than 16 MB and not greater than 20 percent of total disk space.

  The problem may also occur due to a process with a memory leak, in which case that process should be identified and the application fixed.

- Possible problems in the settings of load control parameters

  This may be due to inappropriate load-control parameters settings. Use schedtune command to view or alter the configuration. Refer to 4.10, “The schedtune Command” on page 177, for further details.

- VMM-detected bad memory frames

  It may be necessary to have the memory analyzed. The amount of installed memory should be compared to the accessible memory. If the latter is less than the former, then bad memory has been identified. The command /usr/sbin/perf/diag_tool/getvmparms could be used to check valid memory pages.

- Any host in .nodes becomes unreachable

  This problem may be due to name resolution. Domain Name Service (DNS) configuration files or /etc/hosts should be checked depending on the type of name resolution being used at the environment.

  The command ping could be used to check if the machine has access to other nodes in the same network. Maybe the remote node is down. If it cannot access any other node, cables and connections should be veryfied as well as the routing table of the current machine. This can be accomplished by executing the command netstat -r.

Severity 2 Problems

- Imbalance in the I/O configuration (disks per adapter)

  The number of disks per adapter should be equal whenever possible. This prevents one adapter from being overloaded. A rule of thumb is not to have more than four devices per adapter, especially if the access to the disks is mostly sequential.

- Imbalance in allocation of paging space on physical volumes with paging space

  A substantial imbalance in the sizes of paging spaces can cause performance problems. As stated above, the paging space should be equally distributed throughout the disks.

- Fragmentation of a paging space in a volume group

Paging performance is better if paging areas are contiguous on a physical volume. However, when paging areas are enlarged, it is possible to create fragments that are scattered across the disk surface. The command `reorgvg` could be used to reorganize the paging spaces.

- Significant imbalance in measured I/O load to physical volumes

  Probably the data is not well distributed throughout the disks. The command `iostat` can give more information about the I/O activity of each disk (refer to 2.2, "The iostat Command" on page 17). A disk should not be utilized more than 40 percent over a period of time. Data should be distributed throughout the disks in a manner to balance I/O. The command `filemon` can give more information about the most accessed files and file systems. This can be a good starting point in order to reorganize the data. Refer to 3.4, "The filemon Command" on page 83 , for more information on `filemon`.

- New process is identified as a heavy memory or CPU consumer

  These processes should be examined for unusual behavior. Note that PDT simply checks the process ID. If a known heavy user terminates, then is executed again (with a different process ID); it will be identified as a new heavy user.

- A file in .files exhibits systematic growth (or decline) in space utilization

  Analyze the problem by identifying which user or process is generating the data.

- A host in .nodes exhibits degradation in ping delays or packet loss percentage

  There is probably a performance problem in the host or in the network.

- A getty process consumes too much CPU time

  A getty process that uses more than just a few percent of the CPU may be in error. Normally the solution is to terminate the process.

- A process with high CPU or memory consumption exhibits systematic growth (or decline) in resource use

  If the growth is unexpected, use `vmstat` and `svmon` commands while the process is running to gather more information on its behavior.

- A WORKLOAD TRACKING indicator shows an upward trend

  There are several workload indicators:

  – loadavg

    Refers to 15 minute load average. In general, it indicates that the level of contention in the system is growing. The rest of the PDT report should be examined for indicators of system bottlenecks.

  – nusers

    Shows that the number of users on the system is growing.

– nprocesses

  Indicates that the total number of processes on the system is rising. Should be checked if there are users achieving maxuproc limitation. Perhaps there are ″runaway″ applications forking too many processes.

– STAT_A

  Number of active processes. Indicates that processes are spending more time waiting for the CPU.

– STAT_W

  Number of swapped processes. Indicates that processes are contending excessively for memory.

– STAT_Z

  Number of zombie processes. Zombies should not stay around for a long time. If the number of zombies on a system is growing, this may be cause for concern.

– STAT_I

  Number of idle processes. This might not be of much concern.

– STAT_T

  Number of processes stopped after receiving a signal. A trend here might indicate a programming error.

– STAT_X

  X is any valid character in the ps command output. The interpretation of a trend depends on the meaning of the character X. Refer to 2.4, “The ps Command” on page 30 , for more information on the ps command.

– cp

  Time required to copy a 40-KB file. A trend in the time to do a file copy suggests that degradation in the I/O subsystem is evident.

– idle_pct_cpu0

  Idle percentage for processor 0. An upward trend in the idle percentage might indicate increased contention in non-CPU resources such as paging or I/O. Such an increase is of interest because it suggests the CPU resource is not being well-utilized.

– idle_pct avg

  Average idle percentage for all processors. See comments above for idle_pct_cpu0.

Severity 3 Problems: Severity 3 messages provide additional detail about problems identified at severity levels 1 and 2. This includes the data-collection characteristics, such as number of samples, for severity 1 and 2 messages.

4.1.3 PDT Error Reporting

Errors can occur within each of the different PDT components. In general, an error does not terminate PDT. Instead, a message is written to PDT’s standard error file (/var/perf/tmp/.stderr), and that phase of processing terminates.

Users experiencing unexpected behavior should examine this file.

4.2 The perfpmr Package

The PerfPMR package was developed to help customers report possible performance problems in AIX by supplying enough information to permit problem diagnosis by IBM. It is intended to ensure that the customer receives a prompt and accurate response with a minimum of time and effort. The package consists of a series of scripts that gather configuration and performance data. The output files are suitable for performance analysts as well.

Suspected AIX performance problems should be reported to the IBM Software Service organization, using the normal software problem-reporting channel. Some general information is required at that time, including a description of the problem, what led to the conclusion that the problem is due to a defect in AIX, the hardware and software configuration, the workload characteristics, and the performance objectives not being met. The data collected with PerfPMR package should also be provided. If it is determined that the problem is due to configuration or improper use, the customer may be transferred to a service organization for (billable) help.

PerfPMR is available in AIX V4 as a fileset of the AIX Base Operating System. For AIX V3 PerfPMR, you should contact your IBM Software Service organization to obtain a copy in suitable medium and installation instructions.

4.2.1 Using the perfpmr Command

Before running the `perfpmr` script, the directory that contains the PerfPMR scripts and executables should be added to the PATH variable:

```
# PATH=$PATH:/usr/sbin/perf/pmr:
# export PATH
```

In AIX V3, the directory in which `perfpmr` was installed (instead of /usr/sbin/perf/pmr) and also the directory for the performance tools, /usr/lpp/bosperf, should be added to the PATH variable. In this version, the `perfpmr` script is named `perfpmr.sh`.

The `perfpmr` command captures more information if the `tprof`, `filemon`, and `netpmon` performance tools are available. In AIX V4, these tools are packaged as part of the Performance Toolbox for AIX. In AIX V3, they come with the Base Operating System.

To run `perfpmr`, the collection period must be specified and must be over 60 seconds. A delay time to wait before collecting starts can optionally be chosen. If the delay is not passed in the command line, `perfpmr` begins the collection right away. Root authority is necessary to execute this script. To track the system activity for one hour run:

```
# perfpmr 3600
```

Although a collection period of one hour was specified, `perfpmr` takes about four minutes more to complete its execution.

This command should be executed during the peek utilization hour. So the output files would reflect the worst case in the environment.

Hint

Prior to a significant change in system hardware or software, run perfpmr during the busiest hour of the day. In this way, you have a performance baseline for comparison after the modification took place.

4.2.2 Output Files

All output files are written to /var/perf/tmp directory. These include both interval data (.int files) and summary data (.sum files). In AIX V3, these files are generated in the current directory.

The perfpmr script is the highest-level script of the package. It starts up other collection scripts. The main script that perfpmr executes is monitor. Some commands called by monitor are run in intervals. The duration of the interval depends on the collection-period time specified. If it is less than 10 minutes, the interval is 10 seconds. For higher collection-period times, the interval is set to 1 minute.

The monitor script runs for the collection-period time to generate the following files:

- monitor.int

  This file is the combined output of the:

  - ps -elk and ps gv commands that runs at the beginning and end of the period.
  - sar -A command (for MP machines the option -P ALL is added), run in intervals.
  - iostat command that runs in intervals. The initial cumulative report is omitted.
  - vmstat command that runs in intervals. The initial cumulative report is omitted.

- monitor.sum

  This file contains:

  - Process information
  - The average from sar -A statistics (for MP machines the option -P ALL is added).
  - The average from iostat statistics
  - The average from vmstat statistics
  - vmstat -s differences from the beginning and end of the period

- nfsstat.int

  The output of the command nfsstat -csnr.

- netstat.int

  This file contains the combined output of:

  - netstat -v, before and after run.
  - netstat -m, before and after run.
  - netstat -rs, before and after run.
  - netstat -s, before and after run.
  - netstat command, run in intervals.

- Pprof.flow

Shows the output of the trace command for process-profile data. Example:

```
ps       25848  28673  9.3124  9.519  0.202  8  26690
ksh      25848  28673  9.2968  9.312  0.012  5  26690
swapper      0      3  0.7847  9.786  0.007  2      0
ps       25850  28675  9.5348  9.740  0.202  8  26690
ksh      25850  28675  9.5187  9.535  0.012  5  26690
```

The columns in this file mean:

1. Process name.
2. Process ID.
3. Thread ID of the calling kernel thread.
4. Time of first occurrence of the process within the measurement period.
5. Time of last occurrence of the process.
6. Total process execution time.
7. Describes the beginning and ending state of the process. This column is the sum of Begin + End values, which are:

   Begin:
   - Exec'd:0
   - Forked:1
   - Alive at start:2

   End:
   - Alive at end:0
   - Exec'd away:4
   - Exited:8

   For example, line 1 for the ps command shows a value 8 in the seventh column. This means that the process was exec'd at the beginning of the collection period and was exited when the period ended. A value of 2 for process swapper shows that it was alive at the start and at the end of the measuring period.
8. Parent process ID.

- Pprof.stt

  Contains the starting and stopping time when the trace command was executed.

Besides the execution of the monitor script, perfpmr calls a series of scripts to generate the following files:

- config.sum

  Information about hardware and software configuration.

- trace.fmt, trace.nm, trace.raw

  These files are the output of the trace command (5 seconds of data collection).

- tprof.sum

  Output of the tprof -k -s command (1 minute of data collection).

- filemon.sum

  Output of the filemon -O all command (1 minute of data collection).

- netpmon.sum

  Output of the netpmon -O all command (1 minute of data collection).

- w.int

  Output of the w command executed at the beginning and the end of perfpmr.

As mentioned before, the total execution time for perfpmr is not only the collection-period time specified in the command line. It includes also the periods for the data collection of the commands mentioned above. For this reason, it will take about four minutes more for perfpmr to complete its execution.

Only one trace session may be active at a given time. So the user must make sure there are no traces running prior to starting perfpmr.

The perfpmr command also generates a file named perfpmr.int, which is a duplicate of the information that comes to the screen:

```
PERFPMR: Data collection started...

     CONFIG_: Generating reports....
     CONFIG_: Report is in file config.sum

     Date and time before data collection is Tue Feb 11 10:37:01 CST 1997
     Uptime information before collection:
       10:37AM   up 6 days,  21:30,  10 users,  load average: 0.38, 0.14, 0.08

     MONITOR: Starting system monitors for 60 seconds....
     MONITOR: Waiting for measurement period to end....
     MONITOR: Generating reports....
     MONITOR: Network reports are in netstat.int and nfsstat.int
     MONITOR: Monitor reports are in monitor.int and monitor.sum

     TRACE_: Starting trace for 5 seconds....
     TRACE_: Trace collected....
     TRACE_: Binary trace data is in file trace.raw
     TRACE_: Trcnm data is in file trace.nm
     TRACE_: /etc/trcfmt saved in file trace.fmt

     TPROF_: Starting tprof for 60 seconds....
     TPROF_: Sample data collected....
     TPROF_: Generating reports....
     TPROF_: Tprof report is in tprof.sum

     FILEMON_: Starting filesystem monitor for 60 seconds....
     FILEMON_: Generating report....
     FILEMON_: Finalizing report....
     FILEMON_: Report is in filemon.sum

     NETPMON_: Starting network trace for 60 seconds....
     NETPMON_: Trace stopped....
     NETPMON_: Generating report....
     NETPMON_: Netpmon report is in file netpmon.sum

     Date and time after data collection is Tue Feb 11 10:41:49 CST 1997
     Uptime information after collection:
       10:41AM   up 6 days,  21:35,  11 users,  load average: 0.27, 0.37, 0.21

    PERFPMR: Data collection complete.
```

For each section, messages appear indicating when collection has started, the monitoring period length and when the reports are being generated. Header and footer information includes the date and time the collection began and ended, plus load average.

After collecting the data, you could first check the monitor.int and monitor.sum files. These files have the most information and should be a good starting point to detect if the problem is related to CPU, memory or I/O. Then look at the other files for more detailed information.

Refer to the previous chapters for further information on the commands mentioned in this section.

4.3 The bf and bfrpt Commands

BigFoot is a useful set of tools to analyze the memory requirements of applications running on AIX. It can be run using executable programs without recompilation. Memory footprints can be obtained for processes, system components (libraries and kernel routines) and subroutines of processes.

The BigFoot utility assists in answering the following questions:

- How many text, data, shared library and file pages are referenced by an application?
- How many pages are referenced from each text segment?
- How many pages are referenced from each subroutine with the program?
- How many pages are shared between subroutines?
- How many pages are shared between processes?

Important Notice

This tool places increased load on the system and should only be used to gather detailed information in a non-production or troubleshooting environment.

BigFoot is only available in AIX V4.

4.3.1 How Does BigFoot Work?

The implementation of the BigFoot utility is tightly coupled to the Virtual Memory Manager (VMM) routines, but the VMM algorithms have been left intact. BigFoot operates by enveloping the page-fault and I/O fault-handling routines of the VMM. A pair of kernel extensions are used to envelop the necessary VMM routines.

In the normal kernel (not running BigFoot), the page faults occur when the process tries to access a page that is not in memory or is not mapped in paging space. In the BigFoot kernel, all pages are unmapped; so each time a process accesses a page in memory, there is a BigFoot fault. The BigFoot handler records in a buffer all the faults and the information linked to this fault, such as which process caused it, when and why. Each time there is a BigFoot fault, BigFoot performs its tasks, but after this, it releases the page. So, if the page is present in the main memory, there is not a real page fault each time. If it is not present, BigFoot releases the page so VMM can see it and allow the page-fault handler to do its job.

4.3.2 Using bf

The bf command turns on BigFoot code in the VMM, specifies the user program to be monitored, executes the user program, collects information about pages touched during execution, and then produces an output file. By default, the output file is __bf.rpt and is created in the current directory. The __bf.rpt file contains either the footprint record for all the processes that were running on the system or just the footprint record for a specified user program. In this case the -p option should be used, so bf will only store records of processes with a

specified name. The user program must be an executable module or an executable script. If the user program does not reside in any directory listed in the PATH variable, then the full path to the user program should be provided.

Root authority is necessary to execute this command and only one instance can be active at a time.

There are two ways of limiting capturing page-trace data:

- BufferSize

  Specifies the number of records to store in the BigFoot kernel buffer. If the buffer size is not large enough, an overflow occurs, and the following message is given at the end of the execution of `bf`:

  ```
  bf: A buffer overflow occurred while tracing.  You may want
          to try again with a larger buffer size (-b option)
  ```

  The default buffer size is 5000, which is usually not adequate. So you may want to increase it when running `bf`. When the buffer overflows, the information from the `bf` command is meaningless.

- WindowSize

  The window size causes BigFoot to capture repeated references to the same page, but only logs the reference if the address of the page is not already within the specified window of page references. This parameter can be set anywhere between 2 and 100.

  For example, given a window size of 3, and pages A, B, C and D referenced in the following sequence A B A C D A, the steps performed are:

  1. The contents of the window is initially empty.

     [* * *]

  2. Page reference A is captured and logged.

     [* * A]

  3. Page reference B is captured and logged.

     [* A B]

  4. Page reference A is captured but not logged because A is already in the window.

     [* A B]

  5. Page reference C is captured and logged.

     [A B C]

  6. Page reference D is captured, bumps A out, and is logged.

     [B C D]

  7. Page reference A is captured, bumps B out, and is logged

     [C D A]

  Therefore, the page-trace contains ABCDA not ABACDA. The second reference to A is not logged because A was within the window at the time of the reference. However, the third reference to A is logged.

Note that the use of the `WindowSize` parameter does not cancel the use of the `BufferSize` parameter. Both parameters can be passed in the command line.

Sometimes, only giving a window size cannot prevent a buffer overflow, but can delay it.

When specifying the parameter WindowSize, remember that you will probably loose some data, but at least the algorithm is known. When the buffer overflows, part of the trace is definitely lost, and there is no specified user criteria for which data will be discarded. To gather all possible data, make the buffer size big enough to capture the whole trace. Depending on the processes running on the system when bf is executed, you might have even to increase the paging space to accommodate the proper buffer size.

To run the eatmem process and monitor only its page references until the process completes, enter:

```
# bf -b 220000 -p eatmem -x eatmem
```

The -b flag sets the buffer size to 220000, -p instructs the bf command to filter out all processes that are not named eatmem, and the -x flag specifies the process to run. The -x flag is required and must be the last argument on the bf command line. The program is forked as a child process. The bf command continues collecting page-reference data until the child program ends or the buffer overflows.

To fork the child process sleep 10, wait until the sleep completes before turning off the page monitoring, and record the output in the file bigfoot.out, enter:

```
# bf -W 50 -o /bfdata/bigfoot.out -x sleep 10
```

In the above example, the -W flag was used to set the window size to 50.

Depending on the parameters used to execute the command bf (specially if the buffer size is big), the trace files can get very large. Make sure there is enough space in the file system. If the bf command does not have space to generate the raw file, the following message will be displayed at the end of the execution:

```
Error encountered while turning off bigfoot, exiting.
```

If this happens, increase the size of the file system, and rerun the command. Another option is to use the -p and/or -W flags to reduce the trace file size.

4.3.3 Generating Reports with bfrpt

The bfrpt command is used to format the raw data generated by the bf command into a report. It filters bf files for a global state or for a single process, depending on the type of report specified.

The reports presented in this section were generated based on the output of:

```
# bf -b 1200000 -x eatmem
```

Root authority is required to run this command.

Global Reports: To generate global reports, the bfrpt command uses the global filter to provide a comprehensive view of the system.

To generate a global report, enter:

```
# bfrpt -r global
```

The report is written to __global.rpt and is created in the current directory. The examples shown below are sections of the output of the command above.

The global report consists of:

- Header section

  Summarizes the usage of the 16 segments in a process for reference. Example:

```
Segment Register Usage
   0 : Kernel(text & data)
   1 : Process private text
   2 : Process private data
3-12 : Currently Addressable Files
  13 : Shared Library text
  14 : Kernel
  15 : Shared Library data
```

  This section is a brief reminder for the user about the properties of segment registers. AIX maps 32-bit virtual memory addresses into 16 segments. Four of the 32 bits select one of the 16-bit segment registers. The remaining 28 bits give an offset within the segment. Each segment register contains a 24-bit segment ID, which, when prefixed to the 28-bit segment offset, forms a full 52-bit virtual address.

  Non-kernel programs have read-only access to the two kernel segments, 0 and 14, and the shared library segment, 13.

  Segment 1 contains the executable of the current program. Other instances of the same program share the read-only contents of segment 1.

  Segment 2 contains the private read/write data-space of the current program. Program variables, allocated memory, and the program stack come from the private data-segment of the process. In rare instances, library text resides in segment 2.

  Segment 13 is readable by all users. Therefore, shared libraries with file permissions that do not include read access for all other users on the system are not loaded in segment 13. AIX loads libraries without global read access in the private-data segment of each process that uses them.

  Segment 15 contains the private, shared library data for a process.

  The ten remaining segments, 3 through 12, provide memory access to files and shared memory segments. The system maps files into virtual memory segments when a file is first opened.

  Note

  Though the header information is accurate for most AIX processes, the system can use some segment registers for different purposes. Use the header information as a guideline only.

- Process name table

  Tabulates the number of pages referenced in various segments by the process name. If the bfrpt command reports two or more processes with the same process name, it treats them logically as one process in the process name table. The rows are sorted by the number of referenced pages and show only the top memory consumers. Example:

```
*****************************************************************

*                                                               *

* Number of Pages Referenced by <Process Name>                  *

*                                                               *

*****************************************************************
PNAME (20 chars)          TOTAL      0      1      2   3-12     13     14     15

          eatmem           5240     68      1   5126      0     22      0     23
             ksh            484    227     39     95      8     37      0     78
            init            110     50      8     36      2      8      0      6
         telnetd             38     22      3      5      0      4      0      4
         syslogd             23     10      4      4      0      2      0      3
            afsd              5      5      0      0      0      0      0      0
            wait              1      1      0      0      0      0      0      0
```

The first column shows the process name. Next, the total number of referenced pages is shown, which is a sum of the remaining columns. The column names 0-15 refer to the segment numbers given in the header section.

By analyzing the TOTAL column in the example above, it is clear that the eatmem process is memory bound, and its code should be carefully analyzed. The number of pages referenced by this process is almost eight times the number referenced by all the other processes together.

- Process name/process ID table

  Provides information about the number of pages referenced by process name and ID. Because AIX can reuse process IDs, they are combined with process names to distinguish all processes in the report. Example:

```
*****************************************************************

*                                                               *

* Number of Pages Referenced by <Process Name, PID>             *

*                                                               *

*****************************************************************
PNAME (20 chars)    PID  TOTAL      0      1      2   3-12     13     14     15

          eatmem  15978   5240     68      1   5126      0     22      0     23
             ksh  15980    333    193     29     37      1     33      0     40
             ksh  15978    277    155     23     36      1     25      0     37
             ksh  14668    189     93     37     22      6     16      0     15
            init      1    110     50      8     36      2      8      0      6
         telnetd   5014     38     22      3      5      0      4      0      4
         syslogd   3104     23     10      4      4      0      2      0      3
            afsd   7058      5      5      0      0      0      0      0      0
            wait    516      1      1      0      0      0      0      0      0
```

  The output fields are the same as in the first example, except for the process ID (PID) that was included in this table.

  Notice that the ksh line from the previous example was split into three lines in this table. As you can see only the columns labeled 2 and 3-12 add up to the values from the previous table. These columns are related to the process private area. The remaining columns give information about the kernel and shared memory areas. For multiple instances of a program (in this case ksh), the executable code just has to be paged in once. There are also common areas in the memory that are shared among the processes. In this example the shared pages (pages referenced by more than one process) are added in every line, and in the previous example they are only considered once.

- Unique pages table

  Tabulates the total number of unique pages. A unique page is a page referenced by only one process. The system examines the full 52-bit virtual address, not just 32 bits, to determine the uniqueness of a page. Example:

```
************************************************************

*                                                          *

* Number of Unique Pages Referenced by <Process Name, PID>  *

*                                                          *

************************************************************
PNAME (20 chars)    PID  TOTAL

          eatmem  15978   5135
             ksh  15980    115
             ksh  14668     62
             ksh  15978     36
            init      1     65
         telnetd   5014     12
         syslogd   3104     12
            afsd   7058      4
            wait    516      0
```

  The first column shows the process name. Next, the PID of the process and the total number of unique pages referenced by the process are shown. The total number of unique pages referenced is much higher for the process `eatmem` when compared to the others.

If the command `bfrpt` is executed with the `-p` option, the report is recorded in a PostScript file:

```
# bfrpt -r global -p
```

The file will be written to the current directory as __global.ps and is displayed in Figure 7 on page 150.

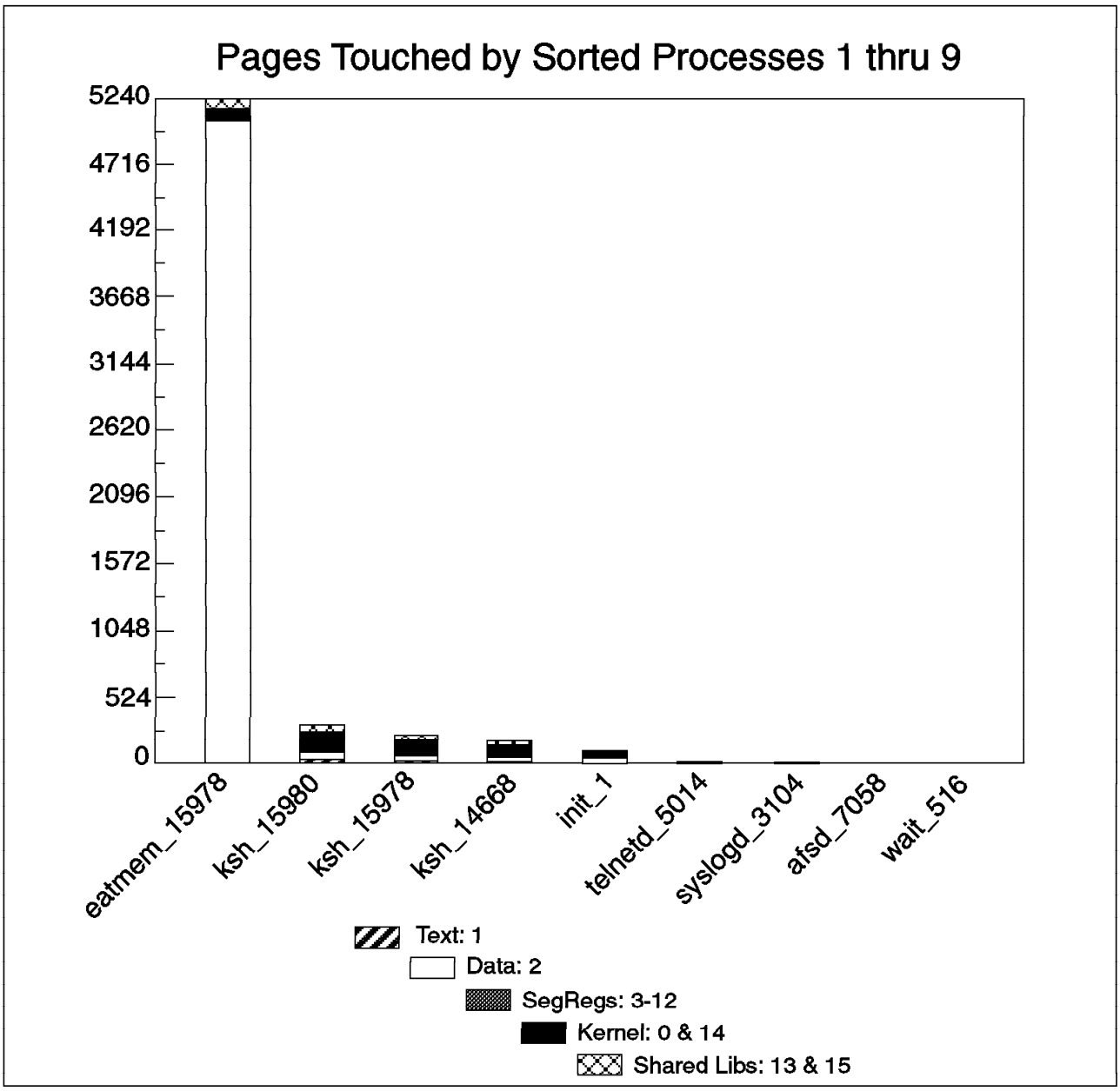


Figure 7. Output File __global.ps

act Reports: The act report provides information about the footprints of each routine in a process. In this case, the bfrpt command uses the address correlation technology filter, or act filter, to analyze data with emphasis on a single process. The act filter translates a system, library, or process address into a symbolic routine name.

To generate an act report based on the eatmen process, enter as root:

```
# bfrpt -r act -P eatmem
```

The report is written to a file named __<process_name>.<process_ID>.rpt in the current directory. It may take a long time for bfrpt to generate the final report. The examples that will be shown are sections of the output of the command above.

The act report has four logical sections:

- Total footprint

  This section is divided into three tables. The first table contains the footprints for all routines other than the shared library and kernel routines. The second table contains footprints for the shared library routines that the main program uses. The third table contains the footprints for the kernel routines that operate on behalf of the process. Example:

```
               ********** Total Footprint ******************

*********** ROUTINES    Start      End    SUM Kernel   Text   Data   File ShrLib  IOdev
             .getmem      26c      317      3      0      1      2      0      0      0
               .back      318      3d3      3      0      1      1      0      0      1
               .exit      580      5a3      3      0      1      1      0      0      1
           .__flsbuf      5a4      5c7      3      0      1      1      0      0      1
             .printf      5ec      60f      3      0      1      0      0      0      2
               .main      3d4      537      2      0      1      1      0      0      0
             .malloc      538      55b      2      0      1      0      0      0      1
               .free      55c      57f      2      0      1      0      0      0      1
             .fflush      610      633      2      0      1      0      0      0      1
             .usleep      634 ffffffff      2      0      1      0      0      0      1
            .__start      1c8      23f      1      0      1      0      0      0      0
                                 TOTAL NUMBER OF PAGES     26

*********** LIBRARIES    Start      End    SUM Kernel   Text   Data   File ShrLib  IOdev
          .malloc_y     3f80     4480  2728      0      0   2723      0      2      3
    .disclaim_free_y    1fa8     2af8  2563      0      0   2561      0      1      1
             ._doprnt    6184     bf44    13      0      0      2      0      4      7
           .__flsbuf     4efc     510c     6      0      0      1      0      2      3
             .free_y     38cc     3f80     6      0      0      3      0      1      2
             .printf     5504     5590     5      0      0      1      0      1      3

(...)
                                 TOTAL NUMBER OF PAGES 5388

*********** KERNEL    Start      End    SUM Kernel   Text   Data   File ShrLib  IOdev
 loader_sect_start   15dbec ffffffff     13     13      0      0      0      0      0
         isync_sc2     3734     37a0      5      5      0      0      0      0      0
          .jfs_cntl   119818   119de0      5      5      0      0      0      0      0
      .ld_usecount   10d328   10d96c      4      2      0      1      0      0      1
          ._crfree    e663c    e66a4      4      4      0      0      0      0      0
       .copyin_pwr    7f8c0    7fa40      3      1      0      1      0      0      1

(...)
                                 TOTAL NUMBER OF PAGES 136
```

  It can be noticed in the SUM column in the second table from the above example that the library function malloc_y is extensively allocating memory, and the library function disclaim_free_y is being used to free this memory and put it back in the free list. At least the program seems not to be generating a memory leak. But the possibly wrong use of malloc_y should be checked in the source code.

- Footprint before start

  Provides footprint for the kernel routines that operate prior to the start of the main program. Example:

```
               ********** FootPrint Before Start ************

*********** KERNEL    Start      End    SUM Kernel   Text   Data   File ShrLib  IOdev
           .xmfree    da5c4    da8a4      3      3      0      0      0      0      0
           .ulimit   125758   1260bc      2      2      0      0      0      0      0
       .copyin_pwr    7f8c0    7fa40      2      1      0      1      0      0      0
      .copyout_pwr    7fa40    7fbc0      2      1      0      1      0      0      0
        .exectrace    818dc    82604      2      2      0      0      0      0      0
      .simple_lock     9500     9900      2      2      0      0      0      0      0
         .fp_close    ec2ec    ec3b0      1      1      0      0      0      0      0
                                 TOTAL NUMBER OF PAGES     14
```

- Footprint after stop

  Provides a footprint for the kernel routines that operate after the stop of the main program. Example:

```
            *************** FootPrint After Stop **************
*********** KERNEL    Start      End    SUM Kernel   Text   Data   File ShrLib  IOdev
         .jfs_cntl   119818   119de0      5      5      0      0      0      0      0
      .ld_usecount   10d328   10d96c      4      2      0      1      0      0      1
          ._crfree    e663c    e66a4      4      4      0      0      0      0      0
        .vnop_rele    e70bc    e710c      3      3      0      0      0      0      0
           .chownx    ec0bc    ec2ec      3      3      0      0      0      0      0
          .acctexit   121e80   1221d4      3      3      0      0      0      0      0
     .fp_opencount    ecaa0    ecc90      2      2      0      0      0      0      0
          .fs_fork    ec7b4    ec9bc      2      2      0      0      0      0      0
       .vnop_unmap    e6c64    e6cb4      2      2      0      0      0      0      0
(...)
                                    TOTAL NUMBER OF PAGES 52
```

- Unique data footprint

  Identifies footprints for unique data segments in the routines of the main program. If two or more routines touch the same page, the page is shared, not unique. Example:

```
             *************** Unique Data Footprint **************
             **** (Routines ONLY, Not Kernel, Not Shr. Lib.) ****

*********** Routines    Start      End       Data     Unique      Difference
             .getmem      26c      317          2          0               2
               .back      318      3d3          1          1               0
               .main      3d4      537          1          0               1
               .exit      580      5a3          1          1               0
           .__flsbuf      5a4      5c7          1          0               1
```

  Note that this table gives only information about the routines. The kernel and shared libraries are not included.

All Reports: To produce both tabular form reports, enter:

```
#bfrpt -r all
```

The global and act reports will be generated in the current directory.

4.4 The stem Command

The meaning of stem is Scanning Tunneling Encapsulation Microscope. This performance tool provides the user with the vehicle to insert user-defined or standard instrumentation at the entry and exit points of selected subroutines, known as target routines. These routines may be within a user program or even within a shared library.

The instrumentation routines are simple C subroutines that may be created or tailored by the user. Sample instrumentation routines are located in the file /usr/samples/perfagent/stem/stem_samples.c, including the default instrumentation routines used by stem. These can be modified to suit specific requirements. Three types of instrumentation are possible:

- Instrument the entry point and exit point
- Instrument only the entry point
- Replace the routine completely

The stem command can instrument application programs that are:

- Stripped
- Optimized

- Running in multiple processes
- In unstripped shared libraries

This tool is only available in AIX V4.

4.4.1 Using stem

Root access is required to execute stem. In AIX V4.2, members of the perf group are also able to run this command. The simplest way to execute stem is by using the -p option:

```
# stem -p /usr/bin/cat
```

This option instructs stem to instrument all target routines of the program /usr/bin/cat with the default routines Stem_Standard_entry() and Stem_Standard_exit(). The -p flag copies the stem_samples.c file to the current directory, unless it already exists. The instrumented version of the program is generated in the /tmp/EXE directory. To specify another directory, use the -exedir option. When the instrumented program is executed, it creates a file called stem_out_xxx (where xxx is the stem thread ID) in the current directory. This is the output file generated after the execution of the instrumented program:

```
# /tmp/EXE/cat
# cat stem_out_001
Seconds.usecs  TID  Routine Names & Seconds.usecs since entering routine.
856218530.325752   1 ->main
856218530.364687   1    ->setlocale
856218530.367411   1    <-setlocale    0.002724
856218530.367613   1    ->catopen
856218530.368368   1    <-catopen      0.000755
856218530.368566   1    ->fcat
856218530.368689   1            ->fstat
856218530.368851   1            <-fstat        0.000162
856218530.368976   1            ->open
856218530.369203   1            <-open         0.000227
856218530.369341   1            ->fstat
856218530.369472   1            <-fstat        0.000131
856218530.369591   1            ->read
856218530.369798   1            <-read         0.000207
856218530.369920   1            ->write
856218530.372705   1            <-write        0.002785
856218530.372914   1            ->read
856218530.373067   1            <-read         0.000153
856218530.373182   1            ->close
856218530.373331   1            <-close        0.000149
856218530.373527   1            ->fclose
856218530.373689   1            <-fclose       0.000162
856218530.373809   1            ->exit
```

The first column contains the time (seconds and microseconds) of the enter or exit event. The second column has the stem thread ID (TID). The indented routine name, prefixed with either the routine entry symbol (->) or the routine exit symbol (<-), appears to the right of the TID column. The indentation is meant to reflect the calling sequence or callgraph of the instrumented program. For exit events, one additional column appears and includes the elapsed time (second and microseconds) since entering the routine.

The callgraph above shows the structure of an AIX command. The calls to setlocale and catopen ensure that the command process is running in the same

National Language Support (NLS) locale and with the same message catalog as its parent process. You might have some problems when using `stem` with system executables. However, this tool is intended to be used in a development environment as a way to detect performance problems in user programs.

The command `stem` creates a subdirectory ./stemdir under the current directory. Under the stemdir directory, other additional subdirectories are created, one for each executable or library to instrument. The previous example would cause `stem` to create the directory ./stemdir/_usr_bin_cat. Notice that each slash (/) passed in the command line was substituted with an underline character (_). Two files in this created directory are of particular interest: instrumented and not_instrumented. The former contains the list of target routines instrumented by `stem`. The latter contains the list of target routines not instrumented and a brief explanation of the problem encountered.

Special instrumentation libraries created for stripped programs are generated in the /tmp/LIBS directory or in the directory specified by the `-libdir` flag. For example, the program `/usr/bin/cat` from the previous example is stripped. After the execution of `stem`, a subdirectory named cat was created under /tmp/LIBS to store the instrumented libraries generated by `stem`.

4.4.2 Shared-Memory Callgraphs

This approach is similar to the `-p` flag. The difference is in the instrumentation and the output mechanism. In this case, the specified program is instrumented with the `Stem_ShmEnter()` and `Stem_ShmExit()` default routines. These routines do not open files to log output; they log output in a specially-made, shared-memory buffer.

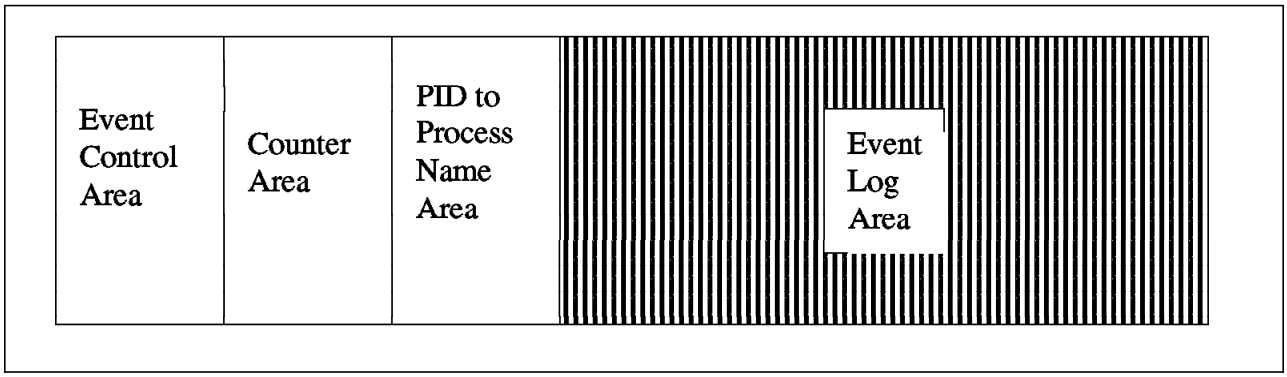


Figure 8. Shared-Memory Buffer Structure

Figure 8 shows the structure of the shared-memory buffer:

- Event Control Area

  Includes the ON/OFF flag, pointers defining the boundaries of other areas, and the WithinInstrumentation flag. This flag determines the present instrumentation state and prevents infinite instrumentation loops. These loops result from instrumenting a target routine and then directly or indirectly calling the target routine from the instrumentation routine, for example, calling the `printf` function in the instrumentation routine code that surrounds the `printf` function. In cases like this, `stem` still generates the instrumented executable, but would not be locked in an infinite loop. If you try to run the instrumented program, it will terminate with a segmentation fault.

- Counter Area

  Reserved for a counter-based implementation. For example, rather than producing an event for each file open, instrumentation routines can add to a counter or set of counters.

- PID to Process Name Area

  This area contains the process IDs and process names for all processes running on the system at the time of the last stem -on call.

- Event Log Area

  Logs events of any type and any length. The stem command defines some event types in the /usr/samples/perfagent/stem/stem_shm.h file.

The following is an example sequence of instructions using the shared-memory flags:

```
# stem -pshm dbapp
# stem -on
# /tmp/EXE/dbapp
# stem -cg /tmp/EXE/dbapp
# stem -shmkill
```

The -pshm flag causes stem to instrument the program with the shared-memory routines. The stem_samples.c file is copied to the current directory unless it already exists.

The logging of events can be controlled with the -on, -off and -noreset options. Logging of events only occurs if the on/off flag of the memory buffer is set to on and the buffer is not full. When created, the current pointer of the buffer is set to full and the on/off flag is off. The -on flag resets the buffer pointer and turns on the on/off flag. It also stores the process IDs and process names of all running processes in the shared-memory buffer. In case you do not want to reset the buffer pointer, the flag -noreset should be used in conjunction with the -on option. The -shmkill option destroys an existing shared-memory segment and disables logging. This flag causes stem to ignore other options that might have been passed in the command line.

The size of the shared-memory buffer can be increased with the -shm flag. The default size is 40960 bytes. Care should be taken when using this flag because stem pins the shared-memory buffer. So make sure to make the buffer size is not too large.

The -cg produces a shared-memory callgraph to stdout:

```
  PID         ElapsedTime    DeltaSecs    IAR    NAME
 17038 Enter    0.000000     0.000000  100004b0   1  main
 17038 Enter    0.033452     0.033452  10000684   1 . signal
 17038 Exit     0.083022     0.049570  10000684   1 . signal
 17038 Enter    0.083139     0.000117  100006a8   1 . alarm
 17038 Exit     0.089919     0.006780  100006a8   1 . alarm
 17038 Enter    0.090040     0.000121  100006cc   1 . select
 17038 Exit     1.090289     1.000249  100006cc   1 . select
 17038 Enter    1.091789     0.001500  100006cc   1 . select
 17038 Exit     2.092053     1.000264  100006cc   1 . select
 17038 Enter    2.093552     0.001499  100006cc   1 . select
 17038 Exit     3.094432     1.000880  100006cc   1 . select
 17038 Enter    3.095932     0.001500  100006cc   1 . select
 17038 Exit     4.096196     1.000264  100006cc   1 . select
 17038 Enter    4.097696     0.001500  100006cc   1 . select
```

The output is different from the previous example, but the information is almost the same. The first column contains the PID of the process. Next is shown if the program entered or exited from a routine. ElapsedTime gives the total time spent since the program started. DeltaSecs gives the time spent until the program either entered the next routine or the total time spent in that routine on exit. IAR stands for Instruction Address Register and can usually be disregarded. The next column has the stem thread ID (TID). The last column shows the name of the routine in an indented form.

The dbapp program simulates a database activity. Notice that each select takes about one second to complete. In a real environment, the time spent in each instruction would be different because it depends on the query complexity. If in your system the execution time for a specific instruction is high, the query structure should be verified. Maybe it can be modified to run faster, or the database might not be well designed.

It can be noticed from this example that stem is very useful for development. Through the time spent in the execution of each routine, it can be seen where the program has hot spots and might be improved.

If you want to execute another instrumented program and collect only the information related to this program, make sure you first run the command stem -on. In this way, the pointers will be reset, and the output will show only the data associated with the last command.

The use of the -pshm parameter for stem has some advantages over the -p option:

- Logging data within the shared-memory buffer is faster than writing to files.
- Logging together of events from multiple instrumented processes in one output stream is possible.
- The ON/OFF shared-memory buffer flag readily controls the logging of events.

The disadvantage is post processing. To view the data, stem has to be executed again with the -cg flag.

4.4.3 Stem Map File

The -p and -pshm flags explained in the previous sections are used to instrument a program with the standard instrumentation routines. The map file (-mf) parameter specifies a user-created file whose contents describe which target routines and/or shared libraries to instrument. Map files contain the names of target routines and of the entry and exit instrumentation routines.

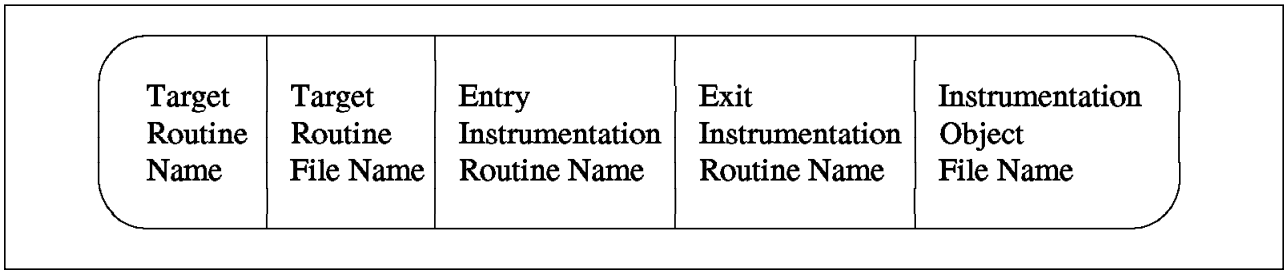


Figure 9. Map File Format

Figure 9 shows the format of the stem map file. The first field is the target routine to be instrumented. The name of the file that contains the routine should

be given next. The two following columns give the names of the instrumentation routines to be executed before and after the target routine. The last field is the file name where the instrumentation routines reside.

To avoid name collisions with routines in the target files, stem mandates that all instrumentation routines begin with the Stem_ prefix. Instrumentation object files in column 5 can have any name, but they should be within the current directory unless when using the default stem_samples.o file. If the file Make.Stem exists in the current directory, stem tries to create the instrumentation object by running the command:

```
make -f Make.Stem <Intrumentation_Object_File_Name>
```

Example of a map file:

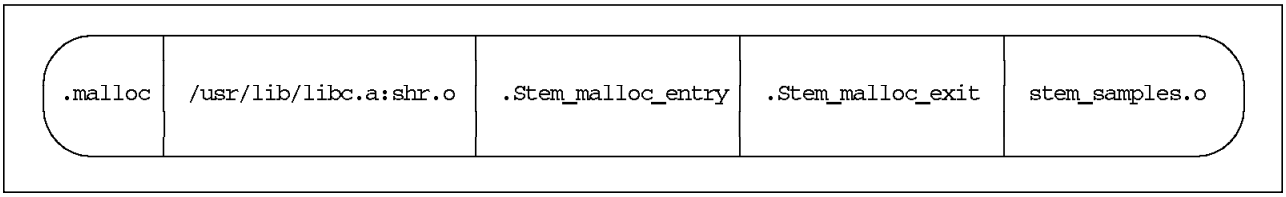


Figure 10. Map File Example

The map file shown in Figure 10 changes the program flow of target routine malloc(). After instrumentation, calls to target routine malloc() are directed first through instrumentation routine Stem_malloc_entry(), then through malloc(), and finally through the Stem_malloc_exit() instrumentation routine. These two instrumentation routines mentioned in the example are given in the file stem_samples.c. This file contains examples for basic instrumentation routines.

Instrumented versions of libraries are created in the directory /tmp/LIBS by default, unless the -libdir option specifies another directory. Both instrumented and uninstrumented programs can make use of these libraries by setting the LIBPATH environment variable. For example:

```
# LIBPATH=/tmp/LIBS /usr/bin/eatmem
```

The uninstrumented program /usr/bin/eatmem will use whatever instrumented libraries exist in the directory /tmp/LIBS.

There are three keywords that can be used in map files:

- StemAll

  Used to instrument all routines in a program or in a shared library. Examples:

  ```
  StemAll /bin/test_prog .Stem_ProgEnter .Stem_ProgEnd function.o
  StemAll /usr/lib/libc.a:shr.o .Stem_ShmEnter .Stem_ShmExit stem_samples.o
  ```

  The first line instructs stem to instrument all routines in the program test_prog with the instrumentation routines Stem_ProgEnter() and Stem_ProgEnd() located in function.o. The second line causes all target routines in the library /usr/lib/libc.a:shr.o to be directed through the default shared-memory instrumentation routines.

- No_Exit

  Used to instrument only the entry point of a target routine. Example:

  ```
  .main /bin/test_prog .Stem_ProgEnter No_Exit function.o
  ```

The above example would cause the entry point of routine main() to be instrumented with the routine Stem_ProgEnter(), but the exit point would not be instrumented.

- Replace

  Used to replace a target routine with an instrumentation routine. Example:

  ```
  .func1 /bin/test_prog .Stem_newfunc1 Replace function.o
  ```

  This example causes all calls to func1() to be directed to the instrumentation routine Stem_newfunc1().

  Note

  Target routine replacement is inherently dangerous. To avoid unpredictable results, the replacement routines must adhere to all pre-conditions and post-conditions of the target routines. For example, you can replace a sorting routine with another as long as the output is properly sorted upon exiting the replacement routine.

To instrument a program using a map file, enter:

```
# stem -mf <name_of_map_file>
```

The output in this case will depend on the code used for instrumentation. If the routines Stem_Standard_entry() and Stem_Standard_exit() are used, after running the instrumented program, the file stem_out_xxx will be generated. In case of instrumenting a program with the default shared-memory routines Stem_ShmEnter() and Stem_ShmExit(), the steps mentioned in 4.4.2, "Shared-Memory Callgraphs" on page 154, should be followed.

The map files approach makes stem a very powerful tool. The default behavior of stem can be very helpful, but if specific information is needed, you need to code your own routines. To accomplish this, knowledge of what the program does, how it does it, and C function programing is necessary.

The stem command has been tested on a number of programming languages, like C, C++ and FORTRAN, although C has received the most testing. FORTRAN programs have one restriction. The FORTRAN programming language supports multiple entry points to subroutines. The stem command cannot currently detect multiple entry points. Therefore, the stem output cannot properly represent entry and exit events from these FORTRAN routines.

The stem command does not work on non-archived libraries and does not support programs or libraries with more than 6200 subroutines. User-created threads are not yet supported by stem, and the result is a segmentation fault.

4.5 The syscalls Command

The syscalls command has the ability to trace system calls for all processes running on the system, or just the ones associated with a particular program. It can also maintain counts for all system calls made over long periods of time. The events are logged in a shared-memory buffer. The syscalls command does not use the trace daemon.

Caution

Since the commands stem and syscalls share the same buffer, do not execute them at the same time.

Capabilities of syscalls:

- Real-time trace of system calls
- Timestamps on a per-second basis
- Ability to see all calls or filter out calls by process ID (PID)
- Executable not modified

Differences from trace:

- trace sees all kernel events
- trace timestamps to nanosecond granularity

This tool is only available in AIX V4.

4.5.1 Using syscalls

Root authority is necessary to run this command. In AIX V4.2, the members of the perf group are also able to execute syscalls.

Some steps are necessary to set up the environment used by the syscalls command. The first thing is to create the shared-memory buffer. Certain options of syscalls automatically create the buffer, if it does not exist. However, the buffer may already exist because it was created by the stem command, for instance. If you try to collect data with syscalls, then no events will be logged. You need to recreate the buffer.

The following is a description of the options used to manipulate the shared-memory buffer:

-enable <bytes>

Creates the system call trace buffer. If the bytes option is not specified, the buffer size gets to be 819200 bytes, which is the default size.

It should always be used if events are not being logged in the buffer. This happens when there is a conflict with another process using the same shared memory buffer ID. As stated previously, if you use stem with the shared memory instrumentation routines, you must use this option prior to initiating syscalls data collection.

Some other options of syscalls automatically enable the buffer. However, by using this flag, the buffer size can be altered. Also, when no events are being logged into the buffer, these other options will not revert the situation. This happens because they only create the buffer if it does not exist. If there is a buffer, it will not be re-created. A good policy is to always use the -enable option before starting data collection. This option should be used any time you are not sure that the stem command has been used.

Again, this option does not start data collection; it only (re)creates the buffer.

-disable

Destroys the system call buffer and disables system call tracing and counting.

-start

Resets the trace buffer pointer. This option enables the buffer if it does not exist, resets the counters to zero and starts tracing.

This flag only creates the buffer if it does not exist. But if the buffer was already created by another command, the -start option will only reset the counters and syscalls will not be able to log events.

-stop

Stops the logging of system-call events and prints to stdout the contents of the buffer. In case a buffer overflow occurred, the following message will be given:

```
Syscall Trace Buffer Overflow
```

When this happens, some events were lost, and you should increase the buffer size.

The handling of system calls is very time and resource consuming to the system. When a program starts a system call, the system-call handler gains control. It then changes the protection domain from the caller (user) to the system call protection domain (kernel) and switches to a protected stack. Next, the system-call handler calls the function supporting the system call. After the function performs its operation, the system-call handler restores the state of the process and returns to the user program.

4.5.2 Examples

To trace system calls for all processes on the system, execute the following commands:

```
# syscalls -enable
# syscalls -start
# syscalls -c
```

The first command (re)creates the system call trace buffer. The next command enables data collection. The -c flag prints a summary of system-call counts to stdout:

```
System Call Counts for all processes
      12203      .lseek
       5486      .kreadv
        297      .sigaction
        134      .sbrk
        111      .close
         74      .kioctl
         72      .kwritev
         60      .open
         55      .kfcntl
         45      .kwaitpid
         39      .select
         31      .getpid
         22      .execve
         20      .statx
         20      .getuidx
(...)
```

The output shows the total number of each system call issued by all processes sorted in decreasing order of usage. The syscalls command can maintain counts for all system calls made over long periods of time. However, if syscalls is executed with the options to manipulate the buffer or stem is used with the shared memory routines, the data will be lost.

To run a program and log only the events referred to that process, execute:

```
# syscalls -x cpubound
PID   System Call
22424            .open  (/dev/null, 200002e0, 0) = 4
22424          .kreadv  (4, 2ff22278, 1, 0) = 0
22424          .kreadv  (4, 2ff22278, 1, 0) = 0
22424          .kreadv  (4, 2ff22278, 1, 0) = 0
22424          .kreadv  (4, 2ff22278, 1, 0) = 0
22424          .kreadv  (4, 2ff22278, 1, 0) = 0
22424          .kreadv  (4, 2ff22278, 1, 0) = 0
22424          .kreadv  (4, 2ff22278, 1, 0) = 0
22424          .kreadv  (4, 2ff22278, 1, 0) = 0
22424          .kreadv  (4, 2ff22278, 1, 0) = 0
22424          .kreadv  (4, 2ff22278, 1, 0) = 0
(... skip 4000 lines ...)
22424          .kreadv  (4, 2ff22278, 1, 0) = 0
22424          .kreadv  (4, 2ff22278, 1, 0) = 0
22424          .kfcntl  (1, 3, 2) = 2
22424          .kfcntl  (2, 3, 6) = 2
```

In this case, the level of detail you can see in the output includes the process ID (PID) of the program that was executed, the system call name, the system call parameters (in parenthesis), and the return code (after the equal sign). The option -x automatically enables the buffer if necessary, resets the pointers and starts data collection for the specified process. If the program is not within the directories of the PATH variable, the full path must be given.

The output shown in the above example was truncated, but still shows the intensive use of system calls by the program. As explained, the use of system calls is very expensive to the system. The program given in the example should be checked. You should try to minimize the use of system calls to achieve better performance.

The -t option prints the time associated with each system-call event alongside the event:

```
# syscalls -t -x cpubound
Time       PID   System Call
17:05:40   22424            .open  (/dev/null, 200002e0, 0) = 4
17:05:40   22424          .kreadv  (4, 2ff22278, 1, 0) = 0
17:05:40   22424          .kreadv  (4, 2ff22278, 1, 0) = 0
(...)
17:05:40   22424          .kreadv  (4, 2ff22278, 1, 0) = 0
17:05:40   22424          .kreadv  (4, 2ff22278, 1, 0) = 0
17:05:40   22424          .kfcntl  (1, 3, 2) = 2
17:05:40   22424          .kfcntl  (2, 3, 6) = 2
```

When used in conjunction with -c flag, the -t flag is ignored. This means that you do not get the time column.

To redirect the output to a file, use the -o option:

```
# syscalls -o syscalls.out -x ls
```

Notice that the option -o when used in conjunction with -x, must be passed first in the command line. Otherwise, the output will be directed to stdout. The -x flag has to be the last one because after this flag the command to be executed is given with a variable number of parameters.

4.6 The fdpr Command

The meaning of fdpr is Feedback Directed Program Restructuring. The fdpr command is a performance-tuning utility that can improve both performance and real memory utilization of user-level application programs. The source code is not necessary as input to fdpr. However, stripped executables are not supported.

The fdpr tool reorders the instructions in an executable to improve instruction cache, Translation Lookaside Buffer (TLB), and real memory utilization by packing together highly executed code sequences (as determined through profiling) and by recoding conditional branches to improve hardware branch prediction.

For example, given an "if-then-else" statement, fdpr may conclude that the program uses the else branch more often than the if branch. It will then reverse the condition and the two branches as shown in Figure 11.

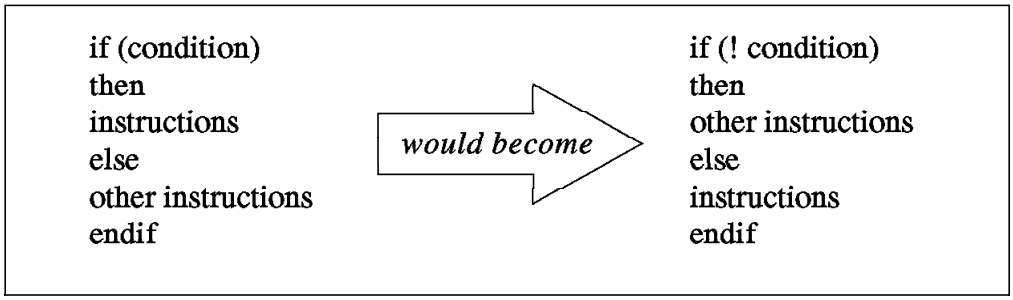


Figure 11. Example of Conditional Branch Re-Coding

Programs can improve execution time up to 73 percent, but typically the performance is improved between 10 and 20 percent. The reduction of real memory requirements for the text pages can reach 61 percent. The average is between 20 and 30 percent.

The optimized program is built in three stages:

1. The executable module to be optimized is instrumented to allow detailed performance-data collection.
2. The instrumented executable is run in a workload provided by the user, and performance data from that run is recorded.
3. The performance data is used to drive a performance-optimization process that results in a restructured executable module.

Attention

The fdpr command applies advanced optimization techniques to a program which may result in programs that do not behave as expected; programs that are reordered using this tool should be used with caution and should be rigorously retested with, at a minimum, the same test suite used to test the original program in order to verify expected functionality. The reordered program is not supported by IBM.

It is critically important that the workload used to drive fdpr closely match the actual use of the program. The performance of the restructured program with workloads that differ substantially from that used to drive fdpr is unpredictable, but can be worse than that of the original executable. The fdpr user should also attempt to eliminate, where feasible, any time-dependent aspects of the program.

This tool is only available in AIX V4. A Programming Request for Price Quotation (PRPQ) is available for AIX V3.

4.6.1 Using fdpr

The typical usage of fdpr is:

```
# fdpr -p testprog -x testprog.sh
```

The -p flag specifies the program to be optimized, which should be within the current directory. Otherwise, the full path must be given in the command line. The -x option specifies the command used for invoking the workload against which the program will be optimized.

In the previous example, the phases of fdpr processing were transparent to the user. However, they can be executed separately with the following set of commands:

```
# fdpr -s -1 -p testprog
# fdpr -s -2 -p testprog -x testprog.sh
# fdpr -s -3 -p testprog
```

The -s option instructs fdpr not to erase the temporary files. This option must be used when running fdpr in separate phases so that the succeeding phases can access the required intermediate files. The -1, -2, -3 flags specify which phase to run. These can be combined together (for example: -12 and then -3) and must be run in order. You cannot run phase 3 before phases 1 and 2 and so on. The default option is -123. The same user has to execute all phases.

After phase 1, fdpr generates the file __testprog.save. This file is a copy of the original program. The file testprog is now the instrumented version, which will be executed in phase 2. When phase 2 ends, the output file __testprog.prof is created, which is the profiled version of the executable being optimized. Phase 3 generates two intermediate files, __testprog.save.bt and __testprog.save.histo, and the optimized executable output file, testprog.fdpr. The testprog file again contains the original program. All files are written to the current directory.

4.6.2 Other fdpr Options

The flags -R0,-R1,-R2 and -R3 can be used to specify the level of optimization. The default option is -R0, while -R3 is the most aggressive optimization. However, the use of higher optimization levels may result in an executable that does not behave as expected. The programs generated with the -R0 and -R2 options are supported by the dbx command. To avoid branch reversing, the option -nI should be passed in the command line.

Executables built with the -qfdpr compiler flag contain information to assist fdpr in producing reordered programs with guaranteed functionality. When this compiler flag is used, the guaranteed functionality advantage of fdpr option -R0 is extended to options -R1, -R2 and -R3. However, if -qfdpr is used, only those object modules built with this flag will be reordered. So it should be used for all object modules in a program.

The fdpr program only reorders the instructions within the executable program specified. Any dynamically linked shared library routines called by the program will not be reordered.

4.6.3 Considerations

As stated before, a full test-verification cycle should be run to ensure that the results are correct and similar to the ones that are expected. Also, if you change the parameters for the program, the gain of performance you have encountered before is probably going to be altered.

This tool can also be used during the development of the program. You can use fdpr to check if the code can be further optimized. If the execution time of the executable generated by fdpr is significantly less than the original program, some sections can probably be recoded to obtain a better performance. Although this use for fdpr is arduous to the developer, it guarantees total control over the code, which can always be checked in case of undesired behavior.

4.7 The lockstat Command

The lockstat command displays lock-contention statistics for SMP systems. Application locks cannot be seen with this command. Only kernel locks generated by the workload can be verified.

This tool is only available in AIX V4.

4.7.1 Locks on SMP Systems

Multiprocessors and thread support make it attractive and easier to write applications that share data among threads. To avoid disaster, programs that share data must arrange to access that data serially, rather than in parallel, as shown in Figure 12 on page 165. If there is a shared variable that can be modified by more than one thread at a time, then modifications to this variable cannot be done simultaneously; instead, the updates to this variable have to be synchronized through the use of locks. The section of code that updates a shared variable is called a critical section, and must not be executed by more than one processor at a time.

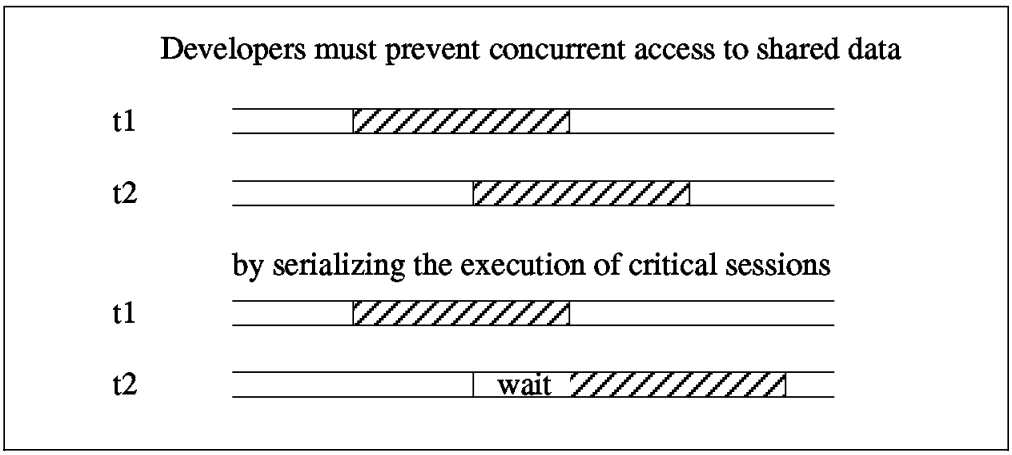


Figure 12. Data Serialization

Conceptually a lock is a bit in memory that threads use to regulate their entry into critical sections. For a processor, the simplest code sequence to take a lock is:

```
test the lock bit;
if lock is free
then
   set it to busy;
else
   wait for it to become free;
```

Since taking a lock requires several operations (read, test and set the lock bit), this operation is itself a critical section. Several threads can test the same lock at the same time. Therefore, multiprocessor hardware must provide a way to perform this test-and-set operation atomically with respect to the other processors. This kind of atomic operation is the basic block upon which all of the locking primitives are built.

When a thread wants to use a lock held by another thread, the thread is blocked. The operating system supports several types of locking strategies:

- Spin Lock

  Allows the waiting thread to keep its time slice and continually recheck the lock bit in a very tight loop. If the thread has a high priority, it could block other threads; therefore, spin locks should be used only when a lock is held for a very short time.

- Blocking Lock

  Suspends the thread until the lock is free and puts it back on the run queue. This type of lock is used when a lock is held for long periods of time.

- Read-Write

  Allows developers to distinguish between threads that need to read data and threads that need to write data. A read-write lock allows multiple readers, but guarantees mutual exclusion for writers.

- Hybrid Lock

  Is a combination of a spin lock and a blocking lock. A thread will spin for a short period of time waiting for the lock. If the lock does not become available, the thread will put itself to sleep.

AIX developers can choose between simple locks, which are either blocking or spin, and complex locks, which are read-write or hybrid. This choice has a big consequence for the scalability of the system. The use of locks made AIX V4 MP-safe, but it is the responsibility of the developers to define and implement an appropriate locking strategy to protect their own global data.

AIX V4 was changed and continues to be enhanced to make it more MP-efficient. This means that the system is optimized to spend the minimum time waiting for and dealing with locks. AIX-defined subsystems are comprised of 256 lock classes in /usr/include/sys/lockname.h.

AIX V4.2 was changed in order to reduce lock contention inside the kernel. Previously there was only one lock for the whole process table; now each process entry can be locked. For the JFS, the global JFS_LOCK has been cut into several smaller locks for operations on cache, directory and inodes.

The lockstat command supports the use of user-supplied lock names in files named /usr/include/sys/lockname_*.h, where * is a wildcard.

4.7.2 Using lockstat

Prior to using the lockstat command, you must create as root a new bosboot image with the -L option to enable lock instrumentation: (Assuming your boot disk is hdisk0)

```
# bosboot -a -d /dev/hdisk0 -L
```

After running the command, reboot the machine to enable lock instrumentation. At this time, lockstat can be used to look at the locking activity.

Attention

The lockstat command can be CPU intensive because there is overhead involved with lock instrumentation. That is the reason why it is not turned on by default. The overhead of enabling lock instrumentation is typically 3-5 percent. Also be aware that AIX trace buffers will fill up much quicker when using this option since there are a lot of locks being used.

Root authority is necessary to run lockstat.

The lockstat command generates a report for each kernel lock that meets all specified conditions. When no conditions are specified, the default values are used. These are the parameters that can be used to filter the data collected:

-c <LockCount>

Specifies how many times a lock must be requested during an interval in order to be displayed. A lock request is a lock operation which in some cases cannot be satisfied immediately. All lock requests are counted. The default is 200.

-b <BlockRatio>

Specifies a block ratio. When a lock request is not satisfied, it is said to be blocked. A lock must have a block ratio that is higher than BlockRatio to appear in the list. The default of BlockRatio is 5 percent.

-n <CheckCount>

Specifies the number of locks which are to be checked. The lockstat command sorts locks according to lock activity. This parameter determines how many of the most active locks will be subject to further checking. Limiting the number of locks that are checked maximizes system performance, particularly if lockstat is executed in intervals. The default value is 40.

-p <LockRate>

Specifies a percentage of the activity of the most-requested lock in the kernel. Only locks that are more active than this will be listed. The default value is 2, which means that the only locks listed are those requested at least 2 percent as often as the most active lock.

-t <MaxLocks>

Specifies the maximum number of locks to be displayed. The default is 10.

If the lockstat command is executed with no options, an output similar to the following would be displayed:

```
# lockstat
Subsys  Name                      Ocn   Ref/s   %Ref   %Block  %Sleep
_____________________________________________________________________

PFS     IRDWR_LOCK_CLASS          259   75356   37.49    9.44    0.21
PROC    PROC_INT_CLASS              1   12842    6.39   17.75    0.00
```

The first column is the subsystem (Subsys) to which the lock belongs. Some common subsystems are:

PROC	Scheduler, dispatcher or interrupt handlers
VMM	Pages, segment and freelist
TCP	Sockets, NFS
PFS	Inodes, icache

Next, the symbolic name of the lock class is shown. Some common classes are:

TOD_LOCK_CLASS	All interrupts that need the Time-of-Day (TOD) timer
PROC_INT_CLASS	Interrupts for processes
U_TIMER_CLASS	Per-process timer lock
VMM_LOCK_VMKER	Free list
VMM_LOCK_PDT	Paging device table
VMM_LOCK_LV	Per paging space
ICACHE_LOCK_CLASS	Inode cache

The field Ocn gives the occurrence number of the lock in its class. Next the reference number (Ref/s - number of lock requests per second) is listed, followed by the reference rate expressed as a percentage of all lock requests (%Ref). The last two columns present respectively the ratio of blocking lock requests to total lock requests (%Block) and the percentage of lock requests that cause the calling thread to sleep (%Sleep).

As a rule of thumb, you should be concerned if a lock has a reference number above 10000. In our example, both classes shown present a very high rate. In this case, you may want to use the vmstat command to investigate further. Refer to 2.1, “The vmstat Command” on page 7 for more information. If the vmstat output shows a significant amount of CPU idle time when the system seems subjectively to be running slowly, delays may be due to kernel lock contention, because lock requestors go into blocked mode. Lock contentions cause wasted

cycles because a thread may be spinning on a busy lock or sleeping until the lock is granted. Improper designs may even lead to deadlocks. The wasted cycles would degrade system performance.

The lockstat output does not show exactly which application is causing a problem to the system. The lock contentions problem can only be solved at the source-code level. For example, if your application has a high number of processes that read and write a unique message queue, you might have lock contention for the VMM subsystem. Adding more message queues may reduce the level of lock contention.

In the given example, many instances of a process that opens the same file for read-only were running simultaneously on the system. In AIX, every time a file is accessed, its inode is updated with the last access time. That is the reason for the high reference number observed for the lock class IRDWR_LOCK_CLASS. There were many threads trying to update the inode of the same file concurrently.

When lockstat is run without options, only the locks with %Block above 5 percent are listed. You can change this behavior by specifying another BlockRatio with the -b option:

```
# lockstat -b 1
Subsys  Name                      Ocn   Ref/s   %Ref   %Block  %Sleep
_____________________________________________________________________

PFS     IRDWR_LOCK_CLASS        258   95660   60.22   69.15    0.16
PROC    PROC_INT_CLASS            1    5798    3.65    4.73    0.00
PROC    PROC_INT_CLASS            2    2359    1.48    1.02    0.00
```

In this case, all the lock requests with %Block above 1 percent will be shown.

If no lock has a BlockRatio within the given range, the output would be the following:

```
# lockstat
No Contention
```

The -a option additionally lists the 10 most-requested (or active) locks:

```
# lockstat -a
Subsys  Name                      Ocn   Ref/s   %Ref   %Block  %Sleep
_____________________________________________________________________

PFS     IRDWR_LOCK_CLASS        259   75356   37.49    9.44    0.21
PROC    PROC_INT_CLASS            1   12842    6.39   17.75    0.00

First 10 largest reference rate locks :

Subsys  Name                      Ocn   Ref/s   %Ref   %Block  %Sleep
_____________________________________________________________________

PFS     IRDWR_LOCK_CLASS        259   75356   37.49    9.44    0.21
PROC    PROC_INT_CLASS            1   12842    6.39   17.75    0.00
PROC    TOD_LOCK_CLASS           --    5949    2.96    1.68    0.00
PROC    PROC_INT_CLASS            2    5288    2.63    3.97    0.00
XPSE    PSE_OPEN_LOCK            --    4498    2.24    0.87    0.00
IOS     SELPOLL_LOCK_CLASS       --    4276    2.13    3.20    0.00
XPSE    PSE_SQH_LOCK             95    4223    2.10    0.62    0.00
```

```
XPSE    PSE_SQH_LOCK            105    4213    2.10    0.50    0.00
XPSE    PSE_SQH_LOCK             75    3585    1.78    0.31    0.00
XPTY    PTY_LOCK_CLASS            6    3336    1.66    0.00    0.00
```

The meaning of the fields is the same as in the previous example. The first table is a list of locks with %Block above 5 percent. Then a list of the top 10 reference-rate locks, sorted in decreasing order, is given. The number of locks in the most-requested list can be changed with the -t option:

```
# lockstat -a -t 3
Subsys  Name                      Ocn    Ref/s   %Ref   %Block  %Sleep
_____________________________________________________________________

PFS     IRDWR_LOCK_CLASS        259   75356   37.49    9.44    0.21
PROC    PROC_INT_CLASS            1   12842    6.39   17.75    0.00

First 3 largest reference rate locks :

Subsys  Name                      Ocn    Ref/s   %Ref   %Block  %Sleep
_____________________________________________________________________

PFS     IRDWR_LOCK_CLASS        259   75356   37.49    9.44    0.21
PROC    PROC_INT_CLASS            1   12842    6.39   17.75    0.00
PROC    TOD_LOCK_CLASS           --    5949    2.96    1.68    0.00
```

In the above example, the -t option specifies that only the top three reference-rate locks will be shown.

The lockstat command can also be run in intervals:

```
# lockstat 10 100
```

The first number passed in the command line specifies the amount of time in seconds between each report. Each report contains statistics collected during the interval since the previous report. If no interval is specified, the system gives information covering an interval of one second and then exits. The second number determines the number of reports generated. It can only be specified if an interval is given.

If the output of lockstat -a looks like:

```
No Contention

First 10 largest reference rate locks :

Subsys  Name                    Ocn   Ref/s   %Ref   %Block  %Sleep
_____________________________________________________________________
```

then an empty most-requested lock list means that the lock instrumentation has not been enabled, which can be done by executing the command bosboot as explained at the beginning of this section.

4.7.3 Improving Lock Performance

According to queuing theory, the less idle a resource, the longer the average time to get it. The relationship is non-linear; if the area that the lock is protecting is doubled, the average wait time for that lock more than doubles. The most effective way to reduce wait time for a lock is to reduce the size of what the lock is protecting. There are also some useful rules to improve performance:

- The frequency with which any lock is requested should be reduced.
- Lock just the code that accesses shared data, not all the code in a component. This will reduce lock holding time.
- Locks should always be associated with specific data items or structures, not with routines.
- For large data structures, choose one lock for each element of the structure rather than one lock for the whole structure.
- Never do synchronous I/O or any other blocking activity while holding a lock.
- If you have more than one access to the same data in your component, try to move them together so they can be covered by one lock-unlock.
- Avoid double wake up. If you modify some data under a lock and have to notify someone that you have done it, release the lock and then post the wake up.
- If you must hold two locks simultaneously, request the busiest one last.

On the other hand, a too fine granularity will increase the frequency of lock requests and lock releases, which therefore will add additional instructions. A balance must be found between a too fine and a too coarse granularity. The optimum granularity will have to be found empirically and is one of the big challenges in an MP system. The graph in Figure 13 shows the relation between the throughput and the granularity of locks. As an initial rule of thumb, try not to hold locks for more than about 300 seconds.

The existence of widely used locks places an upper limit on the throughput of the system. For example, if a given program spends 20 percent of its execution time holding a mutual-exclusion lock, at most five instances of that program can run simultaneously, regardless of the number of processors in the system.

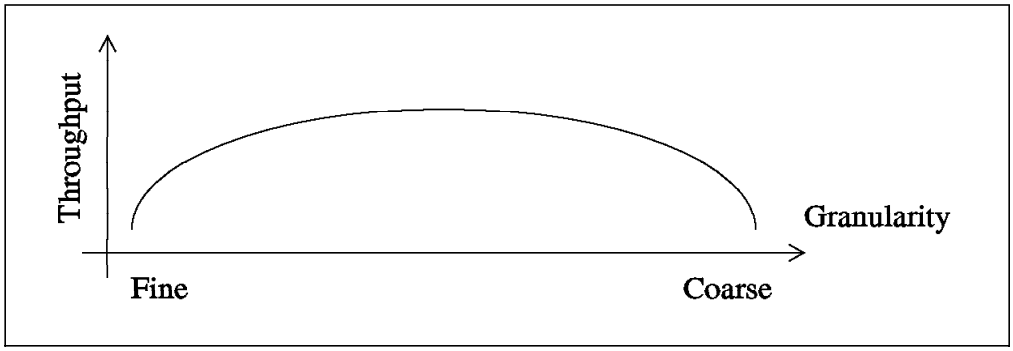


Figure 13. Relationship Between Throughput and Granularity

4.8 The cpu_state Command

The `cpu_state` command controls and lists which processors on a multiprocessor system will be active when the system is next started.

This tool is only available in AIX V4. The `cpu_state` command is only supported on machines that have a service processor. It cannot get status information about the processors without having a service processor.

This tool may be used to size the number of processors required for a given application by varying the number of active processors and monitoring system performance. Note that additional processors do not usually increase system performance in a linear fashion.

4.8.1 Using cpu_state

Root authority is necessary to run this command.

To check the status of the processors, run:

```
# cpu_state -l
```

The following output will be shown:

```
Name    Cpu      Status         Location
proc0   0       enabled         00-0P-00-00
proc1   1       enabled         00-0P-00-01
proc2   2       enabled         00-0Q-00-00
proc3   3       enabled         00-0Q-00-01
```

The Name field is the object data manager (ODM) processor name, shown in the form procX, where X is the physical processor number. Next is the logical processor number. Only enabled processors have logical numbers. Then the status for the next boot is shown. The last field is the ODM processor location code, shown in the form AA-BB-CC-DD (AA is the main unit, always 00; BB is the processor board number 0P, 0Q, 0R or 0S, indicating respectively the first, second, third or fourth processor card; CC is always 00; DD is the position of the processor on the card, it can be 00 or 01).

To disable a processor, enter:

```
# cpu_state -d proc2
# cpu_state -l
Name    Cpu      Status         Location
proc0   0       enabled         00-0P-00-00
proc1   1       enabled         00-0P-00-01
proc2   2       disabled        00-0Q-00-00
proc3   3       enabled         00-0Q-00-01
```

The -d option can be used to disable a processor. Notice that the status field does not display the current processor state, but rather the state to be used for the next boot. In this example, after issuing the command to disable proc2, it still has a logical CPU number, and the scheduler could assign threads to this CPU.

After system reboot:

```
# cpu_state -l
Name    Cpu      Status         Location
proc0   0       enabled         00-0P-00-00
proc1   1       enabled         00-0P-00-01
proc2   -       disabled        00-0Q-00-00
proc3   2       enabled         00-0Q-00-01
```

As can be seen in the above example, after the system reboot, proc2 does not have a logical CPU number, instead a dash (-) is shown in the field. Notice that now proc3 has been assigned a different logical CPU number. It changed from 3 to 2 because proc2 has been disabled.

To enable a processor, enter:

```
# cpu_state -e proc2
# cpu_state -l
Name    Cpu      Status         Location
proc0   0       enabled         00-0P-00-00
proc1   1       enabled         00-0P-00-01
proc2   -       enabled         00-0Q-00-00
proc3   2       enabled         00-0Q-00-01
```

After system reboot, proc2 will be enabled and have a logical CPU number associated to it:

```
# cpu_state -l
Name    Cpu      Status         Location
proc0   0       enabled         00-0P-00-00
proc1   1       enabled         00-0P-00-01
proc2   2       enabled         00-0Q-00-00
proc3   3       enabled         00-0Q-00-01
```

If a processor does not respond at boot time, it is either faulty (an ODM state) or a communication error occurred. In this case, the cpu_state command would show:

```
# cpu_state -l
Name    Cpu      Status         Location
proc0   0       enabled         00-0P-00-00
proc1   1       enabled         00-0P-00-01
proc2   2       enabled         00-0Q-00-00
proc3   -       no reply        00-0Q-00-01
```

This means that during system boot, proc3 failed the power-on tests, and its status was changed to no reply.

Another way of using the cpu_state command is through the diag tool:

```
#diag
```

Next, press **Enter** and choose:

Task Selection
Display or Change Multi-processor Configuration
Display or Change Processor States

After the last selection, the following options will be given:

Display Processor States
Disable a processor
Enable a processor

The first option is equivalent to the command cpu_state -l, the next to cpu_state -d <processor_name> and the last to cpu_state -e <processor_name>. In AIX V4.2, the last two options are not working properly. This should be solved in AIX V4.2.1.

The lsdev command can also be used on any multiprocessor system to query information about processors:

```
# lsdev -Cc processor
proc0 Available 00-0P-00-00 Processor
proc1 Available 00-0P-00-01 Processor
proc2 Defined   00-0Q-00-00 Processor
proc3 Available 00-0Q-00-01 Processor
```

The output above shows that the machine has three processors available (proc0, proc1 and proc3) and one defined (proc2). When a processor is defined, the scheduler cannot assign threads to it.

4.8.2 Differences in AIX V4.1

In AIX V4.1, the user interface for the cpu_state command is different. Instead of passing the name in the command line, the user should only give the physical CPU number:

```
# cpu_state -d 2   (AIX V4.1)
# cpu_state -e 2   (AIX V4.1)
```

Notice that the number following the option is not the logical CPU number. Instead, it is the number associated with the name of the processor (procX). Modifications will be done for AIX V4.2 in order to be able to accept both user interfaces (name and physical CPU number). The changes should be available in AIX V4.2.1.

The difference in the user interface between AIX V4.1 and 4.2 is the reason for the problem with the diag tool in AIX V4.2. This tool tries to execute the command cpu_state with a processor physical number instead of a processor name, as is expected in AIX V4.2. With the planned changes mentioned, this problem should also be solved.

The diag tool in AIX V4.1 has different menus to access the Display or Change Processor States menu:

```
#diag
```

Next press **Enter** and choose:

Service Aids
Multi-processor Service Aids
Display or Change Processor States

The menu that follows has the same interface and functionality as in AIX V4.2.

4.9 The bindprocessor Command

The bindprocessor command should be used to bind or unbind the kernel threads of a process to a processor.

This command is meant for multiprocessor systems. Although it will also work on uniprocessor systems, binding has no effect on such systems.

This tool is only available in AIX V4.

4.9.1 Processor Affinity

If a thread is interrupted and later redispatched to the same processor, there may still be lines in the cache of the processor that belong to that thread. If the thread is dispatched to a different processor, it will experience a series of cache misses until its cache working set has been retrieved from RAM. If the thread has not been blocked long, some of its instructions and data may still be in the cache of the processor it ran on last; so it would be more efficient to run the thread there rather than on another processor. On the other hand, if a

dispatchable thread has to wait until the processor it was previously running on is available, the thread may experience an even longer delay.

Processor affinity is the dispatching of a thread to the processor that was previously executing it. The degree of emphasis on processor affinity should vary directly with the size of the cache working set of the thread and inversely with the length of time since it was last dispatched.

AIX V4 implements processor affinity. When a thread is dispatched on a processor, the identity of the processor is registered in the structure of the thread itself. Each time the dispatcher selects a thread, it knows the processor number on which the thread last ran.

When a processor asks to run a thread, the dispatcher chooses from the priority-ordered run queues the thread with the highest priority and tests if this thread has affinity with the processor. If yes, the thread is dispatched to the processor. If no, the dispatcher tries to find another thread which last ran on the processor. It scans the run queues until it finds one. This scanning is not done indefinitely, it has some limits. Three criterias are used to limit the scanning of the run queues:

1. If the priority difference between the thread with the highest priority and the thread that last ran on the processor is greater than a threshold value, then the thread with the highest priority will be chosen. The threshold value is determined by a parameter called `affinity_priodelta`. The default value is 0, which means that by default the highest priority thread is always chosen to be executed.
2. Scanning is stopped when the number of scanned threads is higher than a defined value. This value is determined by a parameter called `affinity_scandelta`, which has a default value of 3 * number_of_CPUs.
3. Scanning is also stopped when the dispatcher encounters a boosted thread and if a parameter called `affinity_skipboosted` is FALSE.

   When a thread with a low priority holds a lock and if a higher-priority thread is waiting for the same lock, the low-priority thread gets the priority of the higher-priority thread (it is boosted so that the higher-priority thread does not have to wait too much for the lock). This priority inversion is always done by the system.

   When the `affinity_skipboosted` parameter is set to TRUE, the boosted thread is skipped, and the dispatcher goes on finding a thread that has affinity with the processor. The default value is FALSE.

The default values for the parameters mentioned above guard against side effects. For example, if the `affinity_skipboosted` is TRUE, then a low-priority thread may run instead of a boosted thread. Thus, the thread waiting for the lock may wait longer. Also, if the `affinity_scandelta` parameter is too high, the dispatcher may spend too much time scanning the run queues.

A better performance has been achieved on SDET (an industry standard multi-user benchmark) with:

```
affinity_priodelta = 3
affinity_scandelta = 2 x (number_of_CPUs) +1
affinity_skipboosted = TRUE
```

It is possible to change the values of these parameters, but currently there is no available program to change them. The only way to do that is to create a program that writes to /dev/kmem or patch the kernel. An example of a program to alter these values is given in 4.10.5, “Processor Affinity Parameters” on page 185. Changing these values should be done with care.

The highest possible degree of processor affinity is to bind a thread to a specific processor. Binding means that the thread will be dispatched to that processor only, regardless of the availability of other processors. The bindprocessor command binds the threads of a specified process to a particular processor. Binding a thread to a processor can help that process, but it can hurt overall system throughput.

4.9.2 Using bindprocessor

Root authority is necessary to bind or unbind threads in processes you do not own.

To query the available processors, enter:

```
# bindprocessor -q
The available processors are:  0 1 2 3
```

The output shows the logical processor numbers for the available processors, which are used with the bindprocessor command as will be seen.

To bind a process whose PID is 14596 to processor 1, enter:

```
# bindprocessor 14596 1
```

No return message is given if the command was successful. To verify if a process is bound or unbound to a processor, you can use the command ps -mo THREAD as explained in 2.4, “The ps Command” on page 30:

```
# ps -mo THREAD
USER   PID  PPID    TID ST  CP PRI SC    WCHAN        F     TT BND COMMAND
root  3292  7130      - A    1  60  1        -   240001  pts/0   - -ksh
   -     -     -  14309 S    1  60  1        -      400      -   - -
root 14596  3292      - A   73 100  1        -   200001  pts/0   1 /tmp/cpubound
   -     -     -  15629 R   73 100  1        -        0      -   1 -
root 15606  3292      - A   74 101  1        -   200001  pts/0   - /tmp/cpubound
   -     -     -  16895 R   74 101  1        -        0      -   - -
root 16634  3292      - A   73 100  1        -   200001  pts/0   - /tmp/cpubound
   -     -     -  15107 R   73 100  1        -        0      -   - -
root 18048  3292      - A   14  67  1        -   200001  pts/0   - ps -mo THREAD
   -     -     -  17801 R   14  67  1        -        0      -   - -
```

The column BND shows the number of the processor that the process is bound to or a dash (-) if the process is not bound at all.

To unbind a process whose PID is 14596, use the following command:

```
# bindprocessor -u 14596
```

```
# ps -mo THREAD
USER   PID  PPID    TID ST  CP PRI SC    WCHAN        F     TT BND COMMAND
root  3292  7130      - A   2  61  1        -   240001  pts/0   - -ksh
   -     -     -  14309 S   2  61  1        -      400      -   - -
root 14596  3292      - A 120 124  1        -   200001  pts/0   - /tmp/cpubound
   -     -     -  15629 R 120 124  1        -        0      -   - -
root 15606  3292      - A 120 124  1        -   200001  pts/0   - /tmp/cpubound
   -     -     -  16895 R 120 124  1        -        0      -   - -
root 16634  3292      - A 120 124  0        -   200001  pts/0   - /tmp/cpubound
   -     -     -  15107 R 120 124  0        -        0      -   - -
root 18052  3292      - A  12  66  1        -   200001  pts/0   - ps -mo THREAD
   -     -     -  17805 R  12  66  1        -        0      -   - -
```

When the bindprocessor command is used on a process, all of its threads will then be bound to one processor and unbound from their former processor. Unbinding the process will also unbind all its threads. You cannot bind/unbind an individual thread using bindprocessor. However, within a program, you can use the bindprocessor() function call to bind individual threads.

When a process does not exist, the following error is given:

```
# bindprocessor 7359 1
1730-002: Process 7359 does not match an existing process
```

Note

A process cannot be bound until it is started; that is, it must exist in order to be bound.

When a processor does not exist, the following error is given:

```
# bindprocessor 7358 4
1730-001: Processor 4 is not available
```

You should not use the bindprocessor command on the wait processes kproc. That will make the system crash immediately. This problem should be solved with AIX V4.2.1. However, it is still not recommended that you change the characteristics of these processes.

4.9.3 Considerations

Binding can be useful for CPU-intensive programs that experience few interrupts. It can sometimes be counterproductive for ordinary programs because it may delay the redispatch of a thread after an I/O until the processor to which the thread is bound becomes available. If the thread has been blocked for the duration of an I/O operation, it is unlikely that much of its processing context remains in the caches of the processor to which it is bound. It would probably be better served if it were dispatched to the next available processor.

Binding does not prevent other processes to be dispatched on the processor on which you bound your process. Binding is different from partitioning. It is not possible in AIX V4, for example, to dedicate a set of processors to a specific workload and another set of processors to another workload.

This means that a higher priority process might be dispatched on the processor where you bound your process. In this case, your process will not be dispatched on other processors. So, you will not always increase the performance of the

bound process. Better results may be achieved if you increase the priority of the bound process.

In fact, binding a single-threaded process will improve its performance on an idle system. In this case, if the process is not bound, it will bounce around all the processors and then might suffer a high cache miss rate.

Typically, if you bind a single-threaded program on an idle SMP, you will increase its performance. On the other hand, if you bind the same process on a heavily loaded system, you might decrease its performance because when a processor becomes idle, the process will not be able to run on the idle processor if it is not the processor on which the process is bound.

If the process is multithreaded, binding the process will bind all its threads to the same processor. This means that the process will not take advantage of the multiprocessing. You will not improve the performance of the process by doing this.

Attention

Process binding should be used with care because it disrupts the natural load balancing provided by AIX V4, and the overall performance of the system could degrade.

If the workload of the machine changes from that which is monitored when making the initial binding, system performance may suffer. If you use the `bindprocessor` command, take care to monitor the machine regularly because the environment may change, making the bound process adversely affect system performance.

4.10 The schedtune Command

The `schedtune` command can be used to set the parameters for CPU scheduler and Virtual Memory Manager processing. The executable and source for `schedtune` can be found in the /usr/samples/kernel directory. In AIX V3, these files are located in the /usr/lpp/bos/samples directory.

Take Note!!

The `schedtune` command is in the samples directory because it is VMM-implementation dependent. The `schedtune` code that accompanies each release of AIX is tailored specifically to the VMM of that release. This command is not supported under SMIT, nor has it been tested with all possible combinations of parameters. Misuse of this command can cause performance degradation or operating-system failure. If execution of `schedtune` causes a system to crash with flashing 888, then the appropriate level of `schedtune` for that release of AIX needs to be installed. Be sure that you have studied the appropriate tuning sections before using `schedtune` to change system parameters.

The `schedtune` command can only be executed by the root user. Changes made by this tool last until the next reboot of the system. If a permanent change is needed, an appropriate entry should be put in the /etc/inittab. For example:

```
schedtune:2:wait:/usr/samples/kernel/schedtune -m 6
```

In AIX V3, the path to the command schedtune should be altered to /usr/lpp/bos/samples.

Executing the schedtune command with no options shows the current settings for the parameters:

```
# schedtune
     THRASH           SUSP       FORK             SCHED
-h    -p    -m      -w    -e      -f       -d       -r        -t
SYS  PROC  MULTI   WAIT  GRACE   TICKS   SCHED_D  SCHED_R  TIMESLICE
 6   4     2        1      2      10       5        16         2
```

The first five parameters specify the thresholds for the memory load control algorithm, -f defines a wait time value used by the fork() routine; the -d and -r options are used in the scheduling policy algorithm, and -t defines the maximum amount of time a thread can spend on the processor. Each one of them will be discussed in detail in the next sections.

The -D option can be used to restore the default settings:

```
# schedtune -D
         THRASH           SUSP       FORK             SCHED
-h    -p    -m      -w    -e      -f       -d       -r        -t
SYS  PROC  MULTI   WAIT  GRACE   TICKS   SCHED_D  SCHED_R  TIMESLICE
 6   4     2        1      2      10       16       16         1
```

4.10.1 Memory Load Control Parameters

When a page fault occurs, the referenced page must be paged in and, on average, one or more pages must be paged out. AIX attempts to steal real memory from pages that are unlikely to be referenced in the near future, via the page replacement algorithm. Refer to the “The Page-Replacement Algorithm” on page 14, for more information.

At some level of competition for memory, no pages are good candidates for paging out to disk because they will all be reused in the near future by the active set of processes. When this happens, continuous paging in and out occurs. This condition is called thrashing. Thrashing results in incessant I/O to the paging disk and causes each process to encounter a page fault almost as soon as it is dispatched, with the result that none of the processes make any significant progress. The most pernicious aspect of thrashing is that the system may continue thrashing for an indefinitely long time.

AIX has a memory-load control algorithm that detects when the system is starting to thrash and then suspends active processes and delays the initiation of new processes for a period of time. The memory load-control mechanism assesses, once a second, whether sufficient memory is available for the set of active processes. When a memory overcommitment condition is detected, some processes are suspended, decreasing the number of active processes and thereby decreasing the level of memory overcommitment.

The pages of the suspended processes quickly become stale and are paged out via the page-replacement algorithm, releasing enough page frames to allow the remaining active processes to progress. During the interval in which existing processes are suspended, newly created processes are also suspended, preventing new work from entering the system. Suspended processes are not

reactivated until a subsequent interval passes during which no potential thrashing condition exists. Once this safe interval has passed, the threads of the suspended processes are gradually reactivated.

Note

Memory load control is intended to smooth-out infrequent peaks that might otherwise cause a system to thrash. It is not intended to act continuously in a configuration that has too little RAM to handle its normal workload.

Five parameters set rates and thresholds for the algorithm. The default values of these parameters have been chosen to ″fail safe″ across a wide range of workloads. The schedtune command can be used to alter these values. You should not change the memory load-control parameter settings unless your workload is consistent, and you believe the default values are ill-suited to your environment.

The h Parameter: Controls the threshold defining memory overcommitment. Memory load control attempts to suspend processes when this threshold is exceeded during any one-second period. The threshold is a relationship between two direct measures: the number of pages written to paging space in the last second (p) and the number of page steals occurring in the last second (s). The number of page writes is usually much less than the number of page steals. Memory is overcommitted when:

```
p/s > 1/h
```

The default value is 6. This value was chosen because it is comparatively configuration-independent. Any positive value is valid. In AIX V4, if a system has more than 128 MB of memory, the default value is 0. This means that, for these systems, memory load control is by default disabled. Testing has proved that with more than 128 MB of RAM, the normal VMM algorithms could correct thrashing conditions on the average more efficiently than by utilization of memory load control.

To disable memory load control, enter:

```
# schedtune -h 0
      THRASH           SUSP       FORK             SCHED
-h    -p    -m      -w    -e      -f       -d       -r        -t
SYS  PROC  MULTI   WAIT  GRACE   TICKS   SCHED_D  SCHED_R  TIMESLICE
 0    4     2        1     2      10       16       16         1
```

If disabling memory load control results in more, rather than fewer, thrashing situations (with correspondingly poorer responsiveness), then memory load control is playing an active and supportive role in your system. Tuning its parameters may result in improved performance; it would be better to add RAM. The vmstat command can be used to detect if the system is thrashing or if the memory is overcommited. Refer to 2.1.2, “Memory Bound” on page 10, for detailed information.

If you run schedtune -D on any system, the h parameter will be set to 6, independently of the memory size. A copy of the source code for schedtune is shipped with AIX in the /usr/samples/kernel directory and can be altered to change this behavior for systems with more than 128 MB of memory:

```
(...)
struct  cmdtab cmdtab[] = {
/*      flag,  nlst[i],         validation(),        current, default    */

       {'h',  V_REPAGE_HI,  v_repage_hi_x,       0,     6  },
        { 'p',  V_REPAGE_PROC,   v_repage_proc_x,        0,       4  },
        { 'w',  V_WAIT_SECS,     v_wait_secs_x,          0,       1  },
        { 'm',  V_MIN_PROCESS,   NULL,                   0,       2  },
        { 'e',  V_EXEMPT_SECS,   v_exempt_secs_x,        0,       2  },
        { 'f',  PACEFORK,        pacefork_x,             0,      10  },
        { 'd',  SCHED_D,         sched_d_x,              0,      16  },
        { 'r',  SCHED_R,         sched_r_x,              0,      16  },
        { 't',  TIMESLICE,       timeslice_x,            0,       1  },
        NULL,
};
(...)
```

This is the section of the source code that sets the default values used by the -D option. The highlighted line shows the settings for the h parameter. The last column is the default value. For systems with more than 128 MB of memory, you may want to change this value from 6 to 0:

```
{ 'h',  V_REPAGE_HI,     v_repage_hi_x,          0,       0  },
```

To make the changes effective, you need to recompile the program:

```
# cc schedtune.c -o schedtune
```

Make sure you have a C compiler installed along with the appropriate license.

A lower value of h raises the thrashing detection threshold; that is, the system is allowed to come closer to thrashing before processes are suspended. Regardless of the system configuration, when the above p/s fraction is low, thrashing is unlikely.

To alter the threshold to 4, enter:

```
# schedtune -h 4
      THRASH           SUSP       FORK             SCHED
-h    -p    -m      -w    -e      -f       -d       -r        -t
SYS  PROC  MULTI   WAIT  GRACE   TICKS   SCHED_D  SCHED_R  TIMESLICE
 4   4     2        1      2      10       16       16         1
```

In this way, you permit the system to come closer to thrashing before the algorithm starts suspending processes.

If you are using rmss to investigate the effects of reduced memory sizes, you will want to disable memory load control to avoid interference with your measurement. Refer to 3.3, “The rmss Command” on page 79, for more information on this command.

The p Parameter: Determines whether a process is eligible for suspension and is used to set a threshold for the ratio of two measures that are maintained for every process: the number of repages (r - refer to the “The Page-Replacement Algorithm” on page 14) and the number of page faults that the process has accumulated in the last second (f). A high ratio of repages to page faults means the individual process is thrashing. A process is considered eligible for suspension (it is thrashing or contributing to overall thrashing) when:

r/f > 1/p

The default value of p is 4, meaning that a process is considered to be thrashing (and a candidate for suspension) when the fraction of repages to page faults over the last second is greater than 25 percent. A low value of p results in a higher degree of individual process thrashing being allowed before a process is eligible for suspension.

To disable processes from being suspended by the memory load control:

```
# schedtune -p 0
      THRASH           SUSP       FORK             SCHED
-h    -p    -m      -w    -e      -f       -d       -r        -t
SYS  PROC  MULTI   WAIT  GRACE   TICKS   SCHED_D  SCHED_R  TIMESLICE
 6   0     2        1      2      10       16       16         1
```

Note that fixed-priority processes and kernel processes are exempt from being suspended.

The m Parameter: Determines a lower limit for the degree of multiprogramming, which is defined as the number of active processes. Active processes are those that are runable and waiting for page I/O. Processes that are waiting for events and processes suspended are not considered active nor is the wait process considered active.

Each process is counted as one, regardless of the number of threads running in it. Excluded from the count are the kernel processes, processes with fixed priority values less than 60, pinned memory, or awaiting events because no process in these categories is ever eligible for suspension.

The default value of 2 ensures that at least two user processes are always able to be active. Lower values of m, while allowed, mean that at times as few as one user process may be active. High values of m effectively defeat the ability of memory load control to suspend processes. This parameter is very sensitive to configuration and workload. While a value of 2 is appropriate for a desktop, single-user configuration, it is frequently too small for larger, multiuser or server configurations with large amounts of RAM. In these systems, increasing the value of m may result in better performance:

```
# schedtune -m 10
      THRASH           SUSP       FORK             SCHED
-h    -p    -m      -w    -e      -f       -d       -r        -t
SYS  PROC  MULTI   WAIT  GRACE   TICKS   SCHED_D  SCHED_R  TIMESLICE
 6   4     10       1      2      10       16       16         1
```

With these settings, the memory load control has to leave at least 10 user processes running when it is suspending processes.

When the memory requirements of the thrashing application is known, the m value can be suitably chosen. Suppose thrashing is caused by numerous instances of one application of size M. Given the system memory size N, the m parameter should be set to a value close to N/M. Setting m too low would unnecessarily limit the number of processes that could be active at the same time.

The w Parameter: Controls the number of one-second intervals during which the p/s fraction (explained in the “The h Parameter” on page 179) must remain below 1/h before suspended processes are reactivated. The default value of

one second is close to the minimum value allowed, zero. A value of one second aggressively attempts to reactivate processes as soon as a one-second safe period has occurred. Large values of w run the risk of unnecessarily poor response times for suspended processes while the processor is idle for lack of active processes to run.

To alter the wait time to reactivate processes after two seconds, enter:

```
# schedtune -w 2
THRASH                SUSP       FORK             SCHED
-h    -p    -m      -w    -e      -f       -d       -r        -t
SYS  PROC  MULTI   WAIT  GRACE   TICKS   SCHED_D  SCHED_R  TIMESLICE
 0   4     2        2      2      10       16       16         1
```

The e Parameter: Each time a suspended process is reactivated, it is exempt from suspension for a period of e elapsed seconds. This is to ensure that the high cost (in disk I/O) of paging in the pages of a suspended process results in a reasonable opportunity for progress. The default value of e is 2 seconds.

To alter this parameter, enter:

```
# schedtune -e 1
      THRASH           SUSP       FORK             SCHED
-h    -p    -m      -w    -e      -f       -d       -r        -t
SYS  PROC  MULTI   WAIT  GRACE   TICKS   SCHED_D  SCHED_R  TIMESLICE
 6   4     2        1      1      10       16       16         1
```

Suppose thrashing is caused occasionally by an application that uses lots of memory but runs for about T seconds. The default system setting for e (2 seconds) would probably cause this application swapping in and out T/2 times on a busy system. In this case, resetting e to a longer time would help this application to progress. System performance would improve when this offending application is pushed through quickly.

4.10.2 fork() Retry Interval Parameter

Specifies the number of clock ticks to wait before retrying a failed fork() call. The system will retry a failed fork() call five times. For example, if a fork() subroutine call fails because there is not enough space available to create a new process, the system retries the call after waiting the specified number of clock ticks. The default value is 10, and as there is one clock tick every 10 ms, the system would retry the fork() call every 100 ms.

If the paging space is only due to brief, sporadic workload peaks, increasing the retry interval may allow processes to delay long enough to be released:

```
# schedtune -f 15
     THRASH           SUSP       FORK             SCHED
-h    -p    -m      -w    -e      -f       -d       -r        -t
SYS  PROC  MULTI   WAIT  GRACE   TICKS   SCHED_D  SCHED_R  TIMESLICE
 6   4     2        1      2      15       16       16         1
```

In this way, when the system retries the fork() call, there is a higher chance of success because some processes might have finished their execution and, consequently, released pages from paging space.

4.10.3 Priority Calculation Parameters

The kernel maintains a priority value for each thread. The priority value is a positive integer and varies inversely with the importance of the associated thread. That is, a smaller priority value indicates a more important thread. When the scheduler is looking for a thread to dispatch, it chooses the dispatchable thread with the smallest priority value.

The priority of most threads varies with the amount of CPU time the thread has used recently. This implies that, on average, the more time slices a thread has been allocated recently, the less likely it is that the thread will be allocated the next time slice.

The formula for calculating the priority value for a process or thread is:

```
priority value = base value + nice value + CPU penalty due to recent CPU usage
```

The nice value is a fixed value set with the nice or renice commands. For more information on these commands, refer to 2.10, “The nice Command” on page 54, and to 2.11, “The renice Command” on page 56.

The calculation of the CPU penalty is based on two parameters that can be altered with schedtune. The options of schedtune to change these values are the -r and -d options.

The formula used by the scheduler to calculate the amount to be added to the priority of a process/thread as a penalty for recent CPU usage is:

```
CPU penalty = recent CPU use value * r/32
```

The once-per-second recalculation of the recently used CPU value for each process/thread is:

```
new recent CPU use value = old recent CPU use value * d/32
```

This recalculation permits a process/thread to be penalized less for old CPU usage. The recent CPU usage increases by one each time the thread is in control of the CPU at the end of a 10 ms clock tick, up to a maximum value of 120.

The recent CPU usage value is displayed as the C column in the ps command output:

```
# ps -ef | more
    USER   PID  PPID   C    STIME    TTY  TIME CMD
    root     1     0   0   Feb 24      - 12:43 /etc/init
    root  1362     1   0   Feb 24      -  0:00 /etc/srcmstr
    root  1721     1   0   Feb 24      -  0:00 /etc/uprintfd
    root  2483     1   0   Feb 24      -  0:28 /etc/cron
(...)
```

The default value for options -d and -r is 16, and the valid range is from 0 to 32.

The execution of the command:

```
# schedtune -r 0
```

would cause the CPU penalty to be always 0, making priority absolute. No background process would get any CPU time unless there were no dispatchable foreground processes at all. This behavior is only true if the background

process is executed in a shell, where a background process receives a nice value of 24, compared to 20 for foreground processes (the ksh is an example of this). Decreasing the r value makes it easier for foreground processes to compete. Decreasing the d value enables foreground processes to avoid competition with background processes for a longer time.

For example:

```
# schedtune -r 32 -d 32
```

Long-running processes would reach a C value of 120 and stay there, contending on the basis of their nice values. New processes would have better priority, regardless of their nice value, until they had accumulated enough time slices to bring them within the priority value range of the existing processes. The settings of the above example penalize processes/threads to a higher degree on CPU usage.

Another example:

```
# schedtune -r 4 -d 31
```

With this command, the normal priority range for interactive processes would be 60-75. A low r value restricts the priority range to a small value, restricting the impact of CPU usage on the priority. Consider the maximum C value of 120 multiplied by 4/32 would give a maximum CPU penalty of 15. A high value for d implies that the CPU usage is decayed slowly. This means that the CPU penalty value is almost the same after the one-second recalculation (decays 31/32 every second).

Along with the last settings shown in the above example for schedtune, run the renice command on long-running CPU-bound processes to change their initial priority to 80. In this way, the priority range for these processes would be 80-95. This means that these processes would never delay a foreground process that was started with regular 60 priority. This is a procedure that could be used to prevent batch processes from interfering with the online processes.

4.10.4 Time-Slice Increment Parameter

The number of clock interrupts that must occur before the dispatcher is called. The CPU time slice is the period between recalculations of the priority value. Normally, recalculation is done at each tick (10 ms) of the system clock. The number of clock ticks between recalculations (length of the time slice) can be increased by 10 ms (one clock tick) increments.

However a time slice is not a guaranteed amount of processor time. It is the longest time that a process/thread can be in control before it is replaced by another process or thread. A process or thread can lose control of the CPU before its full time slice when the following occurs:

- A process/thread with a higher priority returns from a system call.
- A process/thread with a higher priority completes an I/O request.
- The current process/thread issues an I/O request.
- The current process/thread is suspended.
- The current process/thread yields the CPU with the yield() subroutine.

In AIX V4, the default value is 1, and this parameter only applies to threads with the SCHED_RR scheduling policy (an explanation about scheduling policies is given in 2.5, “The pstat Command” on page 33). This means that variable time slices only affect fixed-priority threads. The command:

```
# schedtune -t 2
THRASH           SUSP       FORK             SCHED
-h    -p    -m      -w    -e      -f       -d       -r        -t
SYS  PROC  MULTI   WAIT  GRACE   TICKS   SCHED_D  SCHED_R  TIMESLICE
 6   4     2        1      2      10       16       16         2
```

would set the time slice to 20 ms; so a fixed priority thread would get at least two full time slices before it is switched out of the CPU.

If you set the time slice parameter to 0, it means that a fixed-priority thread would never get more than one CPU tick, whereas with a value of 1, it would likely get one tick but possibly a bit more of execution time. The performance is probably be better with a time slice of one instead of zero.

In AIX V3, the default value for t is 0, but it is 1 in AIX V4. If you execute schedtune -t 2 in AIX V3, the time slice will be set to 30 ms and, in this version, will affect all processes. Notice that:

- In AIX V3, changing the time slice with the -t option sets the time slice tick value to the number specified plus 1. This setting affects all processes.
- In AIX V4, changing the time slice will only have an effect on round-robin scheduling policy threads, and the value of schedtune -t is the same as that of its time slice value.

It has been found that different values of t have not caused significant changes in the execution times of processes or threads. Although, if your workload consists almost entirely of very long-running, CPU-intensive programs, increasing this parameter may have some positive effect.

4.10.5 Processor Affinity Parameters

There are three variables concerning affinity that are used by the scheduler: affinity_priodelta, affinity_scandelta and affinity_skipboosted. These, along with the processor affinity issues, are discussed in detail in 4.9.1, “Processor Affinity” on page 173. As stated in this chapter, there is no available program to change these values. But the source code for schedtune (located in /usr/samples/kernel) can be easily altered to permit the modification of these parameters. Changes to the default values only apply to MP systems, and not many tests have been done up to now to give recommendations about the best usage of these parameters.

A sample of a modified version of schedtune is given below, but make sure you understand the use of each parameter before issuing any changes. Example:

```
#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
#include <nlist.h>

extern int errno;
extern int optind;
extern char *optarg;

#define V_REPAGE_HI      0
#define V_REPAGE_PROC    1
#define V_WAIT_SECS      2
```

```
#define V_MIN_PROCESS    3
#define V_EXEMPT_SECS    4
#define PACEFORK         5
#define SCHED_D          6
#define SCHED_R          7
#define TIMESLICE        8
#define A_PRIODELTA    9
#define A_SCANDELTA   10
#define A_SKIPBOOSTED 11
#define MAXPARMS      12

static struct nlist nlist[] = {
        { "v_repage_hi" },
        { "v_repage_proc" },
        { "v_sec_wait" },
        { "v_min_process" },
        { "v_exempt_secs" },
        { "pacefork" },
        { "sched_D" },
        { "sched_R" },
        { "timeslice" },
        { "affinity_priodelta"},
        { "affinity_scandelta"},
        { "affinity_skipboosted"},
        NULL
};

struct  cmdtab {
        char    flag;                           /* command flag; see below */
        int     index;                          /* index into nlist        */
        void    (*invalid)();                   /* validation routine      */
        int     current;                        /* current value           */
        int     vdefault;                       /* default value           */
};

static void v_repage_hi_x();
static void v_repage_proc_x();
static void v_wait_secs_x();
static void v_exempt_secs_x();
static void pacefork_x();
static void sched_d_x();
static void sched_r_x();
static void timeslice_x();
static void a_priodelta_x();
static void a_scandelta_x();
static void a_skipboosted_x();

struct  cmdtab cmdtab[] = {
/*      flag,  nlist[i],         validation(),        current, default    */

{ 'h',  V_REPAGE_HI,     v_repage_hi_x,          0,       6  },
        { 'p',  V_REPAGE_PROC,   v_repage_proc_x,        0,       4  },
        { 'w',  V_WAIT_SECS,     v_wait_secs_x,          0,       1  },
        { 'm',  V_MIN_PROCESS,   NULL,                   0,       2  },
        { 'e',  V_EXEMPT_SECS,   v_exempt_secs_x,        0,       2  },
        { 'f',  PACEFORK,        pacefork_x,             0,      10  },
        { 'd',  SCHED_D,         sched_d_x,              0,      16  },
        { 'r',  SCHED_R,         sched_r_x,              0,      16  },
        { 't',  TIMESLICE,       timeslice_x,            0,       1  },
        { 'a',  A_PRIODELTA,   a_priodelta_x,        0,      0  },
        { 's',  A_SCANDELTA,   a_scandelta_x,        0,     12  },
        { 'k',  A_SKIPBOOSTED, a_skipboosted_x,      0,      0  },
        NULL,
};

static void rdwrval();
static void read_values();
static cmd_help();
static display_values();

int     kmem;

#define READ_MODE               0
#define WRITE_MODE              1
```

```
main(argc, argv)
        int argc;
        char **argv;
{
        int     i, c, value;

        nlist("/usr/lib/boot/unix", nlst);

        if (nlst[V_REPAGE_HI].n_value == 0) {
                perror("namelist on /usr/lib/boot/unix failed");
                exit(1);
        }

        kmem = open("/dev/kmem", O_RDWR, 0);
        if (kmem < 0) {
                perror("failed open of /dev/kmem");
                exit(1);
        }

        read_values();

        if (argc == 1)                  /* display current values */
        {
                display_values();
                exit(0);
        }

        while ((c = getopt(argc, argv, "Dt:r:d:h:p:m:w:e:f:a:s:k:")) != EOF) {
                if (c == '?')
                {
                        cmd_help();
                        exit(1);
                }
                if (c == 'D')
                {
                        for (i=0; i<MAXPARMS; i++)
                        {
                                cmdtab[i].current = cmdtab[i].vdefault;
                                rdwrval(WRITE_MODE,i);
                        }
                        read_values();
                        display_values();
                        exit(0);
                }

                for(i=0; cmdtab[i].flag != c; i++);

                if (cmdtab[i].flag == c && nlst[i].n_value){
                        value = atoi(optarg);

                        if (cmdtab[i].invalid)
                                (*cmdtab[i].invalid)(i, value);
                        cmdtab[i].current = value;

                        if (nlst[i].n_value)
                                rdwrval(WRITE_MODE,i);
                }
        }

        display_values();
        close(kmem);
        exit(0);
}
static
cmd_help()
{
        printf("schedtune command\n");
        printf("\n");
       (... skip ...)
        printf("-a n    The maximum number of threads to be scanned in the priority list\n");
        printf("        after the highest one\n");
        printf("-s n    The maximum number of threads to be scanned\n");
        printf("-k n    Skip or not threads which have been boost\n");
        printf("\n");
}
```

```
static
void
read_values()
{
        rdwrval(READ_MODE,V_REPAGE_HI);
        rdwrval(READ_MODE,V_REPAGE_PROC);
        rdwrval(READ_MODE,V_WAIT_SECS);
        rdwrval(READ_MODE,V_MIN_PROCESS);
        rdwrval(READ_MODE,V_EXEMPT_SECS);
        rdwrval(READ_MODE,PACEFORK);
        rdwrval(READ_MODE,SCHED_D);
        rdwrval(READ_MODE,SCHED_R);
        rdwrval(READ_MODE,TIMESLICE);
       rdwrval(READ_MODE,A_PRIODELTA);
       rdwrval(READ_MODE,A_SCANDELTA);
       rdwrval(READ_MODE,A_SKIPBOOSTED);
}
static
void
rdwrval(int mode, int index)
{
(... skip ...)
}

static
display_values()
{
        char    outbuf[120];

        printf("\n");
        printf("     THRASH           SUSP       FORK             SCHED
     AFFINITY\n");
        printf("-h    -p    -m      -w    -e      -f       -d       -r        -t
-a      -s      -k\n");
        printf("SYS  PROC  MULTI   WAIT  GRACE   TICKS   SCHED_D  SCHED_R  TIMESLICE   P
RIO    SCAN    SKIP\n");

        sprintf( outbuf,
       " %-d   %-d     %-d        %-d      %-d      %-d       %d       %d
  %d    %d      %d      %d\n",
                 cmdtab[V_REPAGE_HI].current,
                 cmdtab[V_REPAGE_PROC].current,
                 cmdtab[V_MIN_PROCESS].current,
                 cmdtab[V_WAIT_SECS].current,
                 cmdtab[V_EXEMPT_SECS].current,
                 cmdtab[PACEFORK].current,
                 cmdtab[SCHED_D].current,
                 cmdtab[SCHED_R].current,
                 cmdtab[TIMESLICE].current,
                cmdtab[A_PRIODELTA].current,
                cmdtab[A_SCANDELTA].current,
                cmdtab[A_SKIPBOOSTED].current);
        printf(outbuf);

        return(0);
}

/* validation routines */
(... skip ...)

static
void
a_priodelta_x(int i, int value)
{
        if (value < -2 || value > 127)
        {
                printf("Invalidating -%c %d.\n", cmdtab[i].flag, value);
                printf("New value must be greater than zero and less than 127. S
pecify -1 to disable affinity.\n");
                exit(1);
        }
}
```

```
static
void
a_scandelta_x(int i, int value)
{
        if (value < 0)
        {
                printf("Invalidating -%c %d.\n", cmdtab[i].flag, value);
                printf("New value must be greater than or equal to zero\n");
                exit(1);
        }
}

static
void
a_skipboosted_x(int i, int value)
{
        if (value < 0 || value > 1)
        {
                printf("Invalidating -%c %d.\n", cmdtab[i].flag, value);
                printf("New value must be zero or one\n");
                exit(1);
        }
}
```

The highlighted lines show the modified/added lines in the program. The `cmd_help()` function is truncated, but the lines that should be included are shown. The `cmd_help()` function is called when the user runs `schedtune -?`. The `-?` option displays information about the use of the command.

In this example, the option `-a` was chosen to change the `affinity_priodelta` value, the `-s` for the `affinity_scandelta` and the `-k` for the `affinity_skipboosted`. These new options are used in the same way as the other `schedtune` options.

The default value for the `affinity_scandelta` parameter is 3 multiplied by the number of CPUs. For this example, a four-way SMP was used to test the program, and that is the reason for the default value of 12 for this parameter. You should change this value accordingly to your hardware configuration. The default values are set in the `struct cmdtab`.

The `rdwrval()` routine was truncated because its code was not changed. Some validation routines were not shown, but their code also remains the same.

4.11 The vmtune Command

The `vmtune` command can be used to modify the VMM parameters that control the behavior of the memory-management subsystem. Some options are available to alter the defaults for other AIX components. The executable for `vmtune` is found in the /usr/samples/kernel directory. In AIX V3, this file is located in the /usr/lpp/bos/samples directory.

Prior to AIX V4.2, a copy of the source code for `vmtune` was also shipped along with the executable. But to recompile the program a kernel header file that is needed is not available. So you are not able to make changes to `vmtune`, although you have the source code.

Take Note!!

The vmtune command is in the samples directory because it is VMM-implementation dependent. The vmtune code that accompanies each release of AIX is tailored specifically to the VMM in that release. Running vmtune from one release on a system with a different VMM release might result in an operating system failure. It is also possible that the functions of vmtune may change from release to release. It takes know-how and experience to set vmtune parameters properly. Be sure that you have studied the appropriate tuning sections before using vmtune to change system parameters.

The vmtune command can only be executed by the root user. Changes made by this tool last until the next reboot of the system. If a permanent change is needed, an appropriate entry should be put in the /etc/inittab. For example:

```
schedtune:2:wait:/usr/samples/kernel/vmtune -P 50
```

In AIX V3, the path to the command vmtune should be altered to /usr/lpp/bos/samples.

Executing the vmtune command with no options:

```
# vmtune
vmtune:  current values:
  -p       -P        -r          -R         -f       -F       -N        -W
minperm  maxperm  minpgahead maxpgahead  minfree  maxfree  pd_npages maxrandwrt
   3072    12288       2          8        120      128     524288        0

  -M       -w       -k       -c         -b          -B          -u
maxpin   npswarn  npskill  numclust  numfsbufs   hd_pbuf_cnt  lvm_bufcnt
  13108     1280      320        1       93           64           9

number of valid memory pages = 16384    maxperm=75.0% of real memory
maximum pinable=80.0% of real memory    minperm=18.8% of real memory
number of file memory pages = 3004      numperm=18.3% of real memory
```

The output shows the current settings for the parameters. Each one of them will be discussed in the next sections.

4.11.1 Tuning VMM Page Replacement

The Virtual Memory Manager (VMM) services memory requests from the system and its applications. Virtual memory segments are partitioned in units called pages; each page is either located in physical memory (RAM) or stored on disk until it is needed. AIX uses virtual memory in order to address more memory than is physically available in the system. The management of memory pages in RAM or on disk is handled by the VMM.

In AIX, virtual memory segments are partitioned into 4096-byte units called pages. Real memory is divided into 4096-byte page frames. The VMM has two major functions:

- To manage the allocation of page frames
- To resolve references to virtual memory pages that are not currently in RAM (stored in paging space) or do not yet exist

In order to accomplish its task, the VMM maintains a free list of available page frames. The VMM also uses a page-replacement algorithm to determine which virtual memory pages currently in RAM can have their page frames reassigned to the free list. Refer to the "The Page-Replacement Algorithm" on page 14.

AIX distinguishes between different types of memory segments:

- Persistent segments

  They have a permanent storage location on disk. Files containing data or executable programs are mapped to persistent segments. When a journaled file system (JFS) file is opened and accessed, the file data is copied into RAM.

- Working segments

  They are transitory and exist only during their use by a process and have no permanent disk storage location. Process stack and data regions are mapped to working segments and to shared-library text segments. Pages of working segments must also have disk storage locations to occupy when they cannot be kept in real memory. The disk paging space is used for this purpose. When a program exits, all of its working pages are placed back on the free list immediately.

Computational memory consists of the pages that belong to working storage segments or to program text segments. File memory consists of the remaining pages.

VMM Thresholds: Several numerical thresholds define the objectives of the VMM. When one of these is reached, the VMM takes appropriate action to bring the state of memory back within bounds. The thresholds that can be altered by the `vmtune` command are:

- **minfree**

  Minimum acceptable number of real memory page frames in the free list. When the size of the free list falls below this number, the VMM begins stealing pages until the size of the free list reaches `maxfree`. The default value of `minfree` is the `maxfree` value minus 8. The valid range is from 8 to 204800.

- **maxfree**

  Maximum size to which the free list will grow by VMM page-stealing. The size of the free list may exceed this number as a result of processes terminating and freeing their working segment pages or the deletion of files that have pages in memory. The default value is set by:

  ```
  maxfree = minimum (number of memory pages/128, 128)
  ```

  The valid range is from 16 to 204800, but must be greater than the number specified by the `minfree` parameter by at least the value of `maxpgahead` (this parameter is explained later in this chapter).

- **minperm**

  If the percentage of real memory occupied by file pages falls below this level, the page-replacement algorithm steals both file and computational pages, regardless of repage rates. The default value is calculated by:

  ```
  minperm (in pages) = ((number of memory frames) - 1024) * .2
  ```

The resulting percentage is always around 17 to 19 percent. Any value equal to or greater than one is valid.

- **maxperm**

  If the percentage of real memory occupied by file pages rises above this level, the page-replacement algorithm steals only file pages. The default value is calculated by:

  maxperm (in pages) = ((number of memory frames) - 1024) * .8

  The resulting percentage is always around 75 to 80 percent. Any value equal to or greater than one is valid.

When the percentage of real memory occupied by file pages is between minperm and maxperm, the VMM normally steals only file pages, but if the repaging rate for file pages is higher than the repaging rate for computational pages, computational pages are stolen as well.

The main intent of the page-replacement algorithm is to ensure that computational pages are given fair treatment. For example, the sequential reading of a long data file into memory should not cause the loss of program text pages that are likely to be used again soon. The page-replacement algorithm ensures that both types of pages get treated fairly, with a slight bias in favor of computational pages.

Choosing minfree and maxfree Settings: The objectives in tuning these limits are to ensure that any activity that has critical response time objectives can always get the page frames it needs from the free list and that the system does not experience unnecessarily high levels of I/O because of premature stealing of pages to expand the free list.

If you have a short list of programs you want to run fast, you could investigate their memory requirements with the svmon command and set minfree to the largest memory requirement found. Refer to 3.2, “The svmon Command” on page 71, for more information about the svmon command. This technique risks being too conservative because not all of the pages that a process uses are acquired in one burst.

Another way to investigate the memory requirements of a program is with the vmstat command. If you run vmstat 1 while executing the program on an otherwise idle system, you can see the memory being requested over time:

```
kthr     memory             page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 0  0 16506  3402   0   0   0   0    0   0 151  216  54  1  2 97  0
 0  0 16506  3401   0   0   0   0    0   0 161  377  72  1  2 97  0
 0  0 16506  3401   0   0   0   0    0   0 173  396 112  2  6 92  0
 0  0 16506  3401   0   0   0   0    0   0 171  360  89  1  5 94  0
 0  0 16506  3401   0   0   0   0    0   0 169  347  98  2  5 93  0
 1  0 16762  3139   0   0   0   0    0   0 407 2322 921 28 69  3  0
 2  0 17072  2829   0   0   0   0    0   0 496 2288 905 33 67  0  0
 2  0 17318  2583   0   0   0   0    0   0 420 2318 889 30 70  0  0
 3  0 17582  2319   0   0   0   0    0   0 425 2293 1011 24 76 0  0
 3  1 17854  2047   0   0   0   0    0   0 584 2185 926 25 75  0  0
 2  0 18111  1790   0   0   0   0    0   0 463 2249 1003 21 79 0  0
 2  0 18358  1543   0   0   0   0    0   0 402 2264 996 30 70  0  0
 3  0 18637  1264   0   0   0   0    0   0 443 2284 982 28 72  0  0
 2  0 18916   985   0   0   0   0    0   0 438 2322 983 20 80  0  0
 2  0 16506  3401   0   0   0   0    0   0 324 1807 690 25 56 19  0
 0  0 16506  3401   0   0   0   0    0   0 191  335 111  2 11 87  0
 0  0 16506  3401   0   0   0   0    0   0 200  850 180  2 18 80  0
 0  0 16506  3401   0   0   0   0    0   0 175  427 107  4  7 89  0
```

```
 0  0 16506  3401   0    0    0    0    0   0 162  289  67  2  4 94  0
 0  0 16506  3401   0    0    0    0    0   0 170  412 119  2  7 91  0
```

The highlighted fields show the difference in the active virtual memory when the process was running. As you can see, the program requests each second around 250 pages. The testing system was idle, but on a production environment, the free list would be approximately 120, which is the default value for the system in question. Raising the minfree value would improve the execution time of this application on a production environment:

```
# vmtune -f 270 -F 330
vmtune:  current values:
  -p       -P        -r          -R         -f       -F       -N        -W
minperm  maxperm  minpgahead maxpgahead  minfree  maxfree  pd_npages maxrandwrt
   2457     4915       2          8        120      128     524288        0

  -M       -w       -k       -c         -b          -B          -u
maxpin   npswarn  npskill  numclust  numfsbufs   hd_pbuf_cnt  lvm_bufcnt
  13108     1280      320        1       93           64           9

number of valid memory pages = 16384    maxperm=30.0% of real memory
maximum pinable=80.0% of real memory    minperm=15.0% of real memory
number of file memory pages = 1911      numperm=11.7% of real memory

vmtune:  new values:
  -p       -P        -r          -R         -f       -F       -N        -W
minperm  maxperm  minpgahead maxpgahead  minfree  maxfree  pd_npages maxrandwrt
   2457     4915       2          8        270      330     524288        0

  -M       -w       -k       -c         -b          -B          -u
maxpin   npswarn  npskill  numclust  numfsbufs   hd_pbuf_cnt  lvm_bufcnt
  13108     1280      320        1       93           64           9

number of valid memory pages = 16384    maxperm=30.0% of real memory
maximum pinable=80.0% of real memory    minperm=15.0% of real memory
number of file memory pages = 1911      numperm=11.7% of real memory
```

Notice that maxfree should be greater than minfree by at least 8 or by maxpgahead, whichever is greater. Increasing the maxfree value above the minimum allowed can help reduce calls to replenish the free list. But as a rule of thumb, keep the difference between maxfree and minfree below 100.

Choosing minperm and maxperm Settings: AIX takes advantage of the varying requirements for real memory by leaving in memory pages of files that have been read or written. If the file pages are requested again before their page frames are reassigned, this technique saves an I/O operation. These file pages may be from local or remote (for example, NFS) file systems.

In a particular workload, it may be worthwhile to emphasize the avoidance of file I/O. In another workload, keeping computational segment pages in memory may be more important. To understand what the ratio is in the untuned state, use the vmtune command with no arguments:

```
# vmtune
vmtune:  current values:
  -p       -P        -r          -R         -f       -F       -N        -W
minperm  maxperm  minpgahead maxpgahead  minfree  maxfree  pd_npages maxrandwrt
   3072    12288       2          8        120      128     524288        0
```

```
  -M      -w      -k      -c         -b          -B          -u
maxpin   npswarn  npskill  numclust  numfsbufs   hd_pbuf_cnt  lvm_bufcnt
  13108     1280      320        1       93           64           9

number of valid memory pages = 16384    maxperm=75.0% of real memory
maximum pinable=80.0% of real memory    minperm=18.8% of real memory
number of file memory pages = 1506     numperm=9.2% of real memory
```

The numperm gives the percentage number of file pages in memory, 9.2 percent. If we know that our workload makes little use of recently read or written files, we may want to constrain the amount of memory used for that purpose.

```
# vmtune -p 15 -P 30
vmtune:  current values:
  -p       -P        -r          -R         -f       -F       -N        -W
minperm  maxperm  minpgahead maxpgahead  minfree  maxfree  pd_npages maxrandwrt
   3072    16384       2          8        120      128     524288        0

  -M      -w      -k      -c         -b          -B          -u
maxpin   npswarn  npskill  numclust  numfsbufs   hd_pbuf_cnt  lvm_bufcnt
  13108     1280      320        1       93           64           9

number of valid memory pages = 16384    maxperm=100.0% of real memory
maximum pinable=80.0% of real memory    minperm=18.8% of real memory
number of file memory pages = 1794      numperm=10.9% of real memory

vmtune:  new values:
  -p       -P        -r          -R         -f       -F       -N        -W
minperm  maxperm  minpgahead maxpgahead  minfree  maxfree  pd_npages maxrandwrt
   2457     4915       2          8        120      128     524288        0

  -M      -w      -k      -c         -b          -B          -u
maxpin   npswarn  npskill  numclust  numfsbufs   hd_pbuf_cnt  lvm_bufcnt
  13108     1280      320        1       93           64           9

number of valid memory pages = 16384    maxperm=30.0% of real memory
maximum pinable=80.0% of real memory    minperm=15.0% of real memory
number of file memory pages = 1794      numperm=10.9% of real memory
```

This would set minperm to 15 percent and maxperm to 30 percent of real memory. It would ensure that the VMM would steal page frames only from file pages when the ratio of file pages to total memory pages exceeds 30 percent.

Tuning for Maximum Caching of NFS Data: NFS does not have a data-caching function, but VMM caches pages of NFS data just as it caches pages of disk data. If a system is essentially a dedicated NFS server, it may be appropriate to permit the VMM to use as much memory as necessary for data caching. This is accomplished by setting the maxperm parameter to 100 percent with:

```
# vmtune -P 100
```

4.11.2 Tuning Sequential Read-Ahead

The VMM tries to anticipate the future need for pages of a sequential file by observing the pattern in which a program is accessing the file. When the program accesses two successive pages of the file, the VMM assumes that the program will continue to access the file sequentially, and the VMM schedules additional sequential reads of the file. These reads are overlapped with the program processing and will make the data available to the program sooner

than if the VMM had waited for the program to access the next page before initiating the I/O. The number of pages to be read ahead is determined by two VMM thresholds:

- **minpgahead**

  Number of pages read ahead when the VMM first detects the sequential access pattern. If the program continues to access the file sequentially, the next read ahead will be two times `minpgahead`, the next four times `minpgahead`, and so on until the number of pages reaches `maxpgahead`. The default value is 2. This value can range from 0 through 4096 and should be a power of 2.

- **maxpgahead**

  Maximum number of pages the VMM will read ahead in a sequential file. The default value is 8. This value can range from 0 through 4096. It should be a power of 2 and should be greater than or equal to `minpgahead`.

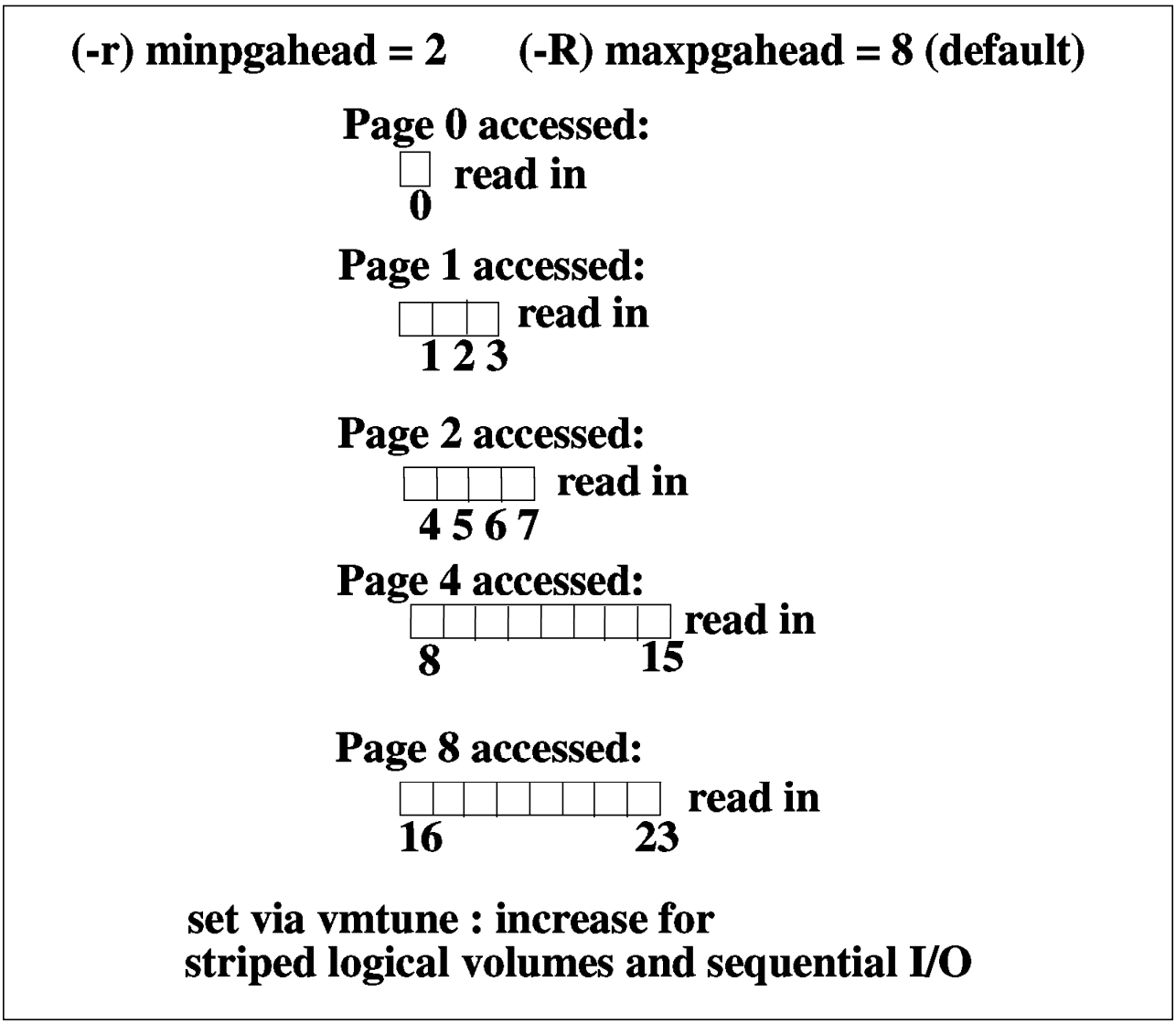


Figure 14. Sequential Read Ahead

The first access to a file causes the first page to be read in. When the second page is accessed, the `minpgahead` number of pages is read in. Subsequent

accesses of first-page read-ahead pages result in a doubling of the pages read in, up to maxpgahead.

If the program deviates from the sequential-access pattern and accesses a page of the file out of order, sequential read-ahead is terminated. It will be resumed with minpgahead pages if the VMM detects a resumption of sequential access by the program.

Occasions when tuning the sequential read-ahead feature (or turning it off) will improve performance are rare. The minpgahead and maxpgahead values can be changed with the vmtune command. If you are contemplating changing these values, keep in mind:

- The values should be one of the set: 0, 1, 2, 4, 8, 16. The use of other values may have adverse performance or functional effects.
- Values should be powers of 2 because of the doubling algorithm of the VMM.
- Values of maxpgahead greater than 16 (reads ahead of more then 64 KB) exceed the capabilities of some disk device drivers.
- Higher values of maxpgahead can be used in systems where the sequential performance of striped logical volumes is of high importance.
- A minpgahead value of 0 effectively defeats the mechanism. This may have serious adverse consequences for performance.
- The default maxpgahead value of 8 yields the maximum possible sequential I/O performance for currently supported disk drives.
- The ramp-up of the read-ahead value from minpgahead to maxpgahead is quick enough that for most file sizes there would be no advantage to increasing minpgahead.
- The VMM currently only allows read ahead of 512 pages; so there is no use in increasing maxpgahead above this value.

VMM page-ahead can be turned off by setting minpgahead to zero. This can be useful in some cases where I/O is random, but the size of the I/Os cause the read-ahead algorithm to take effect. Another case where turning off page-ahead is useful is in the case of NFS reads on files that are locked; on these types of files, read-ahead pages are typically flushed by NFS so that reading ahead is a waste of time.

Tuning for Striped Logical Volume I/O: In benchmarks, the following techniques have yielded the highest levels of sequential I/O throughput:

- Stripe unit size of 64 KB.
- max_coalesce of 64 KB, which is the default value. This parameter limits the largest request, in terms of data transmitted, that the SCSI device driver will build (cannot be altered with vmtune). This is equal to the stripe-unit size.
- minpgahead of 2.
- maxpgahead of 16 times the number of disk drives. This causes page-ahead to be done in units of the stripe-unit size (64 KB) times the number of disk drives, resulting in the reading of one stripe unit from each disk drive for each read-ahead operation.
- I/O requests for 64 KB times the number of disk drives. This is equal to the maxpgahead value.

- Modify maxfree to accommodate the change in maxpgahead. The maxfree value should be greater than minfree by at least 8 or maxpgahead, whichever is greater.
- 64-byte aligned I/O buffers. If the logical volume will occupy physical drives that are connected to two or more disk adapters, the I/O buffers used should be allocated on 64-byte boundaries. This avoids having the LVM serialize the I/Os to the different disks. The following code would yield a 64-byte-aligned buffer pointer:

  ```
  char *buffer;
  buffer = malloc(MAXBLKSIZE+64);
  buffer = ((int)buffer + 64) & ~0x3f;
  ```

The vmtune syntax to accomplish the above recommendations could be:

```
# vmtune -r 2 -R 48 -F 168
vmtune:  current values:
  -p       -P        -r          -R         -f       -F       -N        -W
minperm  maxperm  minpgahead maxpgahead  minfree  maxfree  pd_npages maxrandwrt
   3072    12288       2          8        120      128     524288        0

  -M       -w       -k       -c         -b          -B          -u
maxpin   npswarn  npskill  numclust  numfsbufs   hd_pbuf_cnt  lvm_bufcnt
  13108     1024      256        1        93           64           9

number of valid memory pages = 16384    maxperm=75.0% of real memory
maximum pinable=80.0% of real memory    minperm=18.8% of real memory
number of file memory pages = 2295      numperm=14.0% of real memory

vmtune:  new values:
  -p       -P        -r          -R         -f       -F       -N        -W
minperm  maxperm  minpgahead maxpgahead  minfree  maxfree  pd_npages maxrandwrt
   3072    12288       2         48        120      168     524288        0

  -M       -w       -k       -c         -b          -B          -u
maxpin   npswarn  npskill  numclust  numfsbufs   hd_pbuf_cnt  lvm_bufcnt
  13108     1024      256        1        93           64           9

number of valid memory pages = 16384    maxperm=75.0% of real memory
maximum pinable=80.0% of real memory    minperm=18.8% of real memory
number of file memory pages = 2295      numperm=14.0% of real memory
```

For this example, consider a striped logical volume spread across three physical volumes. The -r option sets the minpgahead value to 2 (default value). The -R option changes the value of maxpgahead to 48 (16 * number of disk drives). The -F option is used to increase maxfree in order to accommodate the new value for maxpgahead.

- **lvm_bufcnt**

  If the striped logical volumes are on raw logical volumes and writes larger than 1.125 MB are being done to these striped raw logical volumes, increasing the lvm_bufcnt parameter of vmtune might increase throughput of the write activity. This parameter specifies the number of LVM buffers for raw physical I/Os. It would take very large I/O combined with very fast I/O devices to cause the bottleneck to be the LVM layer. The default value is 9, and the valid range is between 1 and 64. It can be changed with the -u option and is only available in AIX V4.

4.11.3 Tuning Write-Behind

Write-behind involves asynchronously writing modified pages in memory to disk rather than waiting for the syncd daemon to flush the pages to disk. There are two types of write-behind: sequential and random. Random write-behind was introduced in AIX V4.1.3, while sequential write-behind has always been available.

Sequential write-behind initiates I/O for pages if the VMM detects that writing is sequential. In order to increase write performance, limit the number of dirty file pages in memory, reduce system overhead, and minimize disk fragmentation, the file system divides each file into 16-KB partitions, or four pages. Each of these partitions is called a cluster. The pages of a given partition are not written to disk until the program writes the first byte of the next 16-KB partition. At that point, the file system forces the four dirty pages of the first partition to be written to disk. This helps preventing I/O bottlenecks and fragmentation of the file. The pages of data remain in memory until their frames are reused, at which point no additional I/O is required. If a program accesses any of the pages before their frames are reused, again no I/O is required.

- **numclust**

  The numclust parameter, which can be changed with the -c option from vmtune, can be used to specify the number of 16-KB clusters to be processed by the sequential write-behind algorithm. The default value is 1, and any integer greater than 0 is valid. Increasing numclust results in delaying the write-behind algorithm and may result in better sequential write performance on devices that support very fast writes. Setting the value to a very high number like 500000 will essentially defeat the write-behind algorithm.

If a large number of dirty file pages remain in memory and do not get reused, the syncd daemon writes them to disk, which might result in abnormal disk utilization. To distribute the I/O activity more efficiently across the workload, random write-behind can be turned on to tell the system how many pages to keep in memory before writing them to disk. This causes pages to be written to disk before the syncd daemon runs; thus, the I/O is spread more evenly throughout the workload.

- **maxrandwrt**

  The maxrandwrt parameter, which can be changed with the -W option from vmtune, can be used to specify a threshold (in 4-KB pages) for random writes to be accumulated in memory before these pages are written to disk via the write-behind algorithm. This threshold is on a per-file basis. The default value is 0, which disables random write-behind. By enabling random write-behind (a typical value might be 128), applications that make heavy use of random writes can get better performance due to less of a dependence on the syncd daemon to force writes out to disk.

The vmstat command can be used to verify if the system could benefit from the write-behind algorithm. If the output shows high page outs from vmstat -s and I/O wait spikes on regular vmstat intervals (usually when the syncd daemon is writing pages to disk), adjusting the maxrandwrt value will help spread the I/O more efficiently. Refer to 2.1, “The vmstat Command” on page 7, for more information on the vmstat command.

Note that some applications may degrade their performance due to write-behind, such as database index creations. In these cases, it may be beneficial to disable

write-behind before creating database indexes and then reenabling write-behind after the indexes are created.

4.11.4 Tuning Paging-Space Thresholds

There are two parameters that set the thresholds of when the system paging space is running low that can be set through vmtune:

- **npswarn**

  Specifies the number of free paging-space pages at which AIX begins sending the SIGDANGER signal to processes. If the npswarn threshold is reached and a process is handling this signal, the process can choose to ignore it or do some other action like exit or free-up memory using disclaim(). The default value in AIX V3 is 512 pages. The formula to determine the default value in AIX V4 is:

  ```
  npswarn = maximum (512, 4*npskill)
  ```

  The value of npswarn has to be greater than zero and less than the total number of paging space pages on the system. It can be changed with option -w from vmtune.

- **npskill**

  Specifies the number of free paging-space pages at which AIX begins killing

  processes. If the npskill threshold is reached, then the current policy is to send a SIGKILL signal to the youngest process; this policy has changed over AIX releases and may be enhanced in the future. Processes that are handling SIGDANGER or processes that are using the early page space allocation (paging space is allocated as soon as memory is requested) are exempt from being killed. The default value in AIX V3 is 128 pages. The formula to determine the default value of npskill in AIX V4 is:

  ```
  npskill = maximum (64, number_of_paging_space_pages/128)
  ```

  The npskill value has to be greater than zero and less than the total number of paging space pages on the system. It can be changed with option -k from vmtune.

4.11.5 Miscellaneous I/O Tuning Parameters

These are the other parameters that can be altered with the vmtune command:

- **maxpin**

  Specifies the maximum percentage of real memory that can be pinned. The default value is 80 percent. If this value is changed, the new value should ensure that at least 4 MB of real memory will be left unpinned for use by the kernel. The value for the maxpin parameter must be greater than one and less than 100. The field maximum pinable at the end of the output of vmtune is the maxpin value converted to a percentage. It can be changed with option -M.

- **numfsbufs**

  Specifies the number of file system bufstructs. The default value is 64 in AIX V3. The current default in AIX V4 is 93 and it is dependent on the size of the buf struct. This value must be greater than zero. Increasing this value will help write performance for very large writes sizes (on devices that support very fast writes). In order to enable this value, a file system has to be unmounted and mounted again after changing the value with option -b.

- **hd_pbuf_cnt**

  Controls the number of pbufs available to the LVM device driver. The pbufs are pinned memory buffers used to hold I/O requests related to a journaled file system. Do not set the value too high since it cannot be lowered without a system reboot. The maximum value is 128. It can be changed with option -B.

  In AIX V3, one pbuf is required for each page being read or written. On systems where large amounts of sequential I/O occurs, this can result in an I/O bottleneck at the LVM layer waiting for pbufs to be freed. The default value in AIX V3.2+ will be at least 64. Add 16 for each additional physical disk with an open logical volume on it, excluding the first disk. For example, a system with three disks will have 96 as the hd_pbuf_cnt value.

  In AIX V4, a single pbuf is used for each sequential I/O request regardless of the number of pages in that I/O. Therefore, it is harder to encounter this type of bottleneck. The number of pbufs usually has a default value of 80 because of the way memory is requested.

- **pd_npages**

  Specifies the number of pages that should be deleted in one chunk from RAM when a file is deleted. Changing this value may only be beneficial to real-time applications that delete files. By reducing the value of pd_npages, a real-time application can get better response time since few number of pages will be deleted before a process/thread is dispatched. The default value is the largest possible file size divided by the page size (currently 4096); if the largest possible file size is 2 GB, then pd_npages is by default 524288. It can be changed with option -N. This option is only available in AIX V4.

Chapter 5. Performance Toolbox

This chapter describes the Performance Toolbox for AIX licensed product. The purpose of this chapter is a brief introduction of the tool and its concepts. The main focus is on the usage and customization of this tool.

5.1 Introduction

The Performance Toolbox (PTX) is a comprehensive tool for monitoring and tuning system performance. PTX uses the client/server model to monitor local and remote system performance with several graphical windows that are fully user configurable. It is a Motif-based toolbox that includes 2D and 3D views of performance statistics. Analysis and tuning facilities incorporate existing performance tools into a menu-driven environment.

5.2 Performance Toolbox Concepts

PTX for AIX consists of two major components: the agent and the manager. The agent consists of programs that run on the machine or machines you want to monitor and its role is to obtain and filter performance statistics. The manager has the tools to manipulate the data providing meaningful statistics such as:

- Monitoring of system resource statistics
- Analysis of system resource statistics
- Tuning of the systems performance parameters to balance the utilization of fixed resources

The client/server implementation of PTX is an excellent aid to monitor and tune the performance of various UNIX systems in a network environment from a single graphics workstation.

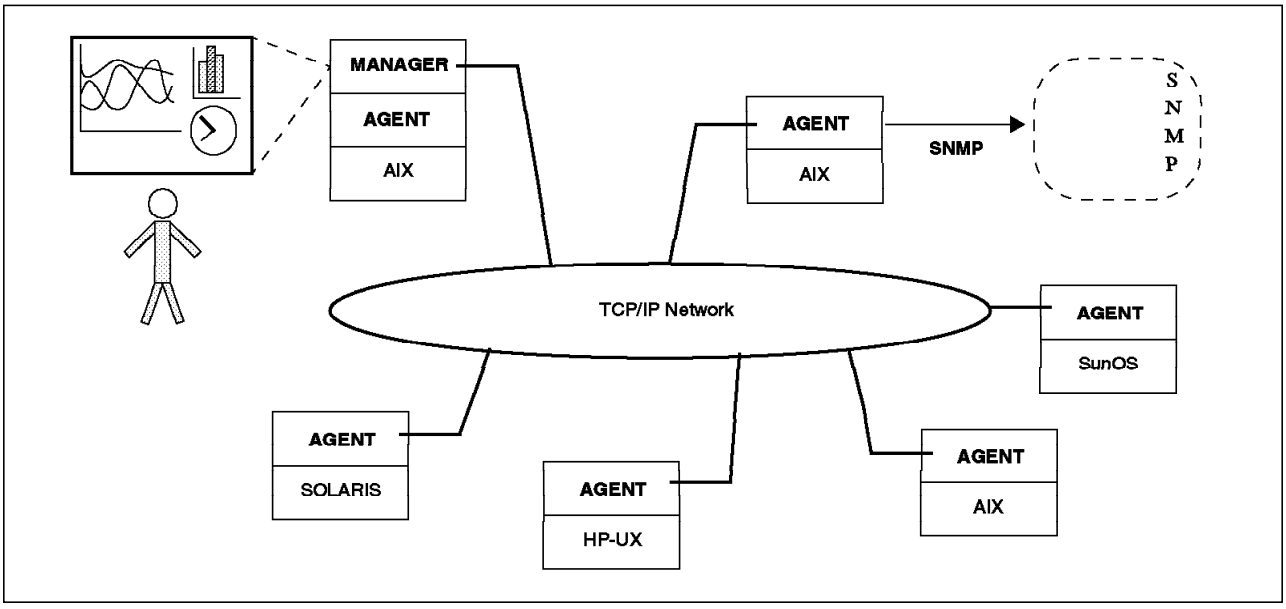


Figure 15. The Performance Toolbox Environment

While you run the manager and the agent on the same machine, it is highly recommended that the manager not run on the system to be monitored. This will minimize the impact of running PTX on the system load and will give a true indication of system performance. Be aware that the manager puts a significant load on the system.

PTX major features are:

- Separately installable manager and agent components that allow a manager to monitor multiple agents and an agent to supply data to multiple managers.
- HP-UX (V9.03), SunOS (V4.1.3) and Solaris (V2.3, V2.4 and V2.5) agents to allow monitoring of performance data on OEM machines.
- Application programming interfaces (API) that allow programmers to access local or remote data as well as to register custom data with the local agent.
- Ability to respond to SNMP requests and to send traps to an SNMP manager (AIX agents only).
- Support for RS/6000 SP systems using the Performance Toolbox Parallel Extensions Feature of the Parallel System Support Programs (PSSP) for AIX/6000.

Performance Toolbox Parallel Extensions (PTPE) is a feature of PSSP 2.2 and is used in conjunction with PTX to simplify the performance analysis and reduce RS/6000 SP administrative overhead by organizing your SP nodes into reporting groups. Each node sends performance data to a manager node, which performs the administrative tasks for the group. It also provides the utilities to monitor, store and retrieve performance data collected on RS/6000 SP nodes about their resources, such as the switch, virtual shared disk and LoadLeveler.

5.3 Benefits of Using Performance Toolbox

There are a number of benefits derived from using PTX, including:

Monitoring System Performance: PTX/6000 provides an easy way of monitoring your local as well as your remote systems, including the performance of the network, which is very important in the SP environment. In network client/server environments, the performance of sets of systems working together can be as important as the performance of an individual system. Likewise, the performance of multiple applications working together can be as important as that of an individual application. Therefore, it is very important to be able to get the “big picture” by graphically viewing many correlated parameters concurrently across multiple nodes in a network. PTX/6000 allows a user to concurrently visualize the live (near real time) performance characteristics of the clients and server applications across the network.

Analysis and Control of System Performance: By providing an umbrella for tools that can be used to analyze performance data and control system resources, the manager program `xmperf` assists the system administrator in keeping track of available tools and in applying them in appropriate ways. This is done through a customizable menu interface. Tools can be added to menus, either with fixed sets of command line arguments to match specific situations or in a dialog window. The menus of `xmperf` are preconfigured to include most of the performance tools shipped as part of the tools option of the agent component.

All performance-related tools already available in AIX can be accessed through this interface. In addition, the ability to record load scenarios and play them back in graphical windows at any desired speed gives ways of analyzing a performance problem.

Features for analyzing a recording of performance data are provided by the azizo program and its support programs. Recordings can be produced from the monitoring programs xmperf and 3dmon during monitoring, or can be created by the xmservd daemon. The xmservd daemon allows for recording with a minimum of overhead. This makes constant recording possible so that you can analyze performance problems after they occurred.

Finally, using the agent component filter filtd, you can define conditions that, when met, could trigger any action you deem appropriate, including alerting yourself and/or initiating corrective action without human intervention. This facility is entirely configurable so that alarms and actions can be customized to your installation.

Capacity Planning: If you can make your system simulate a future load scenario, xmperf can be used to visualize the resulting performance of your system. By simulating the load scenario on systems with more resources, such as more memory or more disks, the result of increasing the resources can be demonstrated.

Network Operation: The xmservd data supplier daemon can provide consumers of performance statistics with a stream of data. Frequency and contents of each packet of performance data are determined by the consumer program. Any consumer program can access performance data from the local host and one or more remote hosts. Any data supplier daemon can supply data to multiple hosts.

SNMP Interface: By entering a single keyword in a configuration file, the data supplier daemon can be told to export all its statistics to a local snmpd SNMP agent. Users of an SNMP manager, such as IBM NetView, see the exported statistical data as an extension of the set of data already available from snmpd.

Note: The SNMP multiplex interface is only available on IBM RS/6000 agents.

5.4 Manager

The manager component of the Performance Toolbox for AIX has the following components:

xmperf The main interface program providing graphical display of local and remote performance information in a menu interface to commands of your choice.

3dmon A program that can monitor up to 576 statistics simultaneously and display the statistics in a 3D graph.

3dplay A program to playback 3dmon recordings in a 3dmon-like view.

chmon Supplied as an executable and in source form. This program allows monitoring of vital statistics from a character terminal.

exmon A program that allows monitoring of alarms generated by the filtd daemon running on local or remote hosts.

azizo A program that allows you to analyze any recording of performance data. It lets you zoom in on sections of the recording and provides graphical as well as tabular views of the entire recording or zoomed-in parts.

ptxtab A program that can format recording files for printed output.

ptxmerge This program allows you to merge up to 10 recording files into one. For example, you could merge xmservd recordings from the client and server sides of an application into one file to better correlate the performance impact of the application on the two sides.

The xmperf Program: The xmperf program is the most comprehensive and largest program in the manager component. It is an X Window-system-based program developed with the OSF/Motif toolkit. The xmperf program allows you to define monitoring environments to supervise the performance of the local AIX system and remote AIX, SUN or HP-UX systems.

Each monitoring environment consists of a number of consoles. Consoles show up as graphical windows on the display. Consoles, in turn, contain one or more instruments, and each instrument can show one or more values that are monitored.

The following terms are used to refer to the xmperf monitoring functions or components:

- A *Console* is a graphical window containing instruments that monitor the system. A console can have one or more instruments.
- An *Instrument* is a graphical view of monitored values, and each instrument can show one or more values that are monitored. The presentation of the values can be in form of graphs, gauges and so on.
- A *Value* is the unit to be monitored. It can be any piece of the system able to be monitored, for example, CPU usage for user processes, disk transfer rates or TCP bytes transmitted.
- *Groups of Statistics* are a functional part of the system. The values are grouped in relation to the functional part of the system they belong to. For example, CPU global user, kernel, wait and idle percentages are a Group of CPU Statistics. However, an instrument can have values from several groups.

5.5 Agent

The agent component is a collection of programs that make it possible for a host to act as a provider of performance statistics across a network or locally. The key program is the daemon xmservd. It supplies the statistics for the monitoring environment through an API called System Performance Measurement Interface (SPMI). The SPMI implementation allows one agent to supply data to many managers, and one manager to request data from many agents. The SPMI interface can be used for any dynamic data supplier program to export their data.

Another agent's program is filtd. This daemon is in charge of filtering data by processing all previously defined expressions that define new statistics. This feature allows to easily combine existing "raw" statistics into new statistics that make more sense in the monitoring environment. The filtd also supplies exmon

with the exceptions that occurred in the monitored system and allows you to define alarms that trigger actions.

The xmservd daemon also acts as supplier of performance statistics to Simple Network Management Protocol (SNMP) managers like NetView. This feature is only available in RS/6000 systems.

5.6 Useful Information

The UNIX platforms that PTX/6000 supports are RS/6000, which includes the SMP and SP environments, Sun (SunOS 4.1.3), Solaris (Sun Solaris 2.3, 2.4 and 2.5), or HP (HP-UX 9.03) environments. You need to install the following filesets on either the agent or manager as listed below for PTX 2.2:

perfagent.server	on the agent
perfagent.tools	on the agent
perfmgr.common	on the manager
perfmgr.local	on the manager, only if you want to monitor your local system
perfmgr.network	on the manager, only if you want to monitor remote systems

To check that you have the filesets installed on either the agent or manager machine:

For the manager: lslpp -l perfmgr.* will provide a list of the installed filesets.

For the agent: lslpp -l perfagent.* will provide a list of the installed filesets.

The fileset perfagent.server contains among other programs, the xmservd daemon. This daemon must be installed and running on all monitored systems. The fileset perfagent.tools contains all the performance tools such as rmss, filemon, netpmon, svmon, lockstat, tprof, fileplace, and others. The perfmgr filesets contain the graphical part of PTX. The commands xmperf and 3dmon are part of this fileset.

Remote AIX, HP-UX 9.03, SunOS 4.1.3, and Solaris 2.3, 2.4 and 2.5 systems can be monitored with the remote option. HP and Sun data-supplier daemons belong to the agent.

Perfagent on Hewlett-Packard 9000/700: The agent code is tested on HP-UX 9.01 and 9.03. For proper disk statistics on HP-UX 9.02, patch number 3325 must be applied. HP-UX 9.03 already includes this patch.

To install the agent, copy the file /usr/lpp/perfagent/hp/hpinstall.tar.Z to a working directory on the HP system. While logged in as root, uncompress and untar the file, and run the script /usr/bin/perfagent.install.scr.

The statistics available from the HP system are a subset of those available on AIX systems. In some cases, the statistics are slightly different, though comparable, to the AIX statistics. The agent code for HP 9000/700 series machines is not guaranteed to work on MP systems, but probably will.

Perfagent on Hewlett-Packard 9000/800: The agent code is developed on HP-UX 9.00. The code has not been through a formal test cycle and is shipped as-is.

To install, copy the file /usr/lpp/perfagent/hp/hp800install.tar.Z to a working directory on the HP system. While logged in as root, uncompress and untar the file, and run the script `/usr/bin/perfagent.install.scr`.

The statistics available from the HP system are a subset of those available on AIX systems. In some cases, the statistics are slightly different, though comparable, to the AIX statistics. The agent code for HP 9000/800 series machines has been tested on a two-processor MP system.

If the HP 9000/800 machine has a remote file system hard-mounted via NFS and that remote file system is unreachable, the SPMI interface may not initialize properly. Attempts to run any program using the SPMI may cause the program to hang and common shared memory to be only partly initialized. The only known way to cope with this is to clean the shared memory areas and remove the unresolved mount or make the remote file system accessible; then retry the program.

Perfagent on Sun SPARCstations: The following versions of Perfagent are available for Sun Microsystem's SPARCstations and SPARCservers. For each version, the corresponding compressed tar file is listed, and the type of system that was used to develop the version:

SunOS V 4.1.3	SunOSinstall.tar.Z	SPARCstation 2
Solaris V 2.3	Solaris23install.tar.Z	SPARC Classic
Solaris V 2.4	Solaris24install.tar.Z	SPARCstation 10
Solaris V 2.5	Solaris25install.tar.Z	SPARCstation 4

Table 4. Table of Perfagents for Sun SPARCstations

Because of the changes to the kernel and libraries between versions of Solaris, you must install the correct version to get correct results. Remember that a Solaris 2.5 system responds to "`uname -a`" with a message like:

```
SunOS <hostname> 5.5 Generic sun4m sparc SUNW,SPARCstation-10
```

The agent code was developed to run on SMP (multiprocessor) systems, but has not been tested on such systems. IBM does not guarantee it will run, but it probably will.

To install, copy the appropriate file from the /usr/lpp/perfagent/sun directory to a working directory on the Sun system. While logged in as root, uncompress and untar the file, and run the script:

```
/usr/bin/perfagent.install.scr
```

The statistics available from the Sun system are a subset of those available on AIX systems. In some cases, the statistics are slightly different though comparable to the AIX statistics.

The `xmservd` daemon on a SunOS 4.1.3 system is NOT capable of recording `xmservd` statistics locally. If you specify a path name that begins with DDS/IBM/XMservd in the /etc/perf/xmservd.cf recording configuration file, the

xmservd daemon will abort with a segmentation fault. This restriction doesn't apply to the agent running under Solaris.

On Solaris platforms, LOCAL data-consumer programs may be interrupted while processing a SIGALRM signal. Such programs should issue a signal(SIGALRM, SIG_IGN) immediately after getting control in their timer signal-handler function and reissue the signal to what it was before exiting the timer signal-handling function. For an example of this, see the shipped sample program /usr/samples/perfagent/server/lchmon.c. The timer signal-handling function is feeding().

On Solaris 2.3, a data-supplier program can't issue the SpmiDdsInit call more than once. This seems to be related to the peculiar way the kstat_open() and kstat_close() functions are implemented and may be fixed in some patch to the operating system.

5.7 Using Performance Toolbox

In many cases, performance management starts at a high level and then zooms in on problem areas when they are reported, generally known as the "Top-down" approach or methodology. PTX/6000 is implemented in such a way that it can take advantage of this methodology, which will become clear as you go through the various facilities of PTX/6000. It is advisable to start with the default consoles provided, and as you become familiar with the product, you could start designing your own consoles.

The performance manager component of PTX/6000 can be started from the command line by entering the xmperf command or xmperf -h <hostname> if you want to monitor your system from a remote host. The two windows as shown below will appear on your screen. These windows are the manager main window and a console of which the name would be prefixed with your hostname. In this case, it was named *ah6000a: Mini Monitor*, ah6000a being the hostname in this case.

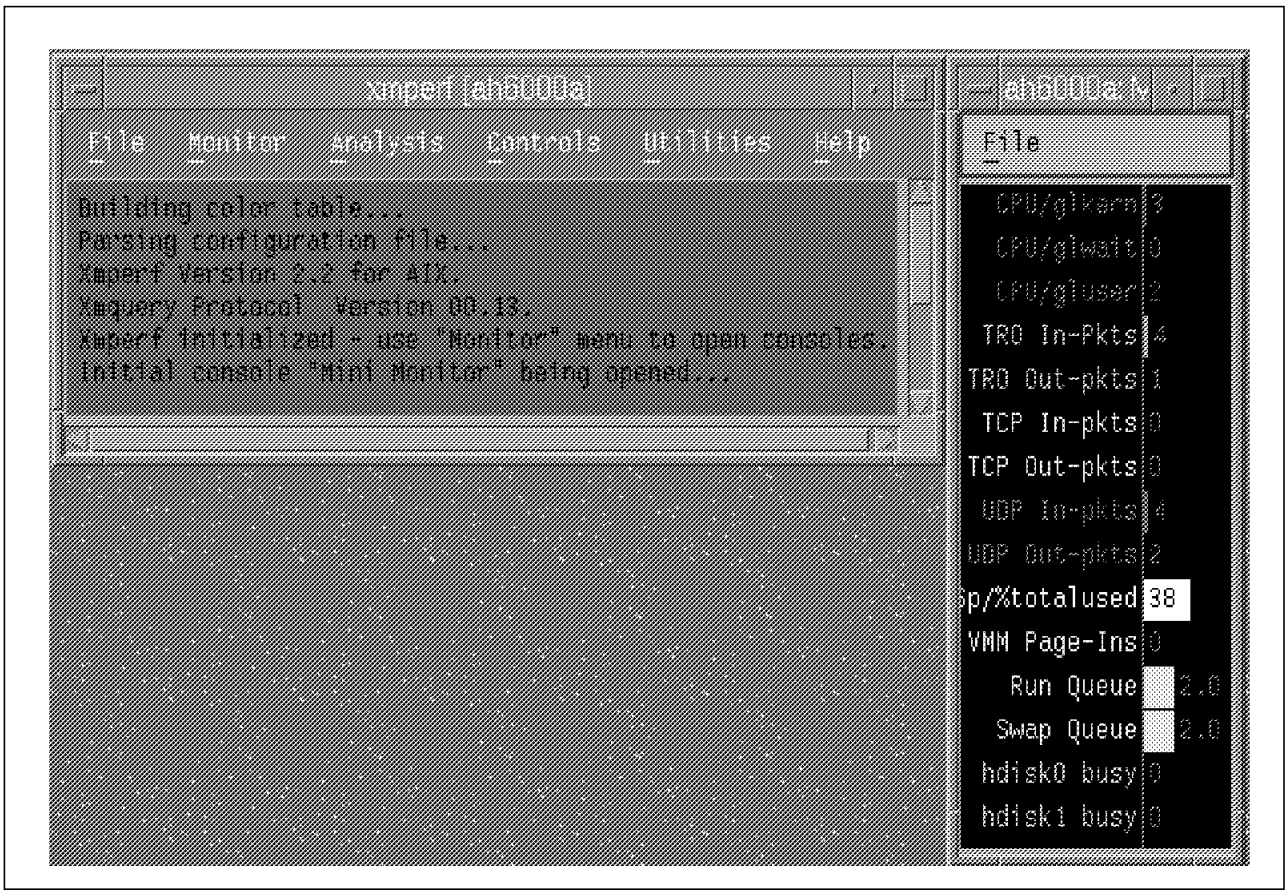


Figure 16. AIX Performance Toolbox Initial Screen

5.7.1 Manager Main Window

At the top of the xmperf main window is a menu bar that provides access to six pull-down menus.

File The menu from where you can exit xmperf, save your changes, start a playback, or refresh the host list.

Monitor The menu from where you open, close or create new consoles.

Analysis The first of the three tools menus from where commands can be executed. The menu and the tools are customizable, and it is included for commands that analyze performance. Each option has extensive help menus associated with it, and you are encouraged to use them.

Controls The second of the three tools menus from where commands can be executed. This menu has a fixed menu item that creates a list of processes in the local system. The rest of the menu is fully customizable, and it is intended to be used for commands that influence the performance of your RS/6000. Each option has extensive help menus associated with it, and you are encouraged to use them.

Utilities The last of the three tools menus from where commands can be executed. This menu has a fixed menu item that creates a list of processes on a remote system that you select. The rest of the menu is customizable, and it is intended to be used for miscellaneous tools and commands. The tools are customizable as well.

Help Provides online help about PTX/6000 and its utilities.

From the main menu, you can observe the names and syntax of the commands that are executed. The output will be presented in a scrollable window so that you can scroll back to see the complete output report, if any.

The console on the right-hand side of the main menu displays the CPU consumption (kernel, wait, user), the network view with the token-ring activity, TCP and UDP activity, the page-space utilization, page-ins, the run and swap queues, and how busy the disks are. These are defined in the configuration file xmperf.cf. You can add or delete the variables easily with the *Edit Console* option of your console window.

5.7.2 Creating a New Console

PTX provides predefined consoles for monitoring your UP or SMP system. You could either add a new console or use the skeleton consoles provided by PTX and modify those to suite your needs. In this example, the *Add New Console* option is used to create a console to monitor CPU statistics on a four-way SMP system.

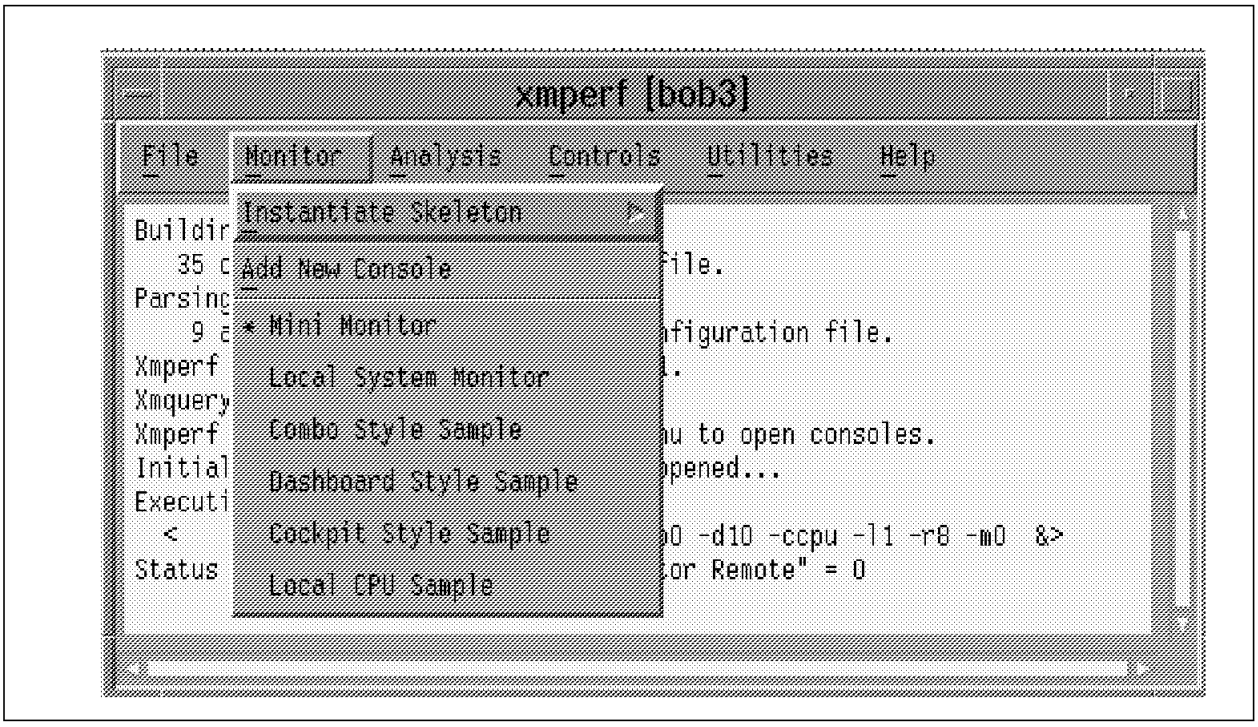


Figure 17. Creating a New Console (Step 1)

Select the **Monitor** pull-down menu to start adding a new console.

- The *Instantiate Skeleton* option contains console skeletons of the most common values used for performance monitoring grouped by categories. Each of these predefined consoles can be modified, and they are an easy and fast way to build a console.
- The *Add New Console* option lets you build a new console from scratch. You have to create your own instruments by selecting the system variables you want to see in the console.

Select the **Add New Console** option and you will get a subwindow inviting you to enter a console name. Give a meaningful name to your console instead of the default name. The default console name is built using the current time and date.

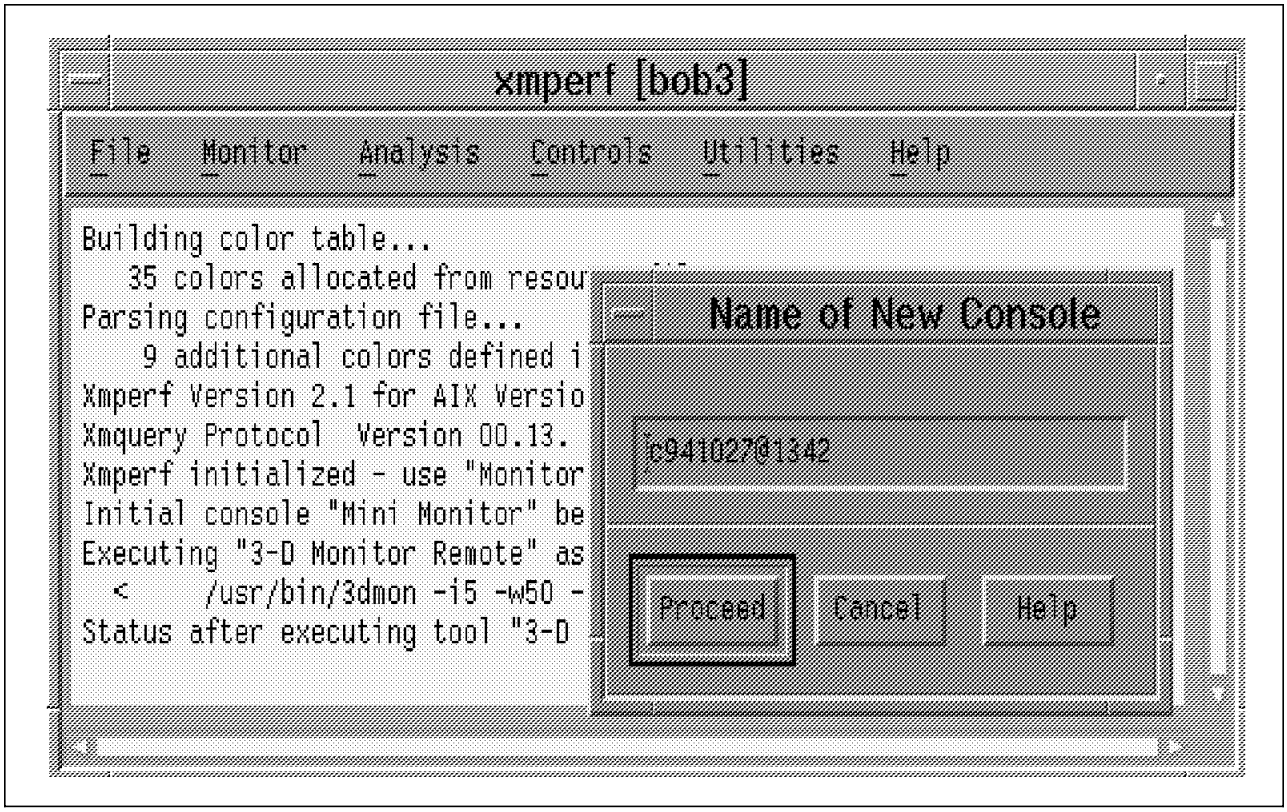


Figure 18. Creating a New Console (Step 2)

Click on the **Proceed** button to continue; you will then get another blank window. Use the *Edit Console* pull-down menu and select **Add Local Instrument**.

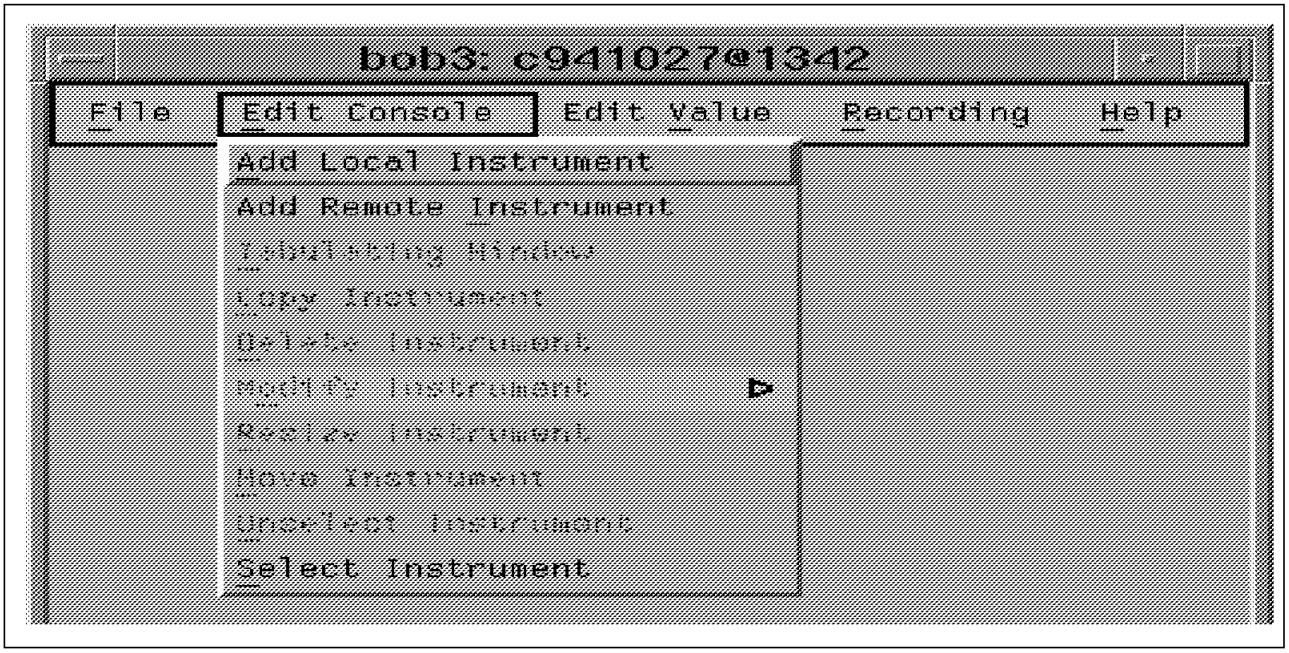


Figure 19. Creating a New Console (Step 3)

- *Add Local Instrument* means you are going to create a console that monitors your local system (the system where xmperf is running).
- *Add Remote Instrument* means you are going to monitor a remote host; the program will ask you to provide the hostname. This host must have xmservd running.

Note: The remote host can be any UP, SP or SMP RS/6000 system. As mentioned earlier, there are also xmservd versions for HP-UX, SunOS, and Solaris.

Having selected **Add Local Instrument**, you will get the following screen that will invite you to select the statistics you want to monitor within your console.

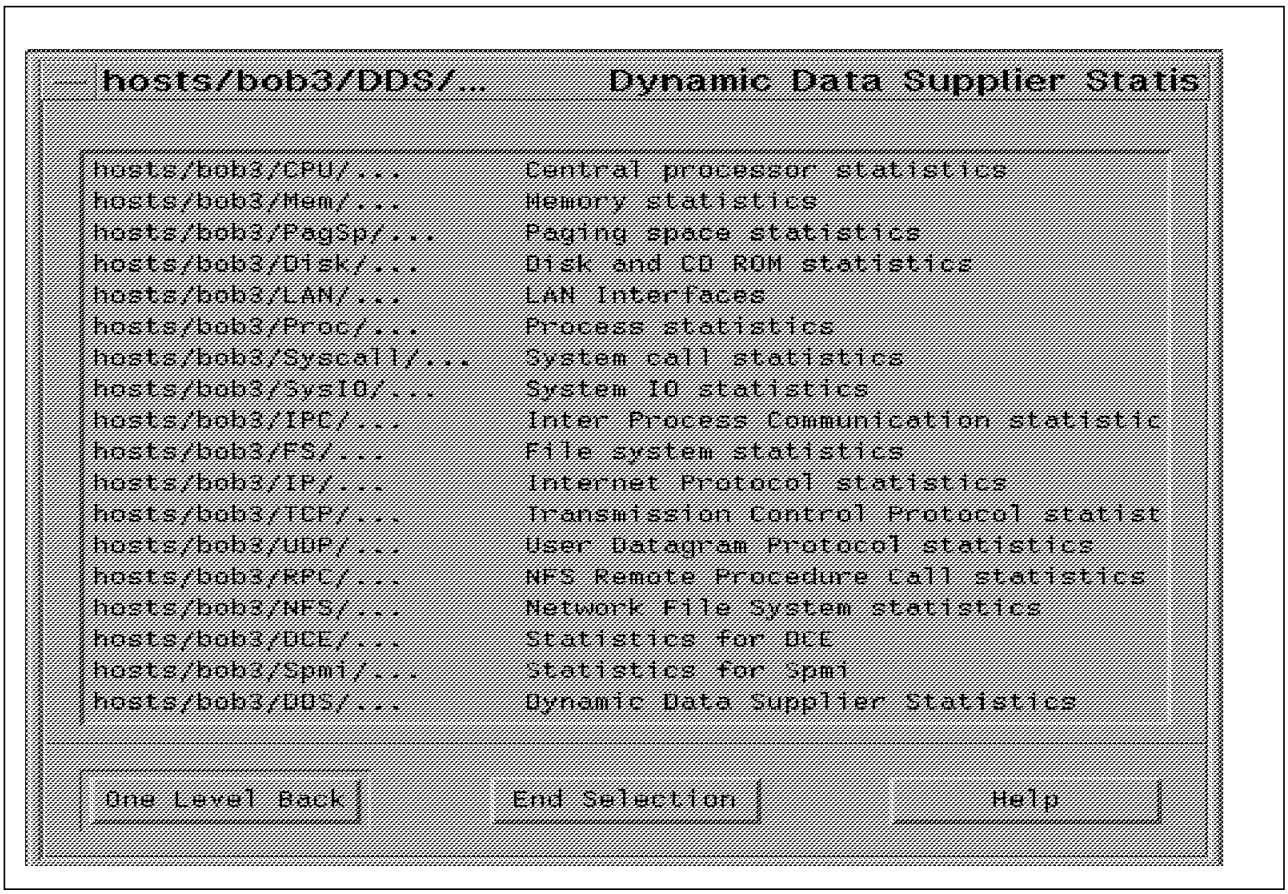


Figure 20. Creating a New Console (Step 4)

- This screen shows the whole set of statistics groups. Each group has variables that are common within a specific function of the system (CPU, disk, memory, TCP/IP, NFS, and so on).
- This is the sample list of the first level of data available. Whenever the name of the value is followed by '...', it means there is another window behind this one to select the fi nal name.

For this example, we want to monitor CPU statistics for an SMP; so we selected **Central processor statistics.**

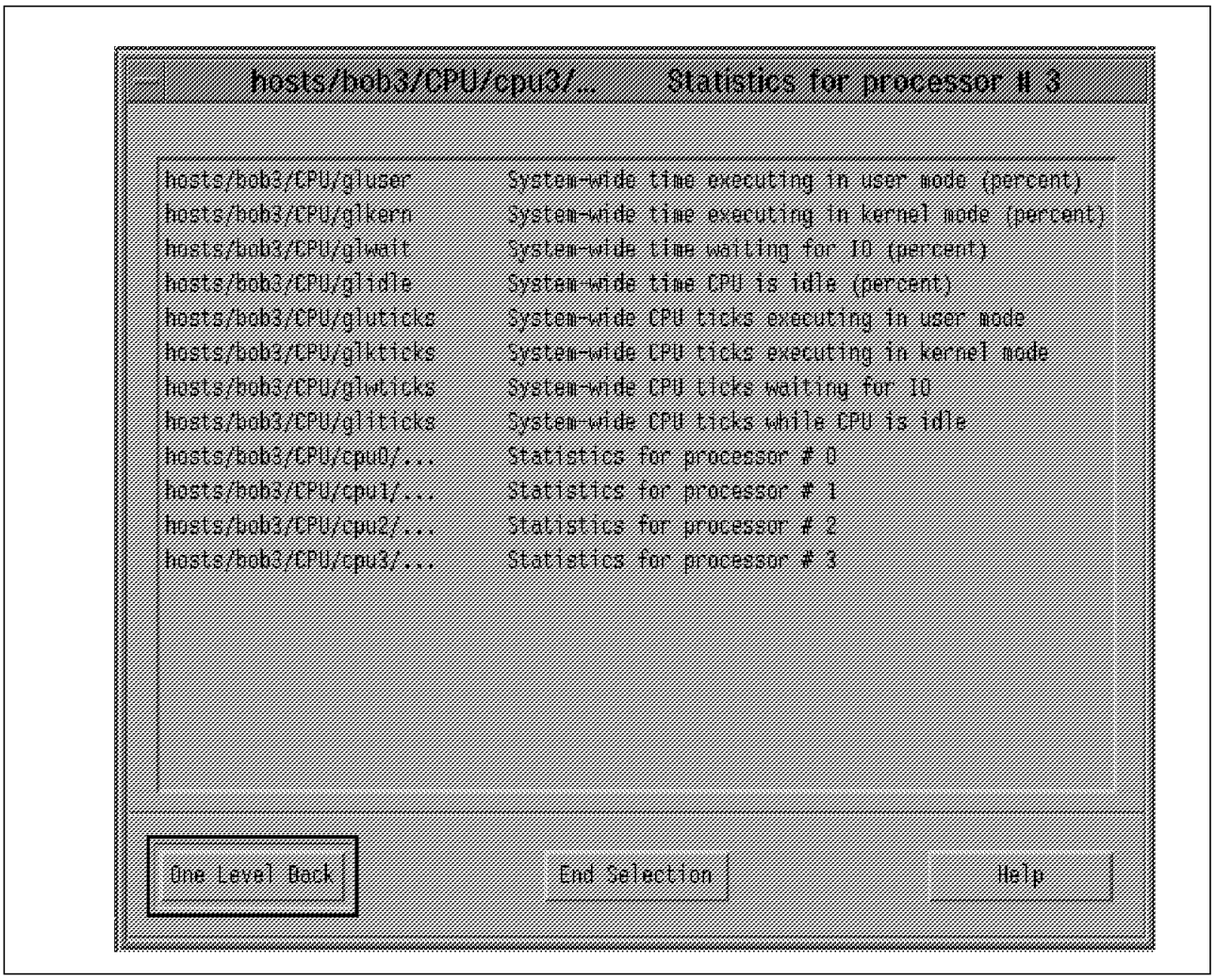


Figure 21. Creating a New Console (Step 5)

As you can see, there are several subgroups within the CPU group. The gl**...** subgroup has values that are global to the system and are the average for the values of each CPU. The cpu# subgroups (one per processor) have the values for a specific processor.

You can create a console with multiple CPU values by just selecting the first **cpu#** subgroup and adding the values you want to see in the instrument for that cpu#. Then go back to this screen and select another cpu# subgroup, and repeat the selection of values, and so on...

For this example, we selected the CPUs one after the other. We will show you cpu3 in the next figure.

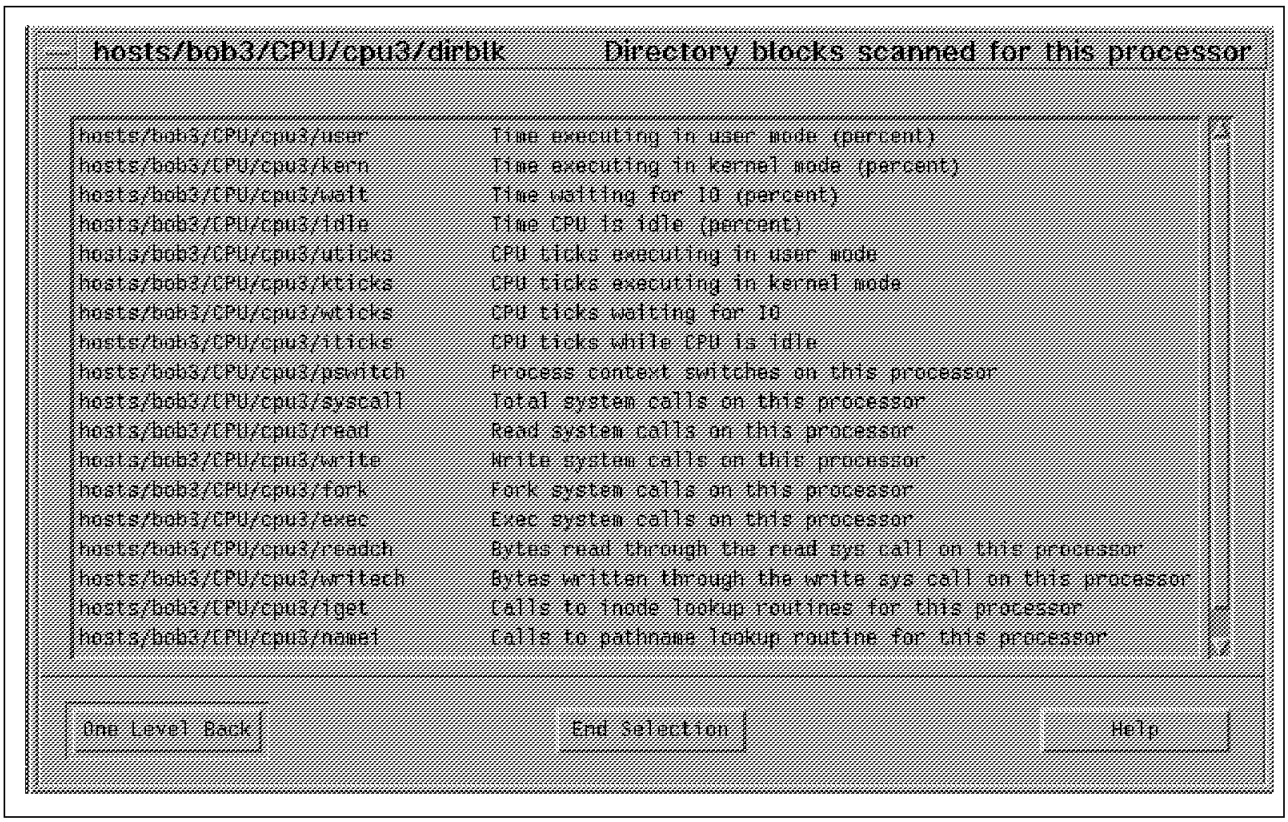


Figure 22. Creating a New Console (Step 6)

When you select the **cpu3** subgroup, a list of the values specific for that CPU is shown. You can add a maximum of 24 values for each instrument; the instrument can contain values from any group or subgroups. The instrument creation finishes when you click on the **End Selection** button.

For example, you can create an instrument showing the CPU percentage time in user mode for each processor in the same graph, or you can create several instruments in the same console showing each one the CPU kernel, user and wait percentages per processor. Each instrument represents a processor (cpu#). It all depends on your needs.

In this example, we selected kernel, user and wait CPU values because we want to show them all in one instrument per processor.

Note

You have to select one value after the other. You cannot select all three in the same window.

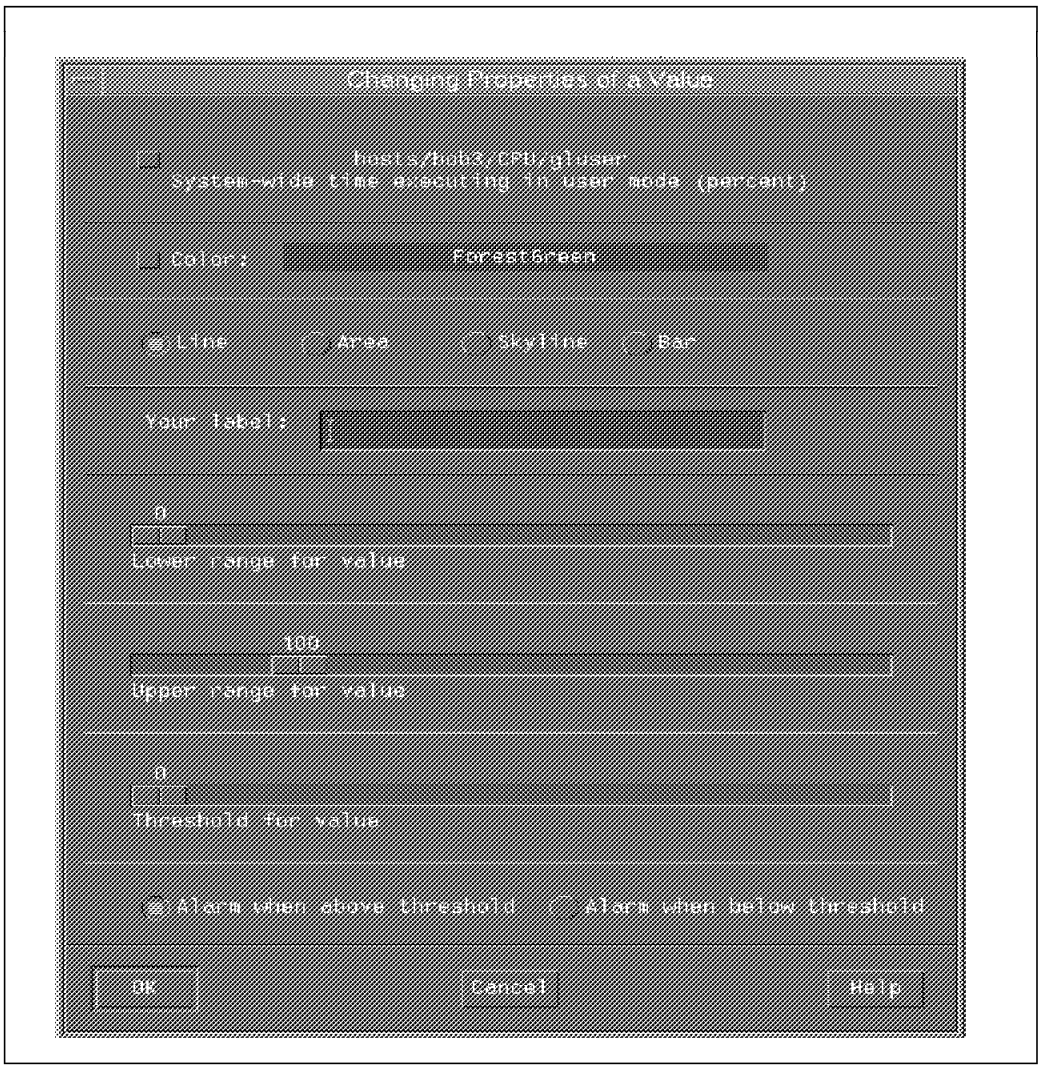


Figure 23. Changing the Properties of a Value

After each selection from the screen on the previous page, the screen above is displayed. You can then customize the properties of the value you selected, such as the color and the type of graph you want (line, area, bars). You will be able to set upper and lower limits, set a threshold, and set an alarm when this threshold is reached. Once you are satisfied with the property values, select **OK**.

Selecting OK takes you back to the screen on the previous page. You can then select the next value, which brings you back to the screen above. When you have selected all the values you require in your console, click on the **End Selection** button, and a console similar the one on the next page will be displayed.

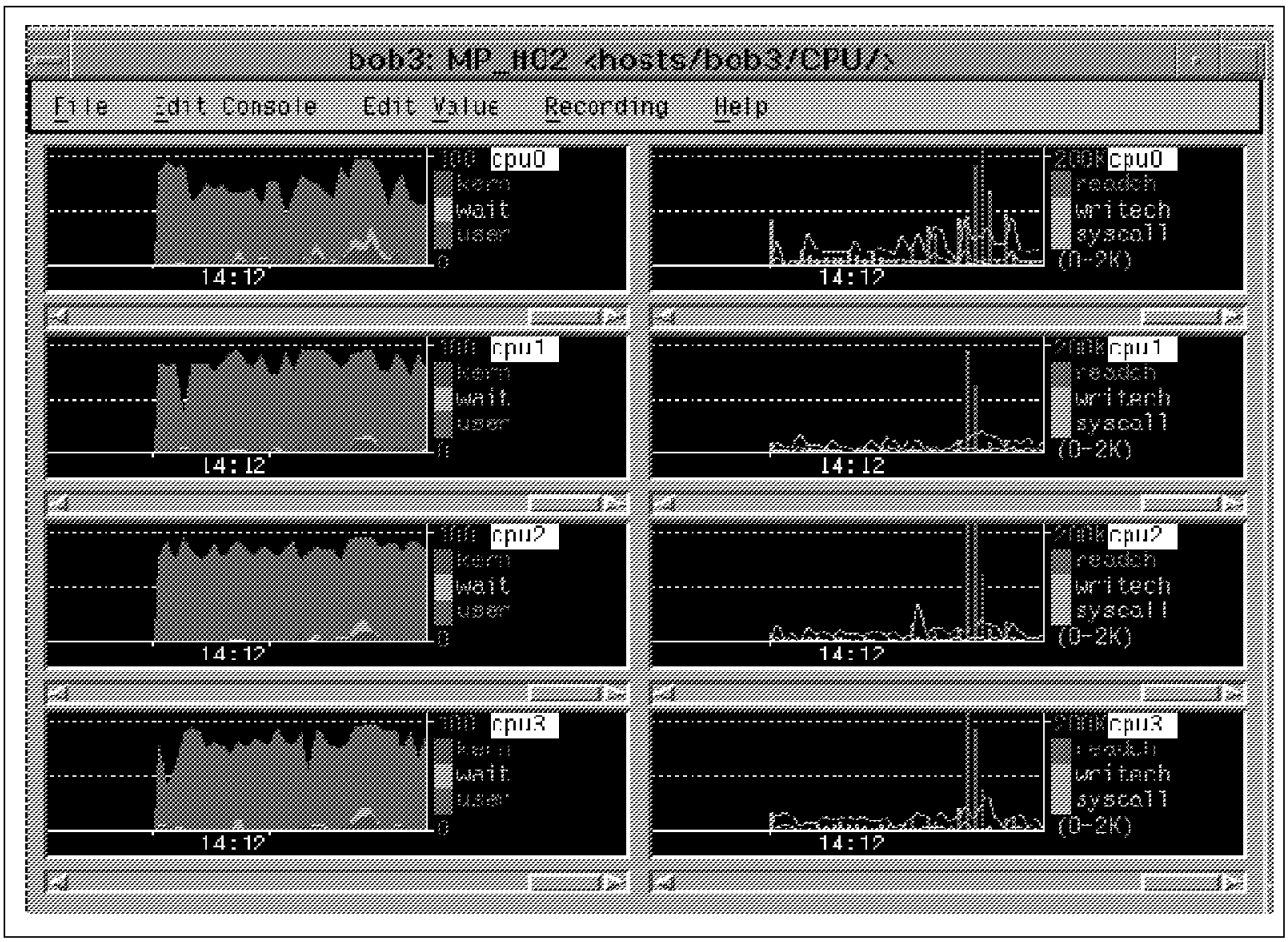


Figure 24. Created New Console

In this console, there are eight instruments, each one belongs to a specific CPU. Instruments in the first column compare CPU kernel, user and wait percentages of each of the processors. Instruments in the second column compare the number of readch, writech and syscall of the processors.

Since each instrument has values from a single subgroup, the graphic title shows this subgroup name (cpu#).

Hints

When you modify the xmperf setup, the changes get saved in the xmperf.cf file in your home directory. If you have created a set of consoles that you wish to use again, it is always advisable to make a copy of this file. You may also edit this file using a text editor to change console parameters. This is sometimes quicker than using the xmperf menus. For more details on configuring and using Performance Toolbox, please refer to the manual, *Performance Toolbox for AIX: Guide and Reference, Version 1.2 and 2,* SC23-2625.

5.7.3 Monitoring a Process

The procedure to create a process-specific console is similar to the procedure that was used to create a new console.

- Select the **Monitor** pull-down menu to start adding a new console. (See Figure 17 on page 209).
- Select the **Add New Console** option (See Figure 17 on page 209).
- When the *Name of New Console* menu is displayed, enter console name and click on the **Proceed** button to continue. (See Figure 18 on page 210).
- From the *Edit Console* pull-down menu, select **Add Local Instrument**. (See Figure 19 on page 211).

At this point, the procedure you need to follow is slightly different from the one used to create the console to monitor the CPU statistics on an SMP system.

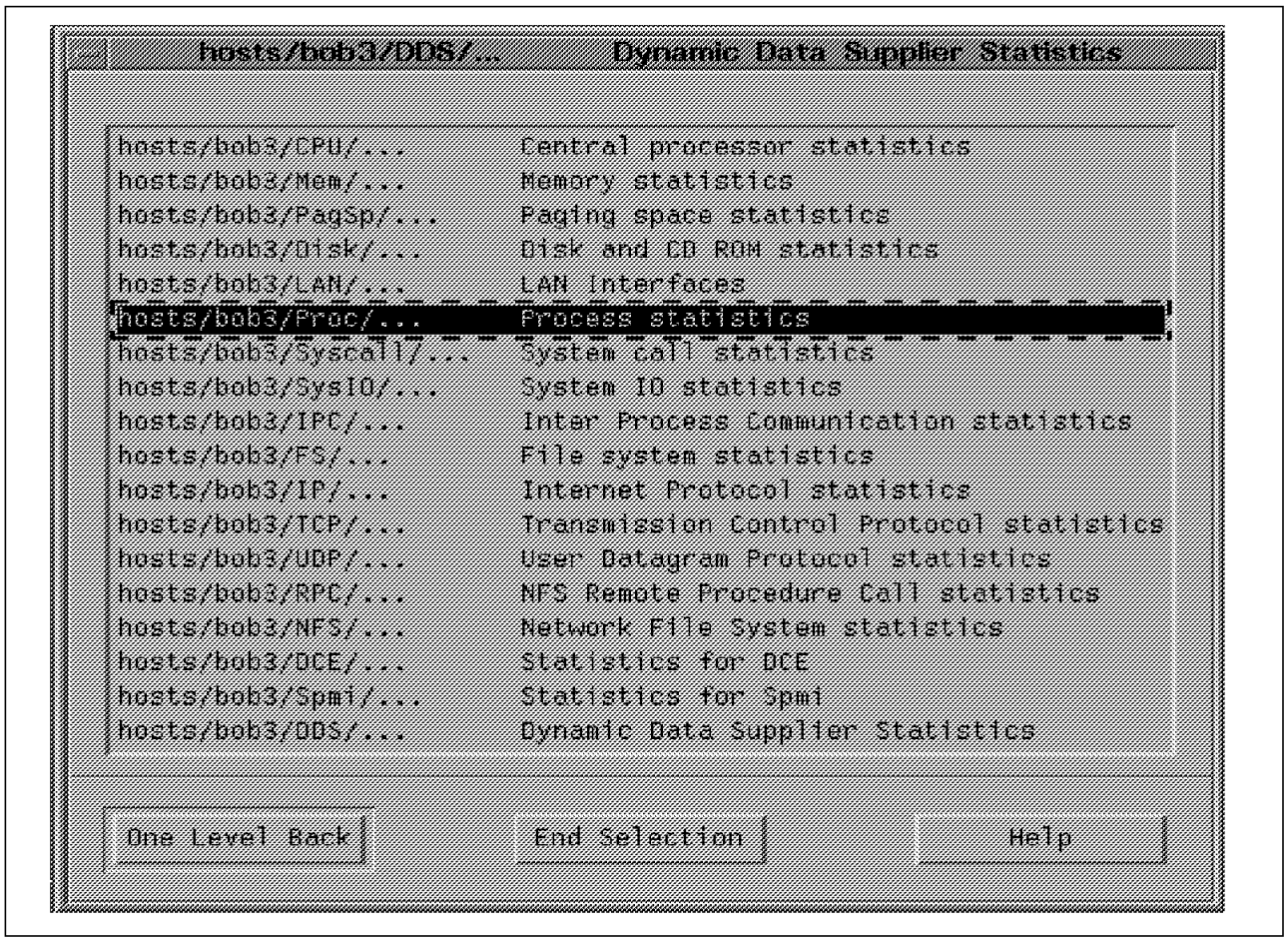


Figure 25. Monitoring a Process (Step 1)

The intention of this example is to see how a process-specific console shows those values that are affected because the process runs across several processors. At this point, select the **Process statistics**.

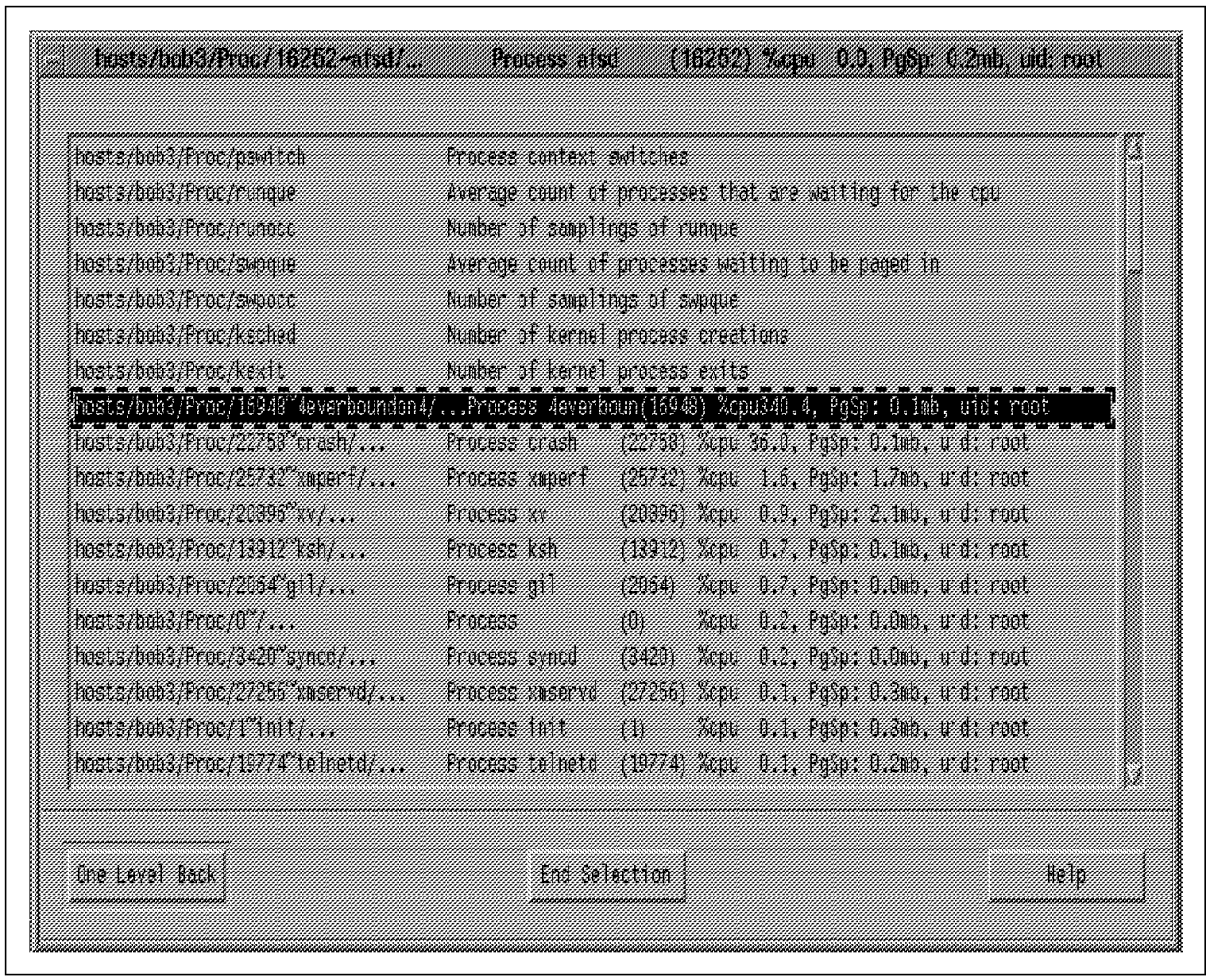


Figure 26. Monitoring a Process (Step 2)

This screen shows a list of all the processes that are running in the system. The list does not reflect threads and does not show a CPU ID. It only shows the process ID, the percentage of CPU usage, paging space and user ID who started the process. Note that the %cpu value for the selected process is 340.4 percent. This is so because the current process is a multithreaded process that runs across four processors at the same time. PTX just adds the %cpu the process is using in each of the processors and presents the total as the result.

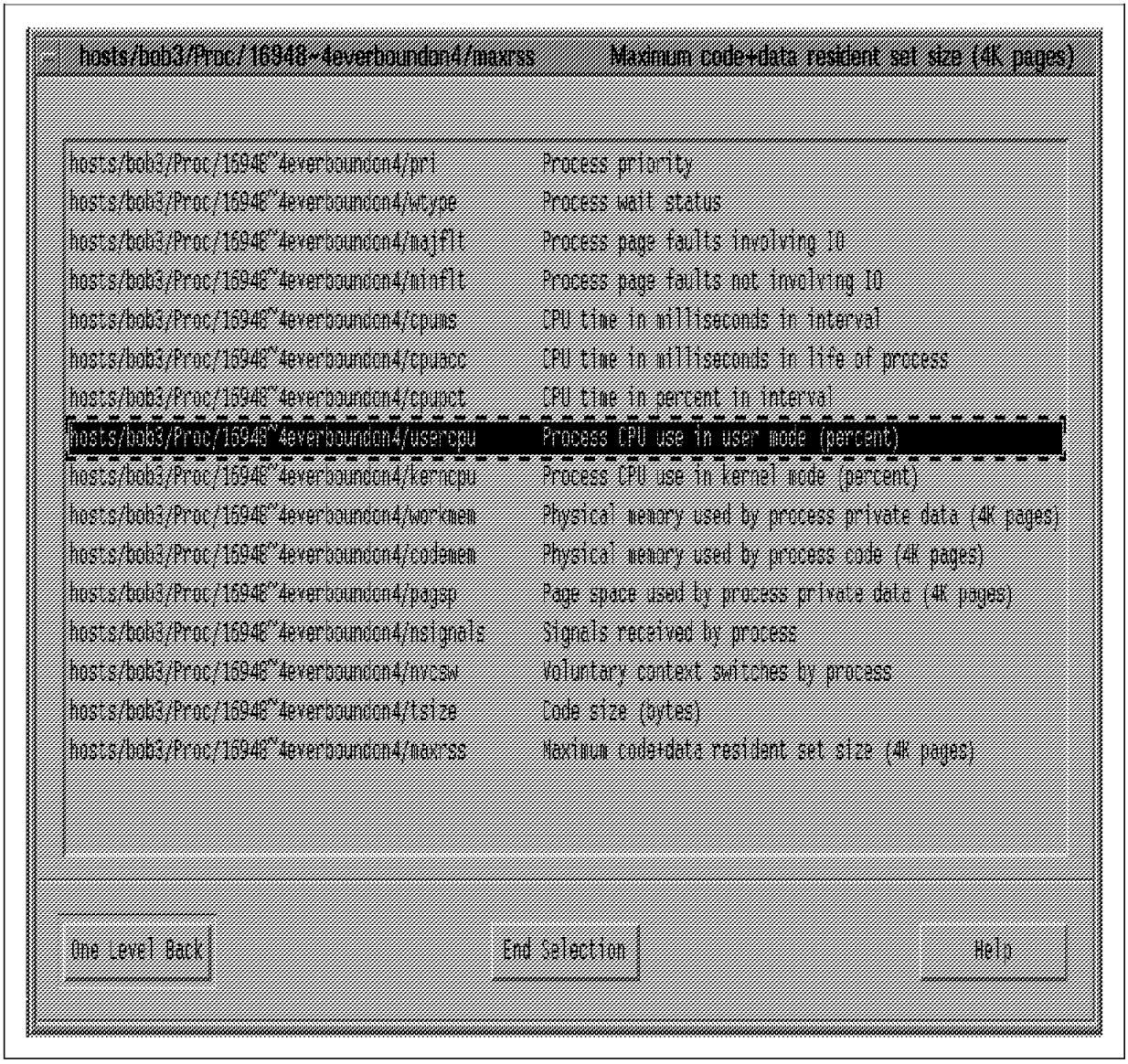


Figure 27. Monitoring a Process (Step 3)

The CPU percentage values are the ones we want to see in this example, and they are SMP related.

After selecting the value you wish to display in your console, in the example above we selected **usercpu**, a screen similar to Figure 23 on page 215 is displayed. This screen allows you to customize the properties of the value you selected, such as the color and the type of graph you want (line, area, bars). You will be able to set upper and lower limits, set a threshold, and set an alarm when this threshold is reached. Once you are satisfied with the property values, select **OK**.

Selecting **usercpu** and **kerncpu** and customizing their properties gives you a console similar to the one on the next page.

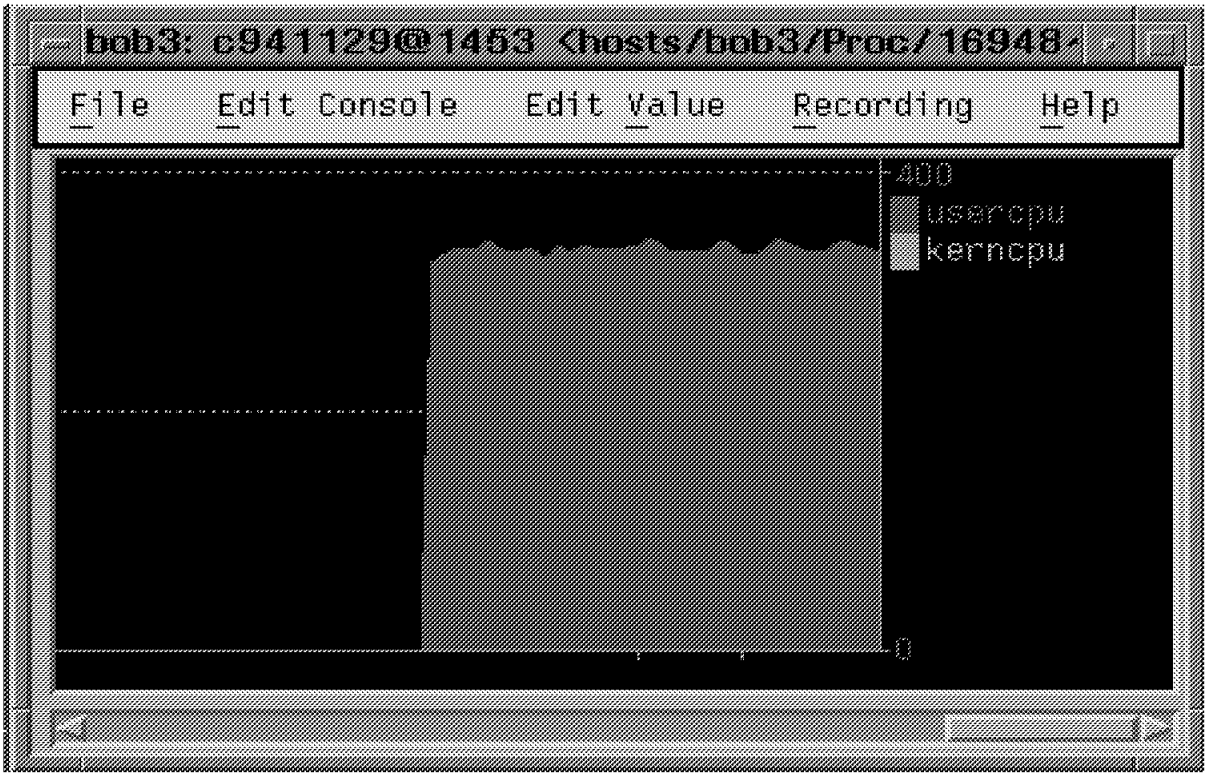


Figure 28. Monitoring a Process (Step 4)

After the CPU percentage values have been selected, the system creates the graph, but the default range is 0-100 percent. Since this process is using 340 percent of the system, the graph would not show a correct draw of the values. You have to change the range of these values to be 0-400 percent, or 0 - (number of processors * 100).

Hints

How do you change the range of the values? Simply move the mouse pointer into the graph area. Click your left mouse button. A dotted box appears around the graph area. Select the **Edit Value** item; then select the **Change Value** item. For more details, see chapter 3, Changing the Properties of a Value in the *Performance Toolbox for AIX: Guide and Reference, Version 1.2 and 2,* SC23-2625.

5.7.4 Monitoring an SMP with 3dmon

The 3dmon monitor provides a quick method of producing statistics about your system in a three-dimensional graphical view.

This monitor may be invoked by going to the **Utilities** menu in the main xmperf window. Inside this menu, you will find both the *3-D Monitor, Single Host* and *3-D Monitor, Multiple Hosts* submenus.

Once you have selected **3-D Monitor, Single Host** from the Utilities pull-down menu, you will see the following screen:

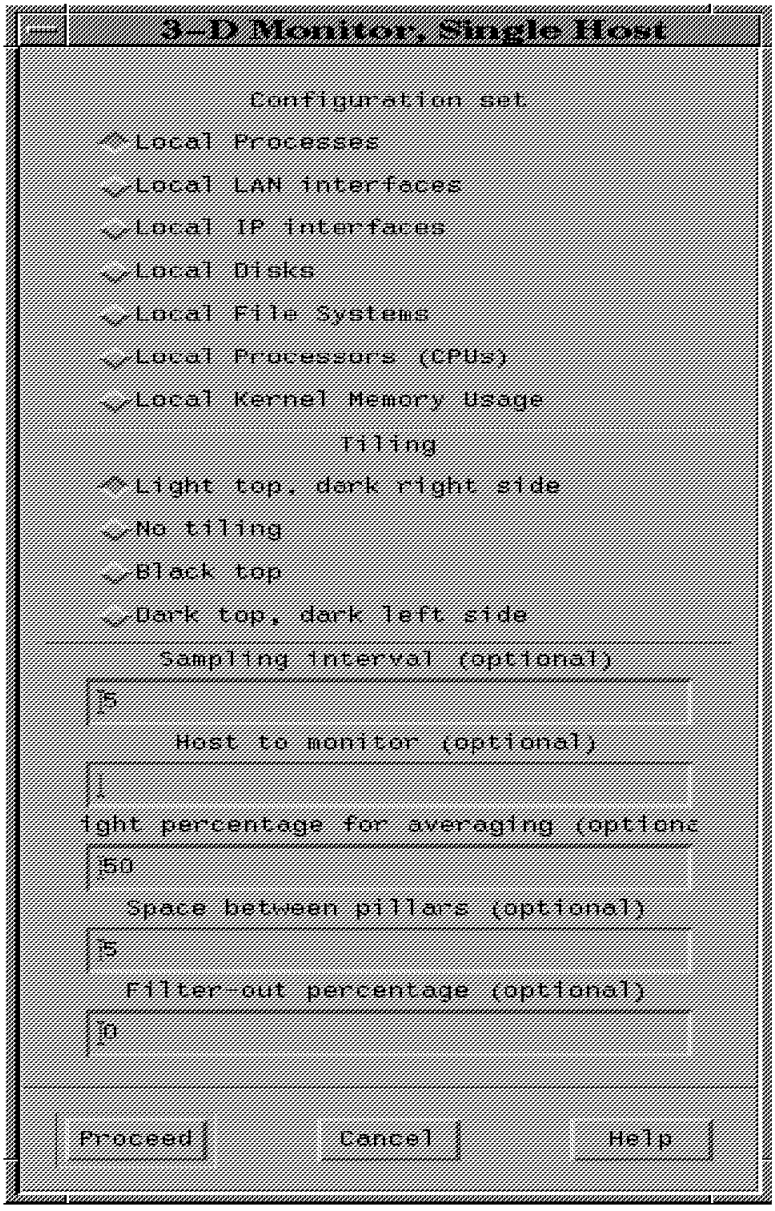


Figure 29. 3-D Monitor Selection Menu

At this step, you can select resources you want to monitor and change the sampling interval. If you select **Local Processors (CPUs)**, you will then see the following screen:

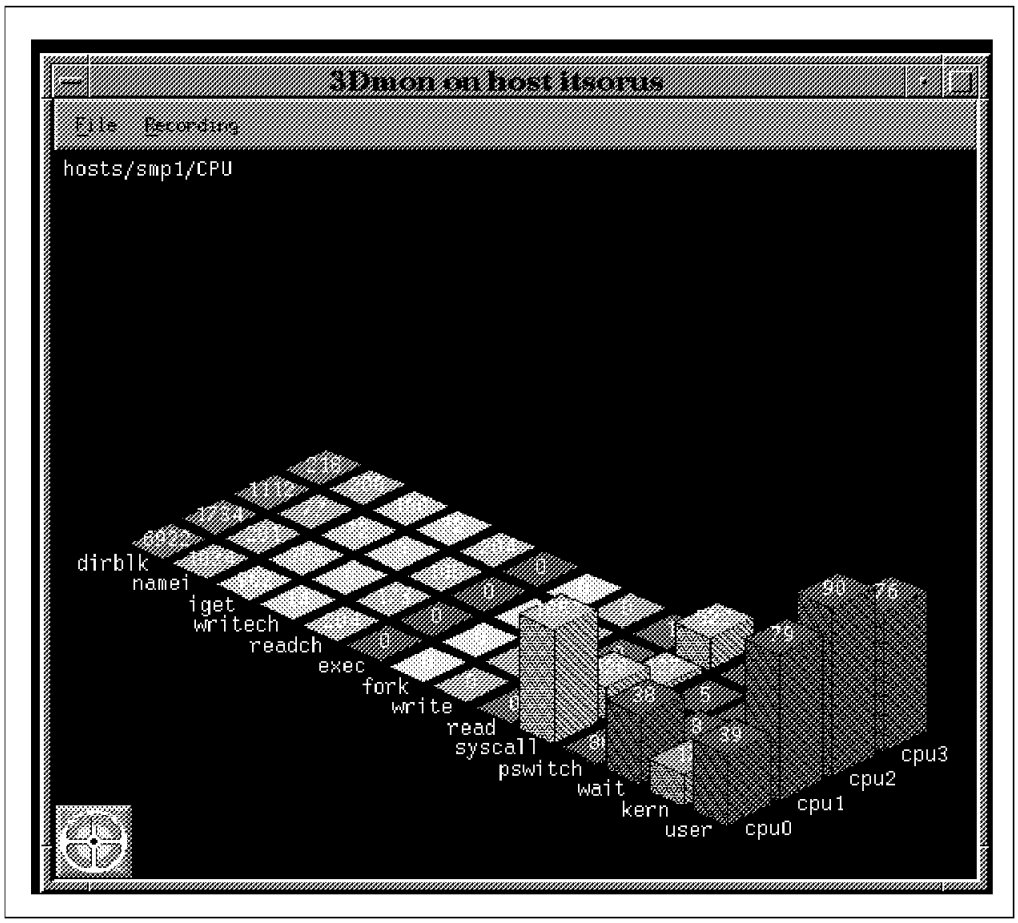


Figure 30. 3dmon Output on a Four-Way SMP

Note: If you cannot read the values behind the first towers corresponding to the user activity, you can move any monitored value to the front by clicking on the name of that value. For example, if you want to read the kern values for all the processors, you can click on kern. It will then move to the first position.

5.7.5 Monitoring Multiple Hosts with 3dmon

This monitor may be invoked by going to the Utilities menu in the main xmperf menu. Inside this menu, you will find both the *3-D Monitor, Single Host* and *3-D Monitor, Multiple Hosts* submenus.

Once you have selected **3-D Monitor, Multiple Host** from the Utilities pull-down menu, you will see the following screen:

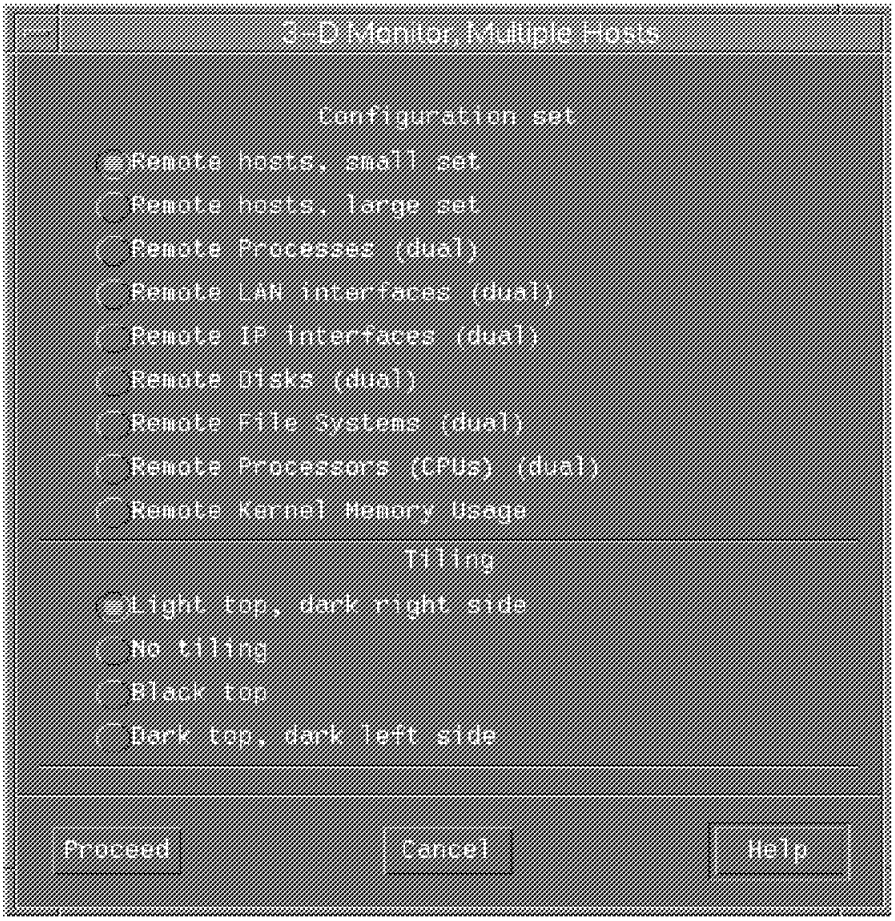


Figure 31. Configuration and Tiling

This screen allows you to pick any of the remote configuration sets supplied in the sample configuration file for 3dmon. Only one set can be selected at a time, but each invocation of 3dmon may use a different set. Sets that are marked as (dual) provide you with the opportunity to select from two selection lists. You can either accept the defaults or select the configuration or tiling options that you require. Click on the **Proceed** button. The *3Dmon on host <hostname>* screen as shown below and an *Inviting Data Suppliers* screen will be displayed. After a few seconds, the *Inviting Data Suppliers* screen disappears, and the *3Dmon on host <hostname>* screen is displayed with the hostnames and IP addresses that you can select for monitoring.

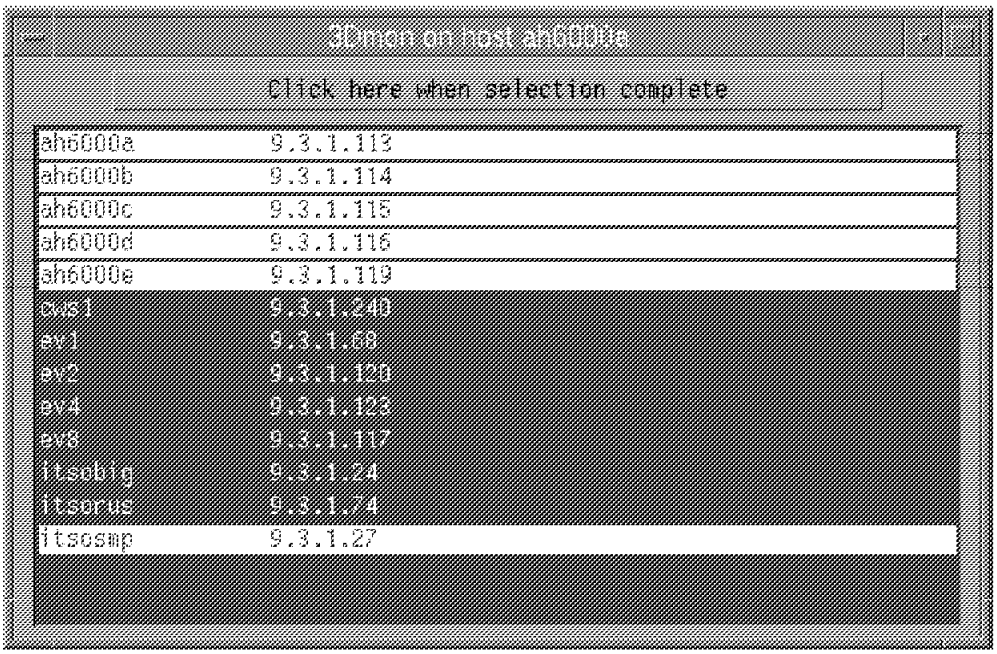


Figure 32. Host Selection

Note: As you can see, six hosts have been selected. Five of them are grouped together, and the last one is at the bottom of the list. Selecting the top five is very easy; all you need to do is hold the left mouse button and move the mouse pointer down to the fifth hostname, then release the left mouse button. To select the last hostname, hold the Ctrl key, move the mouse pointer to the last hostname and click your left mouse button.

Hints

Always use the Ctrl key and the left mouse button after your first selection if you are going to select more than one hostname, and one or more hostnames are to be skipped between the ones you are selecting.

Click on the **Click here when selection complete** button**,** and the following screen is displayed:

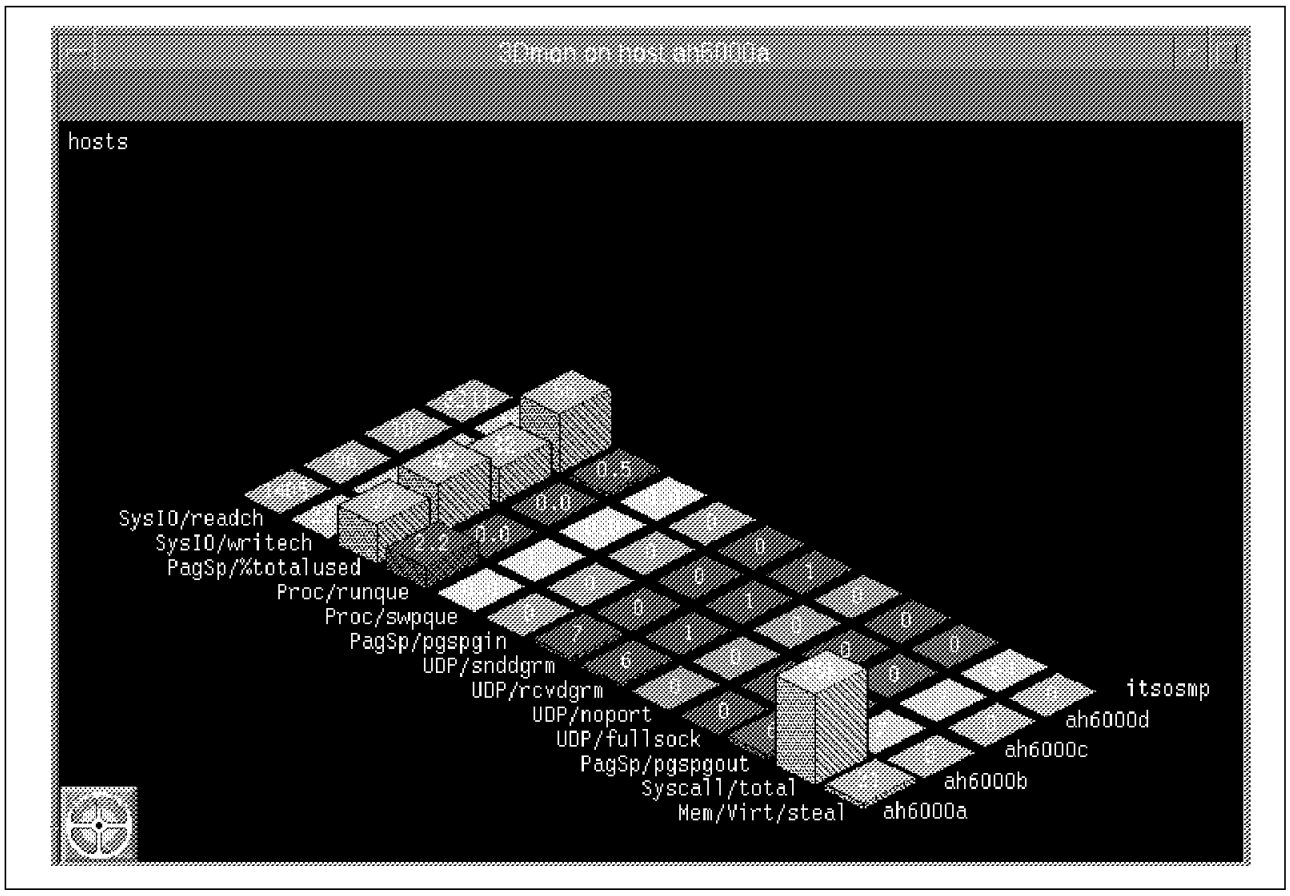


Figure 33. Multiple Hosts 3dmon Graph

When this graph was first displayed, the `Mem/Virt/steal` and `Syscall/total` values were not the first and second row of towers as shown above. By clicking on each of those values, they were moved to the first and second row. See note under Figure 30 on page 222. No values were displayed for the `itsosmp` system because the system was not available during this time.

5.8 Investigating Performance Problems

In the example above, there is a big difference between the `Syscall/total` value for host ah6000a and the other hosts. Also the `Mem/Virt/steal` value is not zero. This may indicate that a performance problem exists on host ah6000a. You need to obtain more information about host ah6000a in order to identify the nature of the problem. The *Local System Monitor* will provide more information about host ah6000a.

Assuming ah6000a is your local system, this is how you would get to the *Local System Monitor* of ah6000a.

- Select the **Monitor** pull-down menu from the main `xmperf` menu.
- Select the **Local System Monitor** option. See Figure 17 on page 209.

If you need to get to the *Local System Monitor* of a remote system, this is the procedure you need to follow:

- Select the **Monitor** pull-down menu from the main `xmperf` menu.
- Select **Instantiate Skeleton** from the *Monitor* pull-down menu.

- Select **S Single-host Monitor** option.
- Select the required host from *Wildcard Selection* menu.
- Click on **Done**, and then click on **Accept Selection**.

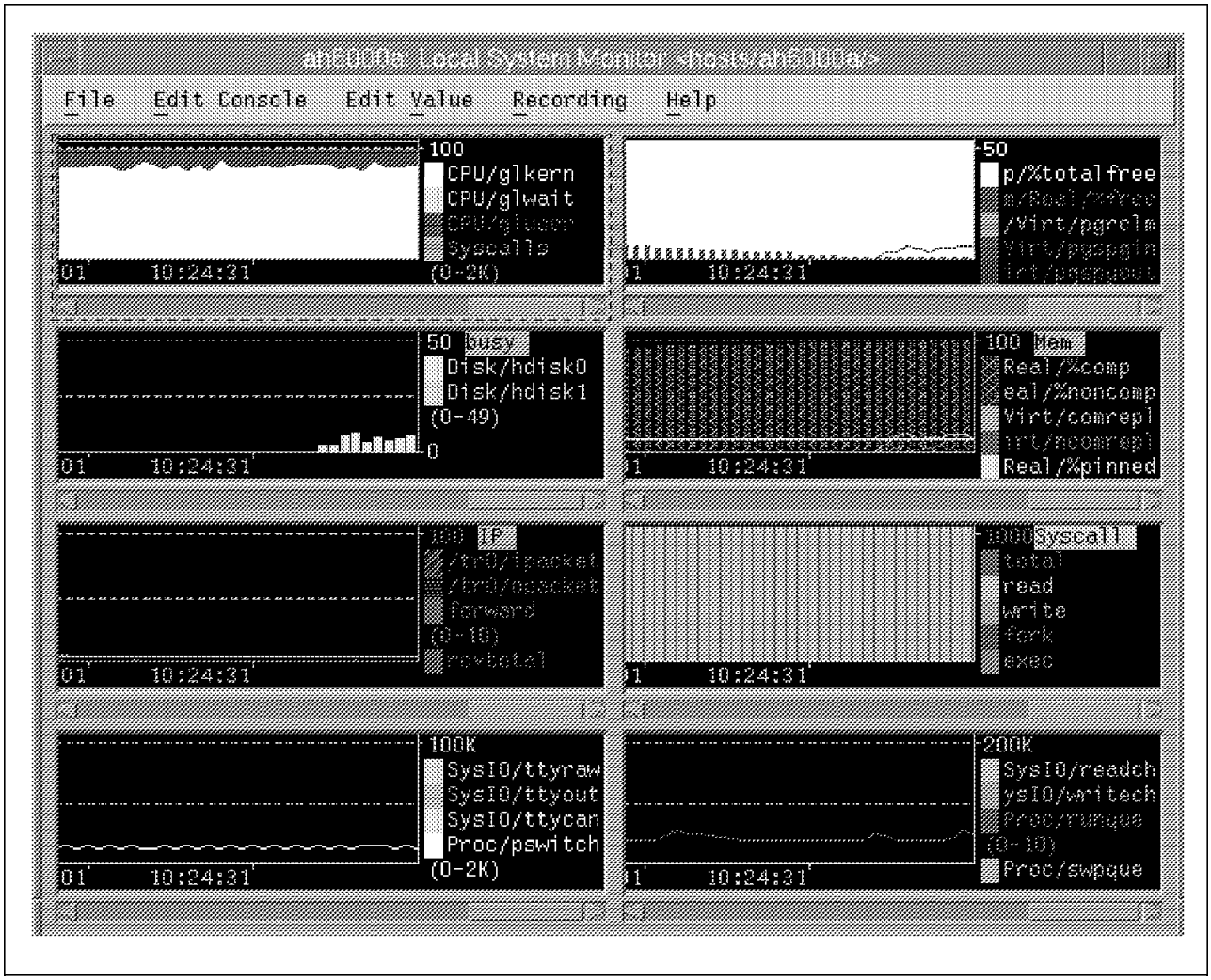


Figure 34. Local System Monitor Console

The graphs above provide much more information about host ah6000a. The CPU instrument (top on the left), the disk busy instrument (second on the left), and the paging/memory instrument (top on the right) are of most interest here. Unfortunately, not all the full value names are displayed on the graphs. However, you can click on **Edit Value**, then click on **Change Value** on the pull-down menu that is displayed, and the full value names will be displayed.

The CPU instrument compares CPU kernel, user and wait percentages of the processor. Notice that the kernel and user values are using most of the CPU cycles, with the wait value being zero. This could indicate that you may have a CPU-related performance problem.

If you look at the disk busy instrument and the paging/memory instrument, you will see that at the end of the disk busy instrument, you have some disk activity on hdisk0 at the same time there is some paging activity (Mem/Virt/pgspgout) in the paging/memory instrument. If you look closely at the paging/memory instrument, you can see the line graph just above the vertical bars

(Mem/Real/%free) near the end of the instrument. This is the paging activity. Notice how the free memory (Mem/Real/%free) is decreasing, and paging activity is commencing. This indicates that you might have a memory-related performance problem.

In order to identify which programs could be causing the problem, you need to have a look at the running processes.

Select **Utilities** from the main xmperf menu. Then select **3D-Monitor, Single Host** from the Utilities pull-down menu**,** and the same screen as Figure 29 on page 221 is displayed. Select **Local Processes**, and press the **Proceed** button. A screen similar to the one below is displayed.

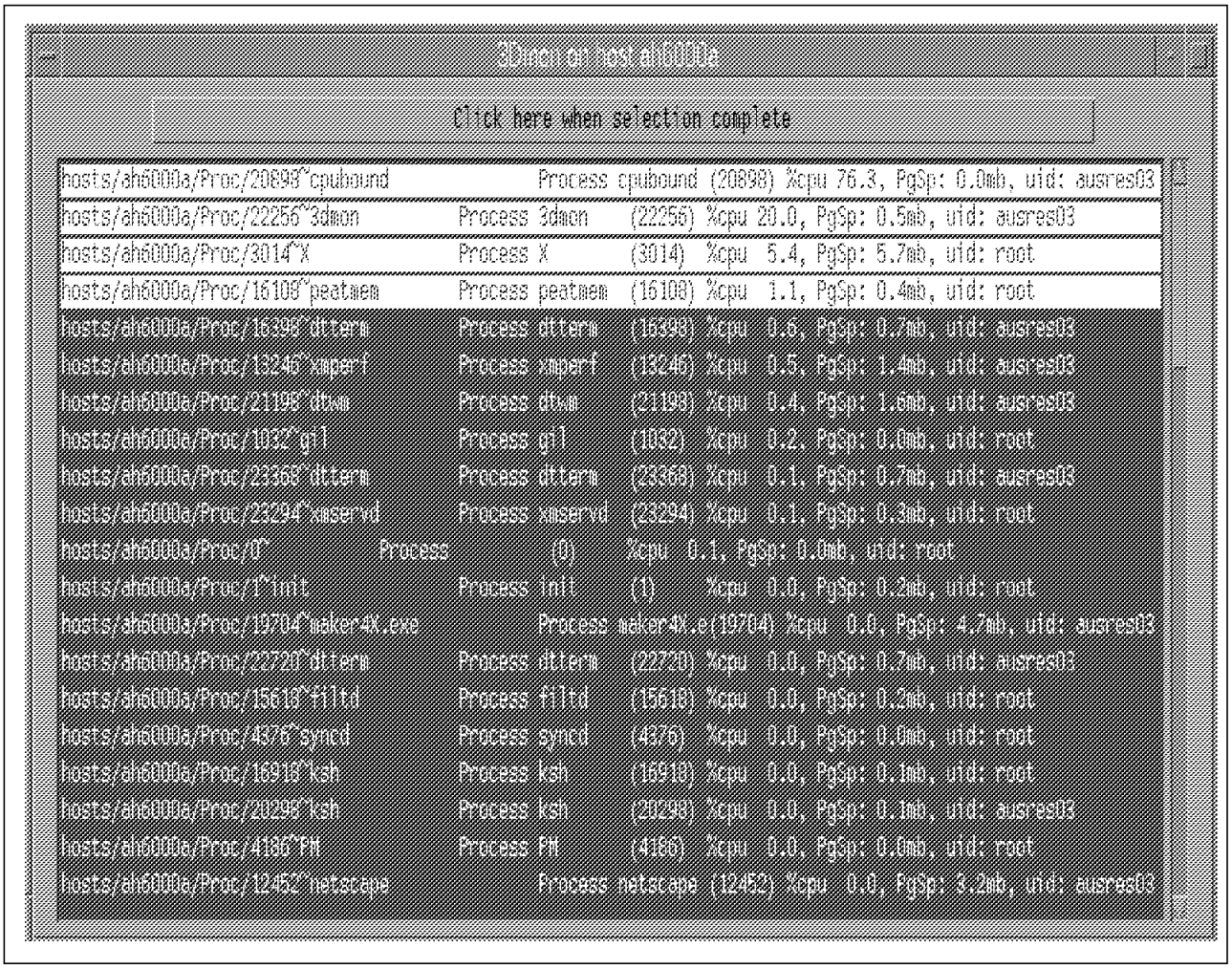


Figure 35. Select Host Processes

This graph displays all the local processes that are executing on your system with the most CPU-intensive process at the top of the list. The list does not reflect threads and does not show a CPU ID. It only shows the process ID, the percentage of CPU usage, paging space, and user ID who started the process. In our example, we have selected the four most CPU-intensive processes for further investigation. Click on **Click here when selection complete**, and the following screen is displayed:

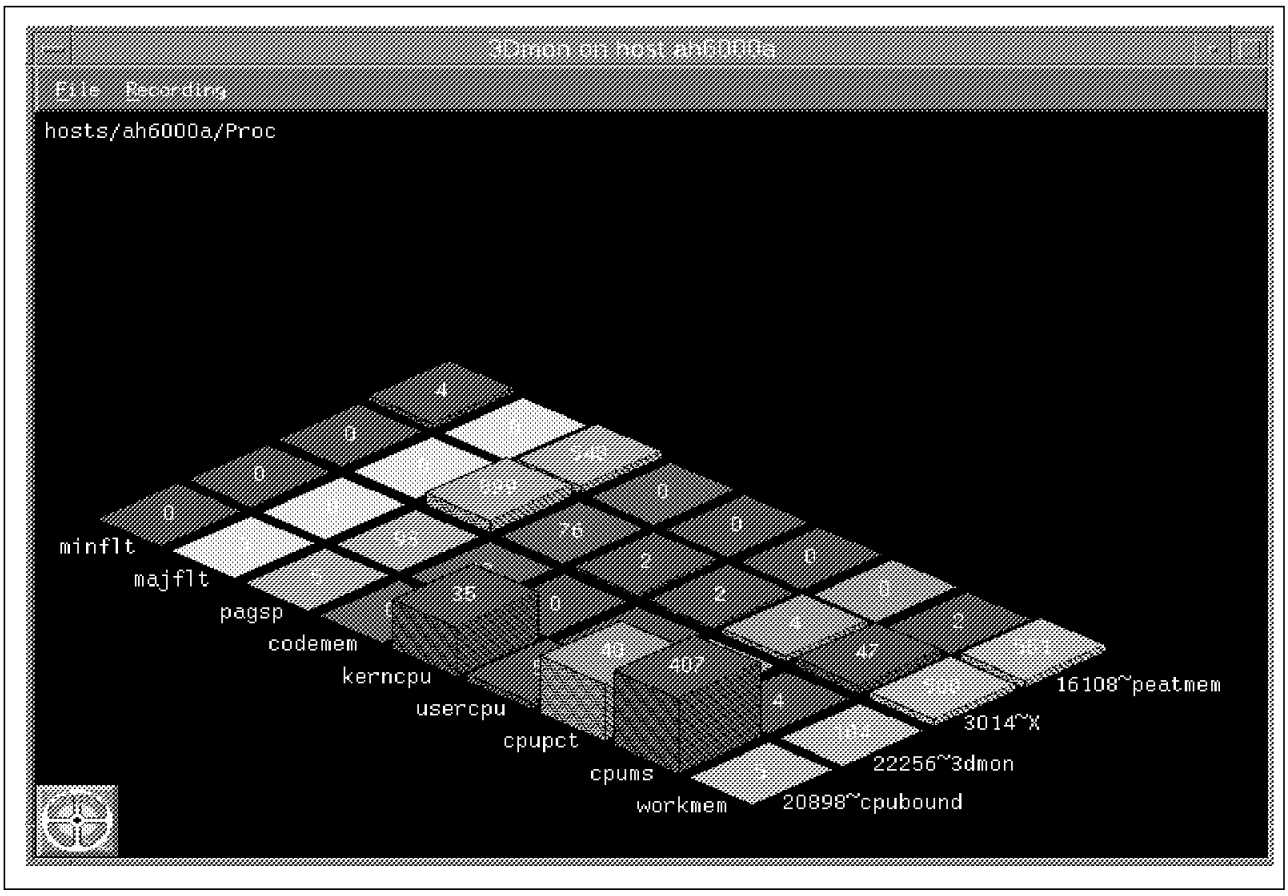


Figure 36. Initial 3dmon Graph of CPU-Intensive Processes

The two processes we are interested in are `cpubound` and `peatmem`.

Let's have a look at the process `cpubound`. It is consuming much more CPU time than any of the other processes. This in itself is not really a problem; however if you look at the CPU values from this graph together with the CPU values from the *Local System Monitor* console, where a zero wait time was displayed, you could have a looping program. Let's use the `vmstat` command to confirm that `cpubound` is looping. The `vmstat` output is in Figure 37 on page 229.

Pay particular attention to the idle (`id`) and wait (`wa`) values. If CPU is 100 percent busy (that is, 0 percent idle and 0 percent wait) for an extended period, there is a good chance that the `cpubound` program is in an infinite loop.

The `vmstat` command can be executed from the PTX main menu.

From the main `xmperf` window, select **Analysis**; from the pull-down menu that is displayed select **Virtual Memory Analysis**, and then select **vmstat Monitor**. On the vmstat Monitor pop-up menu, you can accept the defaults or change the interval value and number of samples as required and click on **Proceed**. A screen similar to the one in Figure 37 on page 229 will be displayed. See 2.1.1, "CPU Bound" on page 7, for more detail about `vmstat` output fields.

```
kthr     memory             page              faults        cpu      disk xfer
----- ----------- ------------------------ ------------ ----------- -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa  1  2  3  4
 0  0 13618   535   0   0   0   1    7   0 169 34357 295 10 28 60  2  0  2  0
 1  0 13622   529   0   0   0   0    0   0 238 90625 2166 19 81  0  0  0  0 0
 1  0 13626   525   0   0   0   0    0   0 207 87476 2599 24 76  0  0  0  0  0
 1  0 13626   525   0   0   0   0    0   0 153 116034 116 16 84  0  0  0  0  0
 1  0 13626   525   0   0   0   0    0   0 126 117427  35 13 87  0  0  0  0  0
 1  0 13626   525   0   0   0   0    0   0 128 117274  35 13 87  0  0  0  0  0
 1  0 13626   525   0   0   0   0    0   0 123 117430  35 14 86  0  0  0  0  0
 1  0 13626   525   0   0   0   0    0   0 122 118332  35 13 87  0  0  0  0  0
 1  0 13626   525   0   0   0   0    0   0 124 117610  35 18 82  0  0  0  0  0
 1  0 13626   525   0   0   0   0    0   0 126 117274  36 16 84  0  0  0  0  0
 1  0 13626   525   0   0   0   0    0   0 125 117190  34 16 84  0  0  0  0  0
 1  0 13626   525   0   0   0   0    0   0 135 116697  37 13 87  0  0  0  5  0
 1  0 13626   525   0   0   0   0    0   0 129 117616  36 14 86  0  0  0  0  0
 1  0 13626   525   0   0   0   0    0   0 128 117216  36 13 87  0  0  0  0  0
 1  0 13626   525   0   0   0   0    0   0 126 117356  36 15 85  0  0  0  0  0
 1  0 13626   525   0   0   0   0    0   0 129 117883  35 13 87  0  0  0  0  0
 1  0 13626   525   0   0   0   0    0   0 128 117821  35 14 86  0  0  0  0  0
 1  0 13626   525   0   0   0   0    0   0 130 117304  35 14 86  0  0  0  0  0
 1  0 13626   525   0   0   0   0    0   0 132 117130  34 15 85  0  0  0  0  0
 1  0 13626   525   0   0   0   0    0   0 129 117688  35 15 85  0  0  0  0  0
```

Figure 37. vmstat Output

The next process you look at is peatmem. The working memory segments for process peatmem and process X are about the same size. However, their working memory segment sizes are greater than the other processes displayed above. You can then monitor the working memory segment of process X and more particularly process peatmem as this is a user initiated process. After a few minutes, compare the working memory segments of process X and process peatmem to the initial values in Figure 36 on page 228. You can see this in Figure 38 on page 230.

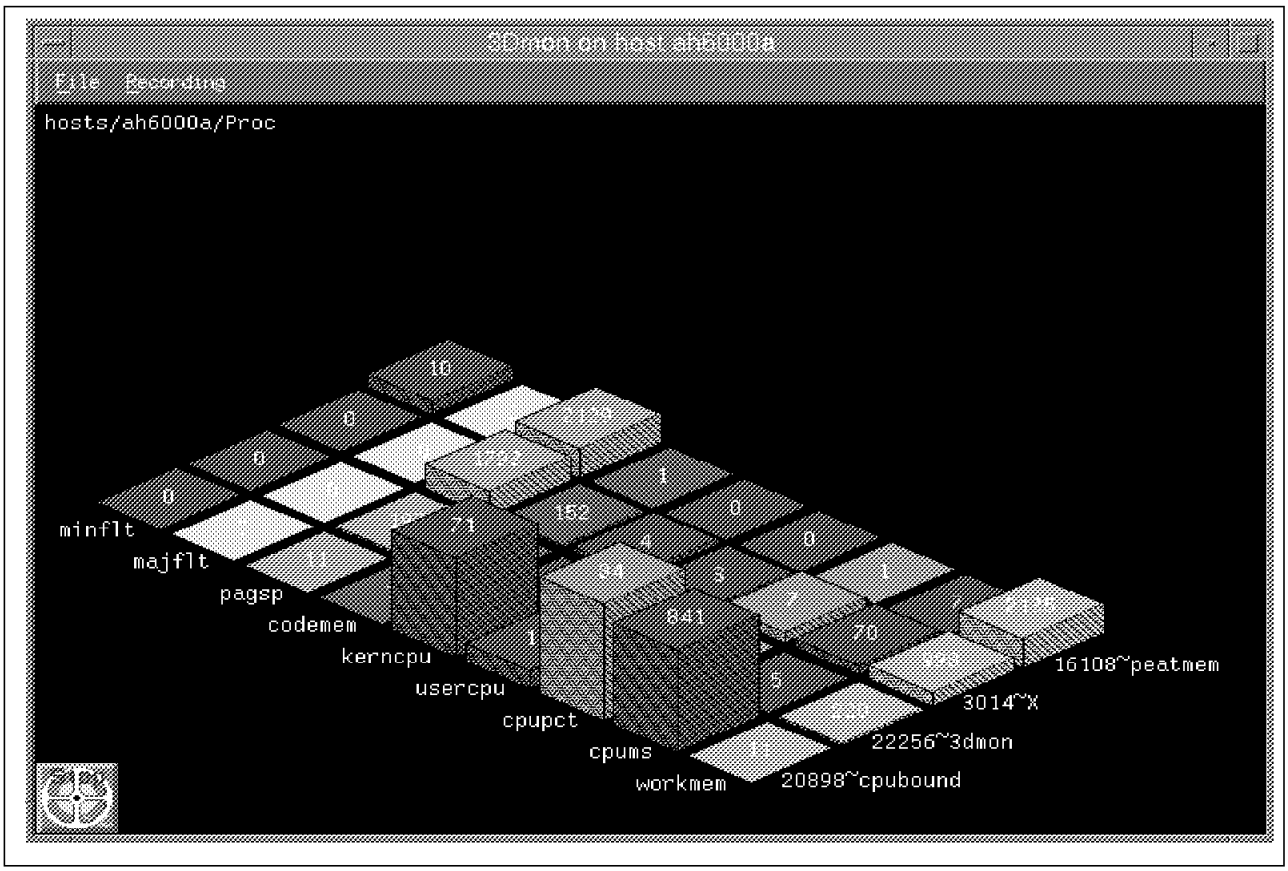


Figure 38. Final 3dmon Graph of CPU-Intensive Processes

Within a space of a few minutes, the working memory segment for process peatmem has grown from 551 pages to 2135 pages. Remember that the *Local System Monitor* showed us that during the memory segment growth of peatmem, the free memory decreased so much that paging activity started. The growth of the working memory segment for process peatmem is significant, and there is a distinct possibility that process peatmem has a memory leak. See 2.1.2, “Memory Bound” on page 10, for more details.

5.9 Customizing Performance Toolbox

A very useful feature of the Performance Toolbox is the ability to monitor the system and react to specific situations. The filter daemon (filtd) allows you to define:

- New statistics from existing ones through data reduction
- Alarms that are triggered by conditions you define, which can execute any command you desire

5.9.1 Data Reduction and Alarms with filtd

To use the filtd daemon, you will need to work with the xmservd.res and filter.cf files. There are sample versions of these files in /usr/lpp/perfagent. It is recommended that you copy these files from /usr/lpp/perfagent to your home directory or to /etc/perf for customization. When xmservd starts, it first checks your home directory for these files. If it does not find them there, it checks

/etc/perf for these files. If these files do not exist in /etc/perf, it then checks /usr/lpp/perfagent.

Since filtd is a dynamic-data supplier program, you may want to always have it running when the xmservd daemon runs. You can cause this to happen by removing the comment character from the line below in the xmservd.res file.

```
# supplier      /usr/bin/filtd -p5
```

If you want the xmservd daemon to be started automatically as part of the boot process, you can add the following two lines at the end of the /etc/rc.tcpip file:

```
/usr/bin/sleep 10
/usr/bin/xmpeek
```

The first line is only necessary when you intend to use the xmservd/SMUX interface (used by IBM NetView) to export statistics to the local SNMP agent. The sleep command makes sure that the start of the snmpd daemon is completed before the xmservd daemon starts.

In the example below, we have created new statistics from existing statistics and three alarms for demonstrations purposes. Use the xmpeek -l command to obtain a list of all available statistics in order to create new ones. Figure 39 shows a partial listing of statistics on host ah6000a after executing the xmpeek -l command:

```
/ah6000a/CPU/                     Central processor statistics
/ah6000a/CPU/gluser                   System-wide time executing in user mod
e (percent)
/ah6000a/CPU/glkern                   System-wide time executing in kernel m
ode (percent)
/ah6000a/CPU/glwait                   System-wide time waiting for IO (perce
nt)
/ah6000a/CPU/glidle                   System-wide time CPU is idle (percent)
. . .
/ah6000a/DDS/IBM/Filters/                 Filters defined by DDS
/ah6000a/DDS/IBM/Filters/user                     Percent User CPU
/ah6000a/DDS/IBM/Filters/allcpu                   Percent CPU, excluding idl
e
/ah6000a/DDS/IBM/Filters/cpubusy                  cpubusy
/ah6000a/DDS/IBM/Filters/cpufree                  New line added to default
filter file
/ah6000a/DDS/IBM/Filters/rwratio                  Read/write ratio, all disk
s combined
/ah6000a/DDS/IBM/Filters/diskmax                  Busy percent - most busy d
isk
/ah6000a/DDS/IBM/Filters/diskmin                  Busy percent - least busy
disk
/ah6000a/DDS/IBM/Filters/diskavg                  Average disk busy percent
/ah6000a/DDS/IBM/Filters/readdistr                Average disk reads in perc
ent of most busy disk
a
```

Figure 39. Partial Listing of Statistics

Copy the default filter.cf file from the /usr/lpp/perfagent directory to the /etc/perf directory, and then add the modifications after the cpubusy entry as listed below.

```
cpubusy = CPU_gluser + CPU_glkern "Line from default filter file"

cpufree = CPU_glwait + CPU_glidle "New line added to default filter file"

@cpubusy:{TRAP22}{EXCEPTION} DDS_IBM_Filters_cpubusy > 80 \
DURATION 60 FREQUENCY 5 SEVERITY 0 \
"Test CPU busy"

@cpufree:{EXCEPTION} DDS_IBM_Filters_cpufree == 0  \
DURATION 60 FREQUENCY 5 SEVERITY 1 \
"Test CPU Not Idle"

@varfsfull:[aixterm -bg red -e ksh -c "(banner Var Full,being increased;
read)";/home/ausres03/increase_var]{EXCEPTION} FS_rootvg_hd9var_%totused > 95\
DURATION 60 FREQUENCY 5 SEVERITY 2 \
"Test var filesystem full"
```

The second line starting with cpufree creates a new statistic named cpufree that is equal to CPU_glwait and CPU_glidle. Notice that the xmpeek output lists these statistics as /ah6000a/CPU/glwait and /ah6000a/CPU/glidle. When we refer to the statistics in the filter.cf file, we remove the hostname (in this case ah6000a) and replace the "/" with "_". When the filter.cf file is used with the filtd daemon, the newly created statistic will be included in the xmpeek output under Filters defined by DDS. (See the cpufree statistic in Figure 39 on page 231.)

The third line starting with @cpubusy creates an exception named @cpubusy. What happens here, is that as soon as DDS_IBM_Filters_cpubusy is greater than 80 percent for more than 60 seconds, the filtd daemon will inform xmservd daemon that the condition was met. You can then use exmon to monitor these exceptions. The *TRAP* action will, when the defined condition becomes true, produce an SNMP trap that is passed on through xmservd daemon to snmpd daemon and eventually to an SNMP manager such as IBM NetView. The keyword DURATION in seconds is used to determine how long a condition must remain true before the alarm is triggered. The keyword FREQUENCY in minutes is the time to wait before checking the condition again after an alarm has been triggered. See the *Performance Toolbox for AIX: Guide and Reference, Version 1.2 and 2*, SC23-2625 for more details about the TRAP, DURATION, FREQUENCY and SEVERITY keywords.

Line four is much the same as the third line except that line four does not specify a *TRAP*. In line four, the newly created statistic cpufree is monitored.

In line five, the var filesystem is monitored to ensure that we are informed before it reaches 100 percent utilization. As soon as FS_rootvg_hd9var is used more than 95 percent for more than 60 seconds, the filtd daemon will inform the xmservd daemon that the condition was met. The keyword DURATION in seconds, is used to determine how long a condition must remain true before the alarm is triggered. The keyword FREQUENCY in minutes is the time to wait before checking the condition again after an alarm has been triggered. When the alarm is triggered, the message in Figure 40 on page 233 will pop up on your console, and the /var filesystem will be increased by ten percent.

The first portion of the @varfsfull exception, up to read, displays the pop-up message, and /home/ausres03/increase_var executes the script, which increases /var by ten percent.

```
# This is the Increase_var script
#
CURSIZE=`df | grep /var | awk '{print $2}'`
ADDSIZE=`expr $CURSIZE / 10`
chfs -a size=+$ADDSIZE /var
```

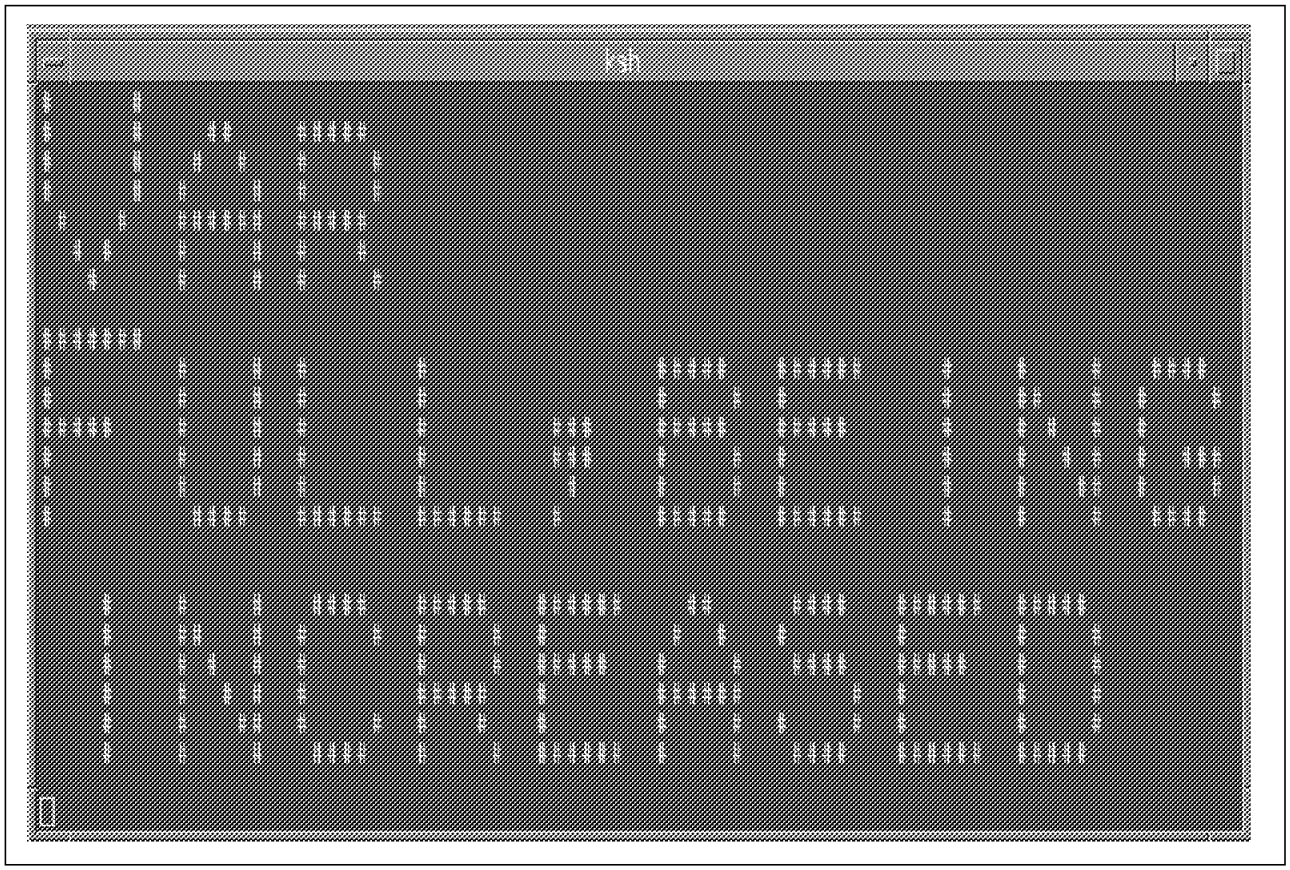


Figure 40. Output From Alarm

Before you start the filtd daemon with the new filter.cf file, you have to ensure that the filtd daemon is not running already. If it is running, kill it (don't use kill -9), use kill -15.

```
ps -ef|grep filtd
    root 14544 17698   0 09:46:43  pts/0  0:00 filtd -f /home/ausres03/filter.cf
    root 18992 17698   2 11:01:13  pts/0  0:00 grep filtd:
```

In this case, the filtd daemon was running; so kill it by executing the following command:

```
kill -15 14544
```

Now you can start the filtd daemon with the new filter.cf file.

```
filtd -f /etc/perf/filter.cf &
```

If there are any errors in your filter.cf file, they will not be displayed on your screen when you execute the above command. Therefore, you need to check the filter.log# (latest log file) in the /etc/perf directory. If you have any errors, correct them, then stop and start the filtd daemon again. Check to see that your filtd daemon is running.

Let's see how the new statistics and alarms that were created can be used. First, create a new console including an instrument with the new statistic, cpufree, and a predefined statistic, cpubusy. Secondly, use exmon to monitor the alarms created above.

The procedure you need to follow to create a new console is the same as the one to create the new console. Refer to 5.7.2, "Creating a New Console" on page 209, if you have any problems following the instructions below:

Select the **Monitor** pull-down menu to start adding a new console.

Select the **Add New Console** option from the Monitor pull-down menu. Then either enter the name of your new console or accept the default name and click on **Proceed**.

A menu similar to Figure 19 on page 211 without the Edit Console pull-down menu will be displayed. Select **Edit Console**. From the Edit Console pull-down menu, select **Add Local Instrument**.

The Dynamic Data Supplier Statistics menu will be displayed. Select **Dynamic Data Supplier Statistics**.

The IBM-defined Dynamic Data Suppliers menu will be displayed. Select **IBM-defined Dynamic Data Suppliers.**

The Filters defined by the DDS menu will be displayed. Select **Filters defined by DDS**.

From the Filters menu, first select cpubusy and a menu similar to Figure 23 on page 215 is displayed. Here you can customize the properties of the values you selected, such as the color and the type of graph you want (line, area, bars). You will be able to set upper and lower limits, set a threshold, and set an alarm when this threshold is reached.

When satisfied with the property values, select **OK**. This takes you back to the Filters menu.

Then select cpufree from the Filters menu. Accept the default property values, and select **OK**. This takes you back to the Filters menu.

From the Filter menu, click on the **End Selection** button, and a console similar to Figure 41 on page 235 is displayed.

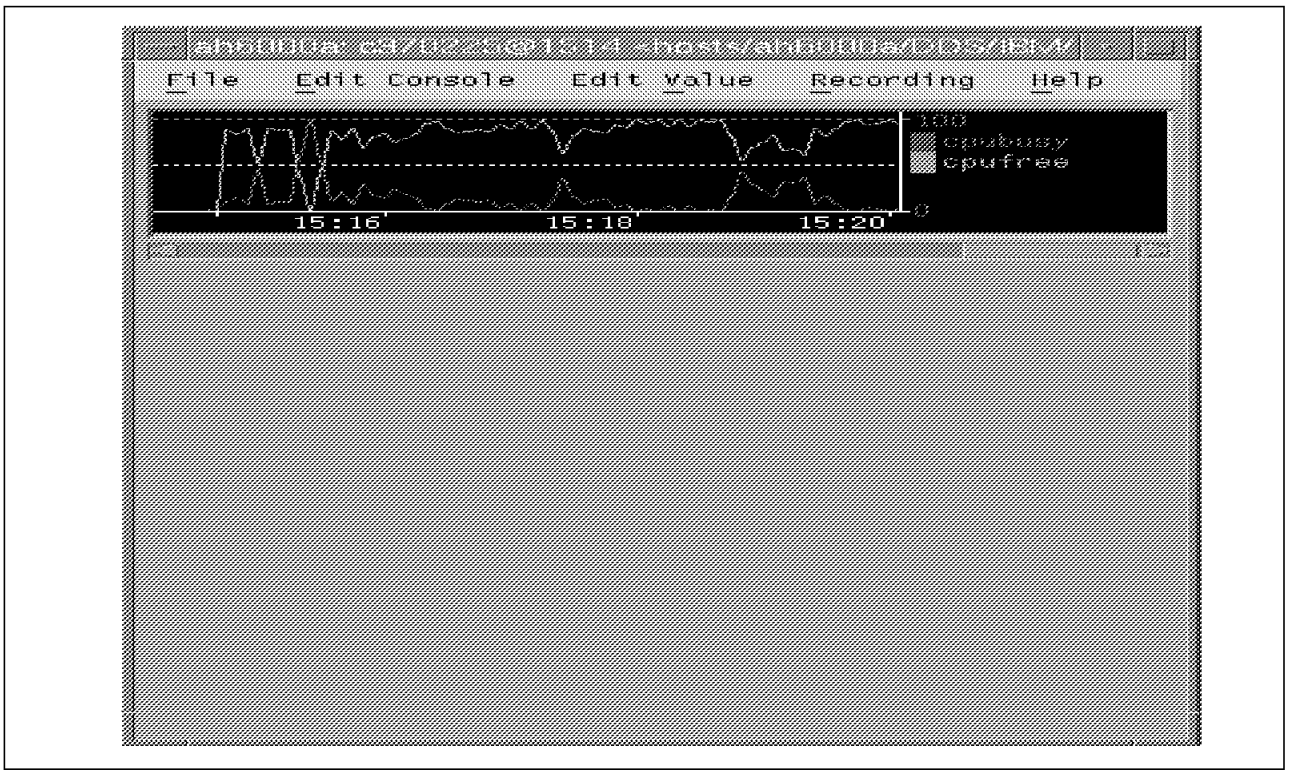


Figure 41. An Instrument Using Filters

Let's change the instrument style to a pie chart and increase the size of the pie chart to make it easier to read.

Place your mouse pointer in the instrument and click your left mouse button. A dotted box appears around the instrument. Select the **Edit Console** option.

Select **Modify Instrument** from the Edit Console pull-down menu.

Select **Style & Stacking** from the Modify Instrument pull-down menu.

Select **Pie Chart** and click on **Proceed**. The instrument style is now a pie chart. To change the size of the instrument, select **Edit Console** again.

Select **Resize Instrument** from the Edit Console pull-down menu.

The instrument will disappear from the console and a dotted box with the box size displayed in the top-left corner will appear in your console. Press the left mouse button and move the mouse pointer to the required size. Release the left mouse button, and the new sized instrument will appear in your console.

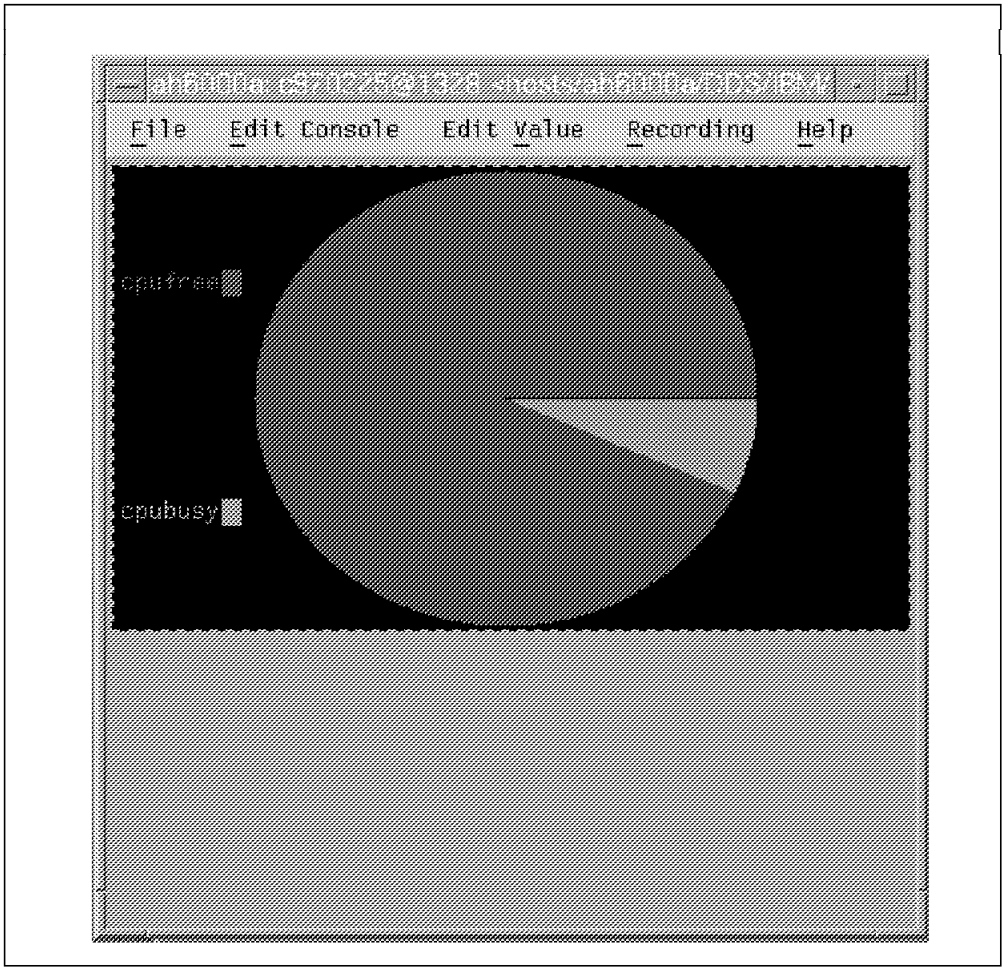


Figure 42. Pie Chart Using Filters

5.10 Monitoring Exceptions with exmon

The exception monitoring program, exmon, is designed to provide a convenient facility for monitoring exceptions as they are detected. It works with the filtd daemon that generates exceptions packets based upon alarm conditions defined in the filtd configuration file (filter.cf). Just below Figure 39 on page 231 the three exceptions we used in the example are shown. Let's now have a look at the exmon monitoring window after starting the filtd daemon with our filter.cf file.

From the main xmperf window, select **Utilities**; then from the pull-down menu that is displayed, select **Exception Monitor.** On the Exception Monitor pop-up menu, click on **Proceed**. Select the host or hosts you wish to monitor on the Add Hosts pop-up menu, and click on **Click here when selection complete**.

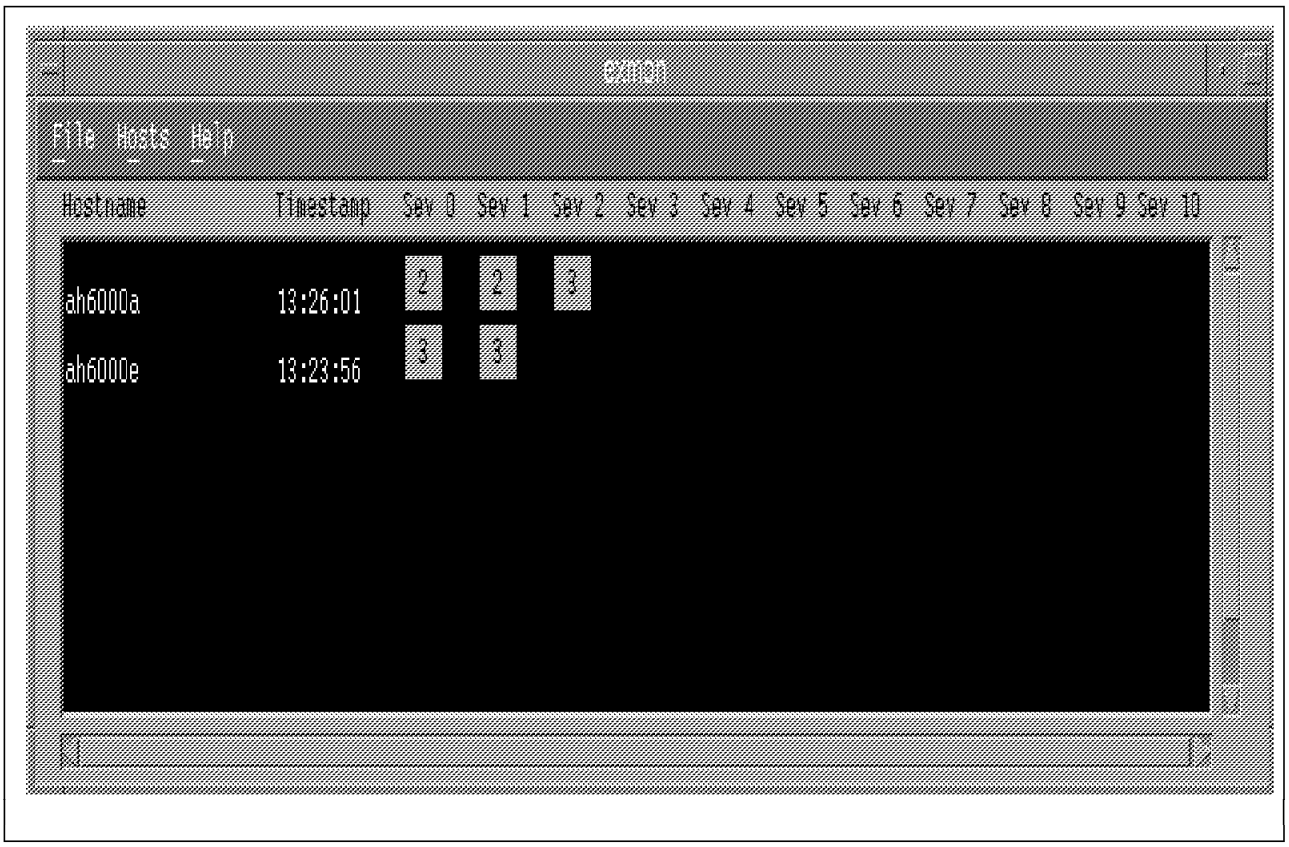


Figure 43. The exmon Main Window

The filtd daemon together with the new filter.cf file is running on two hosts, ah6000a and ah6000e. On host ah6000a, there are three different exceptions, Sev 0, Sev 1 and Sev 2, and on host ah6000e there are two different exceptions, Sev 0 and Sev 1. You need to look at your filter.cf file to understand what these exceptions mean.

It is advisable to have one filter.cf file for all your hosts so that Sev 0 means the same thing on all your hosts. These Sev column headings are not very helpful in terms of describing what type of exceptions they represent. However, you are able to change these column headings. In the /usr/lib/X11/app-defaults directory, you will find the EXmon file, which you can modify. It is strongly suggested that you backup the EXmon file before modifying it. Here is a copy of the EXmon file used to obtain the exmon main window below:

```
#     exmon options
#
*GraphFont:     -ibm-block-medium-r-normal--15-100-100-100-c-70-iso8859-1
*DrawArea.background:   black
*DrawArea.foreground:   white
*Foreground:    black
*Background:    grey

*RangeDisplay:  true
*RangeColor1:   green
*RangeColor2:   yellow
*RangeColor3:   red
```

```
*ValueRange1:   5
*ValueRange2:   10

*ValueColor0:   blue
*ValueColor1:   red
*ValueColor2:   white
*ValueColor3:   green
*ValueColor4:   yellow
*ValueColor5:   orange
*ValueColor6:   grey
*ValueColor7:   pink
*ValueColor8:   magenta
*ValueColor9:   GreenYellow
*ValueColor10:  SkyBlue

*ExceptionText0:        CPU H
*ExceptionText1:        WIC l
*ExceptionText2:        VAR F
*ExceptionText3:        Sev 3
*ExceptionText4:        Sev 4
*ExceptionText5:        Sev 5
*ExceptionText6:        Sev 6
*ExceptionText7:        Sev 7
*ExceptionText8:        Sev 8
*ExceptionText9:        Sev 9
*ExceptionText10:       Sev 10
```

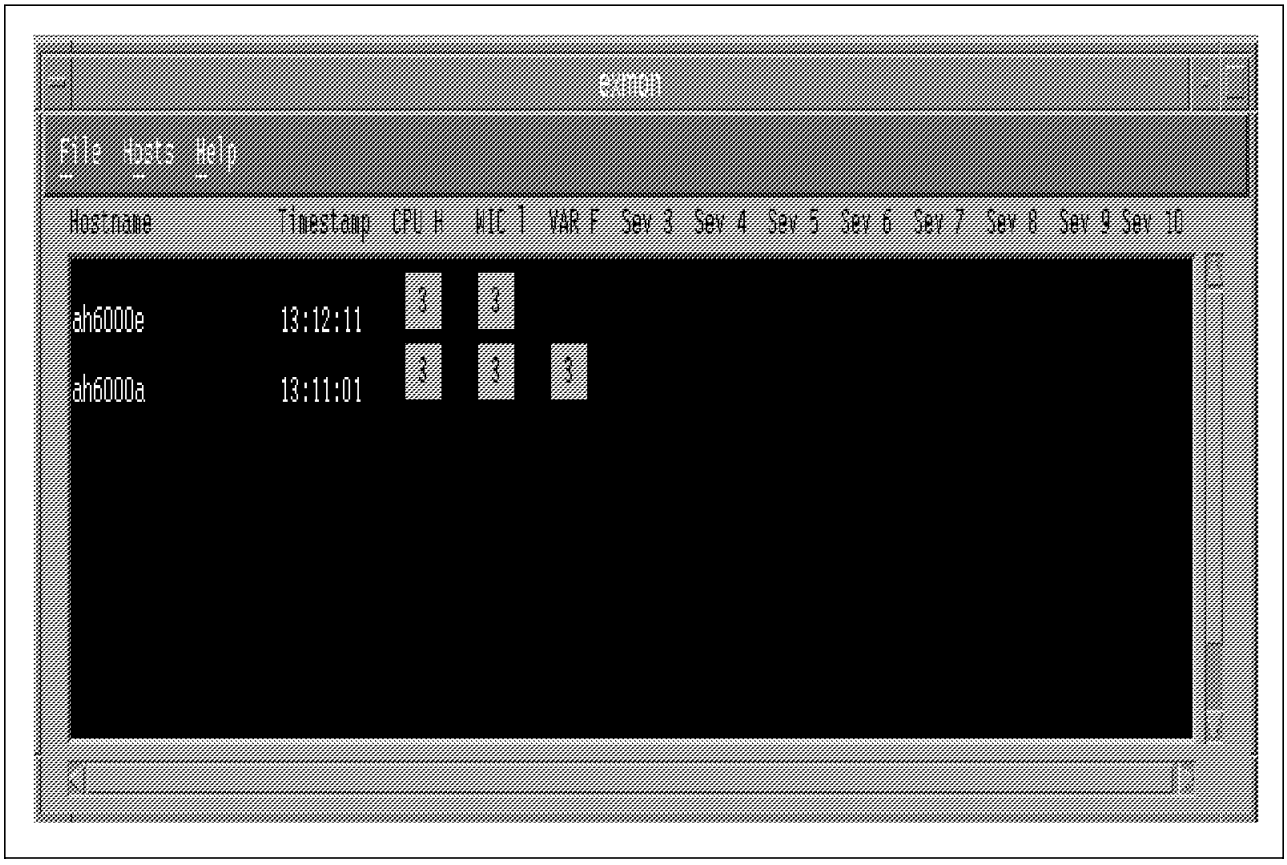


Figure 44. Modified exmon Main Window

1. The ExceptionText0 was changed from Sev 0 to CPU H.
2. The ExceptionText1 was changed from Sev 1 to WIC L.

3. The ExceptionText2 was changed from Sev 2 to VAR F.

At least the column headings make some sense now, CPU H stands for CPU usage is high. WIC L stands for CPU wait and CPU idle is zero. VAR F means the /var filesystem is full.

If you move your mouse pointer to one of the hostnames in the exmon main window and click your left mouse button, the following pop-up menu appears on your screen.

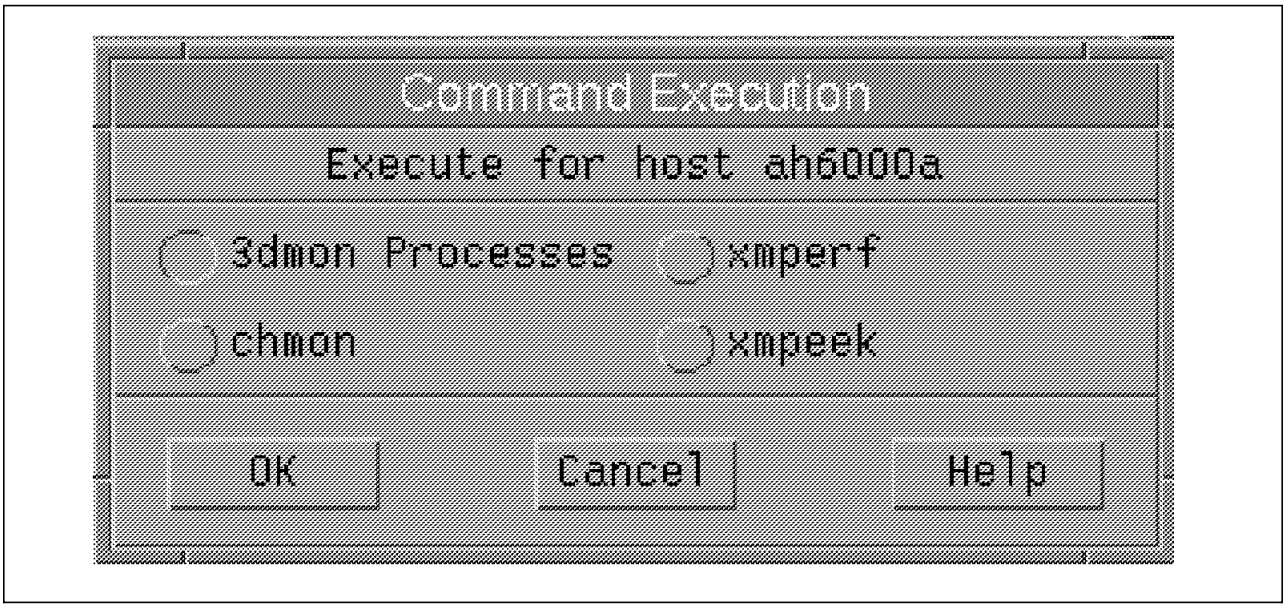


Figure 45. Command Execution Pop-Up

If you select **3dmon Processes** and click on **OK,** you will be presented with a screen similar to Figure 35 on page 227, where all the local processes that are executing on your system are displayed with the most CPU-intensive process at the top of the list. You might then be able to determine which process or processes are causing the problem. If you select **xmperf**, another main xmperf window will be opened for you. If you select **xmpeek**, details similar to the ones below will be displayed. The *Performance Toolbox for AIX: Guide and Reference, Version 1.2 and 2*, SC23-2625 describes the output below in great detail.

```
Statistics for xmservd daemon on   *** ah6000a ***

        Instruments currently defined:    15
        Instruments currently active:      3
        Remote monitors currently known:   7

  --Instruments---   Values    Packets
  Defined   Active   Active      Sent    Internet Address  Port   Hostname
  -------  -------  -------  ----------  ----------------  -----  ----------
      13        1       15      16,234        9.3.1.113    1448  ah6000a
       2        2       40         777        9.3.1.113    2298  ah6000a
  -------  -------  -------  ----------
      15        3       55      17,011
```

If you select **chmon**, a screen similar to the one below will be displayed.

```
Data Consumer API        Remote Monitor for host       Thu Feb 27 16:55:38 1997
CHMON Sample Program        ***  ah6000a ***           Interval:     5  seconds

% CPU                                                EVENTS/QUEUES    FILE/TTY
Kernel   82.5   |######################     |  Pswitch    292   Readch   15313
User     17.3   |#####                       |  Syscall 106575   Writech     29
Wait      0.0   |                            |  Reads   104938   Rawin        0
Idle      0.0   |                            |  Writes       0   Ttyout      29
                                               Forks        0   Igets        0
PAGING counts   PAGING SPACE   REAL MEM  64MB   Execs        0   Namei        0
Faults       0  % Used  26.0   % Comp    75.0   Runqueue   1.0   Dirblk       0
Steals       0  % Free  73.9   % Noncomp 21.0   Swapqueue  0.0
Reclaim      0  Size,MB  192   % Client  16.0

PAGING page/s   DISK      Read   Write     %    NETWORK    Read   Write
Pgspin       0  ACTIVITY KB/sec  KB/sec  Busy   ACTIVITY  KB/sec  KB/sec
Pgspout      0  hdisk0      0.0     0.0   0.0   lo0         0.8     0.8
Pagein       0  hdisk1      0.0     0.0   0.0   tr0         0.0     0.0
Pageout      0  cd0         0.0     0.0   0.0
Sios         0
```

The chmon program allows you to display data on a character-based screen. The default data displayed here are CPU, memory, paging, network activity, and syscalls. The chmon program is a good example of using the data consumer (Rsi) API, and the source code for this program can be found in /usr/samples/perfmgr. This program can be modified to display any values you want or under any format that you will find useful. The sample program refreshes the values on the screen every few seconds, and the program will exit after two thousand observations. You are able to terminate this program before it has done two thousand observations by typing the letter **q** in its window.

It is also possible to start the chmon program from the command line, where you can supply parameters such as interval in seconds between observations, number of processes and the name of the host you would like to monitor.

5.11 Analyzing Recordings with the azizo Program

Recording files are binary files whose first record is a configuration record. This record identifies the file as a recording file, names the source of the recording, and states the version of the file. Recording files are created by one of the agent or manager programs. They can be created by the xmperf and 3dmon programs during monitoring, by the xmservd daemon at any time it is running, by the a2ptx program from ASCII files that adhere to a certain format, or by the ptxrlog program.

The ptxrlog program can produce recordings in either ASCII format, which allows you to print the output or postprocess it with database or spreadsheet programs, or with the a2ptx program to produce a standard Performance Toolbox for AIX recording file, or it can produce a standard Performance Toolbox for AIX recording file in binary format. If the ptxrlog program is executed in the background, the list of statistics to record must be specified in a control file. Below is a copy of a ptxrlog configuration file used to monitor the system for a period of five hours and produce a standard Performance Toolbox for AIX recording file in binary format.

```
# Ptxrlog.cf file
#
CPU/gluser
CPU/glkern
CPU/glwait
CPU/glidle
CPU/cpu0/syscall
Mem/Real/%free
Mem/Virt/pgrclm
Mem/Virt/pgspgin
Mem/Virt/pgspgout
PagSp/%totalfree
Disk/hdisk0/busy
Disk/hdisk1/busy
Proc/runque
Proc/swpque
# end of Ptxrlog.cf file
```

Use the following ptxrlog command line:

```
ptxrlog -f /home/ausres03/ptxrlog.cf -r /home/ausres03/XmRec/R.ptxrlog.out -b
1205 -e 17.12 &
```

The recordings are written to /home/ausres03/XmRec/R.ptxrlog.out and started at approximately 12 noon and stopped at approximately 5:12 pm. The /home/ausres03/XmRec directory was used because this is the default directory used by the azizo program to look for recording files. All recording files done with the xmperf and 3dmon programs will start with "R.". The same naming convention is used, which makes it easier to select the recording file after the azizo program has been started. The -r flag specifies that the output from ptxrlog goes to a binary recording file in standard recording file format. The *Performance Toolbox for AIX: Guide and Reference, Version 1.2 and 2,* SC23-2625 describes the format of the ptxrlog command in great detail. Then the azizo program is used to analyze the recording.

From the main xmperf window, select **Utilities.** Then select **Analyzing Recordings (azizo).** The pop-up window shown below will appear on your screen.

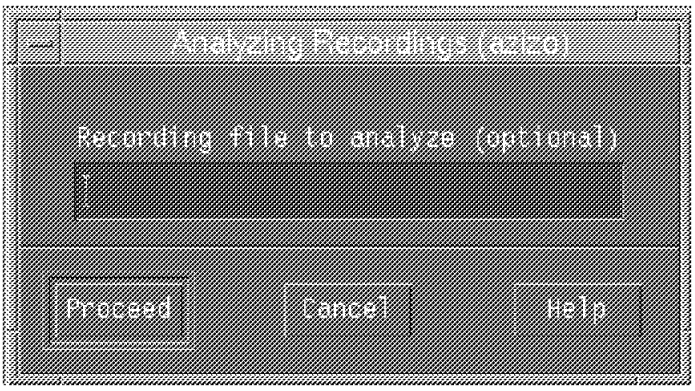


Figure 46. Analyzing Recordings Pop-Up

You can either enter the name of the recording file to analyze on the *Analyzing Recordings (azizo)* pop-up and click on **Proceed,** or just click on **Proceed** and select the recording file later from the azizo main menu.

The azizo main window is shown below:

Figure 47. The azizo Main Window

Now move the mouse pointer to the **Local Files** icon, click on it, and the recording files pop-up is displayed:

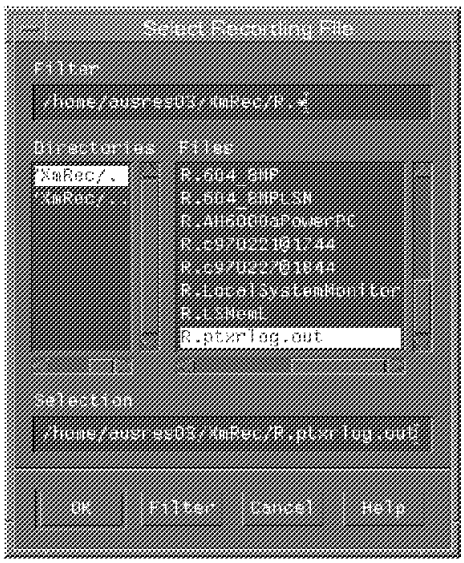


Figure 48. Recording Files Pop-Up Menu

Select the ptxrlog recording file named R.ptxrlog.out and click on **OK**. The azizo main graph window is displayed, and the azizo main window now contains the metrics (list of statistics that were recorded) that were defined in the ptxrlog.cf file. Just above the metrics, you will find information about the metric, such as the style and ticks and the date and time of the recording.

First, the azizo main window is discussed, and then we move on to the azizo main graph window.

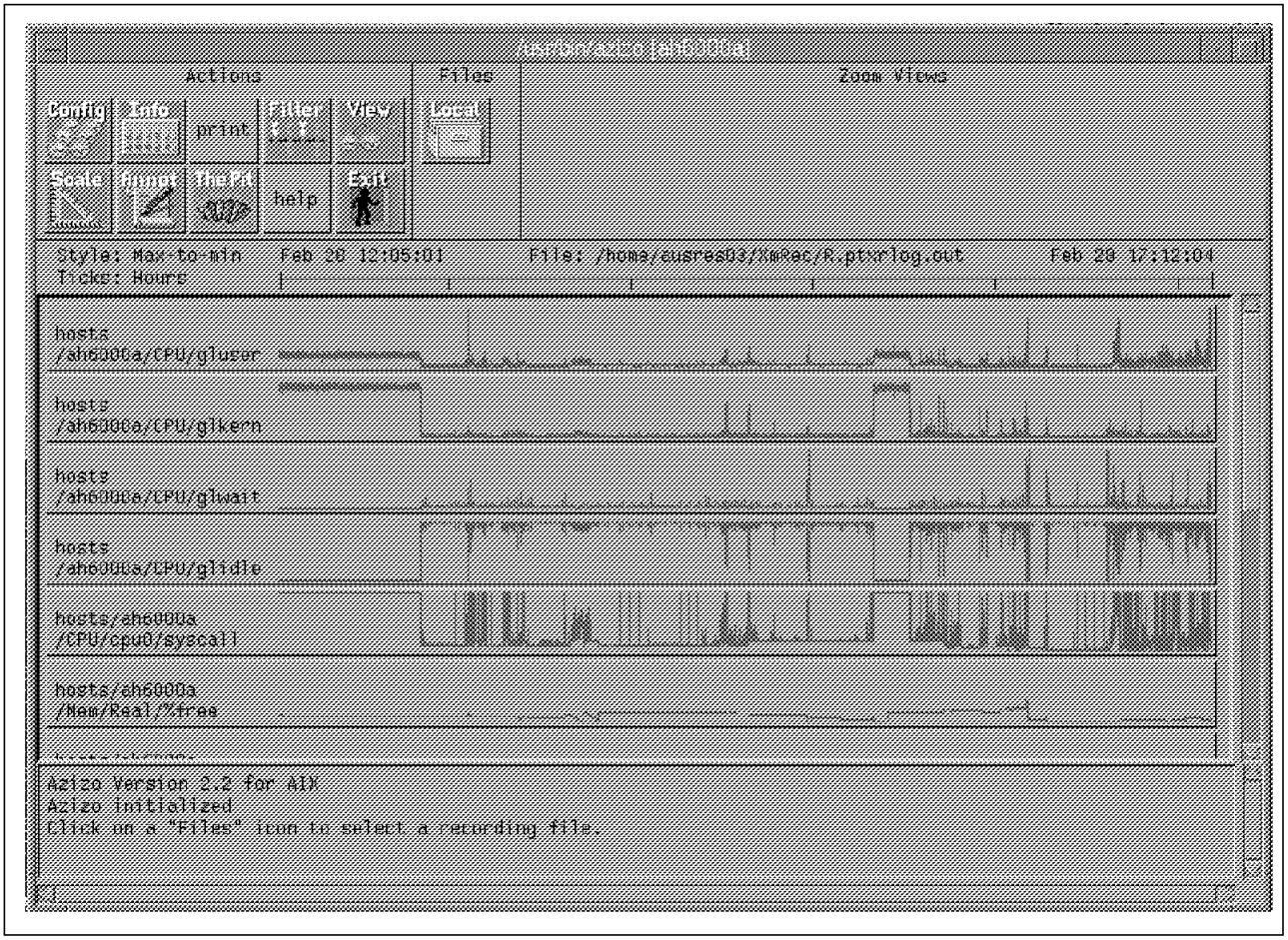


Figure 49. The azizo Main Window

5.11.1 The azizo Main Window

The azizo main window is divided into three sections. They are from top to bottom, the *Actions* section, the *metrics* selection window and the *message* window. From the metrics section, you can drag-and-drop metrics to the Actions section. To demonstrate how the drag-and-drop operation works, all the CPU-related metrics will be removed from the azizo main window. For the purpose of this exercise, we will assume we are looking at a memory-related problem.

Move the mouse pointer anywhere in the block displaying the /ah6000a/CPU/gluser metric, then press mouse button 2 (middle button). The mouse pointer changes to a drag icon (a small metric icon). Hold the middle mouse button down, move the drag icon to **The Pit** icon in the Actions section

and release the mouse button. The /ah6000a/CPU/gluser was removed from the metrics selection window. Do the same with the /ah6000a/CPU/glkern, /ah6000a/CPU/glwait, /ah6000a/CPU/glidle, and /ah6000a/CPU/cpu0/syscall metrics. Now all the memory-related metrics are displayed in one window.

Note: As the CPU-related metrics were removed from the azizo main window, they were automatically removed from the azizo graph window. The reverse does not happen when the metrics are removed from the azizo graph window.

The data for the metrics that were removed has not been deleted from the recording file /home/ausres03/XmRec/R.ptxrlog.out.

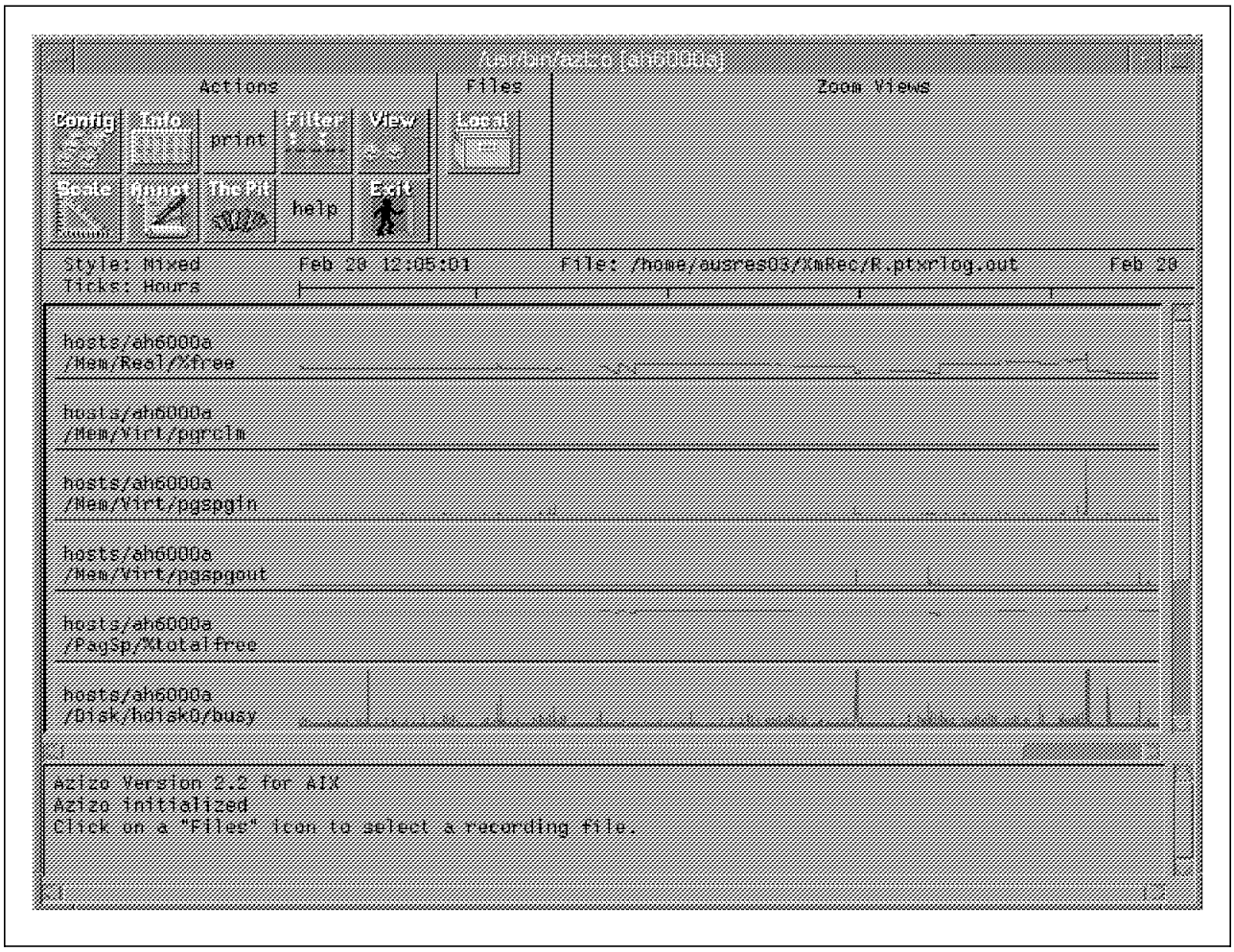


Figure 50. Main Window of azizo with Memory-Related Metrics

When you look at Figure 51 on page 245, it still contains all the metrics from the ptxrlog recording because we want to show the same metric removal procedure we did for the azizo main window.

5.11.2 The azizo Graph Window

The main graph is the principle viewing window of azizo. When azizo reads a recording file, it always displays a top-level main graph that covers the entire time interval of the recording file. The main graph has two sections. To the left is a list of metrics that are included in the graph. The metric names are displayed using the same color as used to draw the data. If the list of metrics is longer than the window can display, a scroll bar allows you to scroll the list of

metrics. On the right is the actual graphical display of the metrics data for the time period covered by the graph. The drag-and-drop operations from the azizo main graph window work with all the Action icons. The next page displays the main graph of the ptxrlog recording.

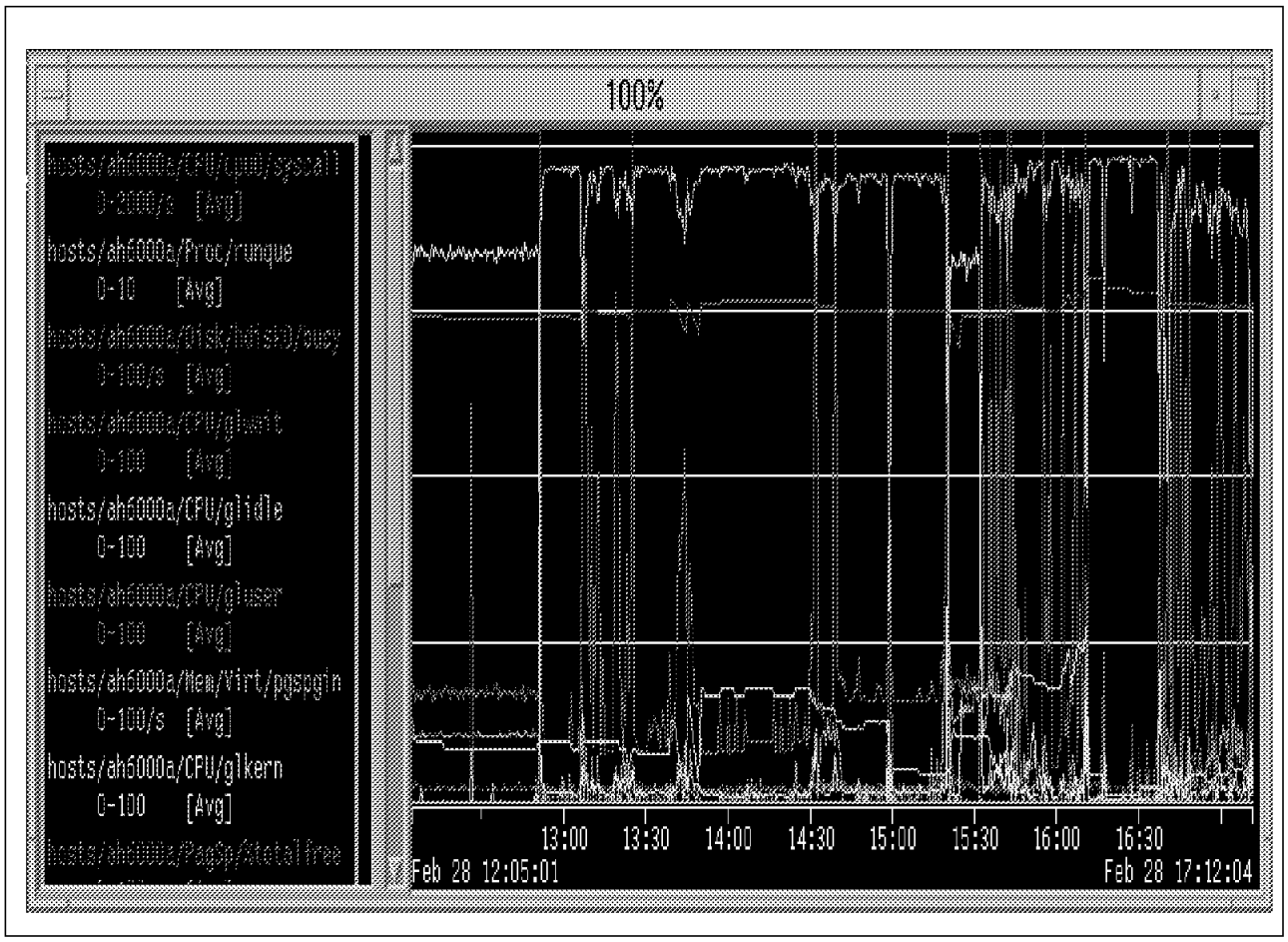


Figure 51. The azizo Graph Window

The azizo main graph can look very busy if you are working with a large number of metrics. It is much simpler to work with the azizo main graph if you have only the metrics displayed that are related to a particular problem. In our case, we are looking at a memory-related problem; so we will remove all the non-memory-related metrics and then save these metrics in a filter file for later use.

Move the mouse pointer to the first CPU metric in the list of metrics to the left of the azizo main graph window. Press and hold the middle mouse button; then move the drag icon to **The Pit** icon in the Actions section. The metric will be removed from the list of metrics to the left of the azizo main graph window. At the same time, the actual graphical display of the metric will be removed from the graph area to the right of the azizo main graph window. Keep repeating this procedure with the rest of the non-memory-related metrics until only memory-related metrics are left.

If you accidently remove a metric that you did not intend to, you are able to add this metric back again. You need to have the azizo main window and the azizo main graph displayed. All you do is move your mouse pointer to the metric in

the azizo main window, press and hold the middle mouse button. Then move the required metric to the graph area of the main graph window. The metric will be added to the metric list and graph area.

This is what the azizo main graph window looks like after removing all the non-memory-related metrics.

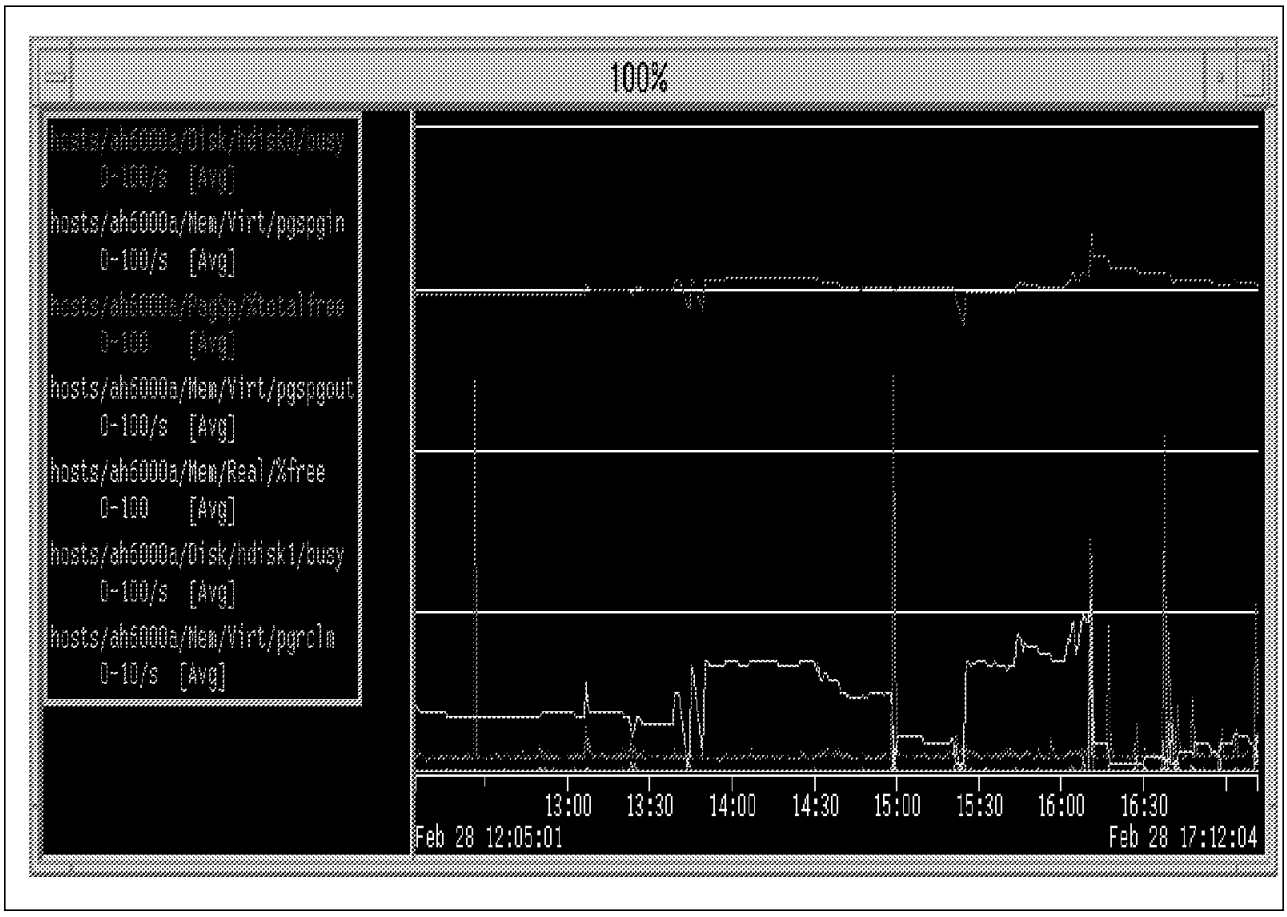


Figure 52. Filtered azizo Main Graph Window

It is much easier to work with this graph. All the metrics are displayed in this window, and the graph is much easier to read. We have created a subset of the recording file, and this is known as filtering when working with the azizo program. Filtering can be done by reducing the time interval, by reducing the number of metrics, or by reducing both the time interval and the number of metrics. Now save the filtered recording in a file.

Move the mouse pointer to the graph area of the main graph window, press and hold the middle mouse button. Then move the drag icon to the Filter icon in the Actions section and release the mouse button. The filter dialog box pop-up menu will be displayed.

Figure 53. Filter Dialog Box

The filter dialog box has the default values for the lowest and the highest time stamps to be included when data values are copied to the filtered output file. You can change either time stamp to extend or reduce the time period covered by the filtered output, as long as the time period covers at least two seconds. The default file name for the filtered output is the source file with ".filt" appended. You can change the file name to anything you want. If the file exists, you are asked whether you want to overwrite it. In our example, we accepted the defaults and clicked on **OK.** The filtered recording file was saved.

5.11.3 Zooming-In on Main Graphs

The next thing you might want to do is select a period during the day when performance was particularly bad. The azizo program gives you the ability to zoom-in on subsections of a main graph. This is achieved by drawing an outline in the main graph area and then selecting the type of zoom-in from a dialog window. The outline around the area of interest would be shaped like a rectangle.

In our case, the area of interest is the last hour of the recording. Draw the outline by moving the mouse pointer to the 16:00 mark on the time scale. Then press the left mouse button and keep it down while moving the mouse pointer to the top right-hand corner of the graph. Release the left mouse button. The zoom-in dialog box shown below will appear.

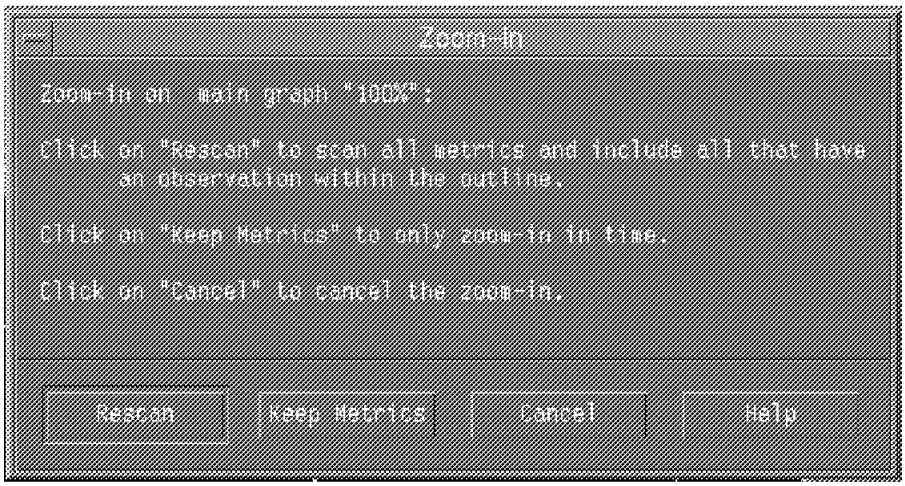


Figure 54. Zoom-In Dialog Box

Click on **Keep Metrics** to obtain the zoomed-in main graph. *Rescan* would have included all the metrics from the original ptxrlog.out file, and this is not what we want right now. The Help option will provide more information about the Keep Metrics and Rescan. You now have a zoomed-in main graph that represents 23 percent of Figure 52 on page 246. When you are done with analyzing the main graph, you could save it as a filter file. The procedure to do this is exactly the same as the one on the previous page.

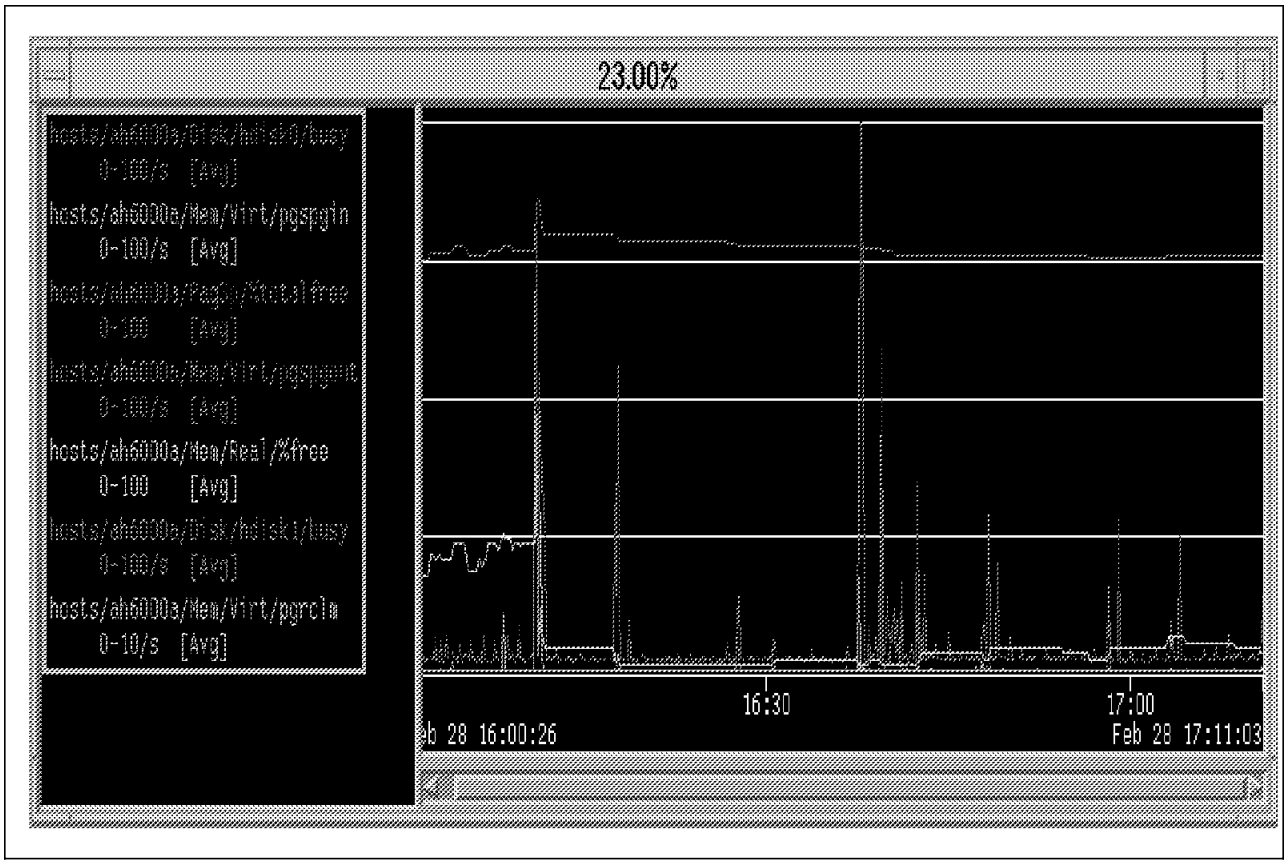


Figure 55. Zoomed-In Main Graph

You could also delete the zoomed-in main graph by putting it in *The Pit* in the Actions section. All you need to do is move the mouse pointer to the graph area of the zoomed-in main graph. Press and hold the middle mouse button. Move the drag icon to **The Pit** icon in the Actions section and release the mouse button. The zoomed-in main graph window will be deleted from the screen. It is not possible to remove the original azizo main graph in this way. When you move the azizo main graph window to *The Pit* icon, the graph will disappear from the azizo main graph window, but the window will remain.

It is also possible to obtain statistical information for the metrics of your zoomed-in main graph or main graph window. You move the mouse pointer to the graph area of the zoomed-in main graph. Press and hold the middle mouse button. Move the drag icon to the **Info** icon in the Actions section, and release the mouse button. Here is an example of the information window for a zoomed-in main graph.

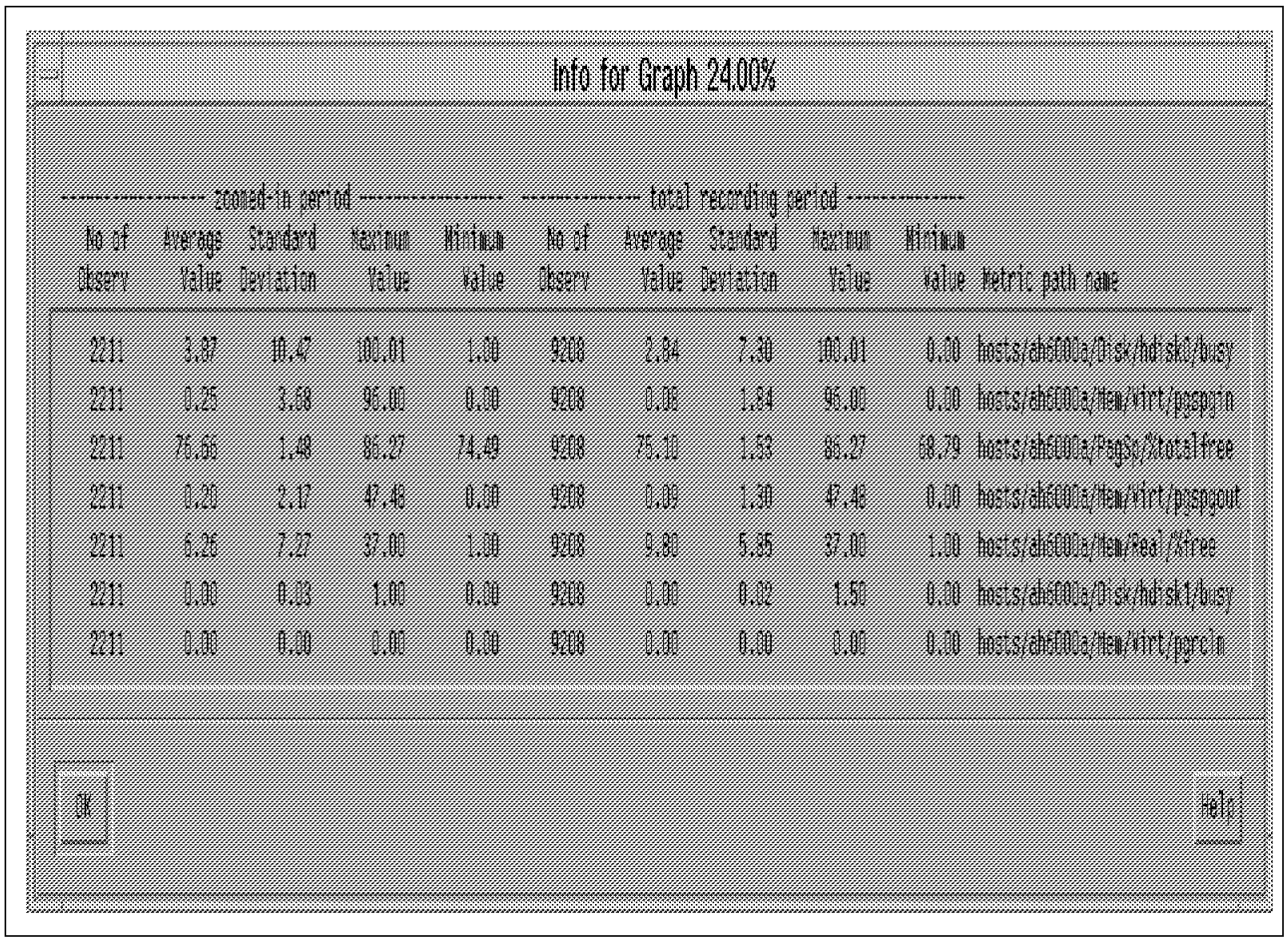


Figure 56. Information Window for a Zoomed-In Main Graph

5.11.4 Other Options

We have looked at a few of the actions that are available to you. You can explore the rest of the actions by simply moving the mouse pointer to the graph area of the zoomed-in main graph. Press and hold the middle mouse button. Move the drag icon to any of the icons in the **Actions** section.

- The *Config* icon is used to save, retrieve, and delete customized views of main graphs. The idea is that by saving a particular view, you can later use it for viewing other recordings and display the graphs in a way that allows you to immediately compare the data in the recordings.
- The *Info* icon is used to display summary information for either a single metric or all metrics in a main graph or in the metrics selection window.
- The *Print* icon is where you drop objects you want to print. This will cause either the ″Print box″ or the ″Report box″ to appear, depending on the object you dropped.
- The *View* icon is used to change the way a given metric (or all metrics) are plotted in any of the graphs where it appears. This is done by dragging a metric's object, a main graph, or the title of the metrics selection window to the icon.
- The *Scale* icon is used to change the scale for one or more metrics in a main graph. This is done by dragging an object to the icon.
- The *Annotate* icon is used to add or modify annotation text to a recording. Annotation text is kept in a separate file, linked to the recording file by a naming convention.
- *The Pit* icon is where you drop objects you no longer need. This will cause the object to be removed from where you dragged it, but it will never cause changes to the recording file. Removal of objects, thus, is only from the viewing environment you are in. It is always possible to start over by re-reading the recording file.

5.12 Conclusion

Performance Toolbox is a specialized tool that works with a suite of system and network management tools. PTX addresses the needs of a wide audience, from novice to expert. It allows you to complete a simple performance tuning cycle, monitoring, recording, analyzing, prescribing a remedy, setting alarm and alert conditions, and adjusting performance resource parameters either manually or automatically.

Chapter 6. Additional Performance Tools

Tuning is an integral requirement for any system. There are a variety of tools included with AIX V4.2 or separately purchasable that can help you in monitoring your system. But there are some other tools available either on the *IBM Developer Connection* CD-ROM, the *IBM Software Development Solutions for AIX Version 4* CD-ROM, or from World Wide Web sites. The objective of this chapter is to introduce some useful performance tools that are available in addition to the ones described in the previous chapters.

6.1 xgprof

The xgprof tool is an *extended graphical user interface profiler for AIX V4*. It is an enhancement to the gprof command, which is the standard AIX graph profiler.

The gprof command produces a text file for the user to read. The text file has two parts, the *flat profile* and the *call graph*. The flat profile is a list of functions sorted by how much time each function used. The call graph is a list or table describing the dynamic call graph. For more information about the gprof command, please refer to 2.13, “The gprof Command” on page 59.

The xgprof tool uses the same information as gprof and works in exactly the same way, but it provides a graphical user interface (GUI). Viewing the flat profile generated by gprof is useful, but the textual call graph table that gprof provides is useless for large programs. It would be very useful if it is displayed graphically. Furthermore, *statement-level profiling* is useful and can be provided through the gprof mechanism, but gprof does not display it.

To use the xgprof tool, the source code has to be compiled with the -pg option. Then use xgprof instead of gprof after the execution of the compiled program. The xgprof tool provides:

- Graphical display of the dynamic call graph.
- Displays source code in a text window, with statement level profiling.
- Displays disassembler code with instruction level profiling.
- Displays the flat profile in a text window.
- Displays the call graph table textual (like gprof).
- It works also with longer running programs which cause arithmetic overflow in gprof.

The graph editing/filter capabilities of xgprof are good, and it allows to filter the dynamic call graph based on paths through the graph. It also allows to cluster functions based on which load-unit they belong to.

The xgprof tool is available internally from the AIXTOOLS disks and externally via the *IBM Developer Connection* CD-ROM.

The following screen-shots show the xgprof windows for profiling the cwhet benchmark program, which was also used in 2.12.1, “The prof Implementation” on page 58.

Figure 57. The xgprof Start Window

The xgprof tool starts up with an overview over the call graph from the profiled program. By selecting a node with the mouse, a pop-up menu is displayed, which provides informations like name, self and descendent (similar to gprof). It allows to display the source code, the disassembler source, parents, and children and allows to zoom in. The following snapshot shows the zoomed-in window for the main routine.

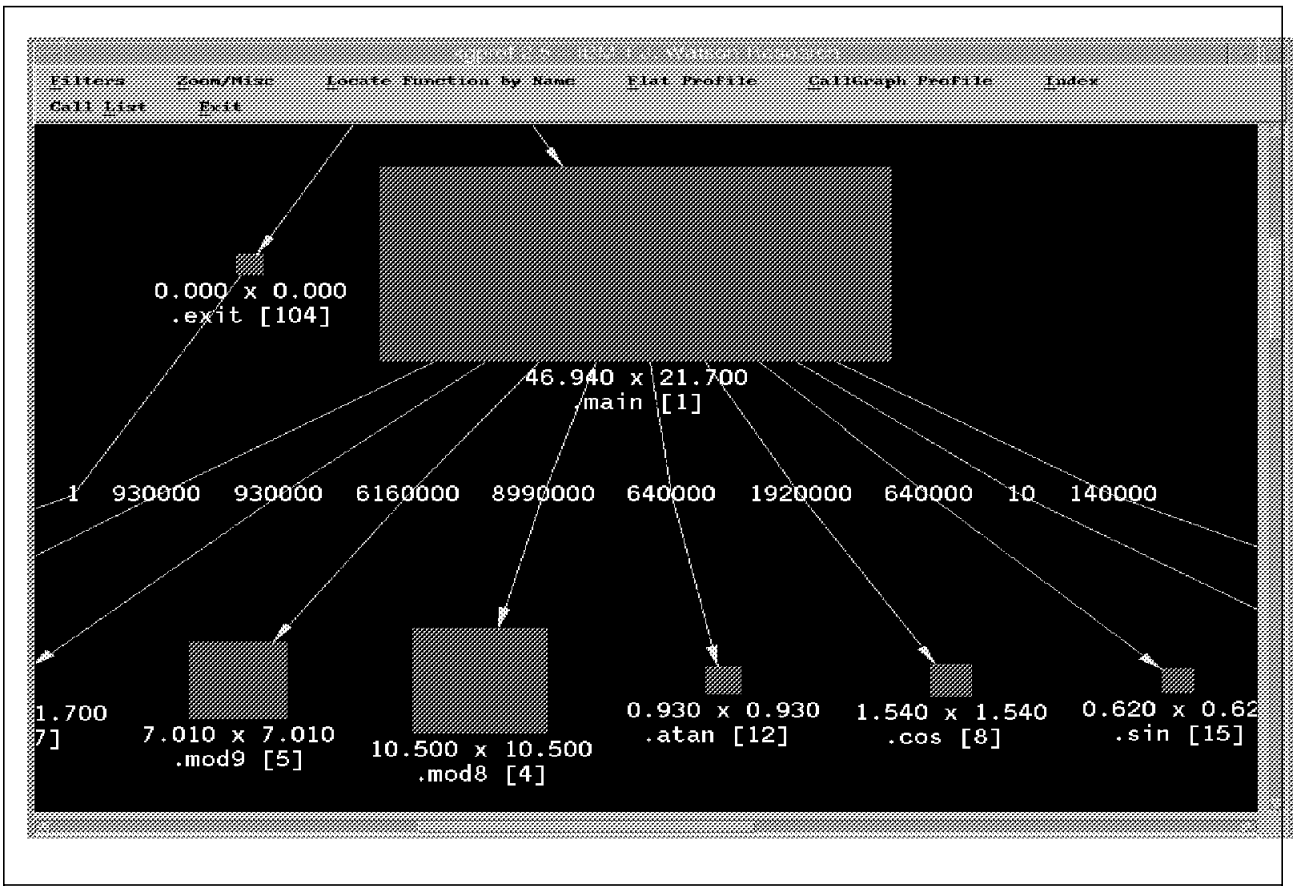


Figure 58. Zoomed-In Window of the Main Routine

It shows which functions are called by the main routine and how often. Via selecting the nodes with the mouse, additional information is available. Also the menu bar allows additional actions to provide more information. Comparing these windows and the additional information provided by the pop-up menu against the gprof textual call graph shows the advantage of the xgprof tool.

Additional information about the xgprof tool is available in the *xgprof User's Guide*, which is provided as a PostScript file on the *IBM Developer Connection* CD-ROM.

6.2 Program Visualizer

Reading trace output is not the easiest way to find performance problems, as you probably saw. But the data that trace provides is very useful. The Program Visualizer (PV) provides graphical, animated views of trace data.

PV for AIX allows you to graphically display the behavior of a target program and the system underlying it. PV is a program visualization system which provides concurrent visual presentation of behavior from all layers including: the program itself, user-level libraries, the operating system, and the hardware, as this behavior unfolds over time. The main features of PV are:

- Aids debugging and performance analysis by showing trends, anomalies, and interesting correlations that help track down pressing problems or reveal unexpected ones

- Helps you to understand the structure and dynamics of large applications, libraries, and frameworks
- Shows application-level activity such as algorithm phase transitions and execution time profiles, correlated to source code, if available
- Shows language and library runtime activity such as parallel-loop scheduling and dynamic memory allocation
- Shows operating-system-level activity such as context switches, address space activity, system calls and interrupts, and kernel performance statistics
- For POWER/2 architectures, shows hardware-level performance information such as instruction execution rates, cache utilization, processor element utilization, and delays due to branches and interlocks

You can use either a PV screen dialog or a PV trace utility named `pvtrace` to record the execution of a target program and create an AIX trace file with appropriate data. Then you use PV to replay the trace file as many times as you want, showing many different views of program and overall system behavior. Zoom-in on the problems in your target program by continually narrowing your focus and looking at more detailed views.

PV trace collection uses the AIX trace facility, plus some additional utility commands to augment the trace data and ensure that a certain subset of trace hooks are enabled during trace collection. A “normal” trace file, collected with the AIX `trace` command alone, can be recast with `trcrpt -r` so that PV will display it; however, usage of the PV trace-collection facilities is recommended.

The Program Visualizer (PV) tool is available internally from the AIXTOOLS disk and externally via the *IBM Developer Connection* CD-ROM or the *IBM Software Development Solutions for AIX Version 4* CD-ROM.

The principal World Wide Web site established for PV is at: `http://www.research.ibm.com/pv`.

6.2.1 pvtrace

The `pvtrace` utility is shipped with the Program Visualizer. The `pvtrace` utility enables the AIX trace facility, runs the given command with arguments, and then disables the AIX trace facility. The trace file is placed in your current directory and is called <command_name>.trc, where <command_name> is the final component of the <command> path name.

The `pvtrace` utility can be invoked from the command line or from a Trace Generation screen dialog available from the PV control panel. An example of invoking the `pvtrace` utility from the command line is shown below:

```
pvtrace /usr/bin/spell /usr/dict/words | wc
```

This command will generate a trace file named spell.trc for the `spell` program. For more information about the `pvtrace` utility, have a look at the *Program Visualizer (PV) Tutorial and Reference Manual* that comes with the tool.

6.2.2 Starting PV

Start PV by typing the following command at the prompt:

```
pv &
```

and press the **Enter** key. After a few seconds, the PV control panel should be displayed.

You may get the following error message: `Cannot allocate enough colors`. This can occur when you run PV with applications that also use custom colors. To correct this error, either end the application in question and restart PV, or you could try starting PV by typing:

```
pv -cmap &
```

You may get something like the following error messages:

```
Could not load program pv
Symbol _system_configuration in ksh is undefined
Symbol _fp_trapstate in ksh is undefined
```

These messages mean that the AIX system on which you are trying to install PV is not at level V3.2.5 or V4. You cannot run PV on a system with AIX level V3.2.4 or earlier.

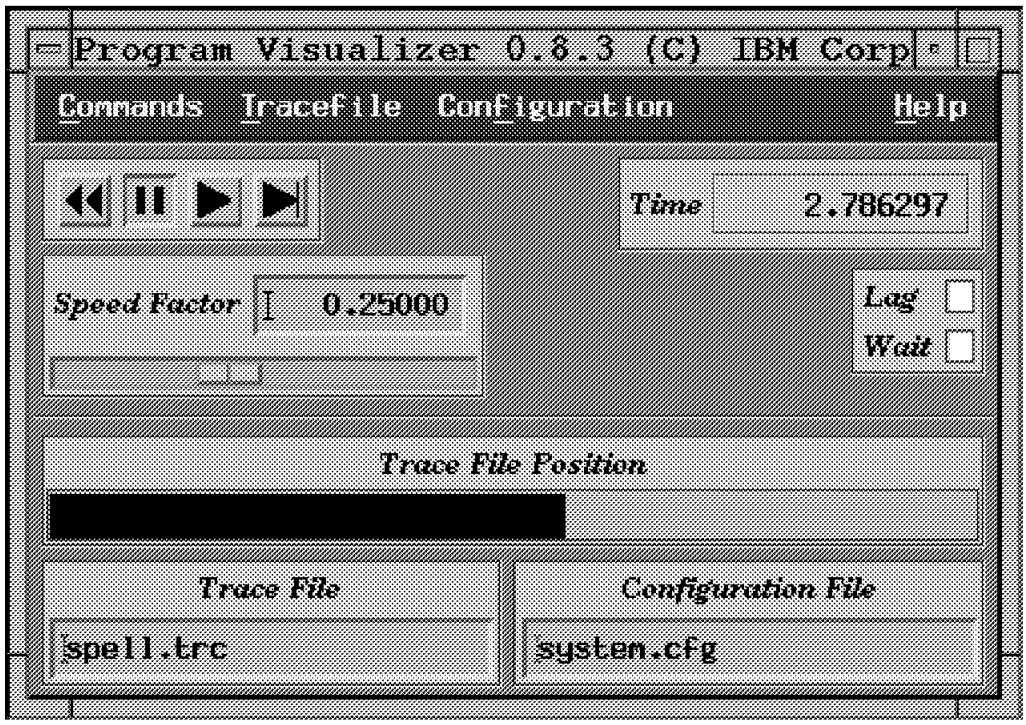


Figure 59. PV Control Panel

The control panel is where you tell PV the source of the information you want to display and how to display it. You enter this information into selection boxes in the control panel, and then you use the control panel to control how the trace file is replayed.

- The four control push-buttons starting from the left are:

Rewind Repositions the trace file to the beginning of the trace and clears out all information in the views

Pause Suspends replay of the trace

Play Replays the trace or resumes replay after the pause

Step Steps through the trace one event at a time

- The Time display shows the amount of original run time that has elapsed between the beginning of the trace and the current position in the trace.
- The Speed Factor slider controls the speed of play.
- The Lag indicator turns red if PV is unable to display events at the rate requested by the speed factor slider. Otherwise, the indicator is white.
- The Wait indicator turns red if PV is blocked, waiting for trace data, when it is consuming the trace file from a pipe and has read and displayed all available trace data. Otherwise, the indicator is white.
- The Trace File Position bar fills slowly to show how far replay has progressed through the trace. When the entire trace has been replayed, the bar is totally filled.
- The Trace File entry field displays the name of the trace file you are currently analyzing. A new trace file may be selected by entering its name in this entry field and pressing Enter.
- The Configuration File entry field displays the name of the configuration file you are currently using. A new configuration file may be selected by entering its name in this entry field and pressing Enter. The configuration file contains a collection of views that you can show using PV.

If you already had a trace file, you would enter its name into the Trace File entry field. Then you need to enter the configuration file name. A number of configuration files are provided with PV. To access these configuration files, select **Configuration.** Then select **Load Config...**. Move the mouse pointer to the configuration file you prefer and click on it. Then click on **OK**. To replay the trace file, click on the **PLAY** button.

If you do not have a trace file for your program, you will have to create one using the PV control panel. From the PV control panel, select **Tracefile.** Then from the Tracefile pull-down menu, select **Generate...**. The Trace Generation pop-up menu will be displayed.

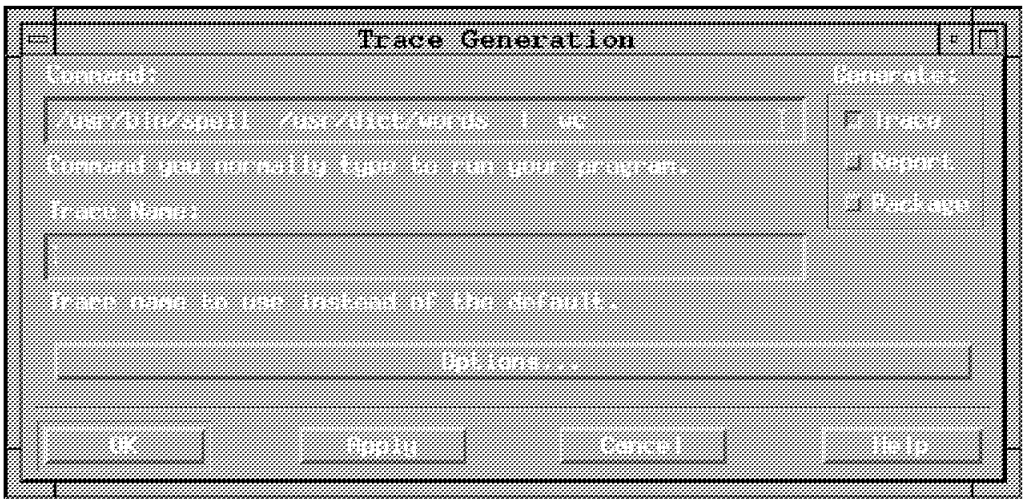


Figure 60. Trace Generation Pop-Up Menu

Enter the command you normally type to execute your program, in this case `/usr/bin/spell /usr/dict/words | wc` in the Command entry field. Then click on **OK**, and after a few seconds, the Question pop-up menu is displayed:

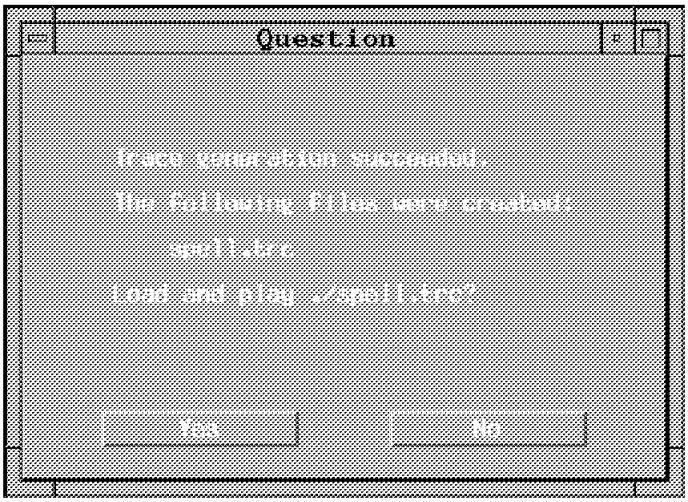


Figure 61. Question Pop-Up Menu

Click on **Yes** and while the spell.trc trace file is playing, six different views are displayed. Unfortunately we are not able to display all six views in one figure; therefore we will show three of these views in Figure 62 on page 258 and three more in Figure 63 on page 259.

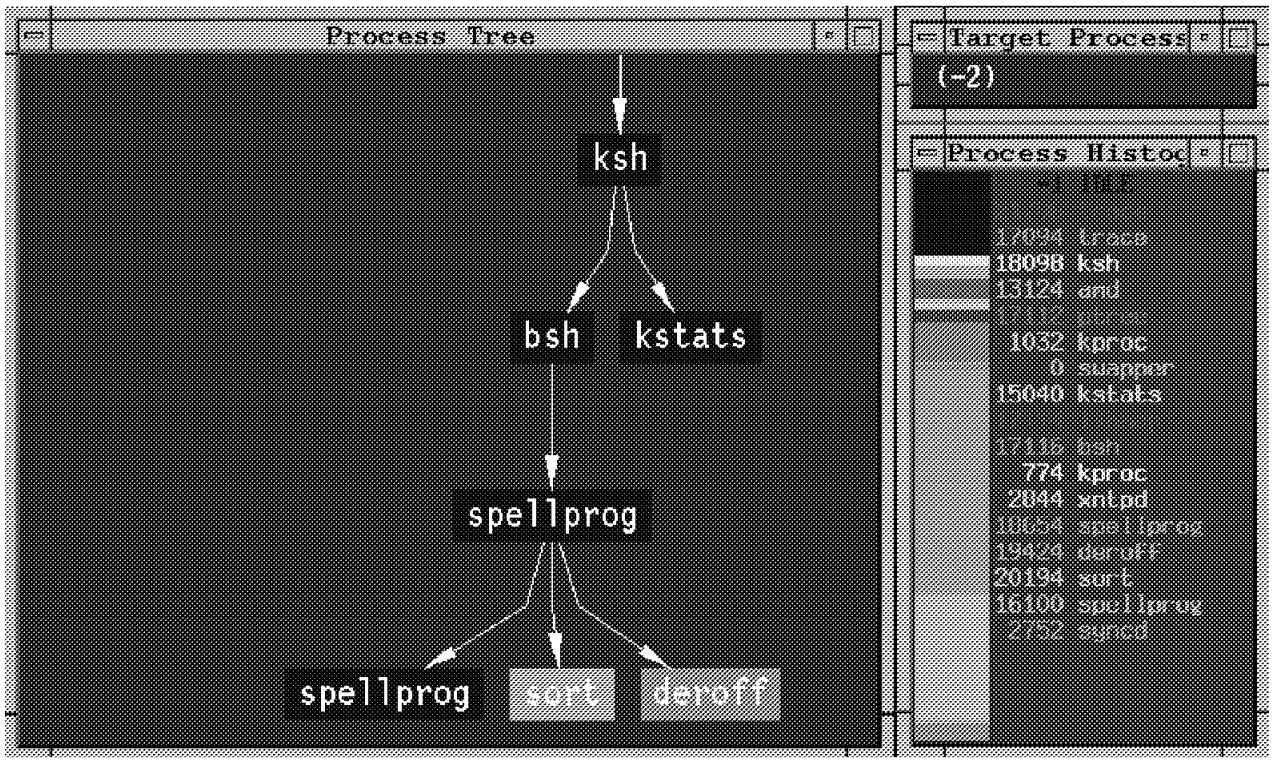


Figure 62. Partial PV Views (One)

The views displayed by PV:

- The Process Tree view shows all the processes that exist at a particular point in the trace. Each process appears as a rectangle, colored cool to warm to indicate the recent CPU consumption of that process.
- The Target Process identifies the target process selected by the user. The target process is the process in the trace that will be the subject of all views that display information specific to a particular process. A process identifier of -2 indicates that no target process has been specified.
- Process scheduling is presented in the Process Histogram view and the Current Process view. The Process Histogram view shows how much CPU time particular processes have accumulated up to the current point in the trace. The Current Process view shows which process was running at each instant in time as the trace was recorded.
- The System State view shows the sequence of system states such as system calls, interrupts, page faults, and user mode as they occurred at each instant in time. A user click in the view will pop up detailed information, as shown in Figure 63 on page 259.
- The kernel performance statistics are presented in the Kernel Statistics view, which shows the rates at which a selected set of kernel activities were occurring as those rates changed over time. The selected set shown depicts the rates at which system calls were made, disk blocks were read and written, and network bytes were received and transmitted at different times.

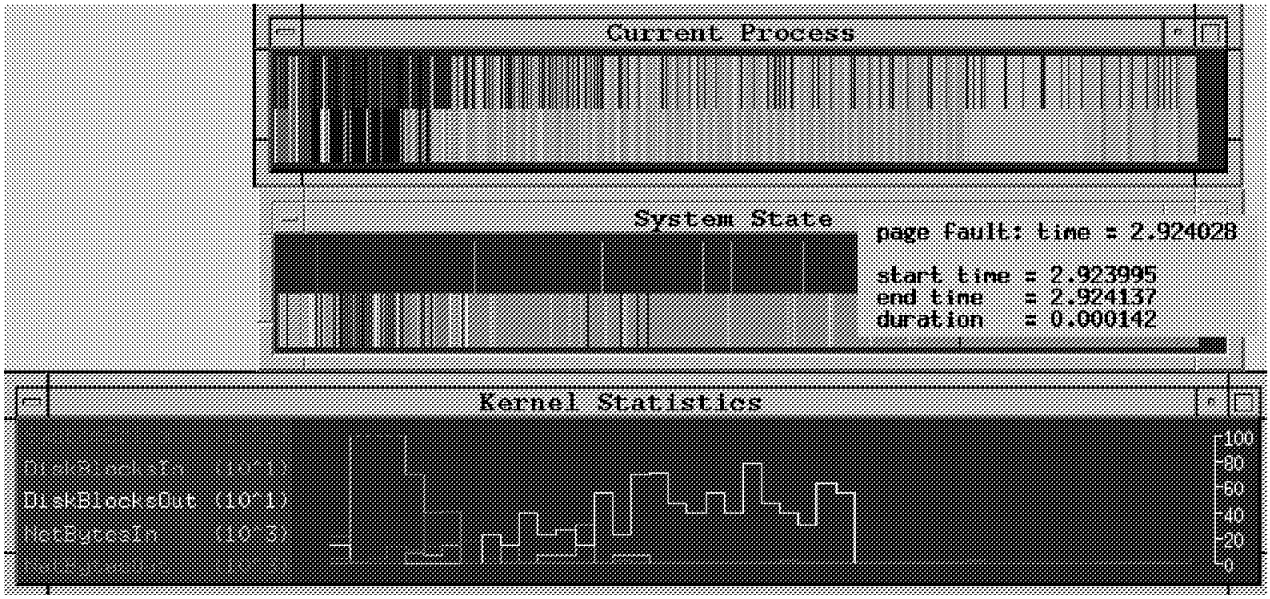


Figure 63. Partial PV Views (Two)

PV juxtaposes these time-oriented strips to allow the user to make visual correlations between system and process activity at any instant in time. Clicking on any of the time strips shows a pop-up with detail about what was happening at that instant in time. PV also provides a sophisticated zooming capability, coordinated among these views, to allow detailed visual examination of activity down to the level of individual trace events.

The tool contains many additional views that show memory operations, heap storage use, execution profiling information, and application function usage.

6.2.3 Packaging the Trace

To package a trace file for visualizing on a different machine, you can use the package utility shipped with PV called pvpackage. The pvpackage utility creates a compressed tar file that contains everything necessary for viewing a trace file on a machine other than the one on which it was created.

You can package the spell.trc trace file by typing the command:

```
pvpackage -v -s spell.trc
```

which creates the spell.pkg.tar.Z file in the current directory.

You are also able to package your trace file from the PV control panel. From the Tracefile option on the PV control panel, you can either package your trace file during the trace generation process or you can generate only the trace package from an existing trace file.

During the trace generation process, click on **Package** when the trace generation pop-up menu is displayed (see Figure 60 on page 257). Trace and Package will now be selected. Then press enter or click on **OK**. The trace file and the compressed tar file will be created.

To generate only the trace package from an existing trace, click on the Trace Name entry field when the trace generation pop-up menu is displayed, and type

the name of the trace file you wish to use to generate the trace package. Click on the **Trace** check button to turn off trace generation. Click on the **Package** check button to select it. Then press **Enter** or click on **OK.** Only the compressed tar file will be created.

The compressed trace package file can then be moved to a different machine, uncompressed and extracted via tar into a clean directory, and viewed by PV in the normal way. The trace file and all the associated symbol table information will be available in that directory.

6.2.4 Making a Textual Trace Report Using pvreport

To format a standard AIX trace file for PV's Trace Report view (or for human viewing), you can use the report utility shipped with PV called `pvreport`. The `pvreport` utility takes a trace file (<file>.trc) and generates a textual report file (<file>.rpt) and a corresponding line index file (<file>.tli). The resulting report file (with its index) can be viewed using the PV Trace Report view. As you use some other PV views to select trace events and examine them, PV can automatically scroll the large trace report to highlight the lines relevant to each event.

To generate a textual trace report called spell.rpt and a index file for PV called spell.tli, type the following:

```
pvreport spell.trc
```

To view the resulting report file using the PV Trace Report view, proceed as follows:

Select **Configuration** from the PV control panel. Then select **Add View ...**. Scroll down to TraceReport.view on the Add View File pop-up menu. Then click on **TraceReport.view** to select it. Click on **OK**. The Trace Report window pops up on the screen. TraceReport.view will be displayed in the Configuration File entry field on the PV control panel.

On the PV control panel, enter the trace file name into the Trace File field. The trace file will be loaded, and its associated textual trace file report will appear in the Trace Report window.

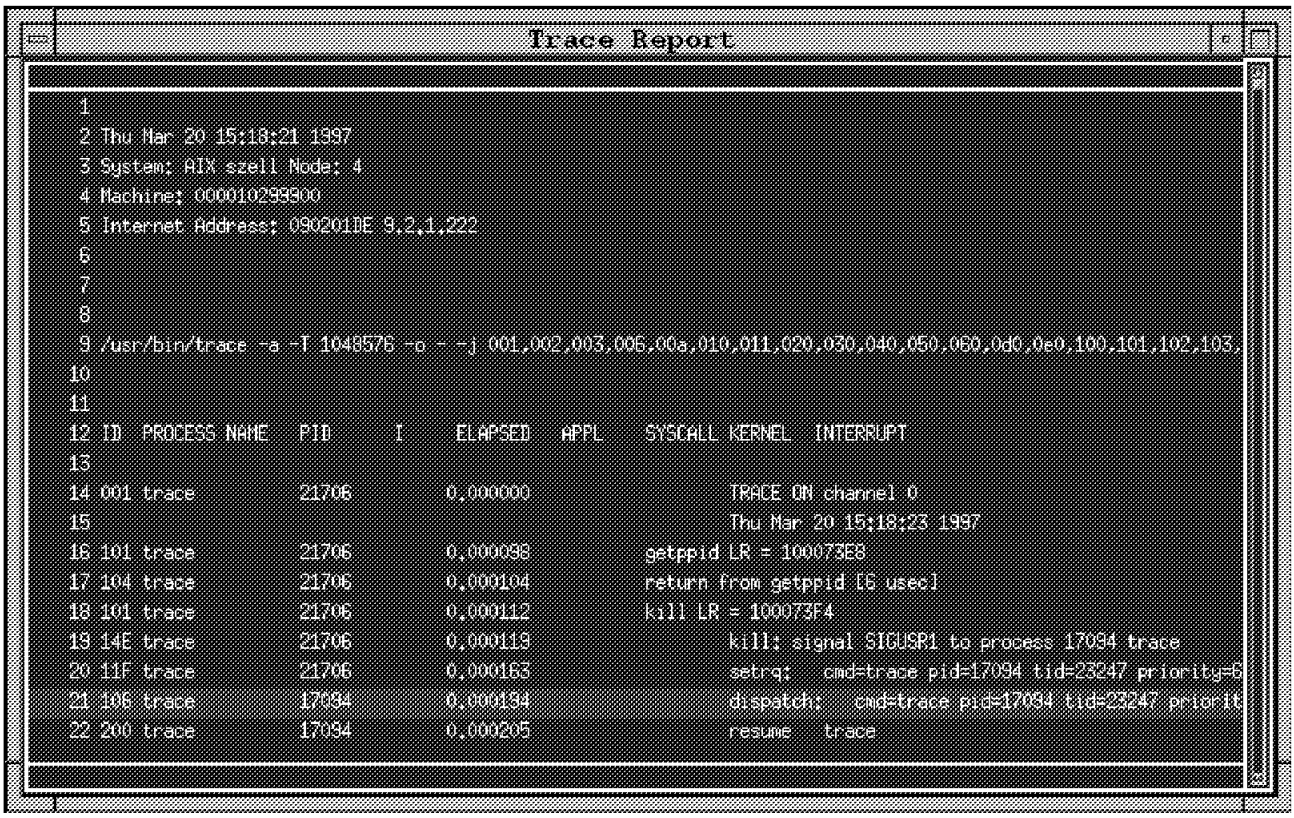


Figure 64. Trace Report View

6.2.5 PV Tutorial

With PV you will receive a tutorial that shows you how to use PV. If you are not familiar with PV, work your way through the lessons. The lessons provide you with a feel for the capabilities of PV by touching on a sampling of the many features and functions of PV. Topics in the tutorials include:

- Examining process context switching
- Studying memory operations and heap storage usage
- Finding execution hot spots
- Using phase markers to trace program activity

PV also includes a full explanatory help facility.

For further information on the World Wide Web, see: `http://www.research.ibm.com/pv`.

6.3 utld

The `utld` command is a trace-based tool that was originally used in the development of AIX, but has now been made generally available; it can be obtained on the Internet via anonymous FTP from `ftp.software.ibm.com` in the directory /aix/tools/perftools.

The `utld` tool reports on system locks and delays. Although intended for use on all AIX platforms, this tool is especially useful in an SMP environment. It reports thread/processor affinity and gives a greater insight into locking that can be

gained by using tools such as lockstat. This tool also provides a very useful summary of system utilization.

Further documentation on this command is included in its respective package.

Important!

Please note that this tool is currently available free of charge and as such is provided without warranty or support. Because it is a development tool, it may change significantly in future releases. The following notes are intended for use with Version 1.x of the tool, for AIX V4.1.4 or later.

Since this tool depends on data captured using the trace command, we first take a look at how to capture trace data and also how to create a trace names file. For more information on trace, see 3.12, “The trace and trcrpt Commands” on page 120.

6.3.1 Generating a Trace for utld

When using trace-based tools, information must first be captured by using the trace command. The trace data for utld can be collected through the following commands:

```
# trace -a -d -f -T 80000000 -L 80000000 -o ./trace.out
# trcon ; sleep 60 ; trcoff
# trcstop
```

The trace command is started detached (-a) and deferred (-d). This puts trace running in the background but not yet collecting trace hooks. This trace command will collect 80 MB of trace data (-T) and stop collecting trace data when the buffer fills (-f). It is important that the system has sufficient free memory (> 80 MB) that can be dedicated to the trace command, or the command will fail or the workload will be perturbed (perhaps by excessive paging). The maximum length for the output file is 80 MB (-L), and the output is written to the file ./trace.out (-o). The trace data is collected in a 60-second interval.

It is unlikely, that a full 60-second trace will be collected due to the use of the -f flag. You will actually get a trace corresponding to one full trace buffer (that is, 80 MB). Please bear in mind that trace adds considerable path length in system time; so using trace will distort the user/system time mix.

The trace data collected will be in binary format that can be translated into ASCII by the trcrpt program. To translate a trace file (for example, trace.out), enter the following:

```
# trcrpt ./trace.out | pg
```

Capturing Names: In order for the trace tools to provide meaningful reports, the “names” of the system must be collected by using the trcnm tool. This tool builds a load map of what’s in the AIX kernel so things like device drivers can be identified in the trace. The following command can be used to collect names:

```
# trcnm > names.out
```

If you are unsure of what device driver names map to adapters, use the command:

```
# lsdev -Cc adapter
```

to get an explanation of the function of each of the adapter drivers in your system. The third step is to unwrap the trace. Since the trace file is collected in a double buffered fashion, it must be put into the correct order. The following command will unwrap the trace:

```
# trcrpt -r trace.out > trace.unwrapped
```

Ensure you have enough space in the file system before entering this command because it effectively duplicates the trace. Note, this example required 260 MB of disk space. More space will be required on an active system.

6.3.2 Using utld

Once the trace has been unwrapped, the utld command can be invoked as follows:

```
# utld -i trace.unwrapped -n names.out > utld.out
```

Note that if the -n option is omitted, the utld command will automatically generate a list of names using the trcnm command as outlined above.

This command produces a summary report in the utld.out file. If you require all locking details, run the command as follows:

```
# utld -i trace.unwrapped -n names.out -d -l lock.out > utld.out
```

Note

In order for the trace command to capture information regarding locks, MP Lock Instrumentation must be enabled. For further details regarding this, please see 4.7, “The lockstat Command” on page 164.

If the -d -l <filename> option is specified when invoking utld, a separate report detailing the locking statistics can be obtained. To look at locking in greater detail, it is necessary to provide utld with a larger dictionary of names. The larger dictionary is created by using the getnames executable provided in the utld package, as follows:

```
# getnames > names.out
```

The getnames facility uses the genkex executable, provided as part of the perfagent.tools section of Performance Toolbox. This must be installed on the system in order for this to work.

Using getnames is only necessary if detailed locking information is required.

This time, utld will create the following files:

utld.out	A summary report
lock.out	A detailed report on process locks
lock.out.details	A detailed report on locks per process

Once the utld command has completed, the lock.out and lock.out.details files will be very large. The utld.out report contains some very useful information for anyone concerned with SMP performance. However, lock.out and

lock.out.details are very in-depth technical reports designed primarily for use by application developers.

Sample utld Report: The following screen shots are examples of output contained in the utld.out report:

The first section of the utld report (see following figure) is a summary of CPU consumption during the trace period. This breaks down CPU usage into five categories:

Application More commonly referred to as user mode

Kernel The time spent in kernel mode

FLIH The amount of time spent dealing with first line interrupt handlers

SLIH The amount of time spent in second line interrupt handlers

DISPATCH The amount of time spent dispatching threads

```
                    SYSTEM SUMMARY

 processing        percent       percent
total (msecs)     tot  time     busy time     processing category
=============     =========     =========     ====================
  4834.901          47.452       47.452       APPLICATION
  3249.240          31.890       31.890       KERNEL
  1907.669          18.723       18.723       FLIH
   132.542           1.301        1.301       SLIH
    64.625           0.634        0.634       DISPATCH
-------------     ---------     ---------
 10188.976         100.000      100.000       CPU(s) busy time
     0.000           0.000                    WAIT
-------------     ---------
 10188.976         100.000                    TOTAL

          Total number of process dispatches          =  5041
          Average time between same process dispatch  =  20.475002 msec.
          Average process to processor affinity       =  0.528291
```

If the trace was collected on an SMP server (as in these examples), the CPU breakdown will include a weighted average of all the processors, as well as information for each processor. For an SMP, the report will also include the number of dispatches and a measure of the processor affinity of the threads.

Following the system summary, there is a section detailing the system usage by processor.

The next section presents a list of all processes/threads that ran during the trace, and for each, specifies the amount of time spent in kernel and user mode (see below).

```
                    APPLICATION and KERNEL SUMMARY (Per Thread/Process)

--  processing total  (msecs) --    -- percent of total processing time --
  combined   application  kernel    combined      app      kernel   process name (proc id / thrd id)
  ========   ===========  ======    ========      ===      ======   =======================
   773.399    773.399      0.000      7.591      7.591      0.000   trace ( 18262  18527 )
   683.106    656.024     27.082      6.704      6.439      0.266   nroff ( 17974  18239 )
   660.407    656.398      4.008      6.482      6.442      0.039    no name ( 18456  18721 )
```

The above information is then condensed into time spent per type of process, headed by title information reporting the total number of threads, as follows:

```
                    APPLICATION and KERNEL SUMMARY (Per Process Type)
                    total number of threads = 57

--  processing total  (msecs) --    -- percent of total processing time --
  combined   application  kernel    combined      app      kernel   process name (count)
  ========   ===========  ======    ========      ===      ======   ======================
   773.399    773.399      0.000      7.591      7.591      0.000   trace ( 1 )
   730.833    698.473     32.361      7.173      6.855      0.318   nroff ( 2 )
  4560.335   2333.526   2226.809     44.758     22.902     21.855    no name ( 14 )
   332.740     15.698    317.042      3.266      0.154      3.112   cpio ( 1 )
   321.571    237.929     83.643      3.156      2.335      0.821   xlcentry ( 1 )
   189.248     62.744    126.504      1.857      0.616      1.242   ed ( 1 )
```

The third part of the report breaks down the total kernel execution time into a list of all system calls that were made:

```
                      KERNEL (System Call) Summary

 processing        percent                - path in msecs         -
total (msecs)     proc time     count     -min-     -avg-     -max-     system call
=============     =========     =====     =====     =====     =====     ===========
    744.791         7.310    000000193     1.598     3.859     7.323     creat
    431.640         4.236    000000659     0.098     0.655     2.109     statx
    317.197         3.113    000000204     0.935     1.555     4.080     chown
    305.065         2.994    000000205     0.902     1.488     3.697     chmod
    269.923         2.649    000000196     0.659     1.377     3.191     utimes
    207.466         2.036    000000284     0.193     0.731     5.273     open
     71.907         0.706    000000544     0.004     0.132     0.506     close
     69.426         0.681    000000016     2.385     4.339     7.686     unlink
```

The next two sections detail the FLIH and SLIH statistics (see below). The final section presents a summary of the idle process for each processor.

```
                    FLIH Summary

 processing       percent                - path in msecs        -
total (msecs)    proc time     count     -min-     -avg-     -max-     flih type
=============    =========     =====     =====     =====     =====     ====================
   1158.986        2.663    000004931     0.005     0.235     0.852     DECREMENTER
    437.893        1.006    000000616     0.034     0.711     1.482     DATA ACCESS PAGE FAULT
    182.108        0.418    000000495     0.010     0.368     0.897     I/O INTERRUPT
     13.889        0.032    000000089     0.003     0.156     0.552     level 50
      0.038        0.000    000000004     0.004     0.010     0.022     FLOATING POINT UNAVAIL
-------------    ---------     -----
   1792.913        4.119    000006135


                    SLIH Summary

 processing       percent                - path in msecs        -
total (msecs)    proc time     count     -min-     -avg-     -max-     slih type
=============    =========     =====     =====     =====     =====     ====================
     62.156        0.143    000000340     0.066     0.183     0.300     ascsiddpin
     50.828        0.117    000000155     0.070     0.328     0.921     tokdd
-------------    ---------     -----
    112.984        0.260    000000495
```

The utld command is an extremely powerful tool. It is beyond the scope of this redbook to cover all the options and reports available. More information can be obtained from the ultd.doc file provided with the package.

6.4 Sources for Additional Tools

xgprof and PV: The Program Visualizer (PV) and xgprof can be obtained from the *IBM Developer Connection*, Z121-0200, CD-ROM. More information can be obtained at: http://www.developer.ibm.com/devcon/index.html

PV can also be found on the *IBM Software Development Solutions for AIX Version 4*, SK2T-2729, CD-ROM.

There also is a Web site for PV: http://www.research.ibm.com/pv

utld: The tool utld can be obtained at this WWW address:

ftp://ftp.software.ibm.com/aix/tools/perftools/utld/utld.obj

General Tools Repository: A general repository for useful tools of all sort can be found at: http://aixpdslib.seas.ucla.edu/aixpdslib.html

Appendix A. Summary of Rules of Thumb

Rules	Interpretations/Actions
vmstat: If cpu sy+us > 80 percent	May be a CPU problem
vmstat: If > 5 processes in run queue	May be a CPU problem
vmstat: If po/fr > 1/6 with large wa and paging space I/O is high	System could be thrashing
vmstat: If fr : sr is high	Memory may be overcommitted
ps v or svmon: If memory use (SIZE in ps v or private + lib data work pages in svmon) increase with time	Memory leak in process PID
lockstat: If Ref/s > 10,000	Lock contention
schedtune and vmtune: Defaults settings	Reasonable in most cases
schedtune: m parameter	Set to number of typical jobs which fits in the RAM comfortably
schedtune: e parameter	Set to the run time of the typical job you like to push through quickly when thrashing
vmtune: minfree	Maximal size of commonly used programs that need quick response
vmtune: maxfree	minfree + max(8, maxpgahead)
iostat: If %iowait > 25 percent	I/O or disk-bound situation
iostat: If %tm_act > 40 percent or high Kb_read & Kb_wrtn	Disk may be overloaded
iostat: If sum of Kbps of disks attached to a SCSI adapter > 70 percent of SCSI adapter's throughput rating	SCSI adapter may be saturated
filemon: If number of read+write sequences approach read+write	Physical disk access is more random than sequential
fileplace: High space efficiency	File less fragmented
fileplace: High sequentiality	Better sequential file access
netstat -i: If Oerrs > 0.01 X Opkts	Increase device driver send queue size
netstat -i: If Ierrs > 0.01 X Ipkts	Use netstat -m to check for lack of memory
netstat -v: If (Max Collision Errors + Timeout Errors) / Transmit Packets > 5 percent	Network should be reorganized to balance the load
netstat -v: If Number of Collisions / Number of Transmit Packets > 0.1	High network utilization
netstat -v: If Received Broadcast Packets / Total Received Packets > 0.2	Indication for high network load
netstat -v: If Max Packets on S/W Transmit Queue = Current Queue Size or S/W Transmit Queue Overflow > 0	Increase driver receive queue size

Rules	Interpretations/Actions
netstat -v: If No mbuf Errors is large	Increase thewall
netstat -m: If there are requests for mbufs denied	Increase thewall
netstat -p ip: If there are bad header checksums or fragments dropped (dup or out of space)	Indicates network corrupting packets or device driver receive queue is not large enough
netstat -s: If ip shows fragments dropped after timeout	Increase ipfragttl if due to network busy or increase sb_max or thewall if due to lack of mbuf
netstat -s on NFS server: If udp socket buffer overflows > 5	Increase server's nfsd, sb_max, udp_recvspace or nfs_socketsize
netpmon: If Network CPU% / CPU% > 0.5 from Process CPU Usage Statistics for NFS server	Majority of CPU usage is network-related
netpmon: If System Wait Time Due to Network Calls are high from the Network Wait Time Statistics for NFS client	Poor performance is due to network
netpmon: If Transmit and Receive Packet Sizes are small from the Device Driver Statistics	Modify current mtu size
nfsstat: If retrans / calls > 0.05 or timeouts are frequent or badxid close to timeout	For NFS client, server overload or network busy
nfsstat -s on NFS server: If nullrecv is large	Too many nfsd daemons are defined
nfsstat: If timeouts and badxids > 5 percent of the total	Increase the timeo parameter
no: sb_max > 2 X the size of the largest socket buffer	Recommended value
Set xmt_que_size to the maximum 150	Recommended value
For servers with more than two NFS clients: number of client's biod=2 per files written simultaneously (4 for file size > 32 Kb) & number of server's nfsd=total number of clients' biod + 20 percent	Recommended value
For dedicated NFS server: Set maxperm to 100 percent	Recommended value
For NFS server: Set sb_max, udp_sendspace and udp_recvspace to 131072	Recommended value

Table 5. Rules of Thumb

Appendix B. Summary of Tunable AIX Parameters

Each of the following sections describes the AIX parameters that can affect performance. The parameters are described in alphabetical order.

- **arpt_killc**

  Purpose: Time before an inactive, complete ARP entry is deleted.

  Values: Default: 20 (minutes), Range: N/A

  Display: `no -a` or `no -o arpt_killc`

  Change: `no -o arpt_killc=NewValue`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `no` command to /etc/rc.net.

  Diagnosis: N/A

  Tuning: To reduce ARP activity in a stable network, `arpt_killc` can be increased. This is not a large effect.

  Refer to: N/A

- **biod Count**

  Purpose: Number of `biod` processes available to handle NFS requests on a client.

  Values: Default: 6, Range: 1 to any positive integer.

  Display: `ps -ef | grep biod`

  Change: `chnfs -b NewValue`. Change normally takes effect immediately and is permanent. The `-N` flag causes an immediate, temporary change. The `-I` flag causes a change that takes effect at the next boot.

  Diagnosis: `netstat -s` to look for UDP socket buffer overflows.

  Tuning: Increase number until socket buffer overflows cease.

  Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″How Many `biod`s and `nfsd`s Are Needed for Good Performance?″.

- **Disk Adapter Outstanding-Requests Limit**

  Purpose: Maximum number of requests that can be outstanding on a SCSI bus. (Applies only to the SCSI-2 Fast/Wide Adapter.)

  Values: Default: 40, Range: 40 to 128

  Display: `lsattr -E -l scsin -a num_cmd_elems`

  Change: `chdev -l scsin -a num_cmd_elems= NewValue`. Change is effective immediately and is permanent. If the `-T` flag is used, the change is immediate and lasts until the next boot. If the `-P` flag is used, the change is deferred until the next boot, and is permanent.

  Diagnosis: N/A

  Tuning: Value should equal the number of physical drives (including those in disk arrays) on the SCSI bus, times the queue depth of the individual drives.

  Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″Setting SCSI-Adapter and Disk-Device Queue Limits″.

- **Disk Drive Queue Depth**

  Purpose: Maximum number of requests the disk device can hold in its queue.

  Values: Default: IBM disks=3, Range: N/A. Default: Non-IBM disks=0, Range: specified by manufacturer.

  Display: `lsattr -E -l hdiskn`

  Change: `chdev -l hdiskn -a q_type=simple -a queue_depth=NewValue`. Change is effective immediately and is permanent. If the `-T` flag is used, the change is immediate and lasts until the next boot. If the `-P` flag is used, the change is deferred until the next boot, and is permanent.

  Diagnosis: N/A

  Tuning: If the non-IBM disk drive is capable of request queuing, this change should be made to ensure that the operating system takes advantage of the capability.

  Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″Setting SCSI-Adapter and Disk-Device Queue Limits″.

- **dog_ticks**

  Purpose: Timer granularity for `IfWatchdog` routines. This value is not used in AIX.

  Values: Default: 60

  Display: N/A

  Change: N/A

  Diagnosis: N/A

  Tuning: N/A

  Refer to: N/A

- **fork() Retry Interval**

  Purpose: Specify the amount of time to wait to retry a `fork` that has failed for lack of paging space.

  Values: Default: 10 (10-millisecond clock ticks), Range: 10 to n clock ticks.

  Display: `schedtune`

  Change: `schedtune -f NewValue`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `schedtune` command to /etc/inittab.

  Diagnosis: If processes have been killed for lack of paging space, monitor the situation with the `sigdanger()` subroutine.

  Tuning: If the paging-space-low condition is only due to brief, sporadic workload peaks, increasing the retry interval may allow processes to delay long enough for paging space to be released. Otherwise, make the paging spaces larger.

  Refer to: N/A

- **ipforwarding**

  Purpose: Specifies whether the kernel forwards IP packets.

Values: Default: 0 (no), Range: 0 to 1

Display: `no -a` or `no -o ipforwarding`

Change: `no -o ipforwarding=NewValue`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `no` command to /etc/rc.net.

Diagnosis: N/A

Tuning: This is a configuration decision with performance consequences.

Refer to: N/A

- **ipfragttl**

  Purpose: Time to live for IP packet fragments.

  Values: Default: 60 (seconds), Range: 60 to n

  Display: `no -a` or `no -o ipfragttl`

  Change: `no -o ipfragttl=NewValue`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `no` command to /etc/rc.net.

  Diagnosis: `netstat -s`

  Tuning: If value of IP: fragments dropped after timeout is nonzero, increasing `ipfragttl` may reduce retransmissions.

  Refer to: N/A

- **ipqmaxlen**

  Purpose: Specify the maximum number of entries on the IP input queue.

  Values: Default: 50, Range: 50 to n

  Display: `no -a` or `no -o ipqmaxlen`

  Change: `no -o ipqmaxlen=NewValue`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `no` command to /etc/rc.net.

  Diagnosis: Use `crash` to access IP input queue overflow counter.

  Tuning: Increase size.

  Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″IP Protocol Performance Tuning Recommendations″.

- **ipsendredirects**

  Purpose: Specifies whether the kernel sends redirect signals.

  Values: Default: 1 (yes), Range: 0 to 1

  Display: `no -a` or `no -o ipsendredirects`

  Change: `no -o ipsendredirects=NewValue`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `no` command to /etc/rc.net.

  Diagnosis: N/A

  Tuning: N/A. This is a configuration decision with performance consequences.

  Refer to: N/A

- **loop_check_sum (AIX Version 3.2.5 only)**

Purpose: Specifies whether checksums are built and verified on a loopback interface. (This function does not exist in AIX Version 4.1.)

Values: Default: 1 (yes), Range: 0 to 1

Display: `no -a` or `no -o loop_check_sum`

Change: `no -o loop_check_sum=0`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `no` command to /etc/rc.net.

Diagnosis: N/A

Tuning: Turning checksum verification off (`loop_check_sum=0`) is recommended.

Refer to: N/A

- **lowclust (AIX Version 3.2.5 only)**

  Purpose: Specifies the low-water mark for the mbuf cluster pool.

  Values: Default: configuration-dependent, Range: 5 to n

  Display: `no -a` or `no -o lowclust`

  Change: `no -o lowclust=NewValue`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `no` command to /etc/rc.net.

  Diagnosis: `netstat -m`

  Tuning: If *"*requests for memory denied*"* is nonzero, increase `lowclust`.

  Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 *"*AIX Version 3.2.5 mbuf Pool Performance Tuning*"*.

- **lowmbuf (AIX Version 3.2.5 only)**

  Purpose: Specifies the low-water mark for the mbuf pool

  Values: Default: configuration-dependent, Range: 64 to n

  Display: `no -a` or `no -o lowmbuf`

  Change: `no -o lowmbuf=NewValue`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `no` command to /etc/rc.net.

  Diagnosis: `netstat -m`

  Tuning: If *"*requests for memory denied*"* is nonzero, increase `lowmbuf`.

  Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 *"*AIX Version 3.2.5 mbuf Pool Performance Tuning*"*.

- **lowthresh**

  Purpose: Specifies the maximum number of bytes (in percentage) that can be allocated by the `thewall` parameter using `allocb()` for the BPRI_LO priority. When the total amount of memory allocated by the `net_malloc()` subroutine reaches this threshold, the `allocb()` request for the BPRI_LO priority returns 0. The `lowthresh` parameter can be set to any value between 0 and 100, inclusively. The default value is 90, indicating the threshold is at 90 percent of the value of the `thewall` parameter.

  Values: Default: 90, Range 0 to 100

Display: `no -a` or `no -o lowthresh`

Change: `no -o lowthresh=NewValue`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `no` command to /etc/rc.net.

Diagnosis: N/A

Tuning: N/A

Refer to: N/A

- **lvm_bufcnt (AIX Version 4 only)**

  Purpose: The number of LVM buffers for raw physical I/Os.

  Values: Default: 9, Range: 1 to 64

  Display: `vmtune`

  Change: `vmtune -u NewValue`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding vmtune command to /etc/inittab.

  Diagnosis: Applications doing large writes to striped raw logical volumes are not getting the desired throughput rate.

  Tuning: If a system is configured to have striped raw logical volumes and is doing writes greater than 1.125 MB, increasing this value may help throughput of the application.

  Refer to: `vmtune` command

- **maxbuf**

  Purpose: Number of (4 KB) pages in the block-I/O buffer cache.

  Values: Default: 20, Range: x to y

  Display: `lsattr -E -l sys0 -a maxbuf`

  Change: `chdev -l sys0 -a maxbuf=NewValue`. Change is effective immediately and is permanent. If the `-T` flag is used, the change is immediate and lasts until the next boot. If the `-P` flag is used, the change is deferred until the next boot and is permanent.

  Diagnosis: N/A

  Tuning: This parameter normally has little performance effect on an AIX system, since ordinary I/O does not use the block-I/O buffer cache.

  Refer to: N/A

- **max_coalesce**

  Purpose: Specifies the maximum size, in bytes, of requests that the SCSI device driver will coalesce from the requests in its queue.

  Values: Default: 64 KB, Range: 64 KB to 2 GB

  Display: `odmget`

  Change: `odmdelete`, `odmadd`, `bosboot`. Change takes effect at next boot and is permanent.

  Diagnosis: N/A

  Tuning: Increase if striped logical volumes or disk arrays are in use.

Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″Modifying the SCSI Device Driver max_coalesce Parameter″.

- **maxfree**

  Purpose: The maximum size to which the VMM page-frame free list will grow by page stealing.

  Values: Default: configuration-dependent, Range: 16 to 204800 (4 KB frames)

  Display: vmtune

  Change: vmtune -F NewValue. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding vmtune command to /etc/inittab.

  Diagnosis: Observe free-list-size changes with vmstat n.

  Tuning: If vmstat n shows free-list size frequently driven below minfree by application demands, increase maxfree to reduce calls to replenish free list. Generally, keep maxfree - minfree <= 100.

  Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″Tuning VMM Page Replacement″.

- **maxperm**

  Purpose: The percentage of memory page frames occupied by permanent pages above which only permanent pages will have their frames stolen.

  Values: Default: 80 percent of (memory size - 4 MB), Range: 5 to 100

  Display: vmtune

  Change: vmtune -P NewValue. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding vmtune command to /etc/inittab.

  Diagnosis: Monitor disk I/O with iostat n.

  Tuning: If some files are known to be read repetitively, and I/O rates do not decrease with time from startup, maxperm may be too low.

  Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″Tuning VMM Page Replacement″.

- **maxpgahead**

  Purpose: The upper limit on the number of pages the VMM will read ahead when processing a sequentially accessed file.

  Values: Default: 8, Range: 0 to 16

  Display: vmtune

  Change: vmtune -R NewValue. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding vmtune command to /etc/inittab.

  Diagnosis: Observe the elapsed execution time of critical sequential-I/O-dependent applications with time command.

  Tuning: If execution time decreases with higher maxpgahead, observe other applications to ensure that their performance has not deteriorated.

Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″Tuning Sequential Read Ahead″.

- **maxpin (AIX Version 4 only)**

  Purpose: The maximum percentage of real memory that can be pinned.

  Values: Default: 80 (percent of RAM), Range: At least 4 MB pinable to at least 4 MB unpinable.

  Display: `vmtune`

  Change: `vmtune -M NewValue`. Change takes effect immediately. Change is effective until next boot.

  Diagnosis: N/A

  Tuning: Only change for extreme situations, such as maximum-load benchmarking.

  Refer to: `vmtune` command.

- **maxpout**

  Purpose: Specifies the maximum number of pending I/Os to a file.

  Values: Default: 0 (no checking), Range: 0 to n (n should be a multiple of 4, plus 1)

  Display: `lsattr -E -l sys0 -a maxpout`

  Change: `chdev -l sys0 -a maxpout=NewValue`. Change is effective immediately and is permanent. If the `-T` flag is used, the change is immediate and lasts until the next boot. If the `-P` flag is used, the change is deferred until the next boot and is permanent.

  Diagnosis: If foreground response time sometimes deteriorates when programs with large amounts of sequential disk output are running, sequential output may need to be paced.

  Tuning: Set `maxpout` to 33 and `minpout` to 16. If sequential performance deteriorates unacceptably, increase one or both. If foreground performance is still unacceptable, decrease both.

  Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″Use of Disk-I/O Pacing″.

- **maxrandwrt (AIX Version 4.1.3 and above)**

  Purpose: The number of dirty file pages to accumulate in RAM before these pages are sync'd to disk via a write-behind algorithm. The random write-behind threshold is on a per file basis.

  Values: Default: 0, Range: 0 to 128 (4 KB pages)

  Display: `vmtune`

  Change: `vmtune -W NewValue`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding vmtune command to /etc/inittab.

  Diagnosis: `vmstat n` shows page out and I/O wait spikes on regular intervals (usually when the `sync` daemon is writing pages to disk).

  Tuning: If `vmstat n` shows page out and I/O wait spikes on regular intervals (usually when the `sync` daemon is writing pages to disk), adjusting the `maxrandwrt` value helps spread the I/O more efficiently. A value of 0 disables random write-behind.

Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″Performance Overview of AIX Management of Fixed-Disk Storage″ and the `vmtune` command

- **maxttl**

  Purpose: Time to live for Routing Information Protocol (RIP) packets.

  Values: Default: 255, Range: N/A

  Display: `no -a` or `no -o maxttl`

  Change: `no -o maxttl=NewValue`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `no` command to /etc/rc.net.

  Diagnosis: N/A

  Tuning: N/A

  Refer to: N/A

- **mb_cl_hiwat (3.2.5 only)**

  Purpose: Specifies the high-water mark for the mbuf cluster pool

  Values: Default: configuration-dependent, Range: N/A

  Display: `no -a` or `no -o mb_cl_hiwat`

  Change: `no -o mb_cl_hiwat=NewValue`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `no` command to /etc/rc.net.

  Diagnosis: `netstat -m`

  Tuning: If the number of mbuf clusters (called ″mapped pages″ by `netstat`) is regularly greater than `mb_cl_hiwat`, increase `mb_cl_hiwat`.

  Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″AIX Version 3.2.5 mbuf Pool Performance Tuning″.

- **medthresh**

  Purpose: Specifies the maximum number of bytes (in percentage) that can be allocated by the `thewall` parameter using `allocb()` for the BPRI_MED priority. When the total amount of memory allocated by the `net_malloc()` subroutine reaches this threshold, the `allocb()` request for the BPRI_MED priority returns 0. The `medthresh` parameter can be set to any value between 0 and 100, inclusively. The default value is 95, indicating the threshold is at 90percent of the value of the `thewall` parameter.

  Values: Default: 95, Range 0 to 100

  Display: `no -a` or `no -o medthresh`

  Change: `no -o medthresh=NewValue`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `no` command to /etc/rc.net.

  Diagnosis: N/A

  Tuning: N/A

  Refer to: N/A

- **Memory-Load-Control Parameters**

Purpose: Customize the VMM memory-load-control facility to maximize use of the system while avoiding thrashing. The most frequently used parameters are:

h - High memory-overcommitment threshold

p - Process memory-overcommitment threshold

m - Minimum level of multiprogramming

Values: h Default: 6, Range: 0 to any positive integer

p Default: 4, Range: 0 to any positive integer

m Default: 2, Range: 0 to any positive integer

Display: `schedtune`

Change: `schedtune [-h NewValue] [-p NewValue] [-m NewValue]`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `schedtune` command to /etc/inittab.

Diagnosis: Heavy memory loads cause wide variations in response time.

Tuning: `schedtune -h 0` turns off memory load control.

`schedtune -p 2` requires a higher level of repaging by a given process before it is a candidate for suspension by memory load control.

`schedtune -m 10` requires that memory load control always leave at least 10 user processes running when it is suspending processes.

Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″VMM Memory Load Control Facility″ and ″Tuning VMM Memory Load Control″.

- **minfree**

  Purpose: The VMM page-frame free-list size at which the VMM starts to steal pages to replenish the free list.

  Values: Default: configuration-dependent, Range: x to any positive integer

  Display: `vmtune`

  Change: `vmtune -f NewValue`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `vmtune` command to /etc/inittab.

  Diagnosis: `vmstat n`

  Tuning: If processes are being delayed by page stealing, increase `minfree` to improve response time. Increase `maxfree` by an equal or greater amount.

  Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″Tuning VMM Page Replacement″.

- **minperm**

  Purpose: The percentage of page frames occupied by permanent pages below which the VMM steals frames from both permanent and working pages without regard to repage rates.

  Values: Default: 20 percent of (memory size - 4 MB), Range: 5 to 100

Display: `vmtune`

Change: `vmtune -P NewValue`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `vmtune` command to /etc/inittab.

Diagnosis: Monitor disk I/O with `iostat n`.

Tuning: If some files are known to be read repetitively, and I/O rates do not decrease with time from startup, `minperm` may be too low.

Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″Tuning VMM Page Replacement″.

- **minpgahead**

  Purpose: The number of pages the VMM reads ahead when it first detects sequential access.

  Values: Default: 2, Range: 0 to 16

  Display: `vmtune`

  Change: `vmtune -r NewValue`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `vmtune` command to /etc/inittab.

  Diagnosis: Observe the elapsed execution time of critical sequential-I/O-dependent applications with `time` command.

  Tuning: If execution time decreases with higher `minpgahead`, observe other applications to ensure that their performance has not deteriorated.

  Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″Tuning Sequential Read Ahead″.

- **minpout**

  Purpose: Specifies the point at which programs that have hit `maxpout` can resume writing to the file.

  Values: Default: 0 (no checking), Range: 0 to n (n should be a multiple of 4 and should be at least 4 less than `maxpout`)

  Display: `lsattr -E -l sys0 -a minpout`

  Change: `chdev -l sys0 -a minpout=NewValue`. Change is effective immediately and is permanent. If the `-T` flag is used, the change is immediate and lasts until the next boot. If the `-P` flag is used, the change is deferred until the next boot and is permanent.

  Diagnosis: If foreground response time sometimes deteriorates when programs with large amounts of sequential disk output are running, sequential output may need to be paced.

  Tuning: Set `maxpout` to 33 and `minpout` to 16. If sequential performance deteriorates unacceptably, increase one or both. If foreground performance is still unacceptable, decrease both.

  Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″Use of Disk-I/O Pacing″.

- **MTU**

  Purpose: Limits the size of packets that are transmitted on the network.

Values: trn (4 Mb): Default: 1492, Range: 60 to 3900

trn (16 Mb): Default: 1492, Range: 60 to 17960

enn: Default: 1500, Range: 60 to 1500

fin: Default: 4352, Range: 60 to 4352

hin: Default: 65536, Range: 60 to 65536

son: Default: 61428, Range: 60 to 61428

lon: Default: 1500 (3.2.5) 16896 (4.1), Range: 60 to 65536

Display: `lsattr -E -l trn`

Change: `chdev -l trn -a mtu=NewValue`. Cannot be changed while the interface is in use. Because all systems on a LAN must have the same MTU, they must all change simultaneously. Change is effective across boots.

Diagnosis: Packet fragmentation stats

Tuning: Increase MTU size for the token-ring interfaces:

trn (4 Mb): 4056

trn (16 Mb): 8500

For the loopback interface lon in Version 3.2.5, increase to 16896. For other interfaces, the default should be kept.

Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″LAN Adapters and Device Drivers″.

- **nfs_chars (3.2.5), nfs_socketsize (AIX Version 4)**

  Purpose: The size of the NFS UDP socket buffer.

  Values: Default: 60000, Range: 60000 to (`sb_max` -128)

  Display: `nfso -a` or `nfso -o nfs_chars` (In 4.1, `nfso -o nfs_socketsize`)

  Change: `nfso -o nfs_chars=NewValue`

  (In 4.1, `nfso -o nfs_socketsize=NewValue`)

  `stopsrc -g nfs`

  `startsrc -g nfs`

  Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `nfso` command to /etc/rc.nfs or /etc/rc.net. The sb_max parameter must change first.

  Diagnosis: `netstat -s`

  Tuning: If `UDP: socket buffer overflows` count is nonzero, increase `sb_max` and `nfs_chars`.

  Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″NFS Tuning″.

- **nfsd Count**

  Purpose: Number of `nfsd` processes available to handle NFS requests on a server.

  Values: Default: 8, Range: 1 to n

Display: `ps -ef | grep nfsd`

Change: `chnfs -n NewValue`. Change normally takes effect immediately and is permanent. The `-N` flag causes an immediate, temporary change. The `-I` flag causes a change that takes effect at the next boot.

Diagnosis: `netstat -s` to look for UDP socket buffer overflows.

Tuning: Increase number until socket buffer overflows cease.

Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 "How Many biods and nfsds Are Needed for Good Performance?".

- **nfs_gather_threshold (AIX Version 4 only)**

  Purpose: Minimum size of a write that sleeps before a sync. Used to disable scatter/gather of writes to the same vnode.

  Values: Default: 4096, Range: x to y

  Display: `nfso -a` or `nfso -o nfs_gather_threshold`

  Change: `nfso -o nfs_gather_threshold=NewValue`. Change takes effect immediately. Change is effective until next boot.

  Diagnosis: N/A

  Tuning: N/A

  Refer to: N/A

- **nfs_portmon (AIX Version 3.2.5), portcheck (AIX Version 4)**

  Purpose: Specifies that NFS is to check whether or not requests come from privileged ports.

  Values: Default: 0 (no), Range: 0 to 1

  Display: `nfso -a` or `nfso -o nfs_portmon`

  Change: `nfso -o nfs_portmon=NewValue`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `nfso` command to /etc/rc.nfs.

  Diagnosis: N/A

  Tuning: This is a configuration decision with minimal performance consequences.

  Refer to: N/A

- **nfs_repeat_messages (AIX Version 4 only)**

  Purpose: Should messages written by NFS be repeated?

  Values: Default: 1 (yes), Range: 0 to 1

  Display: `nfso -a` or `nfso -o nfs_repeat_messages`

  Change: `nfso -o nfs_repeat_messages=NewValue`. Change takes effect immediately. Change is effective until next boot.

  Diagnosis: N/A

  Tuning: N/A

  Refer to: N/A

- **nfs_setattr_error (AIX Version 4 only)**

Purpose: Specifies that NFS is to ignore NFS errors due to illegal PC setattrs.

Values: Default: 1, Range: 0 to 1

Display: `nfso -a`

Change: `nfso -o nfs_setattr_error=NewValue`. Change takes effect immediately. Change is effective until next boot.

Diagnosis: N/A

Tuning: N/A

- **nfsudpcksum (AIX Version 3.2.5), udpchecksum (AIX Version 4)**

  Purpose: Specifies that NFS is to use UDP checksum processing.

  Values: Default: 1 (yes), Range: 0 to 1

  Display: `nfso -a` or `nfso -o nfsudpcksum`

  Change: `nfso -o nfsudpcksum=NewValue`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `nfso` command to /etc/rc.nfs.

  Diagnosis: N/A

  Tuning: Turning checksum processing off may save some processing time, but increases the risk of undetected data errors.

  Refer to: N/A

- **nonlocsrcroute**

  Purpose: Indicates that strict-source-routed IP packets can be addressed to hosts outside the local ring. (Loose source routing is not affected.)

  Values: Default: 1 (yes), Range: 0 to 1

  Display: `no -a` or `no -o nonlocsrcroute`

  Change: `no -o nonlocsrcroute=NewValue`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `no` command to /etc/rc.net.

  Diagnosis: N/A

  Tuning: This is a configuration decision with minimal performance consequences.

  Refer to: N/A

- **npskill (AIX Version 4 only)**

  Purpose: The number of free paging-space pages at which processes begin to be killed.

  Values: Default: 128, Range: 0 to the number of pages in real memory.

  Display: `vmtune`

  Change: `vmtune -k NewValue`. Change takes effect immediately. Change is effective until next boot.

  Diagnosis: N/A

  Tuning: N/A

  Refer to: `vmtune` command.

- **npswarn (AIX Version 4 only)**

  Purpose: The number of free paging-space pages at which processes begin to receive SIGDANGER.

  Values: Default: 512, Range: At least npskill to the number of pages in real memory.

  Display: vmtune

  Change: vmtune -w NewValue. Change takes effect immediately. Change is effective until next boot.

  Diagnosis: N/A

  Tuning: Increase if you experience processes being killed for low paging space.

  Refer to: vmtune command.

- **nstrpush**

  Purpose: Indicates the maximum number of modules that can be pushed onto a single STREAM. This is a loadtime attribute.

  Values: Default: 8.

  Display: Defined in file: /etc/pse_tune.conf

  Change: Change in file: /etc/pse_tune.conf. Change takes effect at next reboot during initial STREAMS load time, when the strload command reads the parameter names and values from the /etc/pse_tune.conf file.

  Diagnosis: N/A

  Tuning: Should be at least 8.

  Refer to: N/A

- **numclust (AIX Version 4 only)**

  Purpose: The number of 16 KB clusters processed by write behind.

  Values: Default: 1, Range: 1 to any positive integer

  Display: vmtune

  Change: vmtune -c NewValue. Change takes effect immediately. Change is effective until next boot.

  Diagnosis: N/A

  Tuning: May be appropriate to increase if striped logical volumes or disk arrays are being used.

  Refer to: vmtune command.

- **numfsbuf (AIX Version 4 only)**

  Purpose: The number of file-system bufstructs.

  Values: Default: 64, Range: 64 to any positive integer

  Display: vmtune

  Change: vmtune -b NewValue. Change takes effect immediately. Change is effective until next boot.

  Diagnosis: N/A

Tuning: May be appropriate to increase if striped logical volumes or disk arrays are being used.

Refer to: `vmtune` command.

- **Paging Space Size**

  Purpose: The amount of disk space required to hold pages of working storage.

  Values: Default: configuration-dependent, Range: 32 MB to n MB for hd6, 16 MB to n MB for non-hdg.

  Display: `lsps -a`

  Change: `mkps` or `chps` or `smit pgsp`. Change takes effect immediately and is permanent. Paging space is not necessarily put into use immediately, however.

  Diagnosis: `lsps -a` If processes have been killed for lack of paging space, monitor the situation with the `psdanger()` subroutine.

  Tuning: If it appears that there is not enough paging space to handle the normal workload, add a new paging space on another physical volume, or make the existing paging spaces larger.

  Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″Placement and Sizes of Paging Spaces″.

- **pd_npages**

  Purpose: The number of pages that should be deleted in one chunk from RAM when a file is deleted.

  Values: Default: largest file size / page size, Range: 1 to largest file size / page size

  Display: `vmtune`

  Change: `vmtune -N NewValue.` Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `vmtune` command to /etc/inittab.

  Diagnosis: This option may be useful if a real-time application is experiencing some slow response time while large files are being deleted.

  Tuning: If real-time response is critical, adjusting this option may improve response time by spreading the removal of file pages from RAM more evenly over a workload.

  Refer to: `vmtune` command

- **Process-Priority Calculation**

  Purpose: Specify the amount by which a process's priority value will be increased by its recent CPU usage and the rate at which the recent-CPU-usage value decays. The parameters are called `r` and `d`.

  Values: Default: 16, Range: 0 to 32 (Note: When applied to the calculation, the values of `r` and `d` are divided by 32. Thus the effective range of factors is from 0 to 1 in increments of 0.03125.)

  Display: `schedtune`

Change: `schedtune -r` or `schedtune -d`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `schedtune` command to /etc/inittab.

Diagnosis: `ps al` If you find that the PRI column has priority values for foreground processes (those with NI values of 20) that are higher than the PRI values of some background processes (NI values > 20), you may want to reduce the `r` value.

Tuning: Decreasing `r` makes it easier for foreground processes to compete. Decreasing `d` enables foreground processes to avoid competition with background processes for a longer time. `schedtune -r 2` would ensure that any new foreground process would receive at least 0.5 seconds of CPU time before it had to compete with any process with NI >= 24.

Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″Tuning the Process-Priority-Value Calculation with `schedtune`″.

- **psebufcalls**

  Purpose: Specifies the maximum number of bufcalls to allocate by STREAMS. In AIX, the STREAM subsystem allocates a certain number of bufcall structures at initialization time. When `allocb( )` fails, the user can register requests for the `bufcall( )`. Lowering this value is not allowed until the system reboots. At reboot, it returns to its default value.

  Values: Default: 0

  Display: `no -a` or `no -o psebufcalls`

  Change: `no -o psebufcalls=NewValue`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `no` command to /etc/rc.net.

  Diagnosis: N/A

  Tuning: N/A

  Refer to: N/A

- **pseintrstack**

  Purpose: Indicates the maximum number of the interrupt stack size allowed by STREAMS while running in the offlevel. Sometimes, when a process running other than INTBASE level enters a STREAM, it encounters stack overflow problems because of not enough interrupt stack size. Tuning this parameter properly reduces the chances of stack overflow problems.

  Values: Default: 0x3000 (decimal 12288).

  Display: Defined in file: /etc/pse_tune.conf

  Change: Change in file: /etc/pse_tune.conf. Change takes effect at next reboot during initial STREAMS load time, when the `strload` command reads the parameter names and values from the /etc/pse_tune.conf file.

  Diagnosis: Stack overflow problems.

  Tuning: N/A

  Refer to: N/A

- **psetimers**

  Purpose: Specifies the maximum number of timers to allocate by STREAMS. In AIX, the STREAM subsystem allocates a certain number of timer structures at initialization time so the STREAMS driver or module can register the `timeout( )` requests. Lowering this value is not allowed until the system reboots. At reboot, it returns to its default value.

  Values: Default: 20

  Display: `no -a` or `no -o psetimers`

  Change: `no -o psetimers=NewValue`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `no` command to /etc/rc.net.

  Diagnosis: N/A

  Tuning: N/A

  Refer to: N/A

- **rec_que_size**

  Purpose: (Tunable only in AIX Version 3.) Specifies the maximum number of receive buffers that can be queued-up for the interface.

  Values: Default: 30, Range: 20 to 150

  Display: `lsattr -E -l tokn -a rec_que_size`

  Change: `ifconfig tr0 detach`

  `chdev -I tokn -a rec_que_size=NewValue`

  `ifconfig tr0 hostname up`

  Change is effective across boots.

  Diagnosis: N/A

  Tuning: Increase size. Should be set to 150 as a matter of course on network-oriented systems, especially servers.

  Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″LAN Adapters and Device Drivers″.

- **rfc1122addrchk**

  Purpose: Specifies whether address validation is performed between communications layers.

  Values: Default: 0 (no), Range: 0 to 1

  Display: `no -a` or `no -o rfc1122addrchk`

  Change: `no -o rfc1122addrchk=NewValue`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `no` command to /etc/rc.net.

  Diagnosis: N/A

  Tuning: This value should not be changed.

  Refer to: N/A

- **rfc1323**

Purpose: Value of 1 indicates that tcp_sendspace and tcp_recvspace can exceed 64 KB.

Values: Default: 0, Range: 0 or 1

Display: no -a or no -o rfc1323

Change: no -o rfc1323=NewValue. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding no command to /etc/rc.net.

Diagnosis: None.

Tuning: Change before attempting to set tcp_sendspace and tcp_recvspace to more than 64 KB.

Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″TCP Layer″.

- **sb_max**

  Purpose: Provide an absolute upper bound on the size of TCP and UDP socket buffers. Limits setsockopt(), udp_sendspace, udp_recvspace, tcp_sendspace, and tcp_recvspace.

  Values: Default: 65536, Range: N/A

  Display: no -a or no -o sb_max

  Change: no -o sb_max=NewValue. Change takes effect immediately for new connections. Change is effective until next boot. Permanent change is made by adding no command to /etc/rc.net.

  Diagnosis: None.

  Tuning: Increase size, preferably to multiple of 4096. Should be about twice the largest socket buffer limit.

  Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″Socket Layer″

- **strctlsz**

  Purpose: Specifies the maximum number of bytes of information that a single system call can pass to a STREAM to be placed into the control part of a message (in M_PROTO or M_PCPROTO block). Any putmsg( ) with a control part exceeding this size will fail returning an ERANGE error code.

  Values: Default: 1024

  Display: no -a or no -o strctlsz

  Change: no -o strctlsz=NewValue. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding no command to /etc/rc.net.

  Diagnosis: N/A

  Tuning: N/A

  Refer to: N/A

- **strmsgsz**

  Purpose: Specifies the maximum number of bytes of information that a single system call can pass to a STREAM to be placed into the data part of a message (in M_DATA blocks). Any write( )

exceeding this size will be broken into multiple messages. A putmsg( ) with a data part exceeding this size will fail returning an ERANGE error code.

Values: Default: 0

Display: no -a or no -o strmsgsz

Change: no -o strmsgsz=NewValue. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding no command to /etc/rc.net.

Diagnosis: N/A

Tuning: N/A

Refer to: N/A

- **strturncnt**

  Purpose: Specifies the maximum number of requests handled by the currently running thread for Module- or Elsewhere-level STREAMS synchronization. In AIX Version 4, the Module-level synchronization works in such a way that only one thread can run in the module at any given time, and all other threads trying to acquire the same module enqueue their requests and exit. After the currently running thread completes its work, it dequeues all the previously enqueued requests one at a time and starts them. If there are large numbers of requests enqueued in the list, the currently running thread must serve everyone and will always be busy serving others thus starvimg itself. To eliminate this problem, the currently running thread serves only the strturncnt number of threads. After that, a separate kernel thread starts all the pending requests.

  Values: Default: 15

  Display: no -a or no -o strturncnt

  Change: no -o strturncnt=NewValue. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding no command to /etc/rc.net.

  Diagnosis: N/A

  Tuning: N/A

  Refer to: N/A

- **strthresh**

  Purpose: Specifies the maximum number of bytes STREAMS are normally allowed to allocate. When the threshold is passed, users without the appropriate privilege will not be allowed to open STREAMS, push modules, or write to STREAMS devices. The ENOSR error code is returned. The threshold applies only to the output side; therefore, data coming into the system is not affected and continues to work properly. A value of 0 indicates there is no threshold.

  The strthresh parameter represents a percentage of the value of the thewall parameter, and its value can be set between 0 and 100, inclusively. The thewall parameter indicates the maximum number of bytes that can be allocated by STREAMS and Sockets

using the net_malloc( ) subroutine. The user can change the value of the thewall parameter using the no command. When the user changes the value of the thewall parameter, the threshold gets updated accordingly. The default value is 85, indicating the threshold is 85 percent of the value of the thewall parameter.

Values: Default: 85, Range: 0 to 100

Display: no -a or no -o strthresh

Change: no -o strthresh=NewValue. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding no command to /etc/rc.net.

Diagnosis: N/A

Tuning: N/A

Refer to: N/A

- **subnetsarelocal**

  Purpose: Specifies that all subnets that match the subnet mask are to be considered local for purposes of establishing, for example, the TCP maximum segment size.

  Values: Default: 1 (yes), Range: 0 to 1

  Display: no -a or no -o subnetsarelocal

  Change: no -o subnetsarelocal=NewValue. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding no command to /etc/rc.net.

  Diagnosis: N/A

  Tuning: This is a configuration decision with performance consequences. If the subnets do not all have the same MTU, fragmentation at bridges may degrade performance. If the subnets do have the same MTU, and subnetsarelocal is 0, TCP sessions may use an unnecessarily small MSS.

  Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 "Tuning TCP Maximum Segment Size (MSS)".

- **syncd Interval**

  Purpose: The time between sync() calls by syncd.

  Values: Default: 60 (seconds), Range: 1 to any positive integer

  Display: grep syncd /sbin/rc.boot

  Change: vi /sbin/rc.boot. Change takes effect at next boot and is permanent.

  Diagnosis: N/A

  Tuning: At its default level, this parameter has little performance cost. No change is recommended. Significant reductions in the syncd interval in the interests of data integrity could have adverse consequences.

  Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 "Performance Implications of sync/fsync".

- **tcp_keepidle**

Purpose: Total length of time to keep an idle TCP connection alive.

Values: Default: 14400 (half-seconds) = 2 hours, Range: any positive integer

Display: `no -a` or `no -o tcp_keepidle`

Change: `no -o tcp_keepidle=NewValue`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `no` command to /etc/rc.net.

Diagnosis: N/A

Tuning: This is a configuration decision with minimal performance consequences. No change is recommended.

Refer to: N/A

- **tcp_keepintvl**

  Purpose: Interval between packets sent to validate the TCP connection.

  Values: Default: 150 (half-seconds) = 75 seconds, Range: any positive integer

  Display: `no -a` or `no -o tcp_keepintvl`

  Change: `no -o tcp_keepintvl=NewValue`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `no` command to /etc/rc.net.

  Diagnosis: N/A

  Tuning: This is a configuration decision with minimal performance consequences. No change is recommended. If the interval were shortened significantly, processing and bandwidth costs might become significant.

  Refer to: N/A

- **tcp_mssdflt**

  Purpose: Default maximum segment size used in communicating with remote networks.

  Values: Default: 512, Range: 512 to (MTU of local net - 64)

  Display: `no -a` or `no -o tcp_mssdflt`

  Change: `no -o tcp_mssdflt=NewValue`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `no` command to /etc/rc.net.

  Diagnosis: N/A

  Tuning: Increase, if practical.

  Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″Tuning TCP Maximum Segment Size (MSS)″.

- **tcp_recvspace**

  Purpose: Provide the default value of the size of the TCP socket receive buffer.

  Values: Default: 16384, Range: 0 to 64 KB if `rfc1323=0`, Range: 0 to 4 GB if `rfc1323=1`. Must be less than or equal to `sb_max`. Should be equal to `tcp_sendspace` and uniform on all frequently accessed AIX systems.

Display: `no -a` or `no -o tcp_recvspace`

Change: `no -o tcp_recvspace=NewValue`. Change takes effect immediately for new connections. Change is effective until next boot. Permanent change is made by adding `no` command to /etc/rc.net.

Diagnosis: Poor throughput.

Tuning: Increase size, preferably to multiple of 4096.

Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″Socket Layer″.

- **tcp_sendspace**

  Purpose: Provide the default value of the size of the TCP socket send buffer.

  Values: Default: 16384, Range: 0 to 64 KB if `rfc1323=0`, Range: 0 to 4 GB if `rfc1323=1`. Must be less than or equal to `sb_max`. Should be equal to `tcp_recvspace` and uniform on all frequently accessed AIX systems.

  Display: `no -a` or `no -o tcp_sendspace`

  Change: `no -o tcp_sendspace=NewValue`. Change takes effect immediately for new connections. Change is effective until next boot. Permanent change is made by adding `no` command to /etc/rc.net.

  Diagnosis: Poor throughput.

  Tuning: Increase size, preferably to multiple of 4096.

  Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″Socket Layer″.

- **tcp_ttl**

  Purpose: Time to live for TCP packets.

  Values: Default: 60 (10-millisecond processor ticks), Range: any positive integer

  Display: `no -a` or `no -o tcp_ttl`

  Change: `no -o tcp_ttl=NewValue`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `no` command to /etc/rc.net.

  Diagnosis: `netstat -s`

  Tuning: If the system is experiencing TCP timeouts, increasing `tcp_ttl` may reduce retransmissions.

  Refer to: N/A

- **thewall**

  Purpose: Provide an absolute upper bound on the amount of real memory that can be used by the communications subsystem.

  Values: Default: 25 percent of real memory, Range: 0 to 50 percent of real memory

  Display: `no -a` or `no -o thewall`

Change: `no -o thewall=NewValue`. `NewValue` is in KB, not bytes. Change takes effect immediately for new connections. Change is effective until next boot. Permanent change is made by adding `no` command to /etc/rc.net.

Diagnosis: None.

Tuning: Increase size, preferably to multiple of 4 (KB).

Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″AIX Version 3.2.5 mbuf Pool Performance Tuning″

- **Time-Slice Expansion Amount**

  Purpose: The number of 10-millisecond clock ticks by which the default 10 millisecond time slice is to be increased.

  Values: Default: 0, Range: 0 to any positive integer

  Display: `schedtune`

  Change: `schedtune -t NewValue`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `schedtune` command to /etc/inittab.

  Diagnosis: N/A

  Tuning: In general, this parameter should not be changed. If the workload consists almost entirely of very long-running, CPU-intensive programs, increasing this parameter may have some positive effect.

  Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″Modifying the Scheduler Time Slice″.

- **udp_recvspace**

  Purpose: Provide the default value of the size of the UDP socket receive buffer.

  Values: Default: 41600, Range: N/AMust be less than or equal to `sb_max`.

  Display: `no -a` or `no -o udp_recvspace`

  Change: `no -o udp_recvspace=NewValue`. Change takes effect immediately for new connections. Change is effective until next boot. Permanent change is made by adding `no` command to /etc/rc.net.

  Diagnosis: Nonzero `n` in `netstat -s` report of `udp: n socket buffer overflows`

  Tuning: Increase size, preferably to multiple of 4096.

  Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″Socket Layer″

- **udp_sendspace**

  Purpose: Provide the default value for the size of the UDP socket send buffer.

  Values: Default: 9216, Range: 0 to 65536. Must be less than or equal to `sb_max`.

  Display: `no -a` or `no -o udp_sendspace`

  Change: `no -o udp_sendspace=NewValue`. Change takes effect immediately for new connections. Change is effective until next boot. Permanent change is made by adding `no` command to /etc/rc.net.

Tuning: Increase size, preferably to multiple of 4096.

Refer to: *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365 ″Socket Layer″

- **udp_ttl**

  Purpose: Time to live for UDP packets.

  Values: Default: 30 (10-millisecond timer ticks), Range: any positive integer

  Display: `no -a` or `no -o udp_ttl`

  Change: `no -o udp_ttl=NewValue`. Change takes effect immediately. Change is effective until next boot. Permanent change is made by adding `no` command to /etc/rc.net.

  Diagnosis: N/A

  Tuning: N/A

  Refer to: N/A

- **xmt_que_size**

  Purpose: Specifies the maximum number of send buffers that can be queued up for the device.

  Values: Default: 30, Range: 20 to 150

  Display: `lsattr -E -l tok0 -a xmt_que_size`

  Change: `ifconfig tr0 detach`

  `chdev -I tok0 -a xmt_que_size=NewValue`

  `ifconfig tr0 hostname up`

  Change is effective across boots.

  Diagnosis: `netstat -i` Oerr > 0

  Tuning: Increase size. Should be set to 150 as a matter of course on network-oriented systems, especially servers.

  Refer to: “LAN Adapters and Device Drivers”

Appendix C. Performance Tools Paths and Filesets

Path and command name	Fileset
/usr/bin/3dmon	perfmgr
/usr/bin/bf	perfagent.tools
/usr/bin/bfrpt	perfagent.tools
/usr/sbin/bindprocessor	bos.mp (V 4.2), bos.rte.mp (V 4.1)
/usr/sbin/chdev	bos.rte.methods
/usr/bin/chmon	perfmgr
/usr/sbin/cpu_state	bos.mp (V 4.2), bos.rte.mp (V 4.1)
/usr/sbin/defragfs	bos.rte.filesystem
/usr/bin/fdpr	perfagent.tools
/usr/bin/filemon	perfagent.tools
/usr/bin/fileplace	perfagent.tools
/usr/bin/genkex	perfagent.tools
/usr/bin/genkld	perfagent.tools
/usr/bin/genld	perfagent.tools
/usr/ccs/bin/gprof	bos.adt.prof
/usr/bin/iostat	bos.acct
/usr/bin/lockstat	perfagent.tools
/usr/sbin/lslv	bos.rte.lvm
/usr/sbin/migratepv	bos.rte.lvm
/usr/bin/netpmon	perfagent.tools
/usr/bin/nice	bos.rte.control
/usr/sbin/netstat	bos.net.tcp.client
/usr/sbin/nfsstat	bos.net.nfs.client
/usr/sbin/nfso	bos.net.nfs.client
/usr/sbin/no	bos.net.tcp.client
/usr/sbin/perf/diag_tool/pdt_config	bos.perf.diag_tool
/usr/sbin/perf/diag_tool/pdt_report	bos.perf.diag_tool
/usr/sbin/perf/pmr/perfpmr	bos.perf.pmr
/usr/ccs/bin/prof	bos.adt.prof
/usr/bin/ps	bos.rte.control
/usr/sbin/pstat	bos.sysmgt.serv_aid
/usr/lpp/pv/bin/pv	aixtools (*, **)
/usr/sbin/renice	bos.rte.control
/usr/sbin/reorgvg	bos.rte.lvm

/usr/bin/rmss	perfagent.tools
/usr/sbin/sar	bos.acct
/usr/samples/kernel/schedtune	bos.adt.samples
/usr/bin/stem	perfagent.tools
/usr/bin/stripnm	perfagent.tools
/usr/bin/svmon	perfagent.tools
/usr/bin/syscalls	perfagent.tools
/usr/bin/time	bos.rte.misc_cmds
/usr/bin/timex	bos.acct
/usr/bin/tprof	perfagent.tools
/usr/bin/vmstat	bos.acct
/usr/samples/kernel/vmtune	bos.adt.samples
xgprof	aixtools (*)
/usr/bin/xmperf	perfmgr
/usr/bin/xmpeek	perfagent.server

* also available on the *IBM Developer Connection*, CD-ROM Z121-0200
Info at: http://www.developer.ibm.com/devcon/index.html

** also available on the *IBM Software Development Solutions for AIX Version 4*, CD-ROM SK2T-2729

Appendix D. Special Notices

This publication is intended to help RS/6000 users, system administrators, and system engineers to understand the available performance monitoring and system-tuning tools on RS/6000, to use them, and to undertake a detailed performance analysis. The information in this publication is not intended as the specification of any programming interfaces that are provided by IBM RS/6000, AIX, and Performance Toolbox for AIX. See the PUBLICATIONS section of the IBM Programming Announcement for IBM RS/6000, AIX, and Performance Toolbox for AIX for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM (″vendor″) products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX	AIXwindows
AS/400	BookManager
HACMP/6000	IBM
IMS	InfoExplorer
LoadLeveler	NetView
POWERparallel	RISC System/6000
RS/6000	SP
System/390	Xstation Manager
400	

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

Java and HotJava are trademarks of Sun Microsystems, Incorporated.

Microsoft, Windows, Windows NT, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

Pentium, MMX, ProShare, LANDesk, and ActionMedia are trademarks or registered trademarks of Intel Corporation in the U.S. and other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names may be trademarks or service marks of others.

486	Intel Corporation
C + +	American Telephone & Telegraph Company, Incorporated
DDS	Sony Corporation
Hewlett-Packard	Hewlett-Packard Company
HP	Hewlett-Packard Company
Motif	Open Software Foundation, Incorporated
Network File System	Sun Microsystems, Incorporated
NFS	Sun Microsystems, Incorporated
OSF/Motif	Open Software Foundation, Incorporated
POSIX	Institute of Electrical and Electronic Engineers
PostScript	Adobe Systems, Incorporated
SCSI	Security Control Systems, Incorporated
Solaris	Sun Microsystems, Incorporated
Sun	Sun Microsystems, Incorporated
SunOS	Sun Microsystems, Incorporated
X Window System	Massachusetts Institute of Technology
Zip	Iomega Corporation

Appendix E. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

E.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How to Get ITSO Redbooks” on page 299.

- *Understanding IBM RS/6000 Performance and Sizing*, SG24-4810
- *RS/6000 SMP Enterprise Servers Architecture and Implementation*, SG24-2583

E.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** and receive updates 2-4 times a year at significant savings.

CD-ROM Title	Subscription Number	Collection Kit Number
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RISC System/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RISC System/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
Application Development Redbooks Collection	SBOF-7290	SK2T-8037
Personal Systems Redbooks Collection	SBOF-7250	SK2T-8042

E.3 Other Publications

These publications are also relevant as further information sources:

- *AIX Performance Tuning Guide, Versions 3.2 and 4*, SC23-2365
- *Performance Toolbox for AIX Guide and Reference, Version 1.2 and 2*, SC23-2625
- *AIX Version 4 Optimization and Tuning Guide for Fortran, C, and C++,* SC09-1705
- *AIX Version 4 System Management Guide: Communications and Networks*, SC23-2526
- *AIX Version 4 Commands Reference*, SBOF-1851

How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at URL http://www.redbooks.ibm.com.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **PUBORDER** — to order hardcopies in United States
- **GOPHER link to the Internet** - type GOPHER.WTSCPOK.ITSO.IBM.COM
- **Tools disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get BookManager BOOKs of redbooks, type the following command:

  ```
  TOOLCAT REDBOOKS
  ```

  To get lists of redbooks:

  ```
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
  ```

  To register for information on workshops, residencies, and redbooks:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
  ```

  For a list of product area specialists in the ITSO:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
  ```

- **Redbooks Home Page on the World Wide Web**

  http://w3.itso.ibm.com/redbooks

- **IBM Direct Publications Catalog on the World Wide Web**

  http://www.elink.ibmlink.ibm.com/pbl/pbl

  IBM employees may obtain LIST3820s of redbooks from this page.

- **REDBOOKS category on INEWS**
- **Online** — send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**

  With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an e-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** (Do not send credit card information over the Internet) — send orders to:

	IBMMAIL	**Internet**
In United States:	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada:	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America:	dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** — send orders to:

IBM Publications	IBM Publications	IBM Direct Services
Publications Customer Support	144-4th Avenue, S.W.	Sortemosevej 21
P.O. Box 29570	Calgary, Alberta T2P 3N5	DK-3450 Allerød
Raleigh, NC 27626-0570	Canada	Denmark
USA		

- **Fax** — send orders to:

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1)001-408-256-5422 (Outside USA)** — ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **Direct Services** - send note to softwareshop@vnet.ibm.com
- **On the World Wide Web**

Redbooks Home Page	http://www.redbooks.ibm.com
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Internet Listserver**

  With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an e-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank).

IBM Redbook Order Form

Please send me the following:

Title	Order Number	Quantity

First name Last name

Company

Address

City Postal code Country

Telephone number Telefax number VAT number

- Invoice to customer number
- Credit card number

Credit card expiration date Card issued to Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

DO NOT SEND CREDIT CARD INFORMATION OVER THE INTERNET.

List of Abbreviations

AIX	advanced interactive executive
API	application program interface
ARP	address resolution protocol
ASCII	American National Standard Code for Information Interchange
azizo	analyzing zoom-in zoom-out
bf	bigfoot
CD-ROM	compact disk read only memory
CDLI	common data link interface
COBOL	common business oriented language
CPU	central processing unit
DB	data base
DDS	dynamic data supplier
DNS	domain name service
exmon	exception monitor
FDDI	fiber distributed data interface
fdpr	feedback directed program restructuring
FORTRAN	formula translation
FTP	File Transfer Program
GB	gigabyte
GUI	graphical user interface
HACMP	high availability cluster multi-processing
HP-UX	Hewlett-Packard Co.-UNIX
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
I/O	input/output
IAR	instruction address register
IBM	International Business Machines Corporation
ICMP	internet control message protocol
ID	identification number
IP	Internet protocol
ITSO	International Technical Support Organization
JFS	journaled file system
K	kilo
KB	kilobyte
KB/s	kilobyte per second
LAN	local area network
LP	logical partition
LPP	licensed program product
LV	logical volume
LVCB	logical volume control block
LVM	logical volume manager
LZ	Lempel-Zev
M	million
MB	megabyte
MP	multiprocessor
ms	milliseconds
MTU	maximum transmission unit
N/A	not applicable
N/A	not available
NFS	network file system
NLS	national language support
ODM	object data manager
OEM	original equipment manufacturer
OS	operating system
PDT	performance diagnostic tool
perfPMR	performance program modification request
PFT	page frame table
PID	process identifier
PMTU	path maximum transmission unit
POSIX	portable operating system interface for UNIX
POWER	performance optimization with enhanced RISC
PRPQ	programming request for price quotation
PSSP	Parallel System Support Programs
PTF	program temporary fix
PTPE	Performance Toolbox Parallel Extensions
PTX	Performance Toolbox
PV	Program Visualizer
RAM	random access memory
RFC	request for comments
RISC	reduced instruction set computer
rmss	reduced memory system simulator
RPC	remote procedure call
RS	RISC system
Rsi	Remote Statistics Interface
RSS	resident set size
sar	system accounting report
SCSI	small computer system interface

SDET	software development environment test
sdev	standard deviation
SMIT	System Management Interface Tool
SMP	symmetric multiprocessor
SMUX	single multiplexor
SNMP	simple network management protocol
SP	Scalable POWERparallel
SPARC	scalable processor architecture
SPMI	system performance measurement interface
stem	scanning tunneling encapsulating microscope
SunOS	Sun operating system
SYN	synchronization
TCP	transmission control protocol
TID	thread identifier
TLB	translation lookaside buffer
TRS	text resident size
TSIZ	text size
TTY	Teletypewriter
UDP	user datagram protocol
UP	uniprocessor
URL	Universal Resource Locator
VM	virtual machine
VMM	virtual memory manager
WWW	World Wide Web
X	X Window System
XCOFF	Extended Common Object File Format
XDR	external data representation
XID	exchange identifier

Index

Numerics
3dmon 203, 220
3dplay 203

A
a2ptx 240
Abbreviations 303
Acronyms 303
azizo 203, 204, 241

B
Baseline 4
bf 144
bfrpt 146
Bibliography 297
BigFoot 144
bindprocessor 173
Bottleneck 1

C
chmon 203, 239
cpu_state 170
cron 5, 24

E
exmon 203, 236

F
fdpr 162
filemon 83
fileplace 95
filtd 5, 203, 204, 230

G
genkex 115
genkld 114
genld 113
getnames 263
gprof 59, 251

I
iostat 17, 29

L
Locks 164, 169
lockstat 164
lslv 98

N
netpmon 106
netstat 35
nfso 50
nfsstat 42
nice 54
no 46

P
Page Replacement Algorithm 14, 178, 191
PDT (Performance Diagnostic Tool) 4, 129
Performance
  Monitoring 1
  Tools 3
perfPMR 4, 140
Processor Affinity 33, 174, 185
prof 58
ps 30, 76
pstat 33
PTX (Performance Toolbox) 201
  Agent 201, 204
  Console 204, 209
  Filesets 205
  Instrument 204
  Manager 201, 203
  Statistics 204
  Value 204
ptxmerge 204
ptxrlog 240
ptxtab 204
PV (Program Visualizer) 253, 266
pvpackage 259
pvreport 260
pvtrace 254

R
renice 56
rmss 79

S
sadc 23, 24
sar 4, 23, 29
schedtune 11, 13, 55, 79, 177
Scheduling Policy 34
stem 152
stripnm 116
svmon 71, 89
syscalls 158

T

Thrashing 11, 13, 126, 178
timex 30
tprof 63
trace 63, 120, 262
trcnm 262
trcrpt 124

U

utld 261, 266

V

vmstat 7, 29, 75, 92
vmtune 11, 14, 15, 17, 189

X

xgprof 251, 266
xmperf 202, 203, 204
xmservd 4, 5, 203, 204, 231

ITSO Redbook Evaluation

RS/6000 Performance Tools in Focus
SG24-4989-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at http://www.redbooks.com
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redeval@vnet.ibm.com

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction _____

Please answer the following questions:

Was this redbook published in time for your needs? Yes____ No____

If no, please explain:

_____

_____

_____

_____

What other redbooks would you like to see published?

_____

_____

_____

Comments/Suggestions: **(THANK YOU FOR YOUR FEEDBACK!)**

_____

_____

_____

_____

_____

Printed in U.S.A.