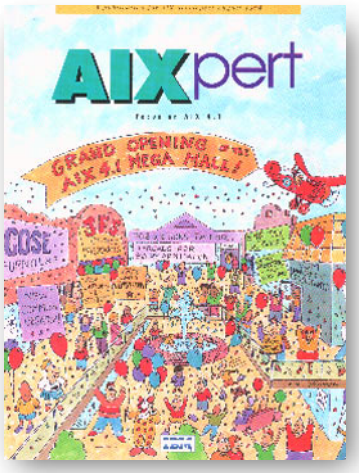
AIXpert
DIGITAL VERSION IBM

TABLE OF CONTENTS



AUGUST
1994

AIXpert BACK TO MAIN MENU

Grand Opening

COMMENTARY

Last month we witnessed the unveiling of AIX® 4.1—in the form of Version 4.1. By now you've probably browsed through the press releases and noted the important new features. You're planning ways to use the new kernel threads support and the expanded filesystem size, or you may be thinking about changing to the new common desktop from Common Open Software Environment (COSE). You also may have noticed the movement to adopt XPG4, giving new meaning to the term, "Grand Opening." This month, we'll clarify issues such as these about AIX 4.1 and provide some insights into its design. We'll also cover other areas of interest for AIX 3.2.5 and AIX/ESA™.

Two articles set the scope for this issue. "Overview of AIX 4.1" summarizes important improvements in the new version of AIX, while the interview with Mike Day presents AIX from the viewpoint of a system architect, which makes very interesting reading.

Providing a more in-depth look at the new operating system, an article on the Journaled File System (JFS) explains the filesystem enhancements, such as data compression, striping, and a larger maximum filesystem size. If you have already been writing programs to the DCE threads available in AIX 3.2.5, check out the article on required changes for making those programs run with the new kernel-based threads. For other portability information, read the XPG4 article to understand current AIX conformance to standards and plans for future conformance. Completing the coverage of AIX 4.1 is an article about the Common Desktop Environment (CDE) and two articles describing the design and features of the Visual System Management (VSM) software.

Interested in tools? The article on fdpr describes a tool that uses program reordering to make AIX programs run faster (up to 73%). Or read the article on performance management tools in a distributed environment to learn about enhancements to an already excellent set of tools in the Performance Toolbox (PTX).

In the hardware corner, you will find performance implications of the new POWER2 Architecture™ machines that were introduced in May, and how those machines compare when running standard performance benchmarks. For those interested in I/O, this issue also contains articles about the AIX terminal server and the new network terminal accelerator adapter.

Rounding out this issue is an article for system administrators that explains the accounting checkpoint concepts in the latest release of AIX/ESA.

George Noren

George Noren, IBM Corporation, Internal Zip 2830, 11400 Burnet Road, Austin, TX 78758. Internet: geo@austin.ibm.com. Since joining IBM in September 1979, Mr. Noren has written manuals for System/34, System/36™, and AIX on both the RT® and RISC System/6000® platforms, and was a member of the InfoExplorer™ design team. He has also worked as system administrator for several AIX server machines and their clients, and is currently responsible for the Prototype Evaluation Labs in Austin. Mr. Noren studied engineering at Illinois Institute of Technology, holds a BA in English from the University of Minnesota and an MBA from St. Edwards University in Austin.

George Noren

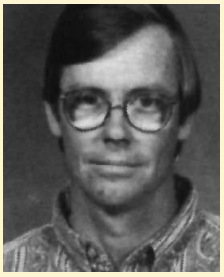
The Architecture Behind AIX 4.1

More than a release, AIX 4.1 is a new version of AIX. It provides additional functions and capabilities in many significant areas—from kernel and application programming interfaces to the "look and feel" that end users will experience—while retaining the functionality and application binary compatibility with AIX Version 3 on which existing customers depend.

We asked Michael Day, a lead AIX system architect in Austin, Texas, to tell us about new features in AIX 4.1 and their implications for developers and customers.

What was your overall objective in designing AIX 4.1?

Day: The overall theme of AIX 4.1 development was to make the AIX experience more appealing to a greater number and more diverse set of customers and markets. We wanted AIX to continue to be the operating system of choice for the existing POWER-based systems and to be a player in the wave of new PowerPC™-based products being introduced by IBM and others. AIX 4.1 is well positioned technically to take advantage of IBM's new Power Personal Systems line of products as well as the upcoming high-end Symmetric Multiprocessor (SMP) systems. While AIX was already well optimized to take advantage of the POWER Architecture™, additional optimization has been added to leverage the family of processors based on the PowerPC Architecture™.



Michael Day

It is quite an undertaking to build one operating system that covers what IBM calls the "laptops to teraflops" range of machines. How have you provided a good operating system for each machine, while avoiding additional overhead costs of supporting a range of platforms?

Day: We started with a good base. AIX Version 3 was already designed with scalability and system flexibility in mind with its use of large virtual address spaces, pageable kernel, dynamically loadable kernel extensions, and self-configuring device and subsystem support. This capability was extended to deal with more diverse system designs required to handle the expanding family of POWER and PowerPC processors, and differing system buses and I/O hardware used by systems being introduced by IBM and others.

What are the key enhancements in AIX 4.1?

Day: I have separated the enhancements and new functions into six categories. From the bottom to the top of the operating system, they are as follows:

- Enhanced kernel functions
- Industry standards
- Application binary compatibility
- Enhanced system management capabilities
- New graphical end-user environment
- New packaging

Let's start with the enhanced kernel functions.

Day: Since AIX 3.2.5, AIX has had platform-specific modules for each processor chip architecture. In Version 3.2.5, we added support for the POWER2 processor and the PowerPC 601™. At boot time, AIX detects the type of system, and then overlays and fills in branch tables for the platform-specific modules required by the system. No memory is wasted by holding unneeded code. In AIX 4.1, we added modules for the

PowerPC 603™ and 604 chips, the upcoming PC-based Power Personal Systems line of products, and the upcoming SMP systems.

Kernel changes were also made to support 2- to 8-way symmetric multiprocessor systems. This is classic SMP support—fully symmetric I/O with shared memory. We used the IEEE POSIX™ 1003.14 MP-specific commands for system administration and management.

At the same time, we implemented threads within the kernel, creating a one-to-one relationship between user threads and kernel threads with the pthread library based on POSIX 1003.4A (Draft7). This should be beneficial to object-oriented, multimedia, real-time, and database programming.

In standard UNIX®, if you had multiple threads of execution, you had to fork and exec processes, using pipes or shared memory to communicate between them. With thread support in the kernel, these applications no longer need that overhead. Developers can create multiple threads of execution within one process, and the threads have access to all the same data and resources. In addition, this change makes it easier for multithreaded applications to exploit multiprocessing. Each thread can automatically be dispatched to the next available processor, since the kernel now handles the execution priorities and dispatching at the thread level.

We have done a lot to parallelize the critical kernel subsystems in anticipation of an SMP machine—including Logical Volume Manager (LVM), Virtual Memory Manager, filesystems, subsystems such as Streams and tty, and many device drivers. Since AIX 4.1 has implemented locking and serialization strategies, many existing applications can run efficiently on an MP system.

We have also replaced our home-grown extensions of the Berkeley Software Distribution (BSD) tty subsystem. AIX 4.1 uses a new tty subsystem based on UNIX System V.4-compatible Streams that makes it easier to port Streams-based applications to AIX. AIX 4.1 still supports the AIX 3.2 tty functionality from an applications perspective.

We have added two new features in the Journaled File System (JFS). One is called *fragmentation support*. Previously, disk space was allocated in blocks of 4 KB. This method often wasted space on small files or files with final blocks containing only a few bytes. AIX 4.1 allows the system administrator to set up fragmented filesystems so that the last block of a file can be allocated in 512-byte blocks instead of 4 KB, resulting in much better disk utilization. In fact, the `/home` and `/var` filesystems are installed as fragmented by default.

The other new feature is an on-the-fly software data compression/decompression option that achieves about a 50% reduction in disk space requirements. It uses a fast LZ1-type algorithm written at IBM that is similar to compression techniques available on personal computers. Even with software compression enabled, performance is very respectable.

Other kernel enhancements include the following:

- Increased filesystem size up to 64 GB (individual files must still be less than 2 GB)
- Support of LVM disk striping for increased performance on very large files
- Improved CD-ROM support including Rockridge filesystem format, photo CD, and multisession capabilities
- Support of NFS 4.2, providing NFS updates and SMP support
- Improved application link times by 2X to 10X
- The newly adopted Open Software Foundation® (OSF®) Common Data Link Interface (CDLI) for LAN-based device drivers enables one driver to support protocols over both sockets and Streams (Previous device drivers could only deal with one or the other; the new interface provides much better path length, and drivers are much smaller.)

AIX 4.1 is well positioned technically to take advantage of IBM's new Power Personal Systems line of products and the high-end SMP systems.

Let's talk about new support for standards.

Day: There are several significant features of AIX 4.1:

- It supports POSIX 1003.1 systems interfaces.
- It supports POSIX 1003.2 shells and commands.
- Thread support is based on Draft 7 of the POSIX 1003.4A threads specification. The upcoming AIX 4.1-based Distributed Computing Environment (DCE) support will provide a mapping layer to convert from Draft 4 interfaces to Draft 7 interfaces.
- AIX is designed to be XPG4 base branded. XPG4 is a superset of the POSIX 1003.1 and 1003.2 standards. It provides more commonality in commands and interfaces, and better internationalization support than XPG3.

- AIX 4.1 is designed to conform to the emerging X/Open™ SPEC™ 1170, which is intended to provide a common UNIX interface specification.

Whenever IBM talks about change, users and developers worry about backward compatibility. What's the situation there?

Day: Since AIX 4.1 is a new version of the operating system with extensive enhancements to core technology, binary compatibility with existing applications was a major development and test theme for the product. For example, we had shifted to System V.3-compatible libcurses and X11R5 support, but we found that many existing application binaries depended on the previous libcurses, X11R3 and X11R4 interfaces. To handle this problem, we added a new Load-Only object type to shared libraries, so that we could transparently support the old interfaces for application binaries while ensuring that any applications rebinding would access only the new interfaces.

We have also identified old commands, links, and libraries that are non-standard or obsolete and have packaged them into optional compatibility packages. If AIX 4.1 is installed as a migration from AIX 3.2, these compatibility packages will be automatically installed to ensure a smooth migration for current customers. However, we do request that application providers refrain from relying on these facilities when releasing new versions of their software.

Although we have an excellent compatibility story for applications executing in the user space, the compatibility picture is not as good for applications with complex kernel extensions. Because of the restructuring of key components of the kernel for threads and SMP support, these complex kernel extensions (such as filesystems, complex device drivers, and communications subsystems) will require a moderate level of porting. "Well-behaved" character and block device drivers—those that do not directly access areas like the U area or `proc` structures, should run without trouble. On SMP systems, we will even run them *funneled,* that is, bound to a specific processor.

We have tested more than 90 of the key AIX 3.2 applications on AIX Version 4 with very good success. We believe this achievement will help quicken software vendor certification of applications on AIX 4.1.

The system management facilities and customization capabilities of AIX 4.1 have been greatly expanded.

System management has always been a strength of AIX. What's new with AIX 4.1?

Day: The system management facilities and customization capabilities of AIX 4.1 have been greatly expanded to support a more diverse set of users with the upcoming introduction of the Power Personal systems. Some system management and GUI enhancements are as follows:

- Repackaging of AIX for custom installation
- Simplified and faster operating system installation
- Installation assistant (graphical and ASCII versions)
- Graphical visual system management
- AIXwindows® Desktop, based on Common Desktop Environment (CDE) technology

These enhancements, along with many others, are a direct response to our customers.

One of the larger efforts in the development of AIX Version 4 was to completely restructure the packaging of the operating system components to support custom installation and improved modularity. Our objective was to automatically install the minimum operating environment for the system being installed, and then provide the facilities for the customer to customize the system by installing additional operating system components when needed.

In AIX Version 3, all the RISC System/6000 device support was installed on all systems, even if the hardware was not present or supported on the system. As the number of systems supported by AIX expanded and diversified, this procedure negatively affected software installation time and disk space usage. To alleviate this problem, AIX Version 4 now automatically installs a customized minimum operating system environment. The device configuration subsystem has been extended to automatically install only the device packages required to support the installed hardware. Custom install will determine if a minimum graphical environment or an ASCII-only based environment should be installed. It will also automatically install the country-specific message packages, locales, and helps for the primary language of the system to support the base operating system and all previously installed applications.

Installation of the base operating system is much simpler and faster than it was in AIX Ver-

sion 3. Now with fewer prompts and screens, it supports an optional no-prompt install using a data file on a diskette. AIX Version 4 supports four installation modes:

- **Overwrite:** Completely re-installs the system
- **Preservation:** Re-creates the root and user filesystems and leaves everything else alone
- **Migration:** Saves all the user's files, configuration information, and applications, then migrates the installed operating system components to the latest level
- **Installation:** Restores the system from a previously created system backup tape

A new Network Installation Manager (NIM) supports centrally administered remote system installation of networked systems. This method offers flexibility for centralized operating system and application installation through push installs to one or more network clients, pull requests from clients, and support of diskless and dataless clients from network servers. NetView® Distribution Manager (NetView DM) also offers remote application installation; however, NIM expands this capability by supporting remote operating system installation as well. These installation capabilities are ideal for customers who have a large number of similar or identical installations. For these customers, a centralized shop can use NIM, system backup, or data-driven Base Operating System (BOS) install to support their clients.

Once the minimum operating environment has been installed on the system, the new Install Assistant is started. This facility helps users through the customization process by taking them through either graphical- or text-based menus to complete the installation and system setup. The graphical version provides drag-and-drop interfaces for tasks such as adding users, installing additional software, and so on. The Install Assistant is essentially a customization help system that launches the appropriate System Management Interface Tool (SMIT) or Visual System Management subtask to perform the required customization.

I previously mentioned the repackaging of AIX and AIXwindows and its custom install. To help customers choose what to install, we have provided a software install facility called *bundles*. Bundles are collections of installable operating system software components and Licensed Program Product (LPP) components that are grouped together and can be installed with one selection. AIX 4.1 supports both system-defined and user-defined bundles. We have created the following prepackaged bundles of the operating system and LPP components:

- **Runtime client:** Primarily runs end-user applications; requires 120 MB of disk space for the ASCII user interface and up to 160 MB for the graphical user interface
- **Server:** Contains components for server functions such as network name serving, routing, more than two concurrent users, directory services, service aids, and performance analysis tools
- **Application Development:** Provides the tools and header files necessary to develop AIX-based applications
- **Personal Productivity:** Provides components for using the AIXwindows Desktop, Windows Application Binary Interface (WABI), Ultimedia® multimedia services, and DOS diskette support

This bundle support depends on the `installp` subsystem provided in AIX Version 4. This subsystem supports the operating system and program product software packaging as a hierarchy consisting of program products that contain installable packages of updatable options or filesets. These filesets are named so that the customer can understand the function provided from a high level. For example, you will no longer find filesets called `bosext1` and `bosext2`, as in AIX Version 3. Instead, you will find filesets named `bos.acct` (accounting services) and `bos.net.tcp.client` (the client support for TCP/IP network services). Textual descriptions are also provided in addition to the fileset name. Enhancements have also been made to the install facility to provide preview and deinstall capabilities.

Installation of the base operating system is much simpler and faster than it was in AIX Version 3.

Does this new packaging and install help others—such as OEMs—incorporate AIX technology?

Day: Certainly. For example, a repackaged AIX is currently being used as part of the IBM 6611 router. I also believe that Apple® and other OEMs are planning to use AIX 4.1 technology in product offerings on PowerPC-based systems. This granular packaging enables OEMs and other IBM groups to utilize the components of the system without costly tailoring and repackaging to

remove unneeded components and add needed ones.

Even the AIX kernel is in its own separate package and offers two flavors: uniprocessor and multiprocessor. The two packages contain different kernel binaries that are compiled from the same source. The operating system auto install determines which package to install on the system. We worked hard to keep the system-specific functions (buses, device support, and diagnostics) out of the runtime environment package containing the core UNIX commands and interfaces. This enables the runtime to be more easily adapted to diverse system definitions that contain a POWER or PowerPC processor. Most system-specific functions are in individual device packages that are custom installed, depending on the target system configuration.

The repackaging of AIX into more granular functions and better understood dependencies provides a solid base for building an excellent update strategy.

What about operating system updates?

Day: The repackaging of AIX into more granular functions and better understood dependencies provides a solid base for building an excellent update strategy. Our early AIX Version 3 update process was less than pleasing to customers. Because the operating system packaging was very coarse, customers often applied updates to software components they did not use. The tools we used for determining co-requisites between packages were also very coarse and conservative. This meant that when updates were applied to fix a particular problem, much of the system itself was updated as well. This side effect, along with our procedure of tracking updates with Program Temporary Fix (PTF) numbers, resulted in lengthy update times and some risk of changing unrelated areas. Using PTF numbers to track the level of software updates was also difficult, because there was no easy way to determine which PTF numbers had superseded previous PTF numbers.

In AIX 4.1, we wanted an easy, effective way to track the update process. In this new AIX release, each installable entity has its own version, release, modification level (for maintenance), and fix level (customer-required change between modifications). We call it a "VRMF" scheme. A higher VRMF number supersedes lower ones. To identify the requirements for a certain fix, you simply need to know the APAR number, available from the product support center. The system uses the APAR number to determine the filesets and the minimum VRMF levels that must be installed to provide the fix. Once the APAR is selected, the system will then install the fixes. In addition, the `oslevel` command knows which filesets and the minimum VRMF level necessary to report that the system is operating at some known maintenance level or release level.

When will we see the next refresh of AIX?

Day: A refresh of AIX 4.1, planned for October, will include features such as additional SMP performance tuning, Streams-based tty performance improvements, additional system hardware support, and worldwide translated message packages. Although the full internationalization facilities are in AIX 4.1, only the English message packages were available for shipment in August. No changes to application interfaces will be made, so there should be no effect on ISV application certification assessments. For existing AIX 4.1 customers, handling of the refresh will be similar to applying a new maintenance level to the system, not a reinstall.

What advice would you give developers as they move to AIX 4.1?

Day: First, ISVs with existing applications on AIX Version 3 should determine if their existing releases can be certified on AIX 4.1 without any changes. We have tested over 90 key AIX applications to ensure operating system compatibility, but we do not certify these applications. We use them with our extensive test cases to flush out incompatibilities.

Here are some suggestions for ISVs.

- If you are planning a new release of your application on AIX 4.1, obtain a copy of the AIX 4.1 publication, *All About AIX 4.1,* from IBM.
- If your applications use complex kernel extensions, you may need to make straightforward modifications to port them to the new kernel environment. You should also make the kernel extension thread and MP safe.
- We recommend that you compile your applications in the default `COM` mode of the new AIX 4-based compiler so that your applications will use the common instruction set. This will enable your application to run efficiently on the existing POWER and PowerPC processors as well as future PowerPC processors.

- Be sure to look at the contents of the `bos.compat` and `X11.compat` packages and determine if you are relying on commands, links, or libraries that we have determined to be obsolete. If so, you should remove your dependency on them.
- Determine if your applications are dependent on compatibility options for commands and convert them to the new POSIX 1003.2 or XPG4 specifications.
- For help or advice concerning porting your application, we have an excellent staff at our AIX porting centers backed by the AIX development team.

What are some points of incompatibility between Versions 3.1 and 4.1?

Day: Complex kernel extensions must be ported to Version 4 because of changes in support for both the MP environment and threads. Extensions that directly access the U area and `proc` structures must also be modified. There are now services for accessing those areas safely.

Moving to the POSIX 1003.2 command and utilities standard changed a few flags and the functions of some commands. We were often able to add the new functions and keep the old ones, but there may be some deletions that particularly affect shell scripts.

Hardware vendors will find that the new CDLI capability makes developing and supporting LAN device drivers quicker, easier, and less costly than the previous Common I/O (CIO) model found in AIX Version 3. However, to take advantage of the performance and common support for Streams- and sockets-based protocols, they must be modified to utilize the new interfaces.

Was anything taken out in AIX 4.1?

Day: We have removed the High Function Terminal (HFT) support that provided multiple sessions on graphics terminals without AIXwindows. It was causing us problems in upgrading to new versions of Xwindows from the X Consortium. In addition, it would have required us to port it to the new Streams-based tty subsystem. We chose to implement a small tty emulator called LFT in place of HFT and concentrate on making AIXwindows more robust and reliable.

A set of packages called *compat options* (such as `X11.compat` and `bos.compat`) contains compatibility support code enabling some existing applications to run unmodified on AIX 4.1. Moving functions to these packages provides an early warning to developers that we plan to eliminate them in future releases of the operating system.

What sets AIX 4.1 apart from other versions of UNIX?

Day: Here are a few examples of the many features that make AIX 4.1 such a great operating system:

- Operating system and compilers highly optimized for performance on POWER and PowerPC processors
- Scalability—the pageable and preemptable kernel enables AIX to scale well
- Dynamically loadable kernel extensions
- Built-in Journaled File System (JFS)
- Built-in Logical Volume Manager (LVM)
- Extensive graphical and ASCII system management facilities
- Custom installation for loading only what you need onto your system
- Automatic device installation and configuration
- Comprehensive 2-D and 3-D graphic support
- Broad range of connectivity types and options

In this new AIX release, each installable entity has its own version, release, modification level, and fix level.

New PowerPC Magazine

PowerPC News, a complimentary publication from IBM, brings you the latest milestones and major successes of the PowerPC microprocessor architecture. The *PowerPC News* is published monthly. To add your name to the mailing list or for additional information, call 1-800-PowerPC. ■

Overview of AIX/6000 4.1

By Mark S. Brown

This article describes the new features in AIX 4.1, a major release now available.

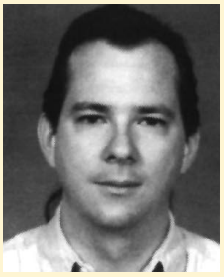
AIX 4.1, the latest release of AIX/6000®, is a major release that includes many new features and enhancements. From the packaging of installation to the kernel-level threads support, it contains a multitude of long-awaited improvements. Even with these improvements, AIX Version 4 still maintains binary compatibility with previous versions. The only exception is a few commands, for which an easy migration process is provided.

Installation

AIX 4.1, now distributed on CD-ROM, has a smaller Base Runtime Environment install package that divides the system into modular sections. This division not only speeds up the installation process, but also enables users to customize installation of AIX 4.1 to fit their needs. As an added benefit, patches and updates are much easier to install because the number of prerequisites has been greatly reduced.

The new Network Install Manager (NIM) gives system administrators central control of installation and configuration by enabling them to install the base OS from a server onto multiple client systems via the network. With NIM, diskless, dataless, and stand-alone systems can be installed.

AIX Version 4 offers backward compatibility packages with AIX 3.2. Compatibility packages support X11R3/R4, old font types, and features such as the directory tree structure and the curses package that have been changed in Version 4.



Mark Brown

Filesystem Enhancements

When configuring Version 4, users will notice changes in the AIX Journaled File System (JFS). AIX 4.1 supports filesystems of up to 64 GB (although files are still limited to less than 2 GB in this release), BSD-style block fragments, and compression. *Fragments* are parts of a disk block; a filesystem created using fragments can store more than one file on a given disk block.

With the new JFS, a filesystem can be created from fragments of different sizes: 512, 1024, 2048, and 4096 bytes. For example, on a Version 3 JFS, a file of 500 bytes occupies an entire 4096-byte disk block. However, using the new JFS in Version 4, up to eight 500-byte files can be stored in the same disk block. The new JFS means a large savings in disk space usage, especially for users whose systems manipulate many small files.

Automatic data compression is another space-saving option in Version 4. This technique creates filesystems with a modified Lempel-Ziv technique that dynamically compresses and restores data written to disk. Users can also utilize the disk striping feature in Version 4 to set up RAID-0, which spreads the logical space of a logical volume over multiple disks. Disk striping enables simultaneous operation of multiple disks for higher performance.

Common Desktop Environment

The new look of Version 4 continues with the Common Desktop Environment (CDE). CDE is part of a co-development project among major UNIX vendors to provide portability and greater usability across platforms. X11R3, X11R4, and X11R5 are fully supported for backward compatibility.

System Management Interface Tool (SMIT)

System administrators have more help in this release through improvements to SMIT, the `mksysb` backup tool, login and password controls, and a new performance tools package. The `mksysb` tool has new extensions (such as `savevg` and `restvg`) that will save and restore user Volume Groups (VGs), save the definition of paging spaces in user as well as root VGs, and also save policy and

map files for logical volumes. SMIT also offers a non-interactive install feature.

In addition, printer support for SMIT has been greatly expanded, easing the administration of print queues. It is now possible to move print jobs from one queue to another, and to hold and release print jobs in a queue.

Open Standards Conformance

One major goal for this release was conformance to the new IEEE® POSIX 1003.2 (commands) and *X/Open Portability Guide,* Issue 4 (system interface and commands volumes). The syntax and output has been modified for some commands to meet these specifications, which will soon be met by most other UNIX vendors.

These modifications are intended to make applications more portable to and from the RISC System/6000. Version 4 conformance to XPG4 command specifications may cause some minor confusion since the syntax and output of certain commands may have changed. To minimize this confusion, IBM developers have tried to use the old syntax wherever possible.

AIX 4.1 is also a step towards compliance with the emerging SPEC 1170 Common Interface specification. As SPEC 1170 becomes final, future releases of AIX will be modified to conform to this important UNIX standard. Developers writing applications for AIX 4.1 should not be concerned since most major changes have already been made, and plans for maintaining compatibility with earlier versions of AIX are already in place.

Application Interface Changes

The curses subsystem has also been revamped, with System V Release 3 curses as its base. Developers who routinely adjust code to meet the idiosyncracies of Version 3 curses will find that the default compilation now uses System V curses as the base, with AIX Version 3 curses available to allow binary portability to Version 4. Moving application code to the new curses also ensures compatibility with SPEC 1170 curses.

A major change to the programming environment is the total rework of the threads subsystem, from the older user-level Distributed Computing Environment (DCE) threads in Version 3 to kernel-based POSIX 1003.4a Draft 7 threads. These are 1:1 (user-level:kernel-level) threads, with M:N threads intended in the near future. The OSF/1® locking model is used, and the same library mechanics (`libc_r.a`, `libpthreads.a`) are still there. The new threads provide much more function under the covers.

Other changes to the application interface include full support for the 64-bit long long `int` and 128-bit long double types, new interfaces to handle regular expressions (from POSIX), full XTI support in addition to TLI, and a full implementation of Streams. The entire tty and pty subsystems have been rewritten to take advantage of Streams.

AIX's International Language Support now encompasses 37 locales, including the bidirectional languages Hebrew and Arabic, Chinese, and many Eastern European locales. The underlying default codeset for characters (that is, the actual byte representation for the visible character) is now industry-standard ISO® 8859-1, with IBM PC-850 still provided as an alternate. Locales have been improved to include more cultural differences, such as multilevel sort capabilities. The locale conversion tool (`iconv`) has been expanded and a developer's toolkit for the Unicode™ (ISO 10646) codeset is also available. Unicode, the future direction for portable locale-independent data exchange, will be supported by AIX.

Enhancements made to the loader, linker, and binder make application development easier. Performance, efficiency in `XCOFF` size, TOC overflow handling, and `LIBPATH` handling have all been improved. Shared library data is now stored in a separate per-process shared library segment to ensure fewer conflicts for shared library data.

AIX 4.1 is a step towards compliance with the emerging SPEC 1170 Common Interface specification.

Compatibility with AIX 3.2

AIX Version 4 is intended to support, with a few exceptions, Version 3.2 binaries. Most applications written to run on AIX 3.2 will also run on 4.1 within the processor families for which they were compiled. With the advent of "common mode" compilation in AIX 3.2.5, binary compatibility extends across most of the POWER, POWER2, and PowerPC processor lines. Applications compiled for POWER processors will run on all platforms, while those compiled specifically for POWER2 or PowerPC (not in common mode) must be run only on those processors. Interface differences are also rare; a compatibility rate of over 90% is expected for existing programs.

Mark S. Brown, IBM Corporation, 11400 Burnet Road, Austin TX 78758. Internet: mbrown@austin.ibm.com. Mr. Brown is a senior programmer with the AIX Base Operating System Development group. He holds a BS in Computer Science from Southwest Texas State University in San Marcos.

Filesystem Enhancements in AIX 4.1

By Bill Baker

Did you ever wonder why small files consumed so much disk space in the AIX Journaled File System (JFS)? Have you ever tried to decrease the block size in JFS? If so, you will be pleased with AIX Version 4. By supporting fragments, the new JFS enables small files to be allocated more efficiently. For even more space savings, JFS is now able to compress files. In addition, the JFS supports filesystems well in excess of 2 GB.

AIX Version 4 introduces several significant filesystem enhancements in the JFS. These enhancements include support of fragments, capacity to compress files, and new 64-bit integers in the compiler that help to relax the former maximum filesystem size of 2 GB.

JFS Fragments

In AIX Version 3, the JFS had a fixed block size of 4 KB. This design made the JFS much easier to integrate into the pager and avoided the classic problem of free space fragmentation. However, in environments with many small files, such as news servers, large fixed-size blocks can result in wasted space.

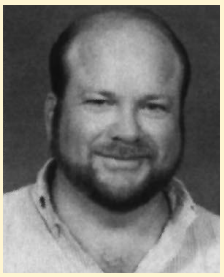
In AIX Version 4, JFS allows disk space to be allocated in smaller blocks known as *fragments*. These partial blocks improve space efficiency for files smaller than the fundamental block size of the filesystem, without losing the performance advantage of large blocks. Fragments have long been implemented in the Berkeley Software Distribution Fast Filesystem[1].

The key to using fragments to reduce space inefficiency of small files lies in partially allocating the last block of a file. On average, the last block of every file is half-wasted. If every file in a filesystem is less than a full block, then 50% of the filesystem space is wasted. However, with fragments, all but the last block have full allocations of contiguous fragments; the last block is only partially allocated.

The partial allocation of the last block of a file or directory occurs when the file has direct geometry—data block addresses in the inode. Since the JFS stores eight direct addresses in the inode, only files less than 32 KB will have their last block fragmented. Files larger than 32 KB will have full block allocations, even in their last block.

For example, on a filesystem with 512-byte fragments, a 4,100-byte file will consume nine fragments. The first logical block of the file will consume eight contiguous fragments. The second logical block, consisting of the four remaining bytes, will consume just one fragment. In AIX Version 3, this second logical block would have consumed another full block.

Partial allocation implies that each block address slot in the inode must be able to represent the number of fragments actually allocated for that slot. Since the maximum full-block-to-fragment ratio is eight 512-byte fragments, each block address slot needs three additional bits to store a partial allocation. These bits are encoded in the upper nibble of the disk block address. The most significant bit is unused. The next three



Bill Baker

[1] McKusick, Marshall Kirk; Joy, William N.; Leffler, Samuel J.; and Fabry, Robert S. "A Fast File System for UNIX," *UNIX System Manager's Manual*. Computer Systems Research Group, University of California at Berkeley, The Regents of the University of California and/or Bell Telephone Laboratories. 1986. SMM 14.

bits record the number of fragments less than a full block.

In the example above, the disk address for the second logical block would encode the number 7 in the upper nibble. Since there are eight fragments per block in this example, the number of fragments actually allocated in that slot is 8-7=1. The rest of the address contains the disk block number for the data.

This organization enables fragmented and non-fragmented filesystems to use an identical disk inode structure. Since these bits are always zero in a non-fragmented filesystem, they will be interpreted as a full block allocation in both fragmented and non-fragmented filesystems.

Another enhancement adopted from the Berkeley Fast Filesystem is the ability to specify the ratio of bytes to the number of inodes in the filesystem. This ratio is described as the Number of Bytes Per Inode (NBPI). In AIX Version 3, the JFS had a fixed NBPI of 4096. If a filesystem had 2,048 4 KB blocks, then it would have 2,048 inodes. More inodes could be added to the filesystem by increasing the size of the filesystem, but the ratio remained constant.

In AIX Version 4, the system administrator can specify different values of NBPI to customize the number of inodes in the filesystem. Raising the value of NBPI to its maximum of 16384 creates filesystems with fewer inodes for a given size. This change reduces the amount of space normally reserved for inodes and benefits filesystems that store a few large files. In contrast, lowering the value of NBPI results in filesystems with more inodes for a given size. This might be useful for a filesystem with many files smaller than 4 KB. When a new filesystem is created, both the fragment size and the NBPI value are set and cannot be changed. The default value of 4096 for both the fragment size and NBPI results in a filesystem that is both disk image-compatible with AIX Version 3 and interchangeable with an AIX Version 3 machine.

One disadvantage of having variable size allocations is free space fragmentation. The inode organization of JFS requires that a 4 KB block of fragments be allocated contiguously. Even though a filesystem may have plenty of free space, if contiguous fragments cannot be allocated, the system call that attempted to allocate more space will fail with the error "ENOSPC".

The defragmentation utility in AIX Version 4 solves this problem. By attempting to coalesce the free space in the disk allocation map, this utility facilitates future allocations of contiguous fragments. This process enables full block allocations, which can only be made from contiguous fragments. The tool called *defragfs* can be used on an active filesystem. Alternatively, you can use its spy mode to view the fragmentation of the filesystem and any changes that would have been made if you had used the tool in active mode.

File Compression

Compression is built on top of JFS fragments. With JFS, compressed filesystems previously available for DOS are now possible in AIX Version 4. In a compressed filesystem, the data for regular files is dynamically compressed to reduce the amount of space consumed by the file. Directories, though not compressed, are fragmented, just as they would be in a fragmented filesystem.

In AIX Version 4, the system administrator can specify different values of NBPI to customize the number of inodes in the filesystem.

Unlike a fragmented filesystem in which only the last block is partially allocated, a compressed filesystem enables each block to be allocated less than a full block of fragments. A block about to be written to disk is first compressed. If the data can be compressed by at least one fragment, then the exact number of fragments required to hold the compressed data—not a full block—is allocated. This process happens independently for each block in the file.

Although it would be more space efficient to compress the entire file instead of compressing each individual block separately, it is impractical because the JFS must support efficient random I/O. To be accessible at any arbitrary offset, a file would have to be completely decompressed in memory and remain in memory. Although this implementation is good at maximizing the compression rate, the performance and system resource utilization would be extremely inefficient.

JFS uses the algorithm based on Lempel-Ziv[2,3], which results in a high compression-efficiency rate. The efficiency of the algorithm is a function of whether data values are uniformly distributed over the range of possible values (truly random data). Since both text files and executables exhibit a fairly non-uniform distribution, they can compress well and can have a compression rate as high as 50%.

[2] Ziv, J. and Lempel, A. "A Universal Algorithm for Sequential Data Compression." *IEEE Transactions on Information Theory* (May 1970) p. 337-343.

[3] Brent, R.P. "A Linear Algorithm for Data Compression," *The Australian Computer Journal* 19 (May 1987).

The JFS uses a conservative space-allocation policy to avoid over-allocating disk resources. In standard UNIX, disk space is allocated synchronously. When the `write` system call extends a file, the space is allocated immediately. If there is not enough free space available, the write fails with the error code ENOSPC. In a compressed filesystem, the minimum amount of space required to store a block is unknown until after the data is compressed. Performance reasons make it impractical to compress the data each time a write is performed.

A good solution is to decompress the data immediately after it is read from the disk and to compress it immediately before it is written back to disk. The file block is decompressed while it is in memory, enabling file reads and writes to occur without compressing or decompressing the data.

The JFS uses a conservative space-allocation policy to avoid over-allocating disk resources.

Delaying compression of data until it is written back to disk also implies delaying the allocation of space. This delayed allocation can lead to problems if the filesystem does not have sufficient free space. For example, if an existing file block is compressed, an application modifies the block via the `write` system call. Then JFS prepares to write the data back to disk by compressing the data. If the new data cannot be compressed as tightly as before, and if the filesystem has no free space, then JFS cannot save the data. In this example, since the write is asynchronous to the application program, JFS must simply discard the data.

To prevent this worst-case scenario, JFS uses a conservative approach that fully allocates the block while it is in memory, and then reallocates the block when it is written out. Even if the filesystem is full, or if the data cannot be compressed, the file will have allocated sufficient space so the operation can succeed. This approach guarantees the synchronous allocation requirements for the filesystem.

JFS compression supports different, user-defined compression algorithms. The compression code itself is completely encapsulated in an independent load module. This module is loaded as a kernel extension for internal use by JFS. The module is also explicitly loaded into application programs such as `mkfs` and `backup` for inode backups.

Large Filesystems

In classic UNIX, the starting logical offset for `read` and `write` system calls is stored as an absolute byte address in the system open file table. It is maintained by the system and can be modified with the `lseek` system call. In the typical 32-bit implementation, the file offset is defined as a signed 32-bit integer. This effectively limits the maximum file offset to 2 GB. Since devices are accessed as files, the same limitation applies to devices.

To allow filesystems larger than 2 GB, both the kernel and application programs must have the ability to address the storage beyond the 2 GB limit. For the kernel, this is not a problem since the kernel uses the block-oriented strategy interface of the device driver to make I/O requests. For the filesystem commands, such as `fsck`, only the byte-oriented `lseek` interface is available.

Several alternatives for extending the range of the file offset are possible. By changing the file offset to an unsigned integer, the effective range can be increased to 4 GB. Some implementations have added a special `ioctl` command, which causes the file offset to be interpreted as a block offset instead of a byte offset. Perhaps the most natural solution is to increase the size of the offset.

In AIX Version 4, the file table offset is re-implemented as a 64-bit integer. A new abstract data type, `offset_t`, derived from the "`long long`" base type, is defined in `<sys/types.h>`. A new system call, `llseek`, is created. Its interface is identical to `lseek`, except that the second parameter and the return value are the `offset_t` type.

In addition to the system open file table itself, the device driver interface must be enabled for device access beyond 2 GB. The `uio` structure is passed to the device driver's read and write entry points. This structure defines the I/O request, including the logical starting offset. The offset field in the structure is changed to be the `offset_t` type.

Although the read and write entry points of a device driver are of interest in understanding the implementation of the `read` and `write` system calls, they are not used by the filesystem when making I/O requests. Instead, the filesystem uses the strategy interface of the device driver. Only device drivers that can support "block" devices need to provide a strategy entry point.

The filesystem constructs a buffer header that describes the I/O to the device driver. Encoded in the `b_blkno` field is the logical 512-byte block number for the request. Since the strategy interface is not byte-oriented, no changes were necessary to allow filesystems larger than 2 GB. The signed 32-bit block number allows block offsets

up to 2^{40}. The maximum device size for AIX Version 4 is effectively 1 terabyte.

The maximum filesystem size for the JFS is now limited by the internal data structures and algorithms of the JFS, not the size of the file table offset. The factors that determine the maximum size include the width of the disk addresses in inode and indirect blocks, as well as the limit on the maximum number of inodes in a filesystem. The new theoretical maximum filesystem size is 256 GB.

The filesystem is not the only beneficiary of the larger offset. Application programs that directly access devices, such as raw logical volumes, can now access data beyond the 2 GB boundary. Applications need only be recoded to use `offset_t` and `llseek`.

Conclusions

These enhancements improve the competitive position of the AIX filesystem. Fragments brings the JFS into line with the capabilities of the Berkeley Fast Filesystem. Compression raises the bar for all UNIX filesystems. Large filesystems ease the system management aspects of multi-disk configurations and make the first step towards files greater than 2 GB.

Bill Baker, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Internet: web@austin.ibm.com. Mr. Baker is a senior programmer in the AIX File System Development department of IBM's RISC System/6000 Division. He previously was a software engineer at Texas Instruments, where he led the team that developed a UNIX implementation for TI's 680x0-based multiprocessor system. He has a BS in Computer Science from the University of Oklahoma.

COMMON Prepares for Blowout Fall Conference in San Antonio

More than 4,000 midrange computer professionals are expected to attend the COMMON Fall 1994 Conference (October 16-20) and the COMMON Fall Expo (October 15-17) at the Marriott Rivercenter, the Convention Center in San Antonio, Texas.

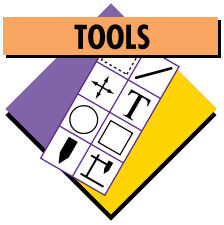
COMMON, the world's largest IBM midrange user group, is composed of over 4,800 installations and users of the IBM AS/400®, System/36, RISC System/6000, and personal computers. Members also include users interested in connectivity or networking solutions within the Enterprise Systems Architecture product family.

Nearly 100 sessions and labs will be devoted to RISC and AIX solutions at COMMON '94. In addition, the Fall Expo will feature 90,000 square feet filled with the latest midrange products and services offered by IBM and more than 120 of the industry's leading vendors.

The conference keynote address will be delivered by IBM CEO and Chairman Lou Gerstner on Monday, October 17. Gerstner is expected to discuss IBM's plans to help customers face the future. Many IBM Austin executives and developers are also expected to attend.

Conference registration is limited to COMMON members; both company and individual memberships are available. Companies that register more than 10 attendees will receive a multiple employee discount. For more information about COMMON, including the 1994 and 1995 conference schedules, contact the following:

COMMON Headquarters
401 North Michigan Avenue ◆ Chicago, IL 60611-4267
phone: (800) 777-6734 ◆ (312) 644-6610
fax: (312) 644-0297



FDPR for AIX Executables

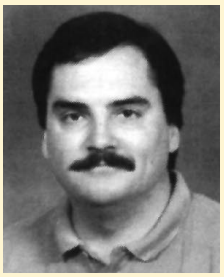
By Randall R. Heisch

The fdpr utility is a performance tuning tool used to optimize a program based on the actual hardware utilization characteristics of an application and its workload. The fdpr tool can improve both the execution time and real-memory utilization of user-level application programs by using a technique called Feedback Directed Program Restructuring (FDPR). Program speedups up to 73% and reductions in real-memory requirements up to 61% have been achieved using the FDPR reordering techniques.

The fdpr tool reorders and modifies the instructions in an AIX executable program by using profile information collected during the actual execution of the program. This improves instruction cache, instruction TLB (Translation Lookaside Buffer), and real-memory utilization by packing together highly executed code sequences and by recoding conditional branches to improve hardware branch prediction. It maximizes speedup by reordering instructions at the basic block level across the entire executable, independent of procedure or csect boundaries.

Highlights of the fdpr tool

- Program speedups up to 73% measured (typically 10-20%)
- Reductions in text memory requirements up to 61% (typically 20-30%)
- Global basic block-level instruction reordering
- Guaranteed functionality
- Debuggable with AIX 4.1 `dbx`



Randall R. Heisch

The AIX 4.1 `dbx` debug program has been modified to support executables reordered with the -R0 and -R2 options of fdpr. Additionally, the fdpr tool provides a reordering methodology[1] (option -R0) that guarantees functionality for reordered programs. The fdpr tool is part of the IBM Performance Aide 2.1 program product.

Why Reorder a Program?

Today's high-performance computer memory architectures are optimized for programs that exhibit high spatial or temporal locality for both instructions and data. Memory hierarchies have evolved in an attempt to minimize cost and maximize performance by exploiting this "locality of reference" program characteristic. Similarly, design assumptions are typically made regarding other program characteristics that result in processor designs optimized for those assumed characteristics (such as branching behavior and hardware branch prediction).

If these program assumptions are valid, processor performance is maximized. However, when a program deviates from these assumed characteristics, the processor architecture is inefficiently utilized. This often leads to reduced performance or excessive use of real memory.

Likewise, compilers attempt to generate optimum code for a specific hardware architecture on the basis of similar program assumptions. However, compiler optimizations are usually limited to a purely static analysis of a program, which includes speculation about how a program will execute on a given hardware platform. Since many programs result from binding together several separately compiled or assembled object modules, the compiler does not usually have an overall view of the final organization of the

[1] Heisch, R. R., "Trace-directed Program Restructuring for AIX Executables," *IBM Journal of Research and Development* 38, no. 4, (1994).

executable image. Therefore, it cannot perform a truly global optimization.

The fdpr tool effectively closes the loop in the optimization process. It attempts to further optimize a program by collecting information on the actual behavior of a program while it is executing. It then uses that information to reorder and modify instructions across the entire executable program image to optimize use of hardware.

Figure 1 shows poor program locality for a typical high-level language code sequence. In this code sequence, the error path (taken when `x` equals `y`) is usually not executed (information not known at compile time).

Figure 2 shows the resulting assembler code generated for a code sequence of the form shown in Figure 1. The example represents a machine with 16 instructions per instruction-cache line. Although only the first four instructions are usually executed (the instructions for the `if (x = = y)` statement), the remaining unexecuted instructions (representing the error handler code) are also loaded into the cache. Since the minimum allocatable unit of a cache (typically a cache line) is usually much larger than a single instruction, poor program locality results in higher miss rates and reduced performance because of inefficient cache utilization. Similarly, real memory space may be wasted on instructions that are loaded into memory (because of their proximity to frequently executed code) but usually not executed.

Figure 2 also shows the results of reordering the instructions in the order they were executed. Based on information collected at runtime, the frequently executed code paths are grouped together. Additionally, the conditional branch instruction has been recoded to improve the success of hardware branch prediction. The result is twofold: a reduction in runtime memory requirements due to improved utilization of real memory pages, and improved performance due to reduced instruction cache and TLB miss rates and improved branch prediction.

Reordering frequently executed code sequences also results in reduced collisions in an N-way set-associative cache. If more than N instructions in a highly executed code loop map to the same cache congruence class, constant cache misses will occur because of the thrashing that results from these collisions. Reordering the instructions in a program based on the actual execution path can produce additional performance improvements by reducing the "conflict misses" in an N-way set-associative cache.

```
if ( x = = y )
                {
                /* Error handler code */
                }
   /* Otherwise, execution continues here */
```

Figure 1. Example of poor locality of reference

Original Text			Reordered Text		
0x10000330	l	r2,0x14(r1)	0x10000330	b	0x10000590
0x10000334	l	r3,0x38(r1)	0x10000334	b	0x10000594
0x10000338	cmp	cr1,r2,r3	0x10000338	b	0x10000598
0x1000033c	bne	1,0x10000374	0x1000033c	bne	1,0x10000374
0x10000340	ai	r3,r31,0x8	0x10000340	ai	r3,r31,0x8
0x10000344	l	r4,0x38(r1)	.		
0x10000348	bl	0x10000530	.		
0x1000034c	l	r2,0x14(r1)			
0x10000350	ai	r3,r31,0x1c			
0x10000354	l	r4,0x38(r1)	0x10000590	l	r2,0x14(r1)
0x10000358	bl	0x10000530	0x10000594	l	r3,0x38(r1)
0x1000035c	l	r2,0x14(r1)	0x10000598	cmp	cr1,r2,r3
0x10000360	ai	r3,r31,0x30	0x1000059c	beq	1,0x10000340
0x10000364	l	r4,0x38(r1)	0x100005a0	l	r3,0x3c(r1)
0x10000368	bl	0x10000530	0x100005a4	l	r4,0x38(r1)
0x1000036c	l	r2,0x14(r1)	0x100005a8	bl	0x100005b0
0x10000370	b	0x10000384	0x100005ac	b	0x100005d8
0x10000374	l	r3,0x3c(r1)	0x100005b0	stu	r1,-64(r1)
0x10000378	l	r4,0x38(r1)	0x100005b4	st	r3,0x58(r1)
0x1000037c	bl	0x10000278	0x100005b8	st	r4,0x5c(r1)
0x10000380	st	r3,0x40(r1)	0x100005bc	l	r3,0x58(r1)
0x10000384	l	r3,0x38(r1)	0x100005c0	srai	r3,r3,0x3
			.		
			.		

Figure 2. Machine code examples for the source code shown in Figure 1

FDPR Process Overview

Figure 3 shows the overall process of applying FDPR. An execution profile is captured for the executable program to be reordered while it runs on the desired workload (W). The profile information is analyzed to determine an optimal instruction reordering. The reordering information from this analysis is then used to create a new, restructured executable. The reordered executable shows varying degrees of performance improvements or reduced instruction memory requirements when run on workload W (or similar workloads).

The fdpr tool follows a similar process and executes in three distinct phases.

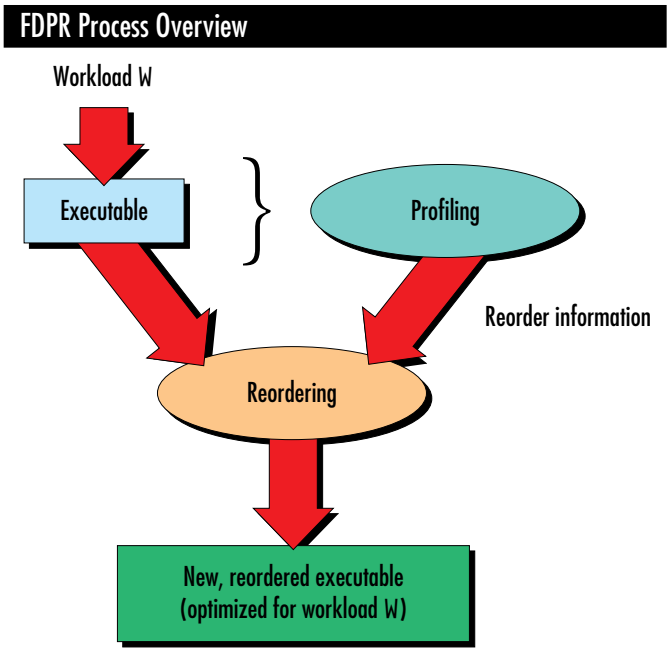


Figure 3. FDPR process overview

Program	POWER (8KIC)	POWER (32KIC)	POWER2	PowerPC 601 (0MB L2)
ksh	+45	+14	+13	+20
awk	+19	+9	+10	+11
vi	+13	+8	+6	+22
sed	+7	+5	+4	+6
SPEC 022.li	+20	+5	+4	+9
SPEC 072.sc	+11	+4	+1	+5
SPEC 056.ear	+9	+9	+4	+2
SPECint92'	-	-	+4[1]	+5[2]
RDBMS1 TPC-A	-	-	+15	-
RDBMS1 TPC-B	+17	+19	-	-
RDBMS2 TPC-C	-	+12	-	-

[1]66.7 MHz RS2G.9, 0 MB L2
[2]80 MHz PowerPC 601, 0 MB L2

Figure 4. Measured program speedups (%)

Phase 1: Creates an instrumented version of the executable
Phase 2: Runs the instrumented version to collect profile data
Phase 3: Reorders the original executable based on the profile data

Phase 1 builds a version of the executable to be reordered with the necessary instrumentation "hooks" required to collect the execution profile data. In phase 2, the instrumented executable created in the first phase is run for the desired workload (that is, used in some typical manner). In the last phase, the profile information is analyzed to determine an optimal instruction ordering, and the reordered version of the program is created. These phases can be run separately or in combination, but they must be in sequence.

In its simplest usage, the fdpr tool is invoked as follows:

```
fdpr  -p program  -x workload
```

where `program` specifies the name of the executable program to reorder, and `workload` specifies the desired invocation required to use that program. If successful, the reordered version of the program, `program.fdpr`, is created.

The fdpr tool provides four levels of optimization, that compare as follows:

- **R0/R2:** Guarantees functionality[2] (R0 only) of a reordered program; preserves debuggability at full program speedup; typically increases the size of the executable by 30-50%
- **R1/R3:** Maintains the size of the original executable; cannot guarantee functionality; marginally debuggable at the expense of speedup

Example fdpr Results

Figures 4 through 7 show typical results for reordering user-level applications. All results were collected using the fdpr -R2 optimization level. However, some results were measured during early prototype testing using an IBM internal-use-only trace tool for profiling.

Figure 4 shows the speedups measured for the 8KIC and 32KIC POWER, POWER2, and PowerPC 601 machines. All speedups shown were calculated by comparing the execution time of the original program to that of the reordered program for the same workload on the same machine. Two different commercially available relational database management systems (RDBMS1 and RDBMS2) were used for the TPC-A, TPC-B, and TPC-C tests. Note that each program shown in Figure 4 was reordered and tested on the same

[2] The -R0 option invokes a reordering methodology (see footnote 1) that can theoretically guarantee functionality; however, reordering may produce programs that behave unexpectedly. To verify expected functionality, programs reordered with the fdpr utility should be rigorously retested with at least the same test suite used for the original program.

workload specific to that test. Figure 7 presents the results for cross-workload measurements.

Source of the Improvements

Figure 5 shows the factors contributing to the 17% speedup measured for the RDBMS1 TPC-B test on the POWER machine. These measurements were taken using a hardware performance monitor that provides exact counts for clock cycles, instructions executed, instruction cache and TLB misses, and so on, during program execution.

The data indicates improved performance due to reduced instruction cache and TLB miss rates, and fewer conditionally issued instructions that were canceled (conditional branches that were predicted incorrectly).

Figure 6 shows the reductions in text real-memory requirements for several user-level applications. The changes in memory requirements were calculated using two different methods (shown as xx/yy). The first number (xx) represents the change in the total number of pages required for executing the program; the second number (yy) indicates the change in the maximum simultaneous pages required during execution. The increases shown for `awk` and `vi` are due to missed branch table modifications that result in additional text memory pages touched during execution (a by-product of -R0 guaranteed functionality). However, the 61% reduction for the RDBMS1 TPC-B test represents an instruction memory savings of more than 512 KB for this program.

To achieve guaranteed functionality and preserve debuggability (using the fdpr -R0 or -R2 options), the reordered instructions are appended to the original executable. This results in increased file sizes for the executable programs as shown in Figure 6. However, for environments in which disk space is not extremely critical, the benefits of improved performance and reduced real-memory requirements more than compensate for the additional disk storage requirements.

Cross-Workload Effects

For certain applications, determining an appropriate workload for reordering a program can be a challenge. If two workloads execute a program differently, finding a single, optimal instruction ordering for both workloads is unlikely.

For example, a program is reordered for workload A; the reordered version is then run on workload A and results in a speedup of Sa. Similarly, a version reordered for workload B is run on workload B and results in a speedup of Sb. Reordering a third version of the program for workloads A and B together—in which the workloads use and exercise the program very differently—and running that version separately on both workloads, usually results in speedups of less than Sa and Sb. Also, running this reordered program on workload C, where workload C was not in the set of workloads used to reorder the program, typically yields little (or possibly negative) improvement, if workload C differs greatly from the other workloads.

Parameter	Reordered	Original
CPI	2.52	2.98
IC Miss	4.20%	5.90%
ITLB Miss	0.150%	0.390%
Can/Cond	23.0%	52.0%

Figure 5. Factors contributing to RDBMS1 TPC-B speedup

Program Size	Text Memory Requirements	Executable Size
ksh	-51/-63	+16
awk	-9/+40	+16
vi	+9/-64	+31
sed	-25/-25	+41
SPEC 022.li	-31/-59	+11
SPEC 072.sc	-28/-24	+19
SPEC 056.ear	-18/-48	+8
RDBMS1 TPC-B	-43/-61	+5

Figure 6. Text working set and executable size changes (%)

The cross-workload effects for two programs, `awk` and `ksh`, are shown in Figure 7. The reordered `awk` programs are as follows:

- `awk.heap`: Reordered for heapsort workload
- `awk.pts`: Reordered for an `awk` Performance Test Suite (PTS) workload
- `awk.comb`: Reordered for both the heapsort and PTS workloads

The reordered `ksh` programs are as follows:

- `ksh.scr`: Reordered for the `ksh` "built-in" commands workload `scr1`
- `ksh.sum`: Reordered for the sequential summation workload `sum.ksh`

Workload	Reordered Program Speedups (%)		
	awk.heap	awk.pts	awk.comb
heapsort	+19	-9	+18
PTS	+18	+22	+18
	ksh.scr	ksh.sum	ksh.comb
scr1	+21	+3	+18
sum.ksh	+11	+45	+30

Figure 7. Cross-workload results

- `ksh.comb`: Reordered for both `scr1` and `sum.ksh` workloads

As shown in Figure 7, running program `awk.pts` on the heapsort workload (a workload not used to reorder the program) actually results in a decrease in performance. However, running `awk.heap` on the PTS workload (again, a workload not used to reorder the program) results in an 18% speedup (slightly less than when `awk.heap` is run on the heapsort workload). The combined reordered `awk` (`awk.comb`) produces significant speedups for both workloads (although `awk.comb` running PTS yields less improvement than `awk.pts` running PTS). The `ksh` cross-workload results are similar to the `awk` results, with only a 3% speedup shown for `ksh.sum` running the `scr1` workload, and still significant speedups for `ksh.comb` on both workloads. The data suggests that careful selection of workloads is critical to achieve good overall speedups from reordering. However, to reach maximum performance improvements for `ksh`, `awk`, and these simple workloads, the program must be reordered for the exact workload for which it will be used.

One possible solution to the problem of cross-workload effects for widely varying workloads is to produce different versions of the program that are each optimized for specific workload types. Using this method, once the workload type is known, the executable program reordered for that specific workload is used.

Debugging Support

To achieve maximum program speedup, highly executed code sequences must be grouped together, independent of procedure or other structural boundaries. The "hot" code paths from multiple procedures or csects are reordered together, away from code that is not generally executed, as well as non-code such as traceback entries. However, a disadvantage of this blurring of procedure boundaries and removal of traceback entries is that program debug capability is reduced or completely disabled.

To solve this problem, the fdpr -R0 and -R2 options employ a unique reordering methodology that preserves debuggability, while requiring only minor modifications to the debugger. This methodology maintains the structure of the original program, which includes the relative location of traceback entries. Also, a new program section is created in the reordered program that contains a mapping of original to reordered addresses. Using this methodology, fdpr produces debuggable, reordered programs that simultaneously achieve maximum speedup by maintaining global instruction reordering.

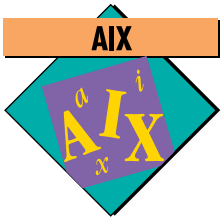
Debug functionality is equivalent to that provided for any stripped or optimized executable. The AIX 4.1 `dbx` program will recognize and utilize the additional program section and the original program text area, and translate between reordered and original instruction addresses for program failures, stack traceback, and assembler single stepping.

Summary

The fdpr performance tuning utility enables improved performance and memory utilization by optimizing a program based on its actual behavior. Large programs with many conditional tests or highly structured programs with multiple, sparsely placed procedures offer the greatest potential for reordering improvements. Significant speedups have also been measured on smaller programs with poor hierarchical virtual memory utilization or branch behavior. For applications in which the workloads are not critical to program behavior, a single reordered executable can be produced to realize these benefits. For applications in which the workloads do change program behavior significantly, using fdpr to provide multiple executables (each reordered for a specific workload type) or reordering for the most common workload may still result in improvements.

Randall R. Heisch, IBM Corporation, RISC System/6000 Division, 11400 Burnet Road, Austin, TX 78758. Mr. Heisch is an advisory programmer in the Processor and System Performance group. He has a BS in Electrical Engineering and an MS in Engineering from the University of Texas at Austin.

Porting DCE Threads Programs to AIX 4.1



By Chary G. Tamirisa

The AIX 4.1 POSIX threads (pthreads) package is based on POSIX.4a Draft 7. Existing applications written to AIX 3.2.4 or 3.2.5 Distributed Computing Environment (DCE) pthreads must be rewritten to work on AIX 4.1. This article provides information to help developers migrate existing AIX DCE pthreads applications to AIX 4.1.

This article assumes that the reader is familiar with POSIX.4a Draft 4 and the AIX DCE pthreads package. The article is written as a set of tables showing the differences between the AIX DCE pthreads and the AIX 4.1 pthreads. Since the reentrant C library is part of the POSIX.4a specification, differences in these interfaces are also presented.

AIX DCE 1.2 pthreads

The DCE pthreads package shipped in AIX DCE 1.2 consists of the POSIX.4a Draft 4 Application Programming Interface (API) and several extensions, shown in Figure 1. It also provides an exception support not specified in POSIX.4a.

The AIX 4.1 threads support (the pthreads and reentrant C library functions based on POSIX.4a Draft 7) introduces the following changes:

- Syntax changes to some APIs in Draft 4
- Return of error codes instead of setting the global `errno` for all pthread functions
- New functions introduced in Draft 7 to enhance thread cancellation and thread scheduling
- Differences in signal handling between Draft 4 and Draft 7
- Specification of cancellation points
- Changes to the reentrant C library API in Draft 7

The pthreads Programming Model

The DCE pthreads provides a set of library functions for parallel programming as well as real-time scheduling. In this model, the programmer identifies sequences of program flow that can be parallelized. These parallel sequences of execution can then be hoisted on the threads.

The pthreads programming model preserves the POSIX.1 functionality, especially Input/Output (I/O), on a per-thread basis. For example, a thread issuing a blocking `read()` only blocks itself. The pthreads programming model is written as an extension to POSIX.4, the standard for Real-Time Extensions to POSIX.1. POSIX.4a preserves the POSIX.4 semantics on a per-thread basis. The pthread model adds the following features to enable parallel and real-time programming:

- Thread management functions
- Synchronization functions
- Scheduling functions to meet real-time requirements
- Thread-specific data functions
- Thread cancellation functions

Base Functionality

POSIX.4a Draft 4 API
Draft 4 signal subsystem
Support for POSIX.1 API
Reentrant C library

Extensions

Non-portable extensions to the pthread API (with suffix `_np`)
Exception returning pthreads API (in addition to error returns)
Exception handling (TRY/CATCH)

Figure 1. Contents of AIX DCE 1.2 pthreads package

Chary G. Tamirisa

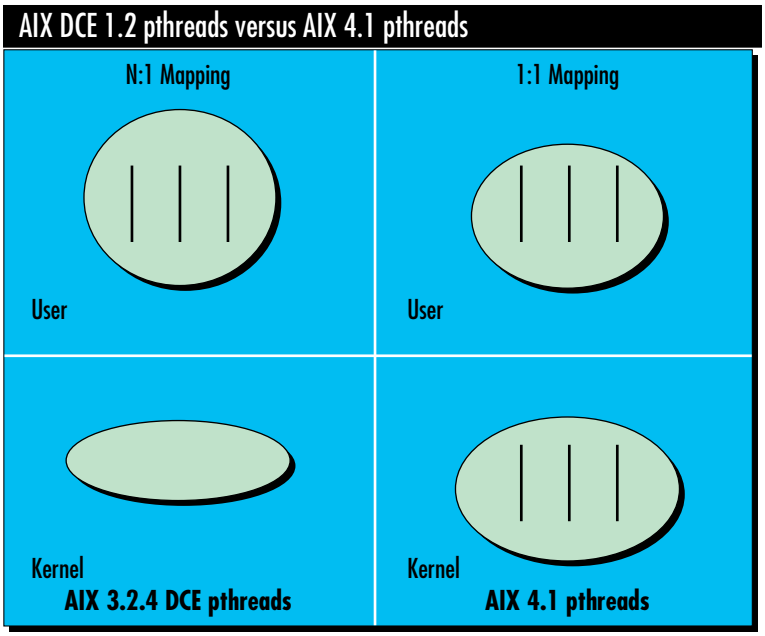


Figure 2. AIX DCE 1.2 pthreads versus AIX 4.1 pthreads

As shown in Figure 2, the AIX DCE pthreads package is purely a user-level threads package; AIX 3.2.4 has no kernel support for threads. AIX 4.1 provides kernel support so that user level threads are supported over these kernel threads. AIX 4.1 provides a 1:1 mapping of user level threads to kernel threads. This 1:1 mapping means that there is a corresponding kernel thread for every user level thread. Currently, AIX 4.1 allows a maximum of 512 simultaneous threads in a user process.

DCE pthreads Package

As shown in Figure 3, DCE pthreads provides the reentrant C library (`libc_r.a`), the pthreads API based on POSIX.4a Draft 4, the signal handling, and the exception handling support. Applications written to DCE pthreads link with `libc_r.a` and `libpthreads.a`. A stanza is provided in `/etc/xlc.cfg` to specify the method of invocation of `cc_r` and `xlc_r`. Multithreaded applications should use these stanzas for compiling threaded programs written to AIX DCE pthreads. Note that special modifications are made to `crt0` for supporting multithreaded processes. These modifications are also specified in the stanzas for `cc_r`.

Porting Issues

This section details the differences between AIX DCE pthreads and the AIX 4.1 pthreads library.

Changes in Error Returns

In AIX DCE, the pthreads functions on error, return a -1, and set the global `errno` variable to the appropriate value. All AIX 4.1 pthreads functions return error number as return values. Applications written to AIX DCE pthreads must be changed to handle AIX 4.1 error returns properly. Figure 4 shows an example.

Data Types

AIX DCE pthreads introduced additional data types, and these data types should be replaced with those specified in POSIX.4a Draft 7, as shown in Figure 5.

Exception Handling

AIX DCE pthreads supports exceptions to handle error conditions and thread cancellations. Exceptions occur in the following situations:

- **When the current thread is canceled**. If a TRY/CATCH block exists around the cancel point (such as a `pthread_join()` call), the appropriate CATCH block is entered. If there is no TRY/CATCH block, the cancelled thread terminates execution.

  POSIX.4a specifies the method of cleanup upon cancellation through two functions: `pthread_cleanup_push()` and `pthread_cleanup_pop()`. The AIX DCE pthreads integrated these two mechanisms so that if a TRY/CATCH and a cleanup handler exist, both are invoked in the Last-In-First-Out (LIFO) order.

- **When the signals `SIGPIPE` or `SIGSYS` are received in the current thread** (unless the signal is set to be ignored). If a TRY/CATCH block exists for handling these exceptions, then the proper block is entered.

- **When the exception raising pthread APIs are used**. These APIs are invoked when a program includes `<pthread_exc.h>` instead of the `<pthread.h>` header file. In `<pthread_exc.h>`, the `pthread_*()` functions are mapped to a corresponding exception-raising API.

AIX 4.1 pthreads does not support exception handling because POSIX.4a does not specify exception handling. Therefore, programs written to handle exceptions must be rewritten to work on AIX 4.1. For cancellation cleanup handling, replace TRY/CATCH with `pthread_cleanup_push()` and `pthread_cleanup_pop()`. Replace all other excep-

tion handling with code that handles error returns from the `pthread_*()` calls.

POSIX.1 Support

All POSIX.1 and C standard functions that suspend the calling process are redefined in POSIX.4a to suspend the calling thread. In AIX DCE this is called *non-blocking I/O support.*

The following functions suspend the entire process: `waitpid()`, `wait()`, `sigsuspend()`, `pause()`, and `tcdrain()`. The DCE pthreads package does not support timed input operations using `VMIN` and `VTIME` (through `ioctl()`).

The AIX DCE pthreads provides support for POSIX.1 I/O functions through a library emulation of non-blocking I/O (there is no kernel support for this in AIX 3.2.4); however, not all functions are supported. For example, no non-blocking support exists for the following: shared memory, System V semaphores, message queue operations, or `poll()`.

This problem is bypassed in AIX 4.1 because all I/O calls are supported to block only the calling thread, not the entire process. Thus, AIX 4.1 now supports all of these functions.

Thread Functions

There are several thread functions described below.

Default attributes for thread: Figure 6 shows a comparison of how a thread is created with default attributes in AIX DCE and AIX 4.1. The figure shows two methods for creating a thread with default attributes in AIX 4.1.

Changes in thread detach: In AIX DCE pthreads, the detach state can be specified by calling the function `pthread_detach()` with the thread ID as the argument. In AIX 4.1, you can specify the thread detach state either at the time of thread creation (by specifying the thread attribute), or after thread creation by using `pthread_detach()`. Programs can use thread attributes to specify the detach attribute or use `pthread_detach()` as appropriate.

There is a difference in the default behavior of detach between AIX DCE pthreads and AIX 4.1 pthreads. In AIX DCE pthreads, a thread can be joined by a call to `pthread_join()` by default; however, in AIX 4.1 pthreads, a thread is created by default in a detached state and cannot be joined as a default. In AIX 4.1, you must set the detach state to `PTHREAD_CREATE_UNDETACHED` by a call to `pthread_attr_setdetachstate()` to

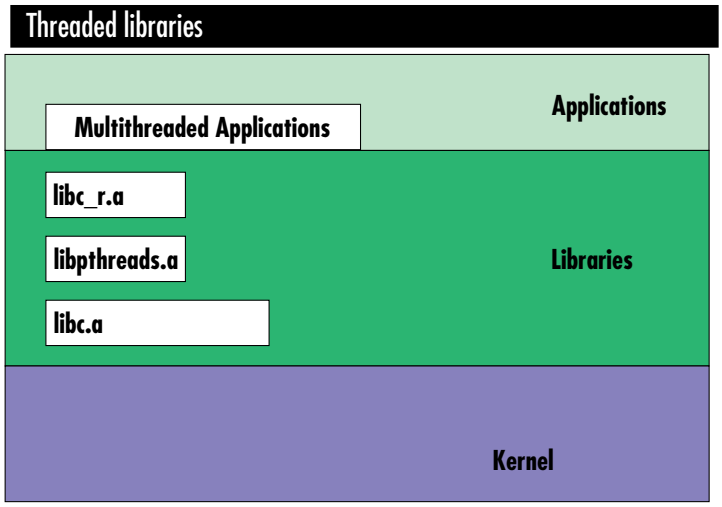


Figure 3. The relationship between threaded libraries

AIX DCE pthreads

```
{int ret;
int ptr;
ret = pthread_join(thread, &ptr);
if (ret== -1) printf("error =%d\n", errno);
}
```

AIX 4.1 pthreads

```
{ int ret;
int ptr;
ret = pthread_join(thread, &ptr);
if(ret) printf("error =%d\n", ret);
}
```

Figure 4. Comparison of error returns in AIX DCE pthreads and AIX 4.1 pthreads

enable you to execute `pthread_join()` on a thread. Figure 7 shows an example.

Dynamic package initialization: The `pthread_once()` initialization symbolic constant is changed from `pthread_once_init` in AIX DCE to `PTHREAD_ONCE_INIT` (lower case to upper case).

Changes in scheduling: Significant changes were made in thread scheduling because of integration of scheduling with the POSIX.4 real-time standard. Specifically, the `sched_param` structure is used in AIX 4.1 to specify scheduling policy and priority. Figure 8 shows the AIX 4.1 structure.

You can use the thread attribute for scheduling to specify scheduling policy and priority. Set `sched_policy` and `sched_priority` in the thread attribute object by specifying a `sched_param` structure with the desired values for policy and priority, and invoke `pthread_setschedparam()`.

AIX DCE pthreads	AIX 4.1 pthreads
pthread_startroutine_t start_routine	void *(*start_routine) (void *)
pthread_addr_t arg	void *arg
pthread_destructor_t destructor	void (*destructor(void *))
pthread_initroutine_t init_routine	void (*init_routine)(void)

Figure 5. Mapping AIX DCE introduced types to AIX 4.1

AIX DCE pthreads

```
{
pthread_t thd;
extern func(int arg);
int arg;
pthread_create( &thd, pthread_attr_default, func, arg);
}
```

AIX 4.1 pthreads Alternative 1 (Recommended)

```
{
pthread_t thd;
extern func(int arg);
int arg;
pthread_create( &thd, NULL, func, arg); /* Note NULL implies default thread attributes */
}
```

AIX 4.1 pthreads Alternative 2

```
{
pthread_t thd;
extern func(int arg);
int arg;
pthread_create( &thd, &pthread_attr_default, func, arg);
}
```

Figure 6. Comparison of thread creation with default attributes

AIX DCE pthreads

```
func(int i)
{
   /* The thread function here.*/
}
main()
{
   int status;
   pthread_t thd;
   pthread_create(&thd, pthread_attr_default, func, 0);
   pthread_join(thd, &status);
}
```

AIX 4.1 pthreads

```
main()
{
   pthread_t thd,
   pthread_attr_t attr;
   /* Initialize the thread attribute and specify the detach state */
   pthread_attr_init(&attr);
   pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_UNDETACHED);
   /* Create a thread that is joinable */
   pthread_create(&thd, &attr, func, 0);
   pthread_join(thd, &status);
}
```

Figure 7. Creation of a joinable thread

In addition, applications that use `pthread_attr_setprio()` or `pthread_attr_getprio()` can be rewritten to use the `sched_param` structure to specify the priority.

In AIX DCE, the function `pthread_setprio()` sets the priority and returns the old priority. A similar functionality can be obtained in AIX 4.1, as shown in Figure 9. Note also that AIX DCE pthreads scheduling symbolic constants are different in AIX 4.1 (see Figure 10).

Differences in Mutex

In AIX DCE, the default value of the mutex attribute is `pthread_mutexattr_default`. In AIX 4.1, you specify this value with `NULL`. AIX 4.1 supports static initialization of mutex variables and has changed the arguments to mutex variable initialization. Figure 11 shows examples.

AIX 4.1 allows static initialization of mutexes, as shown in the following example:

```
pthread_mutex_t mutex =
PTHREAD_MUTEX_INITIALIZER;
```

Figure 12 shows changes in attribute creation and destruction functions.

Figure 13 shows changes in `pthread_mutex_trylock()` return values.

Mutex Types

AIX DCE provides three types of mutexes: fast, non-recursive, and recursive. It also provides `-np` functionality (see Figure 33) to obtain these types of mutexes. The AIX 4.1 `libpthreads` implementation supports only the non-recursive mutex type. Application programs needing other mutex types must provide it themselves.

Differences in Condition Variables

In AIX DCE, the default value is `pthread_condattr_default`; in AIX 4.1, this value is specified with a `NULL` value. AIX 4.1 supports static initialization of condition variables and has changed the arguments to condition variable initialization. Figure 14 shows examples.

Since AIX 4.1 does not support the `{_POSIX_THREAD_PROCESS_SHARED}` options, it also does not support sharing of interprocess condition variables.

Static initialization of condition variables is permitted in AIX 4.1, as shown in the following example:

```
struct sched_param {
     int sched_policy;
     int sched_priority;
}
```

Figure 8. New sched_param structure

```
{
struct sched_param param;
int policy;
pthread_getschedparam(thd, &policy, &param);
/* policy and param structure is filled */
/* Set the new priority */
param.sched_priority = new; /* priority */
pthread_setschedparam(thd, &policy, &param);
}
```

Figure 9. Setting scheduling parameters

AIX DCE pthreads	AIX 4.1 pthreads
SCHED_FG_NP	Use SCHED_OTHER
SCHED_BG_NP	Not Supported
PRI_FIFO_MIN	PTHREAD_PRIO_MIN
PRI_FIFO_MAX	PTHREAD_PRIO_MAX
PRI_RR_MIN	PTHREAD_PRIO_MIN
PRI_RR_MAX	PTHREAD_PRIO_MAX
PRI_FG_MIN_NP	Use PRI_OTHER_MIN
PRI_FG_MAX_NP	Use PRI_OTHER_MAX
PRI_BG_MIN_NP	Not Supported
PRI_BG_MAX_NP	Not Supported
PRI_OTHER_MIN	DEFAULT_PRIO
PRI_OTHER_MAX	DEFAULT_PRIO
PTHREAD_DEFAULT_SCHED	PTHREAD_EXPLICIT_SCHED
PTHREAD_INHERIT_SCHED	PTHREAD_INHERIT_SCHED

Figure 10. Changes to the scheduling-related symbolic constants

AIX DCE pthreads

```
{
pthread_mutex_t mutex;
pthread_mutex_init(&mutex, pthread_mutexattr_default);
}
```

AIX 4.1

```
{
pthread_mutex_t mutex;
pthread_mutex_init(&mutex, NULL);
}
```

Figure 11. Comparison of specifying default mutex attributes

AIX DCE pthreads	AIX 4.1 pthreads
pthread_mutexattr_create() pthread_mutexattr_delete()	pthread_mutexattr_init() pthread_mutexattr_destroy()

Figure 12. Mapping mutex attribute functions

AIX DCE pthreads	AIX 4.1 pthreads
Returns 0 if another thread owns the lock. Returns 1 if the current thread acquires the lock. Returns -1 and sets errno to indicate EINVAL.	Returns EBUSY if another thread owns the lock. Returns 0 if the current thread acquires the lock. Returns EINVAL in case of error in args.

Figure 13. Changes in the return values of pthread_mutex_trylock()

AIX DCE pthreads	AIX 4.1 pthreads

{
pthread_cond_t cond;
pthread_cond_init(&cond, pthread_condattr_default);
}
	{
	pthread_cond_t cond;
	pthread_cond_init(&cond, NULL);
	}

Figure 14. Specification of default attribute of condition variables

AIX DCE pthreads	AIX 4.1 pthreads
pthread_condattr_create() pthread_condattr_delete()	pthread_condattr_init() pthread_condattr_destroy()

Figure 15. Mapping condition variable attribute functions

AIX DCE pthreads	AIX 4.1 pthreads
Returns 0 on success. Returns -1 and sets errno to EAGAIN on timeout. Returns -1 and sets errno to EINVAL on invalid args.	Returns 0 on success. Returns ETIMEDOUT on timeout. Returns EINVAL on invalid args.

Figure 16. Changes in pthread_cond_timedwait() return values

```
pthread_cond_t cond =
     PTHREAD_COND_INITIALIZER;
```

Figure 15 shows the changes in attribute creation and destruction functions.

Changes in `pthread_cond_timedwait()` return values are shown in Figure 16.

Thread Cancellation

A thread has a cancel state and cancel type associated with it. State of cancelability determines whether or not a thread can be cancelled. If the state has been set to enable cancellation, then the type of cancelability will determine when the thread is actually cancelled.

In the AIX DCE pthread function, `pthread_setcancel (int` state) is used to set general cancelability. If a thread's general cancelability is `ON`, then its cancelability is controlled by the state of the async cancelability. If its async cancelability is `ON`, it can be cancelled at any point. In all other cases, the thread can be cancelled only at specific cancellation points, such as condition waits, thread joins, or calls to `pthread_testcancel()`. The default states of cancelability in DCE are general cancelability `ON` and async cancelability `OFF`. This state can be achieved in AIX 4.1, as shown in Figure 17.

AIX DCE-based applications should be rewritten as described in Figure 18 to work on AIX 4.1.

```
pthread_setcancelstate( PTHREAD_CANCEL_ENABLE, &oldstate);
pthread_setcanceltype( PTHREAD_CANCEL_DEFERRED &oldtype)
```

Figure 17. Setting default cancel state and cancel type in AIX 4.1

AIX DCE pthreads	AIX 4.1 pthreads
`pthread_setcancel(CANCEL_ON)`	`pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, &oldstate)`
`pthread_setcancel(CANCEL_OFF)`	`pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, &oldstate)`
`pthread_setasynccancel(CANCEL_ON)`	`pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, &oldtype)`
`pthread_setasynccancel(CANCEL_OFF)`	`pthread_setcanceltype(PTHREAD_CANCEL_DEFERRED, &oldtype)`

Figure 18. Mapping thread cancel functions to set cancel state and cancel type

AIX DCE pthreads	AIX 4.1 pthreads
`void atfork( void *user_state, void (*pre_fork)(), void (*parent_fork)(), void (*child_fork)())`	`int pthread_atfork(void (*prepare)(void), void (*parent)(void), void (*child)(void)`

Figure 19. Atfork handler

Process Primitives

The function that establishes pre- and post-fork handling has been changed in AIX 4.1, as shown in Figure 19. The order of invocation of the pre-fork handlers is now LIFO, while the order of post-fork handlers is First-In-First-Out (FIFO).

In addition, the AIX DCE `atfork()` function raises an exception on error, whereas `pthread_atfork()` returns error numbers.

Signal Subsystem Differences

The following changes were introduced in AIX 4.1:

- **Signal handlers**: AIX DCE pthreads allows two types of signal handlers: a per-thread signal handler for synchronous signals (`SIGILL`, `SIGTRAP`, `SIGIOT`, `SIGEMT`, `SIGFPE`, `SIGBUS`, `SIGSEGV`, `SIGSYS`, and `SIGPIPE`) and per-process signal handlers for all other signals except for `SIGKILL`, `SIGSTOP`, and `SIGVIRT`. These handlers can be installed using the `sigaction()` or `signal()` API.

  All signal handlers in AIX 4.1 are installed on a per-process basis; therefore, programs installing signal handlers must be aware that they may replace existing handlers installed by other threads. You cannot install signal handlers for `SIGKILL`, `SIGSTOP`, `SIGALRM1`, and `SIGWAITING`.

- **Waiting for signals**: In AIX DCE pthreads, `sigwait()` can be executed on all asynchronous signals, excluding the synchronous signals listed above and the signals `SIGKILL`, `SIGSTOP`, and `SIGVIRT`. The signal `SIGVIRT` is used for thread time slicing (scheduling), so it is not available for any applications.

  In AIX 4.1, `sigwait()` can be called for all asynchronous signals except the synchronous signals listed above and the signals `SIGKILL`, `SIGSTOP`, `SIGALRM1`, and `SIGWAITING`.

- **Per-process signal mask**: AIX DCE pthreads allows `sigprocmask()` to be used to block signals for the entire process rather than just the calling thread. In AIX 4.1, the `sigthreadmask()` is used to specify the block mask on a per-thread basis. Use of `sigprocmask()` is unspecified in a multithreaded environment, as per POSIX.4a Draft 7. In AIX 4.1, `sigprocmask()` is mapped to `sigthreadmask()`.

Detailed Differences

Figures 20 through 31 show differences in threads API between AIX DCE and AIX 4.1. Figure 32 shows the remaining pthreads functions in AIX DCE and AIX 4.1 in which the only change is the error-return mechanism. This information can be used when porting existing DCE threads-based applications to AIX 4.1 threads.

AIX DCE provides a set of nonportable extensions to the pthreads functions. These are summarized in Figure 33.

AIX DCE pthreads	AIX 4.1 pthreads
`int pthread_attr_create(pthread_attr_t *attr)`	`int pthread_attr_init( pthread_attr_t *attr)`
`int pthread_attr_delete(pthread_attr_t *attr)`	`int pthread_attr_destroy(pthread_attr_t *attr)`

Figure 20. Mapping pthread attribute functions

AIX DCE pthreads	AIX 4.1 pthreads
None	`int pthread_attr_setdetachstate( pthread_attr_t *attr, int detachstate )`
None	`int pthread_attr_getdetachstate( const pthread_attr_t *attr, int *detachstate )`

Figure 21. New detach state thread attributes in AIX 4.1

AIX DCE pthreads	AIX 4.1 pthreads
`int pthread_attr_setstacksize(pthread_attr_t *attr, long stacksize)`	`int pthread_attr_setstacksize ( pthread_attr_t *attr, size_t stacksize)`
`unsigned long pthread_attr_getstacksize (pthread_attr_t attr)`	`int pthread_attr_getstacksize( const pthread_attr_t *attr, size_t *stacksize)`

Figure 22. Mapping thread stack attribute functions

AIX DCE pthreads	AIX 4.1 pthreads
`int pthread_attr_setprio ( pthread_attr_t *attr, int priority)`	`int pthread_attr_setschedparam( pthread_attr_t *attr, const struct sched_param *param)`
`int pthread_attr_getprio ( pthread_attr_t attr)`	`int pthread_attr_getschedparam ( const pthread_attr_t *attr, struct sched_param *param )`
`int pthread_attr_setsched ( pthread_attr_t *attr, int scheduler)`	`int pthread_attr_setschedpolicy ( pthread_attr_t *attr, int policy)`
`int pthread_attr_getsched ( pthread_attr_t attr)`	`int pthread_attr_getschedpolicy ( const pthread_attr_t *attr, int *policy)`
`int pthread_attr_setinheritsched ( pthread_attr_t *attr, int inherit )`	`int pthread_attr_setinheritsched ( pthread_attr_t *attr, int inherit)`
`int pthread_attr_getinheritsched ( pthread_attr_t attr)`	`int pthread_attr_getinheritsched ( const pthread_attr_t *attr, int *inheritsched)`

Figure 23. Mapping thread scheduling attribute functions

AIX DCE pthreads	AIX 4.1 pthreads
`int pthread_setprio( pthread_t thread, int priority)`	`int pthread_setschedparam ( pthread_t thread, int policy, const struct sched_param *param)`
`int pthread_getprio( pthread_t thread)`	`int pthread_getschedparam ( pthread_t pthread, int *policy, struct sched_param *param)`
`int pthread_setscheduler ( pthread_t thread, int scheduler, int priority)`	`int pthread_setschedparam ( pthread_t thread, int policy, const struct sched_param *param)`
`int pthread_getscheduler ( pthread_t thread)`	`int pthread_getschedparam ( pthread_t pthread, int *policy, struct sched_param *param)`

Figure 24. Mapping thread scheduling functions

AIX DCE pthreads	AIX 4.1 pthreads
`int pthread_create ( pthread_t *thread, pthread_attr_t attr, pthread_startroutine_t func, pthread_addr_t arg)`	`int pthread_create ( pthread_t *thread, const pthread_attr_t *attr, void *(*)(void *), void *arg)`
`int pthread_once (pthread_once_t *once_block, pthread_initroutine_t init)`	`int pthread_once ( pthread_once_t *once_block, void(*)(void)) _`
`int pthread_detach( pthread_t *thread )`	`int pthread_detach( pthread_t thread)`
`int pthread_delay_np( struct timespec *interval )`	None
`int pthread_getunique_np( pthread_t *thread)`	None

Figure 25. Mapping thread functions

AIX DCE pthreads	AIX 4.1 pthreads
`int pthread_keycreate( pthread_key_t *key, pthread_destructor_t destructor)`	`int pthread_key_create ( pthread_key_t *key, void (*destructor)(void *) )`
`int pthread_getspecific( pthread_key_t key, pthread_addr_t *value)`	`void *pthread_getspecific( pthread_key_t key)1`
`int pthread_setspecific( pthread_key_t key, pthread_addr_t value)`	`int pthread_setspecific( pthread_key_t key, const void *value)`

[1]Note that pthread_getspecific() does not return error in AIX 4.1; it just returns NULL in case of error. To differentiate between a NULL thread-specific value from an error situation, call pthread_setspecific() with the same key and a NULL value. Then, check the error return, if any, and pthread_setspecific() will return EINVAL if the key is found to be invalid.

Figure 26. Mapping thread-specific data

AIX DCE pthreads	AIX 4.1 pthreads
`int pthread_mutexattr_create ( pthread_mutexattr_t *attr)`	`int pthread_mutexattr_init( pthread_mutexattr_t *attr)`
`int pthread_mutexattr_delete ( thread_mutexattr_t *attr)`	`int pthread_mutexattr_destroy( pthread_mutexattr_t *attr)`
`int pthread_mutexattr_setkind_np ( pthread_mutexattr_t *attr, int kind)`	None
`int pthread_mutexattr_getkind_np ( pthread_mutexattr_t attr)`	None

Figure 27. Mapping mutex attribute functions

AIX DCE pthreads	AIX 4.1 pthreads
`int pthread_mutex_init( pthread_mutex_t *mutex, pthread_mutexattr_t attr)`	`int pthread_mutex_init ( pthread_mutex_t *mutex, pthread_mutexattr_t *attr)`
`void pthread_lock_global_np()`	None
`void pthread_unlock_global_np()`	None

Figure 28. Mapping mutex functions

AIX DCE pthreads	AIX 4.1 pthreads
`int pthread_condattr_create ( pthread_condattr_t *attr)`	`int pthread_condattr_init( pthread_condattr_t *attr)`
`int pthread_condattr_delete ( pthread_condattr_t *attr)`	`int pthread_condattr_destroy( pthread_condattr_t *attr)`

Figure 29. Mapping condition variable attribute functions

AIX DCE pthreads	AIX 4.1 pthreads
`int pthread_cond_init( pthread_cond_t *cond, pthread_condattr_t attr)`	`int pthread_cond_init( pthread_cond_t *cond, pthread_condattr_t *attr)`
`int pthread_get_expiration_np( struct timespec *delta, struct timespec *abstime)`	None

Figure 30. Mapping condition variable functions

AIX DCE pthreads	AIX 4.1 pthreads
`int pthread_setcancel( int state)`	`int pthread_setcancelstate( int state, int *oldstate)`
`int pthread_setasynccancel( int state)`	`int pthread_setcanceltype( int type, int *oldtype)`

Figure 31. Mapping thread cancellation

```
int pthread_cancel( pthread_t thread)
void pthread_cleanup_push( void (*routine)(void*), void *arg)
pthread_cleanup_pop( int execute)
int pthread_join( pthread_t thread, void **status)
int pthread_mutex_lock( pthread_mutex_t *mutex)
int pthread_mutex_trylock( pthread_mutex_t *mutex)
int pthread_mutex_unlock( pthread_mutex_t *mutex)
int pthread_cond_wait( pthread_cond_t *cond, pthread_mutex_t *mutex)
int pthread_cond_timedwait( pthread_cond_t *cond, pthread_mutex_t *mutex, struct timespec *abstime)
int pthread_cond_broadcast( pthread_cond_t *cond)
int pthread_cond_signal( pthread_cond_t *cond)
int pthread_mutex_destroy( pthread_mutex_t *mutex)
```

Figure 32. Remaining pthreads functions in AIX DCE and AIX 4.1 in which the only change is the error-return mechanism

```
int pthread_signal_to_cancel_np( sigset_t *sigset, pthread_t *thread)
int pthread_delay_np(struct timespec *interval)
int pthread_get_expiration_np(struct timespec *delay, struct timespec *abstime)
void pthread_lock_global_np(void)
void pthread_unlock_global_np(void)
int pthread_mutexattr_setkind_np(pthread_mutexattr_t *attr, int kind)
int pthread_mutexattr_getkind_np(pthread_attr_t attr)
int pthread_cond_signal_int_np(pthread_cond_t *cond)
unsigned int pthread_getunique_np(pthread_t *thread)
int pthread_equal_np(pthread_t thread1, pthread_t thread2)
```

Figure 33. AIX DCE non-portable ("_np") functions

AIX DCE libc_r

```
int localtime_r ( struct tm *result,
   time_t *clock)
int gmtime_r( struct tm *result,
   time_t *clock)
int asctime_r( struct tm *tm, char
   *buffer, int buflen)
char *ctime_r( time_t *clock, char
   *buffer, int buflen)
int rand_r(unsigned int *seed, int *randval)
void utmpname_r(char *newfile, struct utmp_data
   *utmp_data)
int readdir_r( DIR *dirp, struct dirent *result )
```

AIX 4.1 libc_r

```
struct tm* localtime_r (const time_t
   *clock, struct tm *result)
struct tm *gmtime_r( const time_t *clock,
   struct tm *result)
char *asctime_r( const struct tm *tm, char *buf)

char *ctime_r( const time_t *clock, char *buf )

int rand_r(unsigned int *seed)
int utmpname_r(char *newfile, struct
   utmp_data*utmp_data)
int readdir_r( DIR *dirp, struct dirent *entry, struct
   dirent **result)
```

Figure 34. Mapping reentrant C library functions (libc_r)

AIX DCE pthreads

```
int sigwait(sigset_t *set)
int sigprocmask( int how, const
   sigset_t *set, sigset_t *oset)
None
```

AIX 4.1 pthreads

```
int sigwait( const sigset_t *set, int *sig)
int sigthreadmask( int sig, const sigset_t *set, sigset_t *oset)

int pthread_kill( pthread_t thread, int sig)
```

Figure 35. Mapping signal functions

Reentrant C Library Function

Figures 34 and 35 show changes introduced in AIX 4.1. Figure 34 shows the changes made to reentrant C library functions, while figure 35 summarizes the changes made to the pthreads signal-handling subsystem.

Chary G. Tamirisa, IBM Corporation, LAN Systems Division, 11400 Burnet Road, Austin, TX, 78758. Internet: chary@austin.ibm.com. Since January, 1993, Mr. Tamirisa has been the team lead for the threads package on AIX and OS/2® DCE. He has also worked in the fields of communication protocols, system software, and national language support. Mr. Tamirisa holds an MS in Computer Science from McGill University and a BTech in Electrical Engineering from the Indian Institute of Technology in Madras, India.

TPC Ratings for New RISC System/6000s Sizzle

New RS/6000s announced in May of this year have produced industry-leading performance numbers. Results are as follows:

◆ The RISC System/6000 POWERserver™ R24 produced a TPC-A rating of 357.24 tpsA using Oracle® Version 7 (client/server configuration). This is the fastest TPC-A ever reported for a single processor server.

◆ Demonstrating the scalability of RISC System/6000 clusters, a cluster of four POWERserver R24 systems using HACMP/6000 and Oracle Version 7 (client/server configuration) produced a TPC-A rating of 894.08 tpsA.

◆ In the more complex TPC-C benchmark, IBM achieved the fastest Sybase® TPC-C result ever produced. An RS/6000 POWERserver Model 59H using Sybase Version 10.0.1 (client/server configuration) produced a rating of 1122.30 tpmC. ■

The Common Desktop Environment

By Rebecca F. Austen

The Common Desktop Environment (CDE) is a unified graphical user interface for open systems, jointly developed by IBM, Hewlett-Packard® (HP™), Sun®, and Novell®. This article provides an overview of CDE and its components, describes the benefits of CDE to end users and application developers, and explains how CDE will be integrated into AIX.

The desktop, an interactive graphical user environment, has become a natural extension of the computer, as essential to many users as the display, keyboard, or mouse. The desktop enables people to access and share information electronically; it is how we work, study, and communicate with others across networks of computers. Based on industry-standard technology and jointly developed by industry leaders IBM, HP, Sun, and Novell, the Common Desktop Environment (CDE) is a rich and intuitive user interface for open systems. Designed for enterprise computing, CDE unifies the scales across a variety of UNIX platforms and provides a complete desktop appealing to a wide range of users from the novice to the expert.

Target Audiences

The Common Desktop Environment addresses three key audiences: end users, system administrators, and application developers. The following sections discuss its appeal to each group.

End Users

A visually appealing and highly customizable interface, CDE is innovative, consistent, and easy to use. It ensures end users transparent access to data and applications across a network, and supports both stand-alone workstations and distributed, heterogeneous environments. Optimized to take advantage of client/server configurations, CDE enables users to run shared applications and manipulate data at remote servers, even if they do not know the names or locations of these systems. Because CDE provides a common "look and feel" and behavior for different platforms, users familiar with CDE can easily move from one platform to another. To further assist users, CDE is equipped with extensive online help as well as hardcopy manuals for more detailed explanations of the various tools and features.

Rebecca F. Austen

System Administrators

For system administrators, CDE presents an integrated approach to accessing applications, either local or on remote hosts. System administrators can configure the desktop to enable users to access their data and applications without knowing complex commands or host names. Installation of CDE is simple for administrators; most of the setup is handled by tools packaged with the desktop or through ASCII configuration files that are stored for future editing.

Both the runtime and development environments have a well-defined packaging scheme. Although most file names, locations, and formats are the same regardless of the operating system, there can be differences in the actual installation of the desktop on each platform. CDE is designed for distributed computing; users and applications can rely on location transparency for display services, execution, data, and sessions. Configuration files handle the definition and location of session and application servers, and support both standard network filesystems and distributed filesystems. CDE allows for varying points of administrative control—either centralized or distributed.

Tunable and scalable to the needs of the user community, CDE allows administrators to leverage existing hardware and software investments. All systems can participate as application servers, even those that do not have CDE installed. While the initial installation and setup of the desktop

can be simple, administrators deploying CDE in a broad network with many users should have some knowledge of network configuration procedures.

Application Developers

CDE provides a broad and comprehensive set of standardized Application Programming Interfaces (APIs), formats, and policies for developers. The CDE toolkit facilitates development of applications that are seamlessly integrated with the desktop and with one another. Programmers enjoy a rapid development cycle, which enables them to build on the core services provided by CDE and focus their efforts on their applications. The suite of development tools and services increases programmer productivity; that is, they can use tools such as the Graphical User Interface (GUI) builder instead of programming to lower-level APIs.

There is a high degree of application portability across multiple platforms. Since CDE is implemented from a common source base, it is compatible across different systems. CDE fully supports the installed base of applications; existing X Windows, OSF/Motif®, and OpenLook® applications are binary compatible and should run in CDE without modifications.

The COSE Process

The Common Desktop Environment has been developed under the Common Open Software Environment (COSE) process that was introduced in March 1993 to accelerate the development of industry standards in several core technology areas. This process promotes rapid delivery of specifications, while preserving customer choice of products across a range of platforms. CDE, based on the established standard X Window System® (X11) Release 5 and OSF/Motif 1.2, incorporates proven desktop technology from IBM, HP, Sun, and Novell. These four companies are jointly developing a sample implementation of CDE from a single, shared source base; their combined efforts will result in nearly identical desktops on all the supported platforms, including AIX 4.1. (Some differences may exist in operating system-specific areas such as installation and packaging, or value-added tools.)

The CDE technology can also be licensed to other vendors. An open process is used for standardizing the CDE specifications. Through its desktop working group, X/Open is soliciting industry feedback and establishing the branding and certification process for the standard. The result is an open desktop with broad industry acceptance and choice of vendors for the implementation.

A snapshot of the CDE software and documentation was released on CD-ROM for wide public distribution at the CDE Developer's Conference in October 1993; the snapshot was updated and redistributed in April 1994. By providing early access to the technology, this preview enabled end users and developers to begin porting their applications to CDE and to provide direct feedback to the engineering teams before the completion of the joint sample implementation.

Desktop Overview

IBM is introducing a subset of the CDE technology in AIX 4.1. This desktop, based on the content of the April 1994 snapshot, represents a derivative of the CDE joint sample implementation, targeted primarily at end users. It fully replaces the AIX desktop offered in previous versions of AIXwindows/6000; migration tools will be provided in a subsequent release for users who make the transition from previous versions of AIX to the new desktop.

CDE incorporates proven desktop technology from IBM, HP, Sun, and Novell.

Although most features of CDE are available in AIX 4.1, tools such as mail and calendar are offered only as sample programs in the initial release. There is currently limited support for application developers. Subsequent releases of AIX will incorporate the remaining CDE features, including the complete development environment and deployment in multiple national languages. Once the joint sample implementation of CDE is complete, IBM will integrate CDE fully into AIX.

Getting Started

The AIX 4.1 desktop, packaged with the operating system, is the default environment at system initialization (assuming a graphics adapter is present). At startup, the user is presented with a graphical login screen. Upon entering a valid user name and password (optional), the user reaches the default desktop, shown in Figure 1.

Users can begin using the desktop and applications immediately, or customize the desktop for a more personal configuration. Many familiar AIX tools, such as the System Management Interface Tool (SMIT), Visual System Management (VSM), and InfoExplorer, can be launched directly from the desktop. Other products, including third-party applications, are easily deployed on the desktop by using the Create Action tool, which defines an icon and execution instructions for launching the program from the desktop. After invoking Create

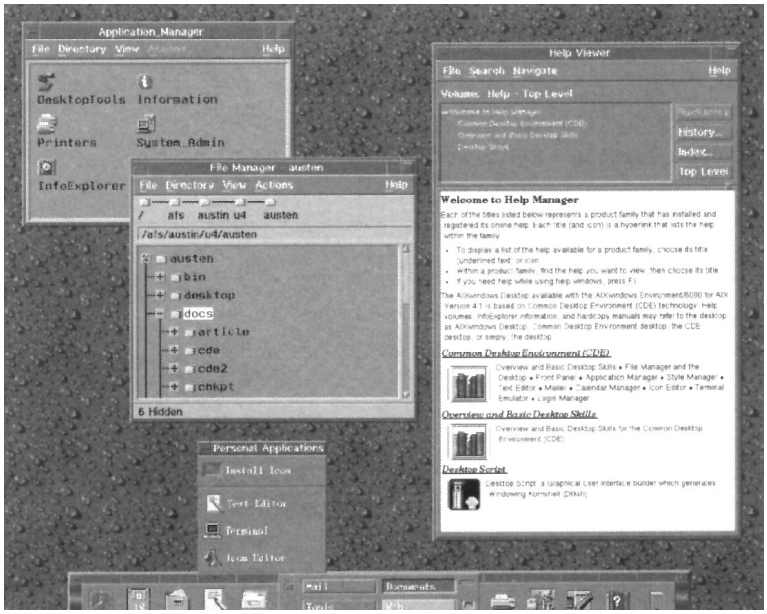


Figure 1. Default desktop

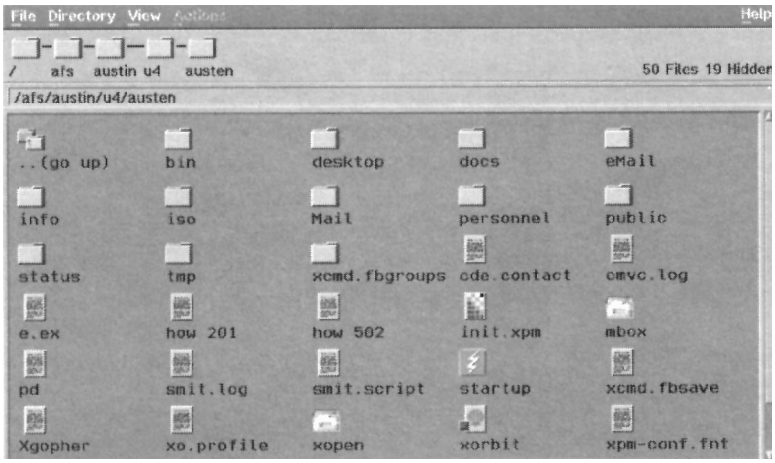


Figure 2. Desktop file manager

Action, the user completes a simple dialog, and an icon is created in the user's home directory (viewed from the desktop file manager, shown in Figure 2).

The user double-clicks on the icon to invoke the application. This icon can also be dragged and dropped into the Personal Applications slide-up on the front panel (see Figure 3). This provides quick access to frequently used programs.

This procedure—the combination of creating an action icon and accessing it through either double-clicking or the slide-up—is the simplest way to begin running applications on the desktop. Creating actions to launch applications can be performed by the end user or preconfigured by the system administrator. To learn more about the desktop, new users are encouraged to navigate through the online help system (Figure 4) or explore the tools in the front panel or in the application manager.

To run traditional command-driven programs, users have two options. They can either launch the terminal emulator or use the desktop tools to easily create a graphical front end.

Desktop Features

The broad scope of the desktop encompasses core services as well as productivity tools and applications. The base support covers areas such as window management, file management, customization, and online help. Advanced programming services for inter-application communication include messaging, drag and drop, data interchange, and session and workspace management. CDE also offers a complete development environment, including libraries, custom widgets, header files, and application building tools.

Not all of these capabilities, particularly the development environment, will be available in the initial AIX 4.1 desktop release, but additional features will be included in subsequent releases once the CDE specification is complete.

Window Management/Front Panel

The window manager and front panel control access workspaces, applications, devices, and frequently used objects. The window manager, based on OSF/Motif 1.2 standards, incorporates extensions to support multiple workspaces (additional areas of screen space). Figure 5 shows the front panel with the workspace selectors in the center and slide-up subpanels that provide access to additional selections.

There are also convenient screen-lock and logout icons. Simple pop-up menus enable the front panel to be customized: users can add, delete, or rename workspaces; create subpanels; and create or delete subpanel entries. By modifying additional configuration files, the front panel can be significantly altered to support specific installation needs, such as changing the front panel dimensions or screen placement, replacing controls, and so on. Although this level of modification is not recommended for the average user, it can be useful when deploying CDE for users

who have different levels of experience or need access to a predefined set of applications.

File Manager

The file manager (Figure 2) is used to browse and manipulate objects, folders, and directories. It includes the following actions: browsing files and directories; creating, moving, copying, and deleting objects; and changing the properties of objects. Most of these actions can be performed either via direct manipulation (drag and drop) or through the menu interface. For example, a file can be deleted by dragging and dropping it to the trash can in the front panel, or by selecting the object and choosing "delete" from the menu.

Style Manager

The desktop and its applications are customizable through the style manager, which enables users to change color palettes, backdrops, mouse and keyboard settings, window behavior, screensavers, and session support.

Users can create their own colors and backdrops or use the defaults supplied with the system. System administrators can also preconfigure the desktop and make these settings available to other users.

Online Help

The desktop provides an interactive online help system that includes browsing and hypertext support for information based on SGML.

The help manager includes an API so that applications can present their own context-sensitive help windows. The author creates the help information in a tagged format. During the application integration process, the application help volumes are registered with the help manager to ensure that keywords can be searched in the central index and that application-specific help information is displayed properly. This eliminates the need for application developers to create a help system. It also ensures that online information is displayed in a consistent format.

Programming Services

The desktop provides application integration services, including application launching, data exchange, drag-and-drop, and cut-and-paste conventions. Object behavior is defined through actions and file type definitions that are generated through the Create Action tool or that can be manually entered in editable ASCII files in a simple

Figure 3. Personal Applications slide-up

configuration language. Object behavior is keyed off the file name, content, or other attributes.

For example, all files with the extension `.ps` (PostScript®) can be defined to automatically launch a PostScript browser when double-clicked (default action), or be sent to the PostScript printer when the Print action is selected. Alternatively, the behavior can be defined by the file's content or format rather than by name. By setting up actions and file types within CDE, the user obtains an intuitive, object-oriented, or data-centered desktop. Objects have predictable behavior when they are manipulated, and applications can be launched naturally through the data types they support. For example, the user can double-click on a spreadsheet data file to launch the spreadsheet program rather than launching the program before opening the data file.

Network services within CDE include the ToolTalk® messaging facility for communication among applications and a remote execution capability that ensures transparent access to network applications. The messaging service handles notification of events and data transfer. All applications should be able to receive and respond to basic desktop messages, and many applications will send messages as well. This enables seamless integration of applications with the desktop and with other programs.

Drag and drop, pervasive in the desktop, is supported through convenience functions and extensions to the OSF/Motif 1.2 drag-and-drop

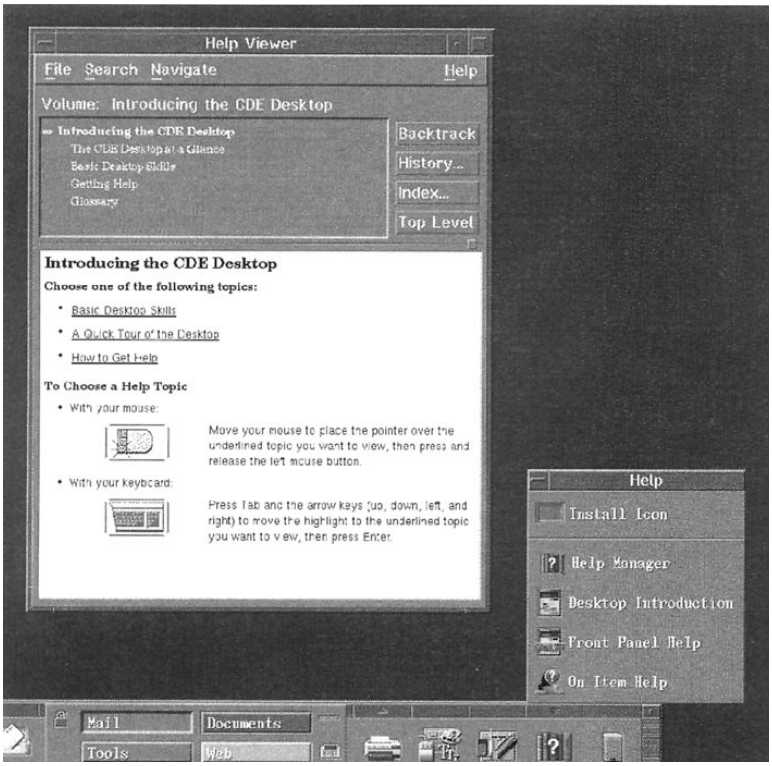


Figure 4. Online help system

Figure 5. Front panel

interface. Once an object registers itself as a drop site, it is notified of events via callbacks. It can then respond with appropriate visual feedback, data transfer, and behavior.

The session manager is based on standard ICCCM conventions for saving and restoring application state; the application is notified when the session is ending so that it can preserve its current state, and then restore it when the session restarts. With the session manager, users can restart a previously exited desktop session and resume working as if they had never left.

The workspace manager ensures that applications behave properly when the user moves from one workspace to another. Default behavior is handled by the workspace manager; that is, the application remains in its defined workspace when the user moves to another space. Applications can control their own behavior across multiple workspaces if necessary, although it is not recommended since it may interrupt the user unexpectedly. Using the workspace manager can be especially useful, for example, if an application needs to post a warning message in the current workspace to get the user's immediate attention.

Productivity Tools

The desktop offers a set of graphical tools for displaying and editing data and collaborating with other users. Figure 6 shows some of these tools: a text editor, an icon editor, and a calendar tool. The mail tool is not pictured.

These applications are tightly integrated with one another and with the desktop services, so that information can be shared and transferred among them. For example, a user can use the editor to compose a mail message (including a meeting notice), use the mail tool to send the message to another user, then drag the embedded appointment and drop it into the calendar for scheduling. A print tool, which acts as a drop site for objects to be printed, displays the status of queues. Additional tools come with the desktop, such as a graphical calculator, a front panel clock, and screensaver support.

Application Development Tools

The desktop provides a dialog and scripting service (based on the windowing Korn Shell technology) and a basic application builder; both support rapid prototyping and development of graphical interfaces for the desktop without requiring advanced programming skills. IBM also offers Desktop Script, a graphical front end to the scripting service, which enables the user to create a graphical interface using drag-and-drop and then automatically generates the script.

Desktop Script is an easy way to create highly portable user interfaces without programming; it is also a convenient tool to use if the developer's environment is not available or installed.

IBM's AIXwindows Interface Composer (AIC) is an advanced graphical user interface builder currently offered as a separate product for AIX. Future versions of AIC will incorporate the CDE style and interfaces.

The User Model

The Common Desktop Environment is based on a consistent user model for interaction, behavior, and application design. It centers on an object-oriented, direct manipulation approach, and its "look and feel" is based on the OSF/Motif 1.2 style guide, with extensions to support OpenLook compatibility and convergence with IBM's Common User Access™ (CUA™) standards. The

desktop has its own style guide, including a checklist to verify that applications comply with the user model.

Internationalization

The desktop is fully internationalized with industry-standard interfaces (XPG4, POSIX, X11R5, and so on). The implementation maintains codeset independence and enables languages to be selected at runtime.

Summary

The Common Desktop Environment brings significant unification to open systems. By presenting a consistent GUI across multiple operating platforms, users have the power of a desktop that was designed for distributed computing from several vendors. The programming interfaces for CDE are fully documented and standardized, promoting a robust and portable environment for application developers.

IBM is introducing an early version of the CDE technology into AIX 4.1 and will follow in subsequent releases with more advanced features and conformance to the CDE joint sample implementation and specification. By offering the desktop with the operating system, CDE will become the pervasive graphical environment for AIX users.

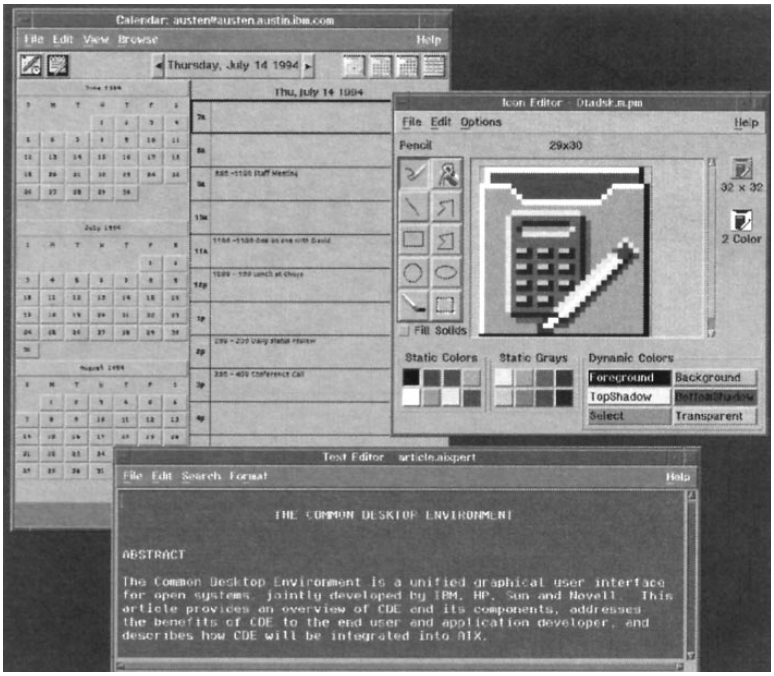


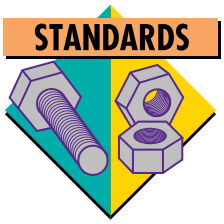
Figure 6. Graphical tools for displaying and editing data

Rebecca F. Austen, IBM Corporation, 11400 Burnet Road, Austin, TX 78758 Internet: austen@austin.ibm.com. Ms. Austen is a senior manager responsible for systems management and user interface development for AIX. She joined IBM in 1984 as a programmer and has been involved in the architecture, development, and management of software products for OS/2 and AIX, in the areas of graphical user interfaces, distributed computing, and systems management. She has a BA in Mathematics from Vassar College.

EVIS Electronic Service is Being Moved to World Wide Web

Effective August 19,1994, the Electronic Vendor Information System (EVIS) will be discontinued. All information on that system will be moved to the IBM Internet Worldwide Web Service.

To access Worldwide Web, you need full Internet service and a browser such as Mosaic. Mosaic is a multimedia web browser, available at no charge from the National Center for Supercomputing Applications's FTP service, ftp.ncsa.uiuc.edu. The IBM Home Page is at URL http://www.ibm.com. The RISC System/6000 Home Page is URL http://www.austin.ibm.com. *AIXpert* information can be found at URL: http://www.austin.ibm.com/services/vendors/tech/aixpert/aixpert.html ■



AIX 4.1 Now Implements XPG4

By Mark S. Brown

One major change in AIX Version 4.1 is the move to conform to X/Open's Portability Guide, Issue 4, popularly known as XPG4. This standard and its upcoming revision, currently known as SPEC 1170, consolidate the UNIX variations offered today. XPG4 and SPEC 1170 are the standards used in AIX to ensure application portability in the open systems market.

X/Open is a consortium composed of most leading UNIX systems suppliers, software vendors, and user groups. Unlike other open standards organizations such as the Institute of Electrical and Electronics Engineers (IEEE) or American National Standards Institute (ANSI®) that only produce standards guidelines, X/Open also provides certification and testing. X/Open test suites are used to certify conformance to these standards. Representatives from the X/Open members form working groups to produce open specifications for operating system interfaces. The X/Open body then votes on whether to accept these specifications.

The *Portability Guides* are collections of X/Open-approved specifications for a given product. Issue 3 (also known as XPG3) contains standards for a set of basic system interfaces and commands known as the *Base*—commonly used programming languages, curses, terminal interfaces, and international language-support features.

X/Open prefers to build specifications based on industry standards already in place, and then expand on those standards (without contradicting them) to meet the needs of its members and the market.

Although XPG3 was based primarily on the System V Interface Definition (SVID)—the only publicly distributed UNIX operating system specification available at the time the *X/Open Portability Guides* were being developed—XPG3 was still considered too proprietary by some. At the time, IEEE POSIX 1003.1 was being developed, and ANSI was finalizing the C language standard. As these standards became accepted, X/Open realized the need to update XPG. The membership also believed that although POSIX and ANSI C were valuable, the two standards were not useful to application developers because they left too much unspecified or undefined, including some functions that UNIX developers relied on daily.

XPG4 and SPEC 1170

XPG4 was developed to address these issues. It is based on POSIX 1003.1 and the new 1003.2 (shell and utilities) standards, with additions from SVID Issue 3 and some invention by the working group to complete gaps in the other standards. It is a superset standard meant to provide a useful set of interfaces available on all certified operating systems. If vendors adhere to XPG4, application developers can develop their products, including previously used interfaces, without fearing the "slight difference" in code portability that always seems to occur between two vendor platforms. For this reason alone, IBM considers XPG4 an important standard for AIX 4.1 to meet.

Users and developers, however, believe that even XPG4 is incomplete, leaving too many uncertainties in UNIX system portability. This belief has led five of the leading UNIX system vendors, including IBM, to sponsor an updated and expanded specification, currently known as SPEC 1170, that will be even more comprehensive. It is hoped that X/Open will adopt SPEC 1170 as the next version of XPG and make it a requirement for using the UNIX trademark. As one of the sponsors of SPEC 1170, IBM intends to develop AIX to meet this specification for easier application portability to and from AIX.

Mark S. Brown

XPG4 Components

Although XPG4 is composed of several different standards, the most important is the Base Specification, which consists of three documents: *System Interfaces and Headers*, *Shell and Utilities*, and *Base System Definitions*. A companion volume, *XPG3-XPG4 Base Migration Guide*, lists the differences between XPG3 and XPG4 for developers who are moving applications. For vendors who wish to certify their systems as conforming, a system conformance profile book describes conforming system configurations.

Currently, there are two different levels of certification for XPG4. Since the commands specification was based on the new 1003.2 standard, X/Open has not yet offered a certification suite for it. Therefore, systems branded "XPG4" are only required to have XPG3 commands. Once the test suite is available, systems will be required to re-brand to XPG4 command level. AIX 4.1 commands and interfaces, already enhanced to meet the XPG4 level, will be certified for commands when the test suite is available.

These standards are modeled on the 1003.1 and 1003.2 standards, even though they are a proper superset. The third standard avoids duplicating certain material that would otherwise have been described in both standards.

System Interfaces and Headers

Because IBM has already implemented changes in Version 3 of AIX to meet XPG4, AIX 4.1 interfaces covered by the XPG4 will look similar to Version 3.2.4 or 3.2.5 interfaces today. In fact, none of the changes required to meet certification will break binary portability. Application sources will require only minor changes to meet the new function prototyping or minor header file modifications. One exception is the curses library discussed later in this article.

Most XPG4 work was "under the covers" or in the development of new interfaces, such as the new regular expression interfaces required by 1003.2. There are new interfaces in AIX 4.1 for shell globbing and parsing, allowing applications to parse input in a similar way to shells. Developers whose applications accept regular or extended regular expressions are encouraged to evaluate these interfaces for their reliability and portability. The portion of the work in international locale support was greatly revised and expanded by 1003.2 and XPG4. Locale source files can now provide much cultural variation, such as multilevel sorting ability.

XPG4 will make locale source files more portable between locales and between platforms. However, locale description file binaries must be remade from the locale source file when moved from platform to platform. A few new interfaces in this area cover date and time functions and character I/O in a more portable manner. The `iconv` converter utility has also been enhanced to be more useful.

The Base specifies several traditional interfaces not found in the POSIX or ANSI standards, such as shared memory and encrypt/decrypt. The intent is to codify what users and developers expect to see in a UNIX-based system.

Shell and Utilities

Users will see the largest difference resulting from the move to XPG4, which is a direct result of XPG4 and compliance with 1003.2. The Shell and Utilities volume codifies the behavior of the shell and standardizes the behavior of over 100 of the most commonly used system commands—from `at` to `yacc`. While much of the groundwork for these changes was laid in AIX 3.2, more was needed to truly meet the specifications.

A common set of commands means that users do not have to relearn commands when switching platforms.

The default AIX XPG4 shell is similar to the Korn Shell (`ksh`), with some modifications in script parsing, regular expression handling, and a few other areas. Dave Korn has developed a new version of `ksh` that conforms to XPG4; IBM intends to support this new version for AIX. (For AIX 4.1, IBM modified the old version of `ksh` to conform to XPG4.) This version specifies several shell utilities in more detail than users have seen in any system man page.

Utilities listed in XPG4 must follow a set of utility syntax guidelines that describes how options and option arguments are to be handled. A common set of commands means that users do not have to relearn commands when switching platforms. Certain command input and output specifications differ from AIX 3.2, causing a potential for user shell scripts to break. To avoid this problem, the old syntax was left intact whenever possible. In fact, many commands with changed syntax completely support the older syntax.

Grammars are provided for the shell, `lex`, `awk`, and `yacc`, making the output of these utilities more easily predicted. Exit codes and `stderr` reporting formats are also now more predictable. All XPG4 commands use `getopt()`, a change that was expanded in AIX to include several non-XPG commands. The new backup utility `pax` has also been extended.

Users of symbolic links will see standardization in how commands parse symbolic links and how interfaces expand pathnames containing symbolic links. The addition of the `lstat()` system call gives status such as the `stat()` interface, with added information for `symlink` files. There is now a consistent behavior based on the Berkeley Software Distribution (BSD) style of symlinks applicable to all commands that may need to handle a symbolic link reference in a special manner (such as `rm`, `find`, or `ls`).

Most changes required by the XPG commands specification are not actually changes from previous standards, but an attempt to codify and make portable behavior that has never been documented explicitly. Users who have argued that AIX behaved in a non-standard manner will be greatly pleased by the changes in Version 4.1, most of them based on this volume alone. In addition, since Sun, Novell/USL, DEC®, IBM, HP, Bull®, Siemens™, and many other major system vendors adhere to or plan to meet this specification, portable scripting between systems will be even more practical than ever.

SPEC 1170 was intended to answer the market demand to make the UNIX environment more open and portable through further specification.

SPEC 1170 and the Future

The next move for AIX is to meet the emerging SPEC 1170 specification, named for its 1,170 interfaces at the time of initial industry review. SPEC 1170 was intended to answer the market demand to make the UNIX environment more open and portable through further specification. SPEC 1170 supersets XPG4 by adding support for Streams, sockets, XTI, an enhanced version of curses for the international community, and many additional SVID3 functions—including signals interfaces, memory mapping, `password/group/utmp` file handling, TCP/IP, and many more. Many of the specifications of SPEC 1170 have already been met in AIX or have been added in Version 4.1. For instance, AIX 4.1 currently supports Streams, XTI, sockets, and most C library functions, with curses on the way. However, since SPEC 1170 was still undergoing review and revision as IBM prepared to ship AIX 4.1, IBM expects to make more changes in future updates of AIX so that it will conform with the final SPEC 1170.

System V Curses

Another big standards change in AIX 4.1 is in the curses library `libcurses.a`. This change is IBM's response to the many requests from users and developers to update its libcurses/terminal interfaces package. IBM could not simply follow SPEC 1170 requirements to meet this request, since SPEC 1170 was still undergoing major revisions in the curses area during the development period of AIX 4.1. Instead, IBM chose System V R3/R4 curses because this source contained most of the changes requested by customers. In addition, using the latest System V sources would make it easier to modify AIX 4.1 to meet the full SPEC 1170 requirements when they became final.

Because several major structural elements and data definitions in these new curses are different from the AIX Version 3 curses, application programs may need modification when recompiled on AIX Version 4. Only recompiles are mentioned, since AIX Version 4 maintains full binary compatibility with earlier releases using shared libraries and compatibility install options provided with the base system package.

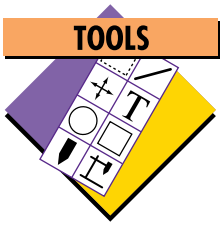
When the version of curses that conforms to SPEC 1170 is available, it will be fully backward compatible with AIX 4.1. It will also contain new functions, such as integrated internationalization support and color. Other benefits will include more application portability, more standard curses, and `terminfo` tools to make portable configuration and installation procedures easier for users and administrators.

Summary

Moving to XPG4 is a major step for AIX, providing many benefits to users and developers seeking cross-platform portability and an industry-standard UNIX operating system on the RISC System/6000. Although AIX will soon conform to SPEC 1170, it will not leave current users behind.

Mark S. Brown, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Internet: mbrown@austin.ibm.com. Mr. Brown is a senior programmer with the AIX Base OS Development group, working in AIX BOS development since 1988. He holds a BS in Computer Science from Southwest Texas State University in San Marcos.

The Next Step for Visual System Management



By Kenneth R. Banning

This article provides an overview of the new features of Visual System Management (VSM) in AIX 4.1. Key among these changes is the addition of VSM applications to support software installation, configuration, and maintenance.

The Visual System Management (VSM) runtime option of AIXwindows was introduced in AIX Version 3.2.5 and described in *AIXpert*[1].

Several enhancements and additions have been made to VSM in AIX 4.1 to improve its usability and extend it into software installation, configuration, and maintenance. In addition, VSM can now support the AIX 4.1 desktop.

Applications for Installation and Configuration

The primary change to VSM was the addition of applications to support software installation and the initial configuration of RISC System/6000s. The task of supporting installation was divided into three applications: Install Assistant, Install and Update Software Manager, and Maintain Installed Software Manager.

Install Assistant

Install Assistant is a unique application of the Help Manager, which can present hypertext-linked online information with graphics and provides links to applications. This technology, derived from Common Operating System Environment (COSE), was first introduced with VSM and now enables Install Assistant to use VSM applications. Install Assistant is displayed upon startup and walks users through all tasks needed to install options and configure the system for use. Install Assistant also details the steps that should be performed by users to ensure that their systems contain all the software needed to complete their configurations. Figure 1 shows a sample Install Assistant screen.

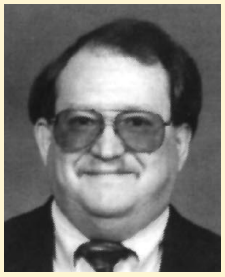
To make changes to the system configuration during installation, the Install Assistant can choose either of two paths. The first option, through the information path ("i" icon before each task in the task list), explains why each task should be executed and then provides step-by-step instructions. The second path is a fast channel that leads directly to the application (the second icon preceding each task). Using this path, the application related to the selected topic can be started and displayed beside the Install Assistant. This second path enables the user to simultaneously read and execute instructions step-by-step. Figure 2 shows an example.

The Install Assistant can also be used to change the system after the initial installation. When making changes, the application provides task descriptions with detailed information that helps users to understand why and how to make the changes.

Install and Update Software Manager

The Install and Update Software Manager provides users with a VSM application to execute the `installp` command. Figure 3 shows the Install and Update Software Manager primary window. The Install and Update Software Manager can perform many tasks.

AIX 4.1 uses bundles to simplify the install task. *Bundles* are collections of software filesets



Kenneth R. Banning

[1] Gibson, Georgia A., "AIX Visual Systems Management: A User's Perspective," *AIXpert* (November 1993).

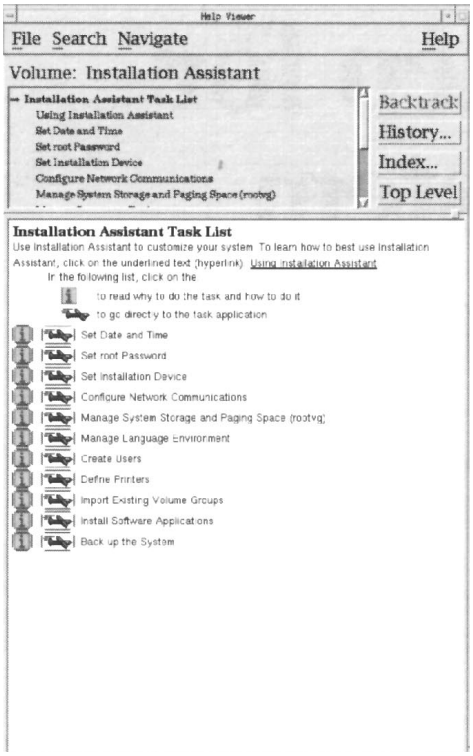


Figure 1. Install Assistant

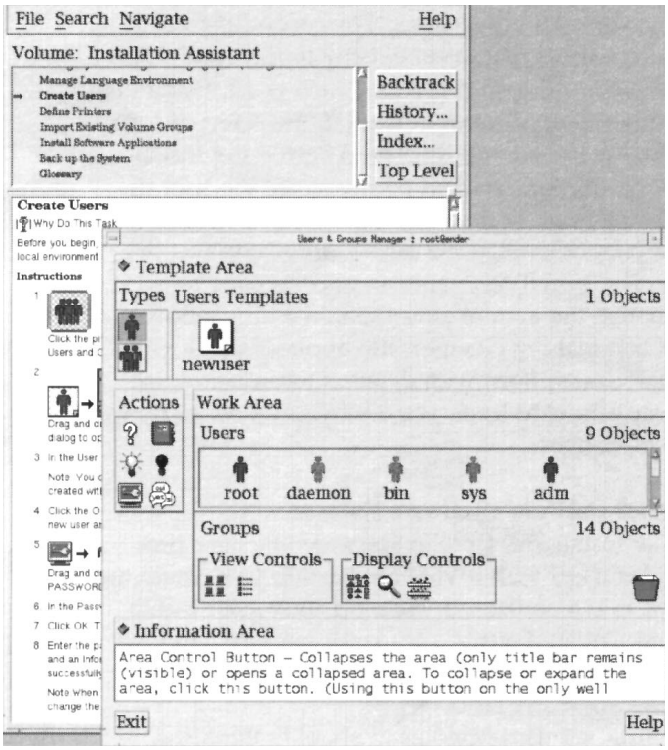


Figure 2. Install Assistant with application

or applications that would be required in a system to perform specific types of jobs. Bundles are given different names, depending on what they include. For example, software for client bundle would refer to the desktop and client versions of common applications, while the server bundle would include the server versions of software in addition to everything in the client bundle. Applications such as spreadsheets and editors would be considered a part of the personal productivity bundle of software. The Install and Update Manager helps users select the correct install device and bundle for installation.

Once users select the install device, the media is read to determine what is available on that media. The list of software available on the media is displayed in the work area. When a user selects a bundle, the Install and Update Manager selects all the software titles in the bundle that are also on the install media. This method enables users to quickly select the software they want to install. Users can select or deselect additional software titles, or specify any of the following actions for the selected titles:

- Install the selected software
- Schedule the installation to occur later
- Preview the selected software for size
- Save the selected objects as a new bundle

Hierarchical information (filesets, files, fixes, and so on) is presented to users in a tree view. Users can modify the list; they can expand or contract the tree as needed, or specify that only special sets of information—such as fileset, maintenance level, and fixes—be displayed for each object.

Users can also specify any of the following actions for the installation task.

- Show the following install settings:
  - Apply and commit
  - Install requisite software
  - Extend the filesystem if needed
  - Include language packages
- Show or change the scheduled activities
- Delete the bundle or object

Maintain Installed Software Manager

The Maintain Installed Software Manager manages software that has already been installed. With this application, users can select objects for the product list and specify any of the following actions: commit the selected software, reject the selected

software, or verify that the selected software is complete.

Again, as with the Install and Update Software Manager, users can customize the hierarchical information presented in the Maintain Installed Software Manager. In addition, they can also find specific objects by name and delete software (de-install). Figure 4 shows the Maintain Installed Software Manager primary window.

Changes to VSM Applications

The primary change to VSM visuals in AIX 4.1 is the introduction of contextual (pop-up) menus for all objects and areas. These menus enable users to view the entire range of actions that can be performed on a specific object with a click of the mouse. The VSM areas—template, action, work, and information—also have contextual menus that list the available actions in each area. These menus may include display controls, view controls, object selection, or object creation.

In addition, the shared library has been modified to improve performance. Other minor changes ensure continued conformance to OSF/Motif. These changes provide a path for future changes in Motif®, in anticipation of the forthcoming Common Desktop Environment from COSE.

Changes to Storage Manager

Storage Manager now has a more granular backup and restore action, which enables users to specify backups at the directory level and then choose to restore at either the directory level or the file level. This new version of Storage Manager still maintains support to the filesystem and volume group backup and restore.

Changes to Device Manager

Two changes were made to the Device Manager in AIX 4.1. First, the System Support Components pane was removed from the work area because the operating system will no longer support virtual terminals. The console control has been placed in a window button action and provides the same function as in AIX 3.2.5.

The second change in Device Manager resulted from the packaging changes in AIX 4.1. AIX 4.1 will no longer automatically install all device drivers on the install media; device drivers on the install media will be installed only if the system detects a need for the specific driver. The immediate effect on Device Manager is that only installed device drivers can appear in the template area, which reduces the number of templates available. If no templates for a specific class are installed, the template type button is grayed and unavailable. Also, if the Hardware Diagnostics bundle is not installed, the command to support the Create Report action may not be valid, as shown by the grayed (unavailable) state of the action icon.

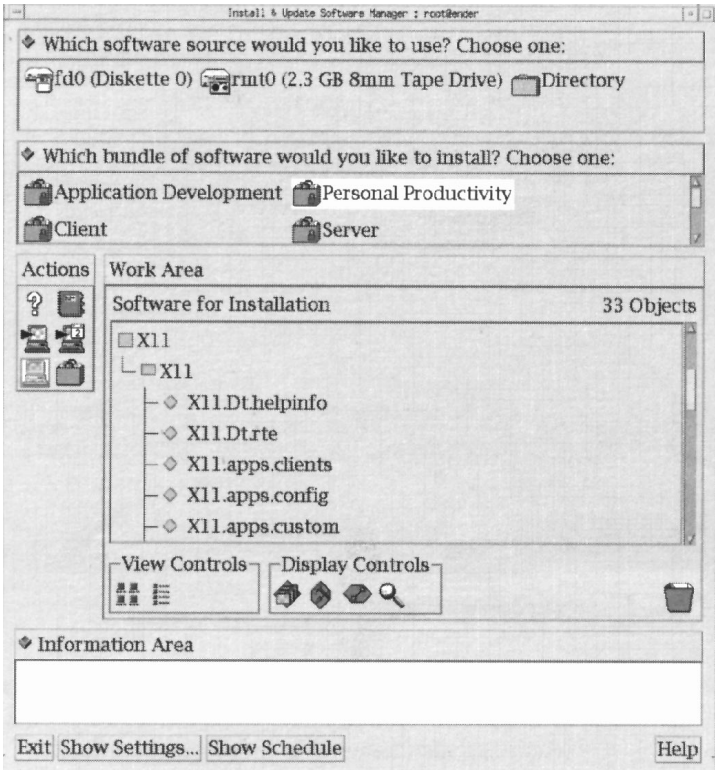


Figure 3. Install and Update Software Manager

Although this restriction on installing device drivers may never impact users, users may need to add a new device after initial installation and configuration. In this case, the driver may need installation. Users can ensure that Configuration Manager will install the needed software on a system initialization by using any of the following methods:

- Inserting the appropriate install media in the install media location
- Selecting the Add Devices Automatically button
- Choosing the Add Device Software button to invoke the install from media

Changes to Print Manager

Changes to the device driver packaging also required some changes to the Print Manager. A new function button, Install Printer Software,

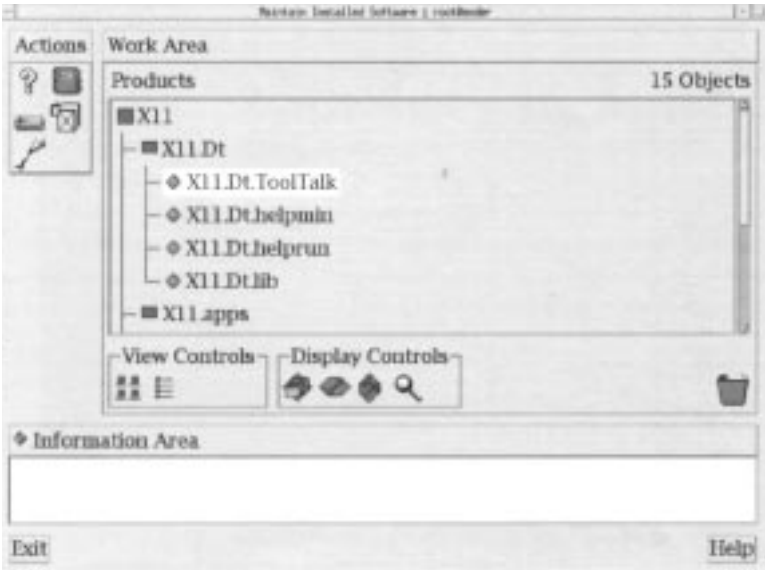


Figure 4. Maintain Installed Software Manager

enables users who have added a new printer type to install the printer driver from the install media. Users can also install this media when a new printer is added by specifying Install New Printer Type in the dialog (or selecting it from the list).

Changes to Users and Groups Manager

The Users and Groups Manager experienced the following changes in AIX 4.1:

- Removal of the Set Initial Interface action, which is not supported in AIX 4.1
- Introduction of the Set Language action, which provides a dialog to set the language environment for a user (This language environment includes the cultural convention and the first three language choices for the interface.)
- Improvement of the set password dialog

Integration with the AIX Desktop

Considerable effort was made to integrate VSM with the AIX desktop being introduced in AIX 4.1. This desktop, based on COSE technology, provides a sound user interface for general system interaction. Although VSM will continue to operate fully with AIXwindows, VSM can utilize user interface technologies—such as the desktop help system and controls, desktop widgets for check boxes and radio buttons, and font and color controls—when it detects the AIX desktop. The following sections describe some of the steps that were taken to provide integration.

Display Characteristics

Because VSM has integrated with the AIX Desktop Style Manager, users can choose from a variety of font sizes and styles to specify the AIX desktop fonts and colors in their VSM applications. The color tool provides common palettes for applications; users can also personalize their displays by changing the color values within their individual palettes. The use of color can also be altered to provide additional user control by specifying selected colors to use for specific parts of the windows and dialogues.

Desktop Technology

In addition to supporting themselves as desktop objects, VSM applications use several technologies from the AIX desktop. For example, VSM uses the desktop extensions to the Motif libraries, including the drag-and-drop widget, the radio button widget, and the check box widget. VSM also uses the AIX desktop help system, which provides a superior hypertext information system.

Application Manager

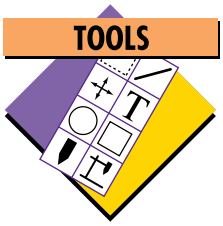
VSM applications will install themselves into the AIX Desktop Application Manager. With these applications in the system management toolbox of the Application Manager, users will have a standard path for finding all VSM applications.

Summary

AIX 4.1 introduces three new applications—Install Assistant, Install and Update Software Manager, and Maintain Installed Software Manager—to the VSM family. These applications provide primary graphical software installation and configuration methods for a graphics system user. The existing applications were enhanced to provide a graphical interface for function changes. Changes also made to existing VSM applications provide better integration with the AIX desktop and also provide the original AIX 3.2.5 and new applications with a step toward full integration in the AIX version of the COSE desktop environment.

Kenneth R. Banning, IBM Corporation, 11400 Burnet Road, Mail Stop 9541, Austin, TX, 78758. With a background in human factors, Mr. Banning is the user interface and usability architect for AIX system management. He has a BS in Psychology and an MS in Industrial Engineering from Texas A&M University.

VSM: A User-Centered Design



By Georgia A. Gibson

This article shows how an IBM team followed a user-centered design process to create a first-class product family for AIX system administrators. As developers of IBM's Visual System Management product family, we learned to appreciate and encourage the individual contributions of each team member. This mutual respect combined with our user focus resulted in a quality product.

Usability of products, a characteristic once important only to users, is now recognized as a feature that results in good business. As you thumb through your favorite computer software magazines, you will notice how frequently terms that describe ease of use are used to differentiate a product. Although customers have asked computer companies for years to make products easier to use, the business side of companies is just now beginning to realize the customer appeal of usable products and the great profit potential.

The User-Centered Design Approach

The heart of a User-Centered Design (UCD) process is a multidisciplinary team following a disciplined user-based process to develop a product "from cradle to grave." Our IBM team followed this UCD model to develop Visual System Management (VSM).

The VSM Product Family

The VSM product family of icon-based, object-oriented Graphical User Interfaces (GUIs) was developed for AIX system administrators. (See "the Next Step for Visual System Managment" in this issue.) It now consists of the following applications:

- Users and Groups Manager
- Device Manager
- Storage Manager
- Print Manager
- Install and Update Software Manager
- Maintain Installed Software Manager
- Easy Install
- Install Assistant

The first four applications were released as a separate package with AIX 3.2.5. The rest of the family was introduced with AIX 4.1.

Since the concept behind VSM was initiated by usability requirements from customers, we started the projects in our AIX Human-Computer Interface (HCI) department, which is primarily concerned with usability, user interfaces, and other customer advocate issues. At the inception of the project, there were two HCI teams—each consisting of one member with traditional human factors skills, one graphics designer, two human factor team leads, and two prototype software engineers. These teams worked together to develop different areas of the VSM project. The system management team focused on the Storage Manager, Device Manager, Users and Groups Manager, and Print Manager, while the install team concentrated on Install and Update Software Manager, Easy Install Manager, Maintain Installed Software Manager, and Install Assistant. These two teams are referred to as one HCI team in this article.

The system management team began working with Storage Manager and the Users and Groups Manager to determine basic metaphors, user mental models, and screen layout issues. Next, the install team began developing their applications. At the height of the project, the department was working on eight new products simultaneously in a collaborative process.

Georgia A. Gibson

The Overall Team and Their Roles

We took a multidisciplinary approach to the design of VSM. The cross-functional team had representatives from the following areas:

- **AIX Software Planning**: The planners assisted in the design by conducting periodic customer councils and providing requirements that fed into the initial designs.
- **AIX Human-Computer Interface**: The HCI department owned the product designs, conducted rapid prototype testing, produced multiple prototype designs, validated metaphors and screen design, documented the designs, followed through with software engineers, conducted User Integration Testing (UIT), and produced announcements with the marketing group.
- **AIX Information Design and Development (IDD)**: The IDD team, which consisted of a team lead and one writer per application, was responsible for the online help design. They were brought in during the design phase to implement the basic concepts and objectives of the design. They collected data in the usability testing efforts and were heavily involved in all stages of the product development, which helped them to design and produce online help, softcopy information, and hardcopy documentation.
- **AIX Software Development**: The Software Development team, consisting of a team lead and one software engineer per application, was also brought in during the design phase. The Software Development team acted as consultants on design and implementation issues. Their participation in this phase familiarized them with the objectives and designs that they would later code.
- **AIX System Management Architecture**: Members of the System Management Architecture group consulted on task domain specialties, such as Logical Volume Management (LVM). From the inception of the high-level design, the User Interface (UI) architect led the GUI widget library team and coordinated the design documentation. The architect's other role was to oversee consistency in the designs of the different applications in the family. HCI, IDD, Product Verification Test (PVT), and Software Development team leads met regularly with the UI architect to resolve issues such as consistency, translation, documentation, and Common Desktop Environment (CDE) integration. These meetings were essential to maintaining open communication between teams.
- **AIX Product Verification Test**: The role of the PVT team was to test the product family after functional verification testing. PVT testing occurred during the implementation phase for product accuracy and consistency across system management families, such as the System Management Interface Tool (SMIT) and VSM. The PVT team started by reviewing the product specification so that each PVT team member would become familiar with the designs at an early stage. Next, the PVT team wrote the PVT test plan, which was reviewed by all team leads for completeness and relevance. When PVT finally implemented the plan by testing the product family, the team also tested online helps and hardcopy documentation for accuracy and consistency across products.

The heart of a User-Centered Design (UCD) process is a multidisciplinary team following a disciplined user-based process.

Disciplined User-Based Process

According to other UCD processes, the disciplined user-based process consists of task analyses, competitive evaluations, high-level design walkthroughs, prototype evaluations, beta surveys, and benchmark assessments. As the following descriptions show, our team followed the UCD process on the VSM development project.

Task Analysis

This phase provides an understanding of future users, their environment, the tasks they currently perform, and the tasks they anticipate performing in the future. In our case, we were interested in learning about the needs of future users of VSM, namely system administrators. Thus, we used the results of an extensive task analysis involving over 100 system administrators[1] that had been conducted for system managers of UNIX operating systems. The study developed job descriptions for system administrators.

The results provided us with information about what tasks were performed by the percentage of the population performing them, and the average time spent on each task. In addition, system administrators rated each task according to various categories—consequences of inadequate performance, time to learn, and task frequency. System administrators also provided information

[1]Gibson, G. A. "System Management: What a Job!" *Proceedings of Share75* 1 (August 1990).

about types of hardware and software used, experience levels, and many other user environment issues and attitudes.

In addition, 57 worldwide customers participated in an AIX system management Quality Functional Deployment (QFD) requirements gathering project.[2] QFD is a systematic process for collecting customer requirements for a product. The collected data was fed into the functional requirements, and user interface and documentation requirements. Each account was profiled, and core QFD team experiences were documented, providing us with an in-depth understanding of our customers and their needs. The information we gained from this project and the task analysis study greatly influenced our designs.

Competitive Evaluation

In conventional UCD processes, this phase involves acquiring a detailed understanding, from a user's perspective, of the design of competitive products. In our case, competitive evaluations were facilitated by IBM's Competitive Evaluation Center (CEC). By using the wide variety of UNIX hardware and software available at the CEC, we were able to conduct heuristic evaluations at a greatly reduced expense.

High-Level Design Walkthrough

This phase defines the high-level design of product aspects seen and touched by users and compares it to that of competitors. We first brought in participants from a sample target population to view sketches and online screen mockups to help us define the high-level design of VSM products. Rapid prototyping techniques iterated the design; design changes were a direct result of this feedback. We changed the design at every iteration until we gained a favorable consensus from the users. The resulting high-level design became a solid basis from which we could develop more design details.

The HCI, planning, and architecture groups then compared this user-centered, high-level design with the competition. Once we all agreed that we knew the basic functions in the product, we were able to further develop the details of the design.

Prototype Evaluation

The prototype evaluation methodology of conventional UCD has two tenets:

- GUIs must be designed iteratively; a GUI cannot be designed "right" the first time.
- Design iterations must involve users—that is, user feedback must influence the design iterations.

Iterative prototyping was first performed at a high level and then throughout the design phase as a result of user testing. Once we had defined the high-level design, we quickly began producing prototypes of each product. Since our project was large and funding came in stages, each product was in a different stage of development; the products were being designed, prototyped, and tested concurrently. We soon realized that we would have consistency problems.

To combat these problems, we instituted "WHIM" meetings to communicate design issues and test results. With everything flying at a rapid pace, all of us were wondering "What Have I Missed" (WHIM). Any team member could call a WHIM meeting, but we also had regular, weekly meetings—sometimes once a week, twice a week, whenever we needed them. We documented the meetings in the product design specification to help us keep track of issues. The WHIM meetings proved useful with many benefits, including good communication lines.

Toward the end of the design phase, we validated the application designs with a traditional usability test and measures. Since the designs were still only prototypes, we defined sample tasks for each application and developed the prototypes to perform those tasks. To test the designs, we brought in users who matched target customer descriptions. We then changed the designs based on the results of user testing. Once changed, we documented the designs in product design specifications. At this point, we had completed over 1,100 hours of user testing.

Implementation Phase

Although this phase is not described in traditional UCD processes, we found it to be a critical part of our process. This was the coding phase, in which designs were converted into "real" products. Designs also changed in smaller ways during this phase in response to feedback from actual users of our applications. The designs became *even more* user centered.

During the implementation phase, the application teams consisted of UI designers, the IDD team, and software engineers. Each application

[2] Ashford, J.; Boschult, D.; Gibson, G.; Kim, G.; Lubart, N.; Rother, D; Smith, M. "UNIX Systems Management Voice of the Customer." *Unite 91 Conference Proceedings* (August 1991).

team met regularly, sometimes every other day, to review the coded design and to discuss design and documentation issues. Defects were documented and tracked using IBM's AIX Configuration Management Version Control (CMVC). We used these forums to discuss any defect or feature that might affect users. These discussions were especially beneficial to our user interface designers because they enabled designers to discuss implementation issues and trade-offs as a team. By participating in the discussions, the IDD team acquired new understanding of the design from the inside out.

User Integration Testing

The UIT phase is an important phase that is not described in conventional UCD processes. By conducting a usability test of the applications using real-code versions and up-to-the-minute documentation, the UIT phase provided additional user feedback to further improve the designs.

Defects were documented and tracked using IBM's AIX Configuration Management Version Control.

During the UIT phase, a PVT team member worked jointly with the HCI team members to test the implemented products. In independent testing efforts, PVT tested the functions and features, while HCI tested the usability of the designs with real users. This collaboration ensured that any design trade-offs had not negatively impacted the usability of the design. The result was a very profitable test of the usability and functionality of both the interfaces and documentation designs.

Beta Survey

The purpose of the UCD beta survey was to gather data from early users of the product before making the product generally available to customers. Typically in a beta program, all product channels are exercised (including service, support, and marketing). VSM also had a beta project in which users of all types used the products in their work. We then incorporated their feedback as product improvements using the CMVC tool.

Benchmark Survey

In the benchmark survey phase, the new product is compared to competitive products. This comparison is performed in a controlled testing environment with users and tasks identified in the Task Analysis phase. Information sought includes competitive advantages, traditional usability testing measures, and early identified usability objectives. The VSM products have not undergone any benchmarking activities to date.

Benefits of UCD

All participants in our VSM development project benefited from working on a cross-functional, user-centered team. First and foremost, we developed a high-quality, user-centered family of products. Individual team members experienced benefits such as first-hand knowledge of customer interaction with the system, early design input, multiple focuses on consistency issues, and close relationships that allowed for quick turnaround on design issues.

The ultimate gain to the product was a high focus on usability, which resulted in high product quality.

The PVT team also reported multiple benefits from working together, such as faster turnaround on problems, early design input, increased ability for software engineers to anticipate usability problems and how to prevent them, and a noticeable difference in the number and type of problems in the products released in IBM's AIX 3.2.5 and AIX 4.1 operating systems.

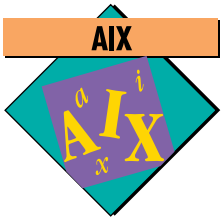
Because every team member was intimately involved in all stages of the project, each member was able to give impromptu demos or planned presentations of the products. This spread the responsibility of presenting the project to non-team members across organizational boundaries.

Team members gained many benefits through their interdependent experience on the VSM development project. Everyone was encouraged to introduce new ideas and opinions; communication between the teams was open and free-flowing. The diversity of personalities, experience, and training brought multidimensional qualities to the team.

In such an environment of cooperation, the interface blossomed. The result was VSM—a product that hides the pain of system management under designs that are fun and easy to use.

Georgia A. Gibson, IBM Corporation, 11400 Burnet Road, Austin, TX, 78758. Internet: ggibson@ausvm1.vnet.ibm.com. A graduate of the University of Texas in Austin, Ms. Gibson is a software planner for the Power Personal Systems Division. Her experience includes hardware and software usability, as well as user interface design for the SMIT, Distributed SMIT, and VSM for AIX.

Using Accounting Checkpoints in AIX/ESA



By Bob Gensler

System administrators can now charge users for resources used by their applications during the normal accounting cycle, even if the applications have not yet terminated. This article discusses accounting checkpoint concepts as they relate to the Accounting Checkpoints Enhancement for AIX/ESA Version 2 Release 2. It also provides instructions for installing this package on an AIX/ESA system and for effectively using the new functions.

Accounting Checkpoints improves AIX/ESA accounting facilities by enabling the system to capture accounting information for both running and completed applications. A history of accounting information can now be obtained for long-running applications. System administrators can charge running applications for resource usage before the applications complete. Even if the operating system should fail, administrators can still charge incomplete applications for resource usage before the system failure. The package also gives administrators the flexibility to charge differing rates based on differing levels of CPU priorities used by these applications.

This package gives the AIX/ESA accounting system greater accuracy, historical accounting for running applications, and greater billing flexibility.

Overview

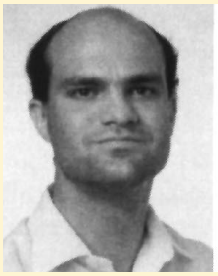
The AIX/ESA Accounting Checkpoints package creates interim process accounting records, which can be obtained for any user process. Using the package, system administrators can request that records be created at the following times:

- **At specific intervals**: For example, they can request that the operating system create interim accounting records 30 minutes after the process begins.
- **Upon demand**: The system administrator can request interim records by name for any process or user.
- **Whenever the `nice` command or system call changes the priority of a process**: A record is created containing the resources used under the previous priority level.

Interim process accounting records report changes in resources used by a process. The accounting information in each record shows the additional resource used by a process after the previous interim accounting record, or from the beginning of the process if no previous record exists. These records are placed in the process accounting output file, usually named `/var/adm/pacct`. The total record of resources used by a process can be obtained by adding the accounting information from all interim accounting records for that process.

The package also provides a new total accounting record format. These records, which report the complete accounting statistics for each user, now contain an additional field for process `nice` value. Using the proper commands, CPU-based resources can be separated by process `nice` value into different total accounting records for the same user. Since these records divide the CPU-based resources into different priority levels, system administrators can charge different rates for CPU resources used at different priorities.

Accounting Checkpoints adds four commands and two new C library system calls to AIX/ESA. In addition, to take advantage of new features in the package, seven existing commands have been extended. The package also provides complete man pages for the new and enhanced commands. Figure 1 shows highlights of these modifications.



Bob Gensler

Name	Type	Description
acctint()	New system call	Enables or disables the Accounting Checkpoints function; sets the default accounting checkpoint interval.
acctint	New command	Represents a command-line interface to the acctint system call.
acctch()	New system call	Creates interim process accounting call checkpoint records for one or more processes or users.
acctch	New command	Represents a command-line interface to the acctch system call.
ckptmerg	New command, migration aid	Summarizes all interim process accounting records for completed processes; imitates the behavior of the traditional AIX accounting package after Accounting Checkpoints has been enabled.
tacctconv	New command, installation tool	Converts existing files in the total accounting (tacct) record format to the extended total accounting record format; must be executed after installing the Accounting Checkpoints package.
acctcms	Enhanced command	Represents two new command-line options, -m and -w, which have been added to load and save the command's working storage between executions.
acctprc1	Enhanced command	Adds a reason code and a process nice value to the command's output.
acctprc2	Enhanced command	Handles the new, 12-column command input from acctprc1; a new option, -n, sorts output by uid/logname/nice.
acctcom	Enhanced command	Summarizes, by default, all interim process accounting records into summary records before processing the input; a new -x option processes input (interim records) instead of summarizing the raw input; a new -p option reports information specific to interim accounting records.
lastcomm	Enhanced command	Summarizes all interim process accounting records for completed processes before generating its output; a new -s option skips this summation step, generating only a list of completed commands.
acctmerg	Enhanced command	Represents a new option, -n, that summarizes output by process nice value, in addition to the summation technique selected by other options to this command; a nice value column has been added to the command's output.
runacct	Modified shell script	Adds an option -n to sort the daily accounting file by uid/logname/nice; by default, sorting is by uid/logname.

Figure 1. New and enhanced commands in Accounting Checkpoints

Starting It Up

If your AIX/ESA system has been configured to enable Process Accounting during system startup, the Accounting Checkpoints can also be activated at that time. The Accounting Checkpoints package installs a new file called `/etc/acct_interval` on your system that indicates whether Accounting Checkpoints should be automatically activated. This ASCII file contains a single-integer value that is used as the argument to the `acctint` command during system initialization.

If the `/etc/acct_interval` file contains 0, Accounting Checkpoints will not be activated during system initialization; Process Accounting will behave in the traditional manner, creating process accounting records only if a process terminates.

To activate Accounting Checkpoints during system initialization, change the value in the `/etc/acct_interval` file to the desired interval in minutes. This interval specifies the time between automatic creations of interim accounting records for an application. During system initialization, the accounting checkpoint interval will be set to match this value.

If your system does not automatically enable Process Accounting during system initialization, you can activate Accounting Checkpoints after system initialization. Simply use the `acctint` command and specify the accounting checkpoint interval as the argument. For example, to create interim accounting records for each process every 30 minutes, enter the following:

```
/usr/lbin/acct/acctint 30
```

The `acctch` command can create interim process accounting records even if Accounting Checkpoints has not been activated. If you do not

want the system to automatically create interim records at frequent intervals, you can choose not to activate Accounting Checkpoints and still execute the `acctch` command to create interim accounting records for processes or users. Accounting Checkpoints must be active for records to be created whenever a process changes its own `nice` value. Process Accounting must be activated for any interim records, either automatic or manual, to be created.

Selecting the Checkpoint Interval

Accounting Checkpoints was designed to capture accounting information at regular intervals for long-running applications. Although the accounting checkpoint interval can be any positive, non-zero integer, this value should be selected with care. If the interval is too small, medium-sized applications may create records more frequently, causing the accounting output files to fill rapidly. If the interval is too large, you may lack up-to-date information when performing accounting duties for applications that are still running.

The choice of an accounting checkpoint interval depends on your definition of a long-running application and when you perform accounting and chargeback duties. Select an interval that will cause the system to create at least one interim process accounting record for every long-running application by the time you perform your accounting duties.

An initial interval of 60 minutes is recommended. This interval will avoid creating interim accounting records for short applications, such as the `ls` command, while creating records for applications running longer than one hour. After you become comfortable with the Accounting Checkpoints concept and have a history of application runtimes for your system, you can then tailor the accounting checkpoint interval to your specific needs.

The `acctint` command adjusts the accounting checkpoint interval. To change the accounting checkpoint interval from 30 minutes to one hour, enter the following:

```
/usr/lbin/acct/acctint 60
```

The changed accounting checkpoint interval takes effect on all user applications that begin after the `acctint` command has been issued. Applications that were already running will continue to use the previous accounting checkpoint interval until that interval expires. Once the interval expires, the revised accounting checkpoint interval will be used to schedule the creation of the next interim record.

Using Interim Records

Interim process accounting records capture resource information for running applications. AIX/ESA then collects these interim records during the accounting chargeback cycle and uses them to create total accounting records for each user. By collecting interim records during chargeback cycles, total accounting records allow system administrators to charge running applications for system resources used during previous accounting periods as well. In addition, if an operating system should fail while applications are still running, this process enables system administrators to charge any uncompleted processes for all resources used before the system failure.

To create these records automatically, system administrators must set the accounting checkpoint interval using either the `/etc/acct_interval` file or the `acctint` command. AIX/ESA will then create an interim accounting record for all user processes begun after this interval is set, either when the application's `nice` value changes or when its interval expires.

Interim records can be created upon demand with the `acctch` command. System administrators can request interim accounting records in the following categories: a single process, a list of processes, a process group, a user, a list of users, or all current user processes.

Using interim process accounting records is not difficult; all accounting commands and scripts have been modified to handle these new records and to take advantage of the record's new features. To begin using interim process accounting records, simply activate the Accounting Checkpoints feature. The accounting system will then take over, requiring very little intervention from system administrators.

When creating interim accounting records automatically, the accounting checkpoint interval and the start time of the process determine the exact time that an interim record will be created.

By adding the accounting checkpoint interval to the start time of the process, the AIX/ESA kernel calculates the time at which the first record will be created. When this time expires, the kernel adds it to the accounting checkpoint interval to determine when the next record will be created. The cycle repeats until the process completes.

Occasionally, system administrators may not want to wait for these records to be automatically

Accounting Checkpoints was designed to capture accounting information at regular intervals for long-running applications.

created, preferring instead to create interim accounting records for one or more processes. They can use the `acctch` command to target processes, process groups, or users. The kernel will schedule interim process accounting records to be created for all processes named by the command, or for all processes owned by the user named by the command.

For example, consider the following scenario. As system administrator, you want to perform billing for the user bobgens at 1 A.M. Since you want the billing to be up-to-date, you need the most recent information on any running applications for this user. However, some applications used by bobgens may not yet have created interim process accounting records. Instead of waiting for these records to be created automatically, you could issue the following command to gain the latest accounting information for this user:

```
/usr/lbin/acct/acctch -u bobgens
```

The total accounting record contains resource usage information for all users that ran applications on the system during the accounting period.

Within a few seconds, all running processes owned by user bobgens will have created interim process accounting records in the process accounting output file.

Although the command schedules the creation of interim accounting records, not all records may be created by the time the `acctch` command completes. To have its interim record created, an application must be using the CPU at that time; suspended or waiting applications will not create interim accounting records until they begin running again. You should allow a small amount of time—usually just a few seconds—after the completion of the `acctch` command before expecting interim process accounting records to be available.

Using the Extended Total Accounting Record

The total accounting record provides information for charging customers for system usage. These records contain resource usage information for all users that ran applications on the system during the accounting period.

With Accounting Checkpoints installed, the operating system can create multiple total accounting records for a single user. One total accounting record can be created for each `nice` value selected by the user during the previous accounting period. CPU-related resources, such as primetime and non-primetime shift work, are split according to the `nice` value used by these resources and reported in separate records for each `nice` value. All non-CPU related resources, such as printer or disk resource usage figures, are reported in the total accounting record for the zero `nice` value. The total CPU-related resources for a user can be calculated by adding the CPU-resource usage figures from all the user's total accounting records.

By dividing the CPU-based resources into separate total accounting records, Accounting Checkpoints enables system administrators to charge different rates for CPU resources used at different levels of priority. Charging less expensive rates for CPU-based resources used at a lower priority can encourage users to lower the priority of their applications more frequently.

Installing the Package

The Accounting Checkpoints package can be installed only on AIX/ESA Version 2 Release 2 operating systems. There are no special hardware requirements for this package, but a complete list of software requirements and requisites is provided in the Accounting Checkpoints installation documentation.

You should schedule some system downtime to install the Accounting Checkpoints package. Installing the package requires a rebuild of the kernel, and some existing accounting data files must be converted to a new format. Single-user mode is recommended to perform the installation and file conversions. The package is installed using the `installp` command.

Accounting Checkpoints will change the format of the total accounting (`tacct`) record. If the system has been operational before the installation of Accounting Checkpoints, you will probably have several files residing on the system that use the older total accounting record format. These files, created by the `runacct` shell script and the `acctprc2` command, are used as input to the `acctmerg`, `prdaily`, and `monacct` commands. Each of these commands has been modified in the Accounting Checkpoints package. For these commands to function properly, the existing data files must be converted to the new total accounting record format after the Accounting Checkpoints package has been installed, but before bringing the system back into multi-user mode.

If applications generate total accounting record files, these data files may be in some nonstandard locations. These files must be manually located and converted to the new `tacct` format. By default, the AIX/ESA accounting commands generate the following total accounting record files:

```
/var/adm/acct/fiscal/tacct.mm
/var/adm/acct/sum/tacct
/var/adm/acct/sum/tacct.mmdd
/var/adm/acct/sum/tacct.prev
/var/adm/acct/sum/ptacct.mmdd
/var/adm/acct/nite/ctacct.mmdd
/var/adm/acct/nite/daytacct
/var/adm/acct/nite/ptacctn.mmdd
```

Convert these files to the new total accounting record format with the `tacctconv` command, before bringing the system back to multi-user mode and before using any accounting commands. Figure 2 shows an example of how to do this. Once these data files are converted, the system can be brought to multi-user mode, and the accounting commands can be used.

Migration and Compatibility

Until now, one process accounting record represented the entire execution of an application, because such records were created only when an application terminated. With the Accounting Checkpoints enhancement, an application can now create one or more process accounting records during its life. The accounting commands have been changed to remove the basic assumption of "one record, one execution," but your system may contain local applications that were designed using this assumption. If such applications exist on your system, they need to use the migration aids included in the Accounting Checkpoints package to work properly.

Accounting Checkpoints also implements a new process accounting record format called the *acct structure*, described in the `<sys/acct.h>` header file. Your system may have applications that use this record; any applications that examine the process accounting output file `/var/adm/pacct` use this record format.

The `ckptmerg` command is provided for those who do not wish to update their local applications to handle the new record format or interim process accounting records. This command converts the file to the format expected by existing applications. Summary process accounting records are created for completed applications in either the new extended record format or the Release 2 format. Existing commands can access the accounting output file by using this command as a filter, as shown below (enter as a single command):

```
/usr/lbin/acct/ckptmerg -c
/var/adm/pacct |
old_accounting_appl
```

```
mv /var/adm/acct/sum/tacct /var/adm/acct/sum/tacct.OLD
/usr/lbin/acct/tacctconv /var/adm/acct/sum/tacct.OLD >
    /var/adm/acct/sum/tacct
```

Figure 2. Using tacctconv after installation

When using this command as a filter, you should realize that some long-running applications may have created interim process accounting records that exist in several accounting output files. It is possible for an application to begin while one accounting output file was active, and complete when another output file was active. For the `ckptmerg` command to function properly, it must locate all interim process accounting records for the application. You may need to supply several files as arguments to the command to obtain summary records for certain long-running applications.

The package also changes the format and nature of the total accounting record for users. Users may now have more than one total accounting record generated for them. Applications that manipulate these records must be updated to handle the new format of the total accounting record and to expect multiple records for each user for the same accounting period. The new total accounting record format is specified by the tacct structure in the `<tacct.h>` header file. Applications that use the ASCII version of total accounting records should also be redesigned.

Using the Accounting Checkpoints Package

The following example demonstrates how Accounting Checkpoints can be used in your daily accounting routines. This example is made with the following assumptions:

- The operating system is AIX/ESA Version 2 Release 2 with the Accounting Checkpoints enhancement installed and all necessary file conversions made.
- The system administrator's definition of a long-running job is any application that takes longer than an hour to complete.
- Process accounting is started automatically during system Initial Program Load (IPL).
- The `runacct -n` shell script is scheduled by `cron` to run at 11 P.M. daily.

Before the last IPL of the system, the system administrator changed the contents of the

UID	LOGNAME	PRI_CPU	NPRI_CPU	PRI_MEM	NPRI_MEM	...	NICE
				:			
				:			
101	bobgens	58.7800	10.3460	4255566	890040	...	0
101	bobgens	6.4240	0.0000	0	0	...	2
101	bobgens	12.3570	3.7550	0	0	...	5
				:			
				:			

Figure 4. /var/adm/acct/sum/rprt.1118 file entries for user bobgens (edited)

```
(non-CPU resources used * non-CPU resource usage rate) +
(CPU resources used at default nice value * default CPU usage rate) +
(CPU resources used at nice value 2 * nice value 2 usage rate) +
(CPU resources used at nice value 5 * nice value 5 usage rate)
```

Figure 5. Total charge to user bobgens

etc/acct_interval file from 0 to 60. Since process accounting was automatically started by the operating system during IPL, Accounting Checkpoints was also automatically activated during the last IPL of the operating system. Interim records are now created every 60 minutes for applications that run longer than an hour.

To ensure that the system has the most recent process accounting information at its disposal before the runacct script begins, the system administrator schedules the acctch command to create interim records at 10:55 P.M. for all running applications with the following crontab line:

```
55 22 * * * /usr/lbin/acct/acctch 0
```

With this command, the system will force the creation of interim process accounting records for all running applications at 10:55 P.M. These interim records will be placed in the accounting output file var/adm/pacct and will be processed by the runacct script.

10:55 P.M., November 18: The acctch command forces interim process accounting records to be created for all applications currently running on the system.

11 P.M. that same night: The runacct shell script begins and performs all the accounting chores, creating the /var/adm/acct/sum/rprt.1118 and var/adm/acct/sum/tacct.1118 files. The runacct shell script specified the -n option, resulting in one total accounting record per user, per process nice value, to be created in the output files.

Morning, November 19: The system administrator processes the output file var/adm/acct/sum/tacct.1118 and begins to charge back resource usage to the system's users. The chargeback program finds multiple records for a single user in the ouput file, /var/adm/acct/sum/rprt.1118, shown in Figure 4. The chargeback program assigns a lower usage rate for the CPU resources used by bobgens at nice values of 2 and 5, rewarding that user for his conscientious behavior. Figure 5 shows the total charge made to user bobgens.

Bob Gensler, IBM Corporation, Enterprise Drive, Kingston, NY 12401. Internet: bobgens@minnie.nic.kingston.ibm.com. Mr. Gensler is a software developer for Power Parallel Systems and former lead developer on the AIX/ESA Accounting Checkpoints project team. He has a BS in Computer Science from Central Connecticut State University in New Britain.

The IBM POWER2 Architecture Implementations

By Sohel R. Saiyed and Jacob Thomas

This article describes some of the important characteristics of IBM POWER2 Architecture implementations used in products announced by IBM in September 1993 and May 1994.

All POWER2 Architecture implementations (shown in Figure 1) are completely binary compatible. Applications written for the POWER Architecture will also run on the POWER2. The major differences among POWER2 implementations are the size of the data cache, the use of a second level (L2 cache), width of memory interface, and width of processor interface (processor-to-data-cache interface).

These characteristics have different performance impacts on the three classes of codes: integer, floating point, and commercial Transaction Processing (TP). Performance discussions in this article are based on the results of SPECint92™, SPECfp92™, LINPACK, and TPC-C™ benchmarks.

POWER2 Architecture Implementations

In September 1993, IBM announced three RISC System/6000 (RS/6000) servers based on the first implementation of the POWER2 Architecture (the *original* POWER2). Although IBM customers received these three servers with enthusiasm, many hoped for and requested even more—a desktop implementation and greater capacity for commercial TP. IBM set a major goal of satisfying these two requests for the May 1994 announcement.

Meeting these two requests was not easy. The stringent packaging and price targets needed for a desktop product meant several design trade-offs. Achieving greater capacity for commercial TP at a reasonable cost required several other design trade-offs. In the end, IBM achieved the results requested by its customers in the form of two new POWER2 Architecture implementations: the *new-desktop* POWER2 and the *new-server* POWER2. Figure 1 summarizes the three implementations of POWER2 Architecture and the corresponding RS/6000 model numbers.

RS/6000 Model	Package	POWER2 Implementation	Announcement Date
58H, 590	Deskside	Original	September 1993
990	Rack	Original	September 1993
59H	Deskside	New Server	May 1994
R20, R24	Rack	New Server	May 1994
380, 390	Desktop	New Desktop	May 1994

Figure 1. POWER2 Architecture implementations

The three implementations are identical at the processor level, making them completely binary compatible. All three can execute up to six instructions (branch, conditional register, two fixed point, and two floating point) per cycle and have the same instruction cache size (32 KB). Figure 2 summarizes the major differences among these implementations:

- **Data cache size:** Depending on the implementation, data cache size can be 64, 128, or 256 KB.
- **L2 cache:** This is standard on Models 59H, R20, and R24, and an option on Model 390. L2 cache is not available on Models 58H, 590,

Sohel R. Saiyed

Jacob Thomas

POWER2 Implementation	RS/6000 Model	Data Cache/ Line Size	L2 Cache Size	Memory Interface	Processor Interface
Original[1]	58H,590,990	256 KB/256 bytes	N/A	8 words	8 words
New-Server	59H,R20,	128 KB/128 bytes	1 MB	4 words	8 words
	R24		2 MB		
New-Desktop	380	64 KB/64 bytes	N/A	2 words	4 words
	390		0-1 MB		

[1]Four or more memory cards assumed. With less than four cards, the effective cache size and memory interface width are 128 KB and 4 words, respectively.

Figure 2. Key aspects of the POWER2 implementations

990, and 380. Existing application programs will generally exploit the L2 cache without recompilation.

- **Memory interface:** Width of the memory data bus could be eight, four, or two words, depending on the implementation. The width of L2 cache to memory interface (on models with L2 cache) is the same as the memory interface width.
- **Processor interface:** Width of the processor to data cache bus is four words on Models 380 and 390, and eight words on all other models.

Characteristics of Different Classes of Codes

Integer code (SPECint92) typically does not access large amounts of data and usually fits in any large data cache. A reduction in data cache size may reduce performance slightly, but adding an L2 cache may compensate for the performance loss in some application codes. For integer codes, a four-word processor interface is sufficient to keep the two fixed-point units busy. Also, a 32 KB instruction cache is usually sufficient for these types of codes.

Typical floating-point code (SPECfp92 and LINPACK benchmarks) accesses large amounts of data. Smaller data caches and the reduced interface widths have a greater impact on floating-point code than on integer code. The L2 cache can compensate for some lost performance, but not all, particularly on the new desktop models. For some floating-point code, the four-word processor interface may not be sufficient to keep the two floating-point units busy. However, the 32 KB instruction cache is usually sufficient for these types of code.

Commercial TP workloads consist of integer code, but their characteristics differ from those of SPECint92. TP code typically contains a large instruction footprint that does not fit in a 32 KB instruction cache. In this environment, adding an L2 cache significantly reduces the penalty for instruction cache misses. Since TP data is primarily 32-bit integer data, a four-word processor interface can feed the two fixed-point units. Therefore, the new-desktop and new-server POWER2 implementations show significant performance gains on TP code. Although this behavior of code classes is typical, the behavior of customer application codes may differ from these examples.

Performance Differences

This section discusses the performance of RS/6000 models selected from the three POWER2 implementations: Model 590 (original POWER2), Model 59H (new-server POWER2), and Model 390 (new-desktop POWER2). Since these models operate at about the same frequency, the performance results (Figure 3) reflect implementation differences. Discussion of these results is limited to the performance of benchmarks SPECint92, SPECfp92, LINPACK, and TPC-C.

The Models 590 and 59H have nearly identical performance on SPECint92, while the SPECint92 result on the Model 390 is about 7% lower. Since the four-word CPU cache interface (4x32 bit) of the Model 390 should be sufficient to feed the two integer units, the most likely sources of the performance decline are the smaller data cache and the memory interface.

On LINPACK SP (100x100 Single-Precision Benchmark), all three models offer the same performance. The LINPACK SP data fits in the 64 KB

data cache. The L2 cache and the memory interface have minimal impact on this benchmark result. The four-word processor interface is sufficient to keep both floating-point units busy.

On LINPACK DP (100x100 Double-Precision Benchmark), the Models 590 and 59H performance results are the same. Since the LINPACK DP data fits in the 128 KB data cache, the differences in cache size and memory interface do not affect performance. The performance of the Model 390 LINPACK DP is less than half that of Model 590. The eight-word processor interface can deliver two quad-word, floating-point storage reference instructions (two load quads or two store quads) per machine cycle. Since the Model 390 four-word interface operates at half the capacity of the eight-word interface on the Model 590, a 50% drop in performance should be expected. Furthermore, the DP data does not fit in the 64 KB data cache, which causes the performance to drop.

The Model 590 offers slightly better performance than the Model 59H on SPECfp92. The reduction in memory interface width and data cache size is not fully compensated by the addition of the L2 cache. The SPECfp92 of the Model 390 is about 20% lower than the Model 59H results. The smaller data cache and narrower interfaces contribute to the decline in performance compared to Models 590 and 59H.

The new-desktop and new-server POWER2 implementations have smaller data cache lines than the original POWER2 implementation. The smaller cache lines can affect the performance of floating-point code with contiguous memory access.

On double-precision, floating-point code (typical of engineering and scientific applications), the original POWER2 implementation (such as the Model 590) delivers better performance than the new-server implementation (such as the Model 59H).

The Models 59H and 390 show significant performance gains over the Model 590 on the TPC-C benchmark. These gains are the result of adding L2 cache, application tuning, and operating system enhancements. The addition of an L2 cache accounts for a large part of the performance gain. Because the L2 cache is significantly larger than the instruction cache, it reduces the effect of instruction cache misses by containing the majority of code paths executed by the transaction processing software. A four-word processor interface is sufficient to feed the two integer units. The performance loss (if any) caused by a reduced memory interface is inconsequential compared to the performance boost from the large L2 cache. By adding the L2 cache, the 59H performance gain over the 590 is estimated to be about 20% for workloads similar to TPC-C.

	Implementations		
RS/6000 model:	Original 590	New-Server 59H	New-Desktop 390
Clock cycle:	66.7 MHz	66.7 MHz	67.0 MHz
Processor interface:	8 words	8 words	4 words
Data cache size:	256 KB	128 KB	64 KB
L2 cache size:	N/A	1 MB	1 MB
Memory interface:	8 words	4 words	2 words
SPECint92	121.4	122.4	114.3
SPECfp92	254.2	250.7	205.3
LINPACK SP	73.1	73.1	73.0
LINPACK DP	132.0	132.0	55.1
tpmC(c/s)	726.1[1]	1122.3	901.5
$/tpmC	1395[1]	968	909

[1]The Model 590 measurement, performed in 1993, does not include recent enhancements in the TPC-C application and AIX 3.2.5.

Figure 3. POWER2 implementation performance comparison

The TPC-C benchmark results confirm that the new-desktop and the new-server POWER2 implementations are more suitable for commercial applications than the original POWER2 implementations.

Finally, the eight-word memory interface of the original POWER2 implementation requires a minimum of four memory cards to achieve full bandwidth potential. With less than four memory cards, the effective memory interface width and the data cache size become four words and 128 KB, respectively.

Sohel R. Saiyed, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Saiyed is an advisory engineer in systems architecture and performance in the RISC System/6000 Division. Mr. Saiyed has an MS in Computer Engineering from Clemson University and a B.Tech. degree in Electrical Engineering from the Indian Institute of Technology in Kharagpur, India.

Jacob Thomas, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Thomas is an advisory programmer in processor performance in the RISC System/6000 Division. Mr. Thomas has an MS in Statistics from Michigan State University and an MS in Physics from Birla Institute of Technology and Science in Pilani, India.

AIX Terminal Server

By Eddie Ho and Dave Phipps

As a multi-user system, AIX enables multiple users to connect to the system either locally via direct attachment, or remotely using a teleprocessing facility or a Local Area Network (LAN). The new Serial Communication Network Server can accelerate the use of AIX in multi-user distributed environments by enabling remote LAN-based workgroups to access system resources.

The classic definition of a terminal server is a device used to multiplex serial peripherals—ASCII terminals, printers, plotters, or modems—onto either a LAN cable or a host computer. Most UNIX systems have at least one built-in serial port. For additional users to be connected, systems must use multiport adapters, such as the RISC System/6000 (RS/6000) asynchronous adapters, which include the 8-Port Asynchronous Adapter and the 16-Port Asynchronous Adapter.

To use these adapters, each peripheral must be cabled to the host. Host computers have decreased their footprint size and can now support many more peripherals. However, because cabling is inherently bulky, connecting a large number of peripherals to a "small footprint" host can become a cabling nightmare.

The IBM 128-Port Asynchronous Controller Subsystem and the Serial Communication Network Server eliminate the difficulty and expense of hard-wiring these devices by packaging multiple serial ports in a single distribution box, rather than through point-to-point connections. The 128-port subsystem is used for wide area point-to-point connections between distribution boxes, while the Serial Communication Network Server is primarily Ethernet™ LAN-based.

Network resources are usually partitioned into server groups, in which the clients are connected to various network servers. These input/output devices can be located wherever they are needed, making it very convenient to relocate people and equipment on a network. Figure 1 summarizes the capabilities of the 7318 Serial Communication Network Server. In this scenario, all users connected to the 7318s can access the resources of all hosts.

Common methods of communication to a host system using TCP/IP are through the `telnet` and `rlogin` commands, which enable remote users to connect to hosts across a network. The communication layers of a host system dynamically create a pseudo-device that simulates a locally attached terminal for network communications. Therefore, users attached to a terminal server typically have the flexibility to connect to several hosts concurrently. In general, a terminal server can be classified as a *communication server* if it has a full set of network protocol layers and exhibits network attributes such as the following:

- Network management support
- Access to other communication nodes
- Advanced routing
- Security features
- Remote printing support

The 7318

There are two models of the 7318—the P10 and the S20. The P10 is a terminal server, while the S20 is a communication server. Both models include the following hardware features:

- 16 serial ports with RJ-45 connections; each port provides an RS-423 electrical interface that is compatible with RS-232, RS-423, and RS-422 devices
- Serial data rates up to 115.2 Kbps on all 16 ports
- Multiple session support for terminals, including one session for transparent printing and six display sessions (Each display session is selected via a user-definable hot-key based on the terminal type.)

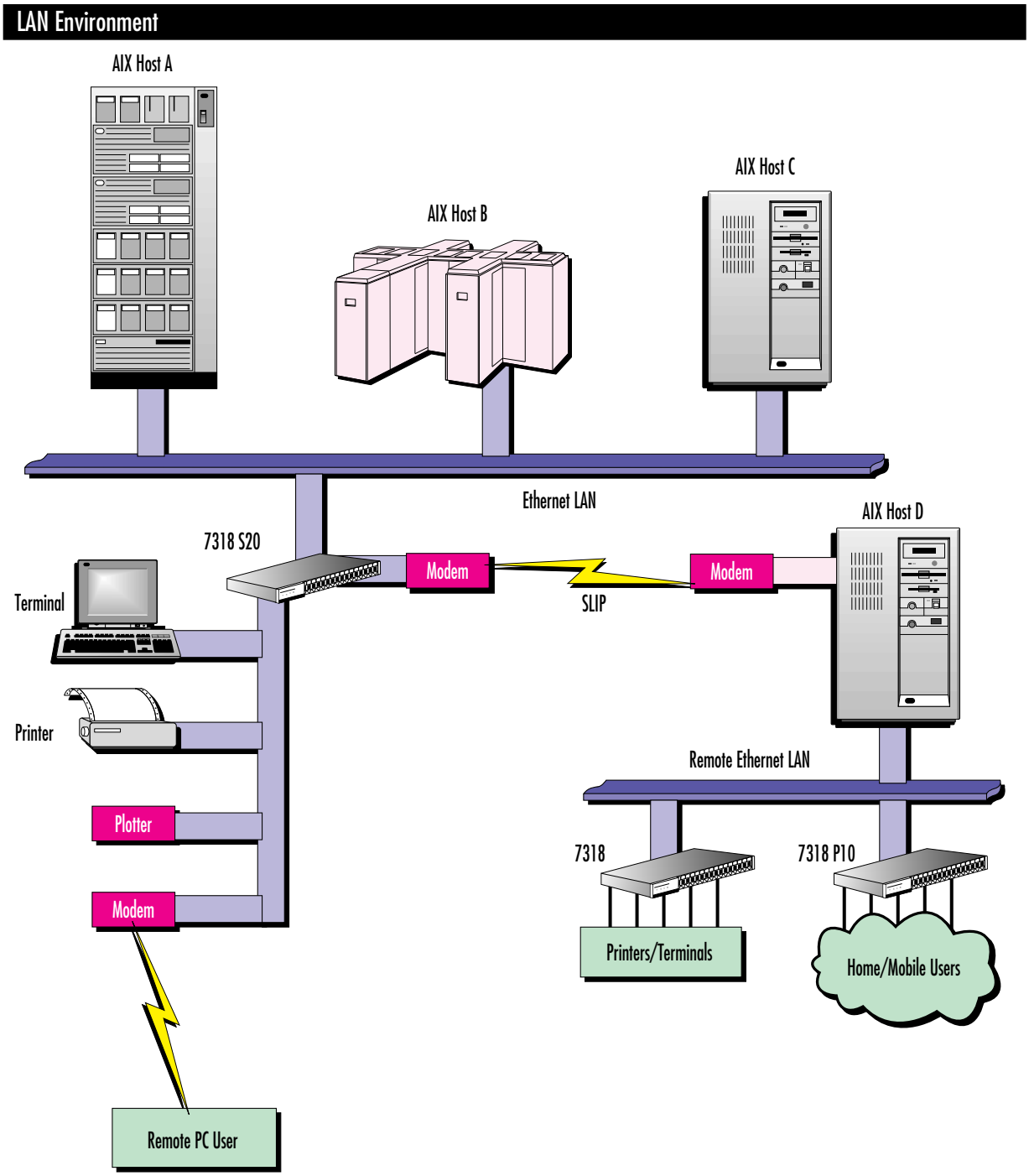


Figure 1. Serial communication network server in a LAN environment

- Two parallel ports, each with data rates up to 250 Kbps
- Capacity to daisy-chain up to four 7318s on one Ethernet connection
- Ethernet attachment using all existing RS/6000 Ethernet adapters (This is an efficient means for attaching serial devices because this attachment does not occupy a Micro Channel® slot in the RS/6000 system, as compared to the multiport attachment approach. Any of the following can serve as attachment media: 10BaseT, 10Base2, and 10Base5.)
- Desktop, rack, or wall-mounting

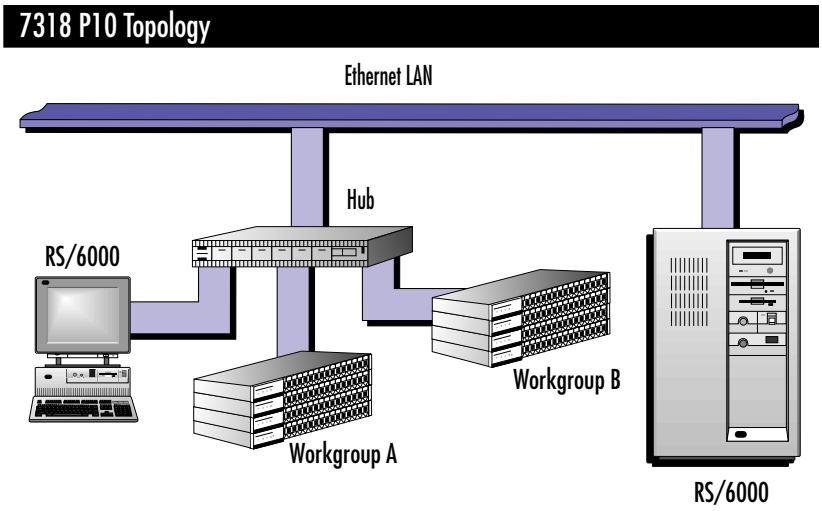


Figure 2. Model 7318 P10 topology in a hub environment

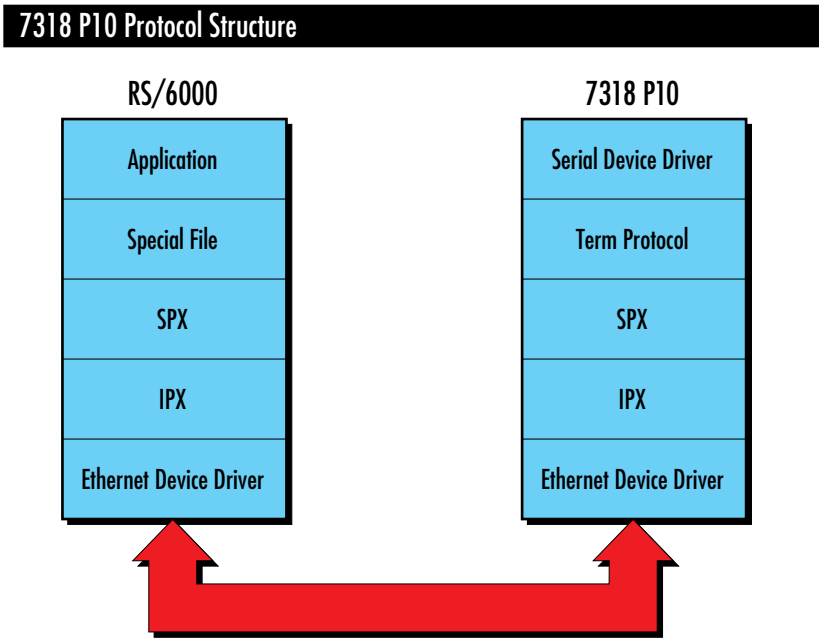


Figure 3. Model 7318 P10 protocol structure

Model 7318 P10

The Model 7318 P10 provides for the attachment of serial ports to the RS/6000. These ports operate much like the multiport adapter or the native ports. The network topology is similar to the Remote Asynchronous Node of the 128-port asynchronous controller; the 7318 connects to the RS/6000 through a coax cable running the standard Ethernet protocol. Figure 2 shows a simple Model P10 topology.

The RS/6000 LAN connection can use either a Micro Channel Ethernet adapter or an integrated Ethernet port. The LAN protocol uses SPX/IPX, selected for its simplicity and low overhead for the RS/6000. Figure 3 shows both protocol layers.

The device driver creates tty devices very similar to traditional tty devices such as `/dev/ttyxx`. When a login program such as `getty` is started on a tty device connected to a 7318, a login banner appears on the terminal. These static tty devices are created at system boot time. Application programs that use a `/dev/ttyxx` device as their interface can also use the ports without any programming changes.

Some tty terminal control processing occurs with the P10. During its power-on phase, these functions are downloaded from the RS/6000.

Model 7318 S20

The Model 7318 S20 is a communication server that uses TCP/IP as the transport protocol. It supports all the functions of the Model P10 plus additional communication protocols. It is a complete TCP/IP networking node with key node attributes such as security, network management, and other applications, including the following:

- Simple Network Management Protocol (SNMP)
- Security (Kerberos Version 5)
- Communication programs:
  - `telnet`, `rlogin`, `tn3270`
  - Serial Line Interface Protocol (SLIP)
  - Compressed SLIP (CSLIP)
  - Point-to-Point Protocol (PPP)

Since the S20 contains its own TCP/IP protocol stack and applications, it does not consume any RS/6000 CPU resources after receiving its program load from the RS/6000. It provides stand-alone support for attaching serial devices with a focus on asynchronous TCP/IP node-to-node communication. The common protocols used for this are SLIP, CSLIP, and PPP.

The common terminal emulation protocols used with TCP/IP are `telnet` and `rlogin`. The 7318 S20 can communicate with any system that supports the same protocol.

The 3270 data stream is a commonly used terminal protocol in the IBM mainframe area. It is a blocked data stream in EBCDIC form. The 7318

S20 can process this data stream and translate it into ASCII format for terminal presentation.

The S20 supports the SNMP management protocol and can participate in a network management system as a full agent. Using an SNMP manager such as NetView/6000™, network administrators can retrieve event statistics and error logs as well as set event traps.

Network security is also supported by the S20 system with password and data encryption during runtime. The Kerberos security protocol is used with the `rlogin` protocol for better security control. The S20 uses Kerberos Version 5, which is compatible with the Distributed Computing Environment (DCE) standard. This provides a secure environment for workgroup users while accessing data.

Historically, running SLIP on a point-to-point link was seriously limited by the modem bandwidth. This limitation has been significantly reduced by the availability of V.32bis modems providing 14400 bps connections over dial-up lines. The 7318 supports SLIP as well as the compressed format (CSLIP) for better performance. Since SLIP was formed in the early 1980s, it lacks many useful link-level functions; most known deficiencies have been documented by RFC 1055 of IETF.

Point-to-Point Protocol evolved from earlier work on SLIP. PPP is a more robust, full-featured protocol that is quickly becoming a standard, as defined by RFC 1331.

Conclusions

Terminal servers provide the most cost-effective way to add users to a network. By offering a better wiring scheme, they enable terminal connections to be located wherever they are needed, instead of being confined to the back of an RS/6000. In addition, communication servers are next-generation devices designed to meet the ever-evolving connectivity needs of today's networks. These servers enable remote access, a more mobile workforce, and modem access from PCs or X terminals.

Eddie Ho, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Internet: eddieho@chang.austin.ibm.com. Mr. Ho is a senior programmer in the AIX Communications area. He has a BS in Computer Science from the University of Wisconsin and an MS in Computer Science from North Dakota State University.

Dave Phipps, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Internet: dlphipps@bach.austin.ibm.com. Mr. Phipps is a staff programmer in the AIX Communications area. He has a BS in Applied Mathematics, Engineering, and Physics, and an MS in Computer Science from the University of Wisconsin.

Terminal servers provide the most cost-effective way to add users to a network.

PowerPC 604 Development Completed

IBM and Motorola™ have completed development and fabrication of the PowerPC 604 microprocessor, the most powerful high-volume microprocessor in the industry. The PowerPC 604 will provide close to twice the performance of competing microprocessors, with the power to support new classes of advanced multimedia, graphics, and other applications. It is designed to deliver exceptional performance for high-end desktop systems, midrange servers, and high-performance graphics workstations.

The PowerPC 604 at 100 MHz achieved an estimated SPECint92 rating of 160 and an estimated SPECfp92 rating of 165. It is designed in a 0.5-micron, 3.3-volt Complementary Metal Oxide Semiconductor (CMOS) technology, and incorporates 3.6 million transistors. This superscalar, multiprocessor-enabled chip issues up to four instructions in parallel every clock cycle to six execution units. Its three-stage, double-precision floating-point unit enables users to take advantage of graphical and multimedia applications, providing tremendous performance capabilities that were previously available only through expensive add-on hardware. ■

Network Terminal Accelerator

By Eddie Ho, Jim Gallagher, Kent Malave, and David Phipps

Users can be attached to multi-user AIX systems either by terminal adapters or by a LAN. The challenge for any multi-user system is to apply processing power to each user's application efficiently. IBM's Network Terminal Accelerator can improve overall system performance for users on an Ethernet LAN by off-loading the LAN protocol execution from the host.

The growing demand and competition for CPU resources require managers to search for ways to deliver computing services at a minimal cost. One way to reduce costs is to use a Micro Channel multiport adapter to attach terminals to a system. Terminals are often attached across a LAN for convenience and flexibility; however, this method is inefficient because of the software protocol overhead of network connections. Implementing this software in an adapter eliminates this inefficiency, resulting in lower CPU costs.

One question facing many managers is how to increase the number of regular AIX users without overloading the CPU. There are several potential solutions to this problem:

- **Migrate to a faster system**. Since new systems are often expensive, this solution may not be the most cost-effective.
- **Add more processors**. Although this method can often meet the needs of end users more efficiently, it can be programming intensive and may increase system administration requirements.
- **Provide extra capacity through loosely coupled systems on a network**. Such systems are too unstructured for many applications. Some applications can use a client/server model to off-load some processing to distributed client processors. However, since the client/server model does not fit all applications, this approach may involve extensive programming changes.
- **Add tightly coupled special-purpose processors to off-load the main processor**. This is often the most desirable approach because it has the least impact on programming environment, current applications, and migration. One example of this approach is the use of communication or storage I/O processors on mainframes. The RISC System/6000 has a variety of intelligent Micro Channel adapters that can perform some level of off-loading. For example, the graphics adapters off-load much of the vector or complex image processing from the graphics subsystem, and the 128-port asynchronous subsystem adapter can off-load all the tty character processing from the AIX tty subsystem. Similarly, the Network Terminal Accelerator off-loads the networks and terminal protocol to the firmware.

Network Terminal Acceleration

Network terminal is the remote terminal access between two systems in a LAN environment. Traditional network protocols—System Network Architecture (SNA), TCP/IP, and Open Systems Interconnection (OSI)—provide terminal emulation for peer-to-peer host access to data and applications.

In the AIX TCP/IP environment, the network terminal access commands are `telnet` and `rlogin`. The overhead for TCP/IP protocols and `telnet/rlogin` processing can decrease performance of a business application. Figure 1 shows the data path differences between a locally attached ASCII terminal and a network-attached device. The overall path length for a direct attach-

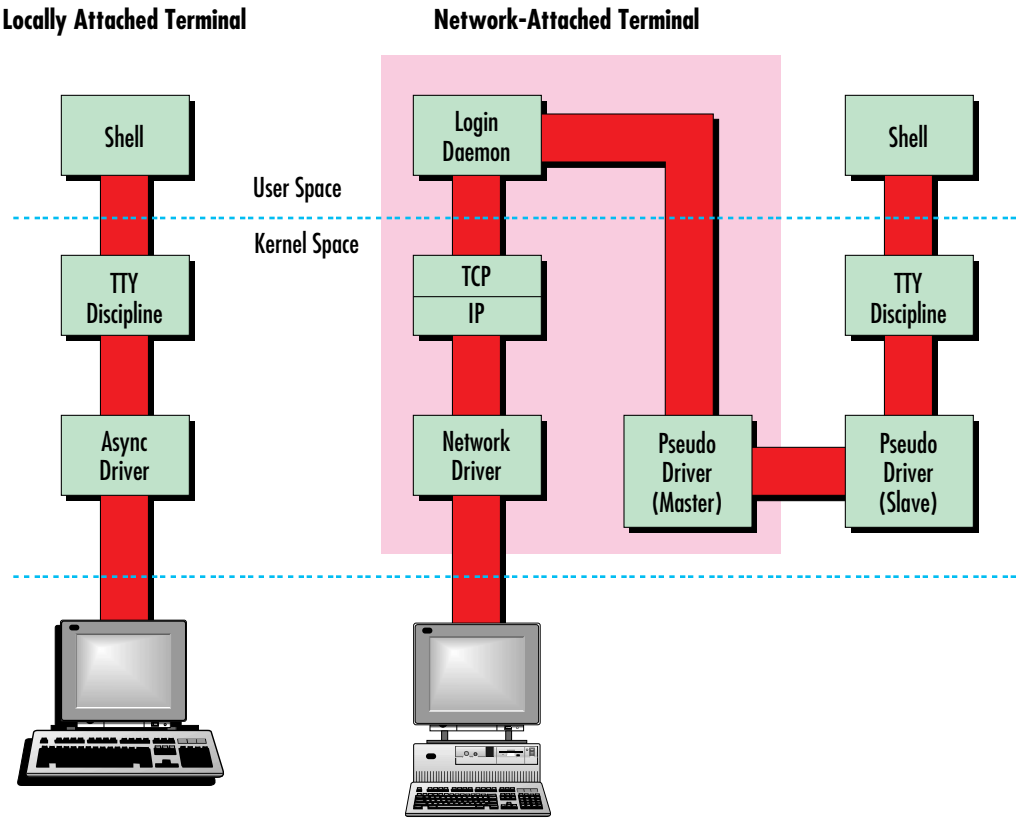


Figure 1. Local versus network terminal data path

ment is about one-third that of the network-attached terminal.

Network-attached devices using TCP/IP over Token-Ring, Ethernet, or Fiber-optic Data Distribution Interface (FDDI) networks can provide higher bandwidths than a directly attached async terminal. However, network attachment causes a heavy protocol processing load for the host, including the following:

- Kernel-level processing
  - Network driver
  - TCP/IP protocol
  - The tty protocol
  - Pseudo-drivers
- User-level processing
  - Login daemon execution

This heavy load can be reduced by shortening the code path and off-loading the network protocol and daemon processing to firmware (the shaded area in Figure 1). The shell environment in traditional AIX uses pseudo-tty (pty) as I/O simulation for remote access. In the accelerated environment, a new type of reengineered pseudo-device, high-performance tty (hty), can provide equivalent functions with special firmware assistance. Because the Network Terminal Accelerator adapter and software can provide this capability and efficiency, they are ideal for a host with many users.

Highlights of the Terminal Accelerator

The Network Terminal Accelerator adapter can be attached to any standard Ethernet with an appropriate transceiver. Since the adapter supports both IEEE 802.3 and Ethernet Version II packet formats, it can replace the existing Ethernet adapter. Figure 2 shows the network topology with this adapter.

The software support includes a device driver, configuration methods, and diagnostics. The Network Terminal Accelerator requires AIX 3.2.5. The adapter can support two IP addresses: one

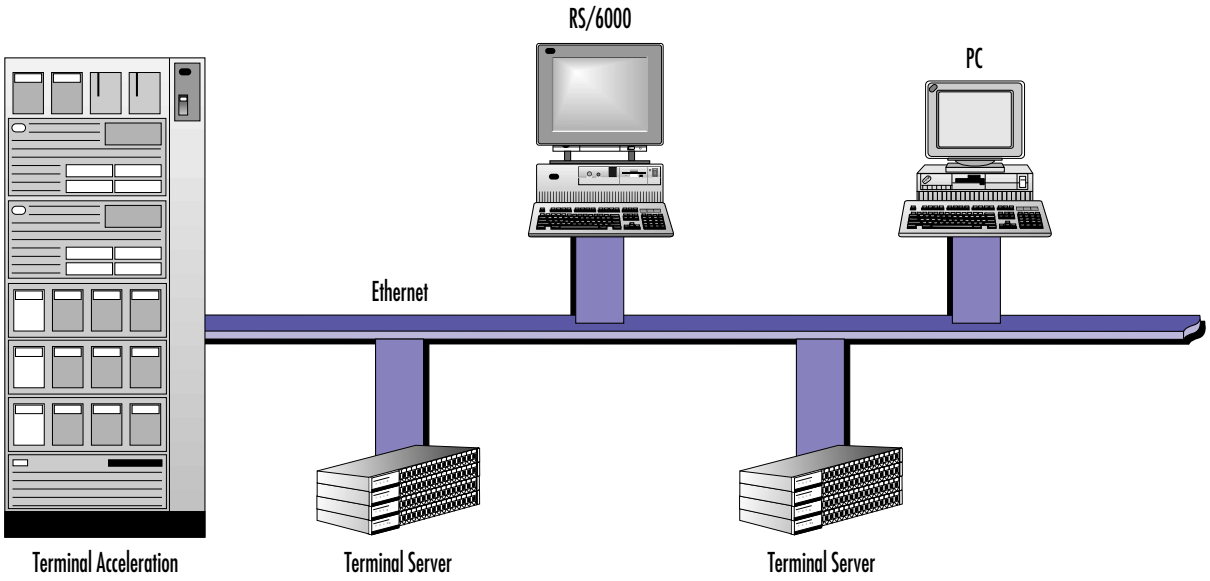


Figure 2. Large multi-user environment topology with terminal acceleration

for the off-loaded TCP/IP protocol layer with login support, and the other for existing AIX TCP/IP protocol running in AIX kernel space. The system administrator must be aware of both protocol stacks with their functional placements, capabilities, and limitations.

In a typical terminal server host environment, the off-loaded IP address will handle all the terminal users, while the other IP address and resources are intended for other uses, such as NFS, network routing, and so on. Figure 3 shows functional placements and packaging of both TCP/IP protocol layers. The adapter can be configured using System Management Interface Tool (SMIT) to define the IP address for the Ethernet connection, and also to define the number of hty devices available at system boot. Each hty device maps onto a terminal user access shell and becomes available for login.

The telnet/rlogin Sessions

The `telnet` or `rlogin` sessions can be initiated from any TCP/IP host on the network: a PC, another RISC System/6000, or a terminal server (such as the 7318 S20 described on page 58). The adapter has two models that support 256 or 2048 maximum sessions. Each hty device can be configured as a "call out" or "call in" port. If configured for call out, a `telnet` or `rlogin` session is automatically established to a destination IP host during the open system call. In a distributed workgroup environment, this support can connect to a modem pool or shared printers that are directly connected to the network.

Ethernet packets for protocols other than `telnet` or `rlogin` will be passed on to the protocol stacks in the host for processing. The address assignment is set up using SMIT. This enables the system to support all existing protocols, yet the adapter can still efficiently handle the LAN terminal protocol processing for the users attached to the network.

The adapter also supports the Simple Network Management Protocol (SNMP) agent function and can be centrally managed by a manager product such as NetView/6000.

Conclusion

With a dedicated microprocessor, the adapter can efficiently off-load the LAN protocol stacks for TCP/IP. Each CPU interrupt from the adapter to AIX can contain multiplexed data from many sessions. The adapter, which supports the 32-bit busmaster feature, can move large volumes of data efficiently. The protocol off-loading also reduces the number of context switches required to process incoming data. This solution, transparent to both users and application software, can

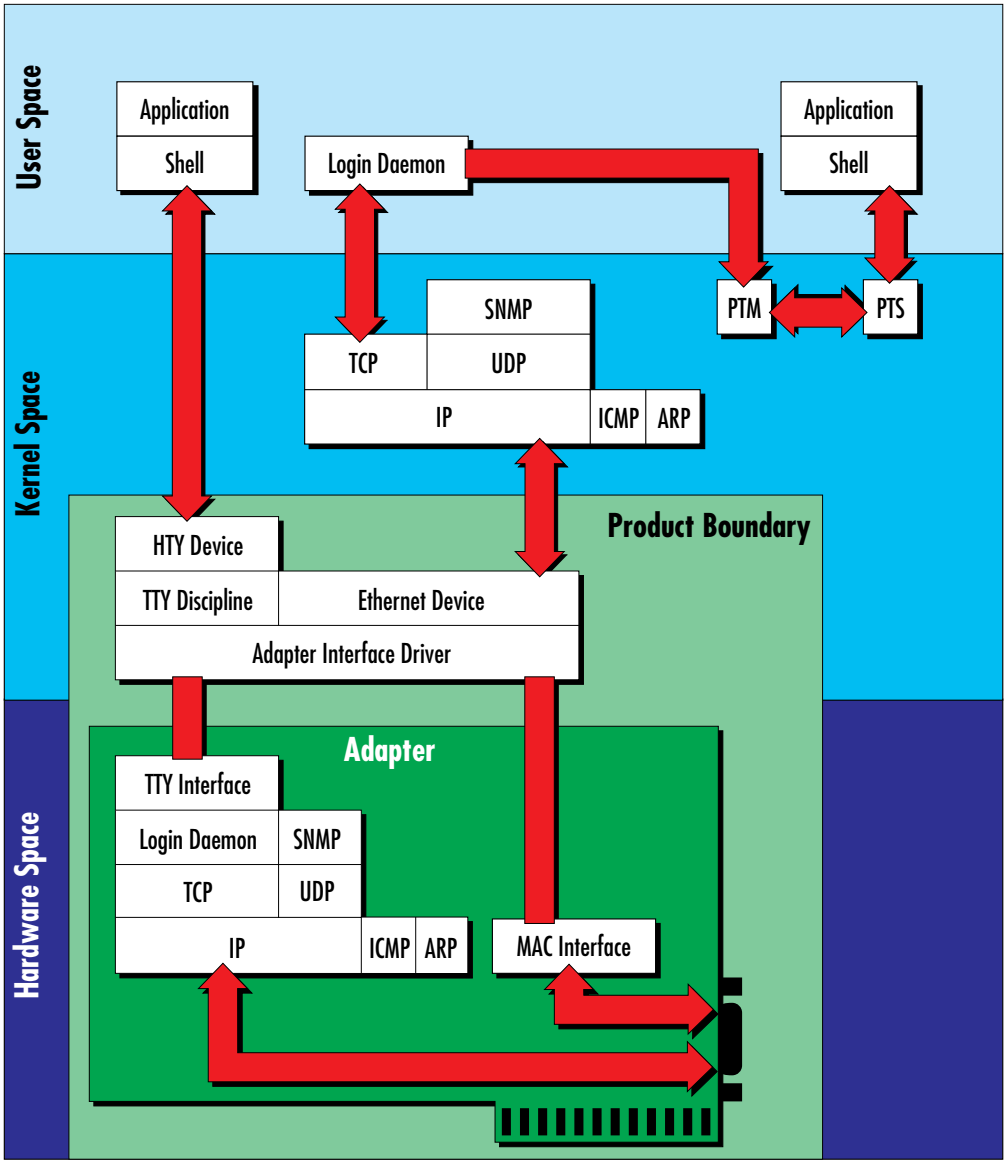


Figure 3. Structural diagram for both TCP/IP protocol stacks

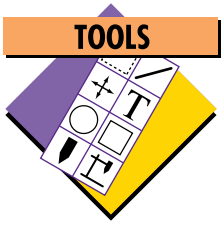
increase `telnet` and `rlogin` efficiency by 300%, freeing the CPU for more application processing.

Eddie Ho, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Internet: eddieho@chang.austin.ibm.com. Mr. Ho is a senior programmer in AIX Communications. He has a BS in Computer Science from the University of Wisconsin and an MS in Computer Science from North Dakota State University.

Jim Gallagher, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Internet: jrg@bach.austin.ibm.com. Mr. Gallagher is an advisory engineer in the Async Communication Development area. He has a BS in Electrical Engineering from Auburn University in Alabama.

Kent Malave, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Internet: kent@bach.austin.ibm.com. Mr. Malave is a staff programmer in the TCP/IP Communications area. He has a BS in Electrical Engineering and Computer Science from the University of Oklahoma.

Dave Phipps, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Internet: dlphipps@bach.austin.ibm.com. Mr. Phipps is a staff programmer in the AIX Communications area. He has a BS in Applied Mathematics, Engineering, and Physics and an MS in Computer Science from the University of Wisconsin.



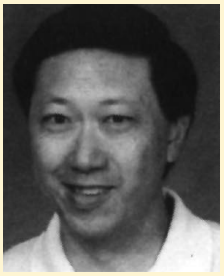
Distributed Performance Management Tools

By James N. Chen and Niels Christiansen

This article presents an overview of performance management concepts that are useful in understanding how the IBM AIX system, network, and performance management tools work together in a heterogeneous UNIX environment. New performance management tools in the Performance Toolbox for AIX (PTX) are highlighted.

In a client/server environment, smooth and optimal performance is achieved when all available resources exist in harmonious balance. Reaching this balance is difficult because it requires juggling many machines across systems and networks. Clearly, any attempt to attain this system and network management harmony depends on effective performance management.

Effective performance management can appear to be an elusive goal. To find this "sweet spot" of optimal performance, system administrators must continually adjust and readjust a multitude of parameters. The resources of an enterprise are in balance when the utilization of all system resources is high but not overstressed, all user service goals are met, response times are minimal, and throughputs are maximized. In a perfect world, optimum performance management means *no waiting*; resources are always available.



James N. Chen

Management in a Client/Server Environment

In a distributed client/server environment, system, network, and performance management disciplines can be viewed as an interlocking triad. (See Figure 1.) Each discipline has some functions and data in common with the other two, but each discipline also has individual focus, scope, data, and process model requirements. Although system overheads, protocols, and audience characteristics can vary widely, there is a strong need for commonality of Application Programming Interfaces (APIs) and User Interfaces (UIs) so that various tools can be used together without major compatibility problems among applications. The following are examples of IBM tools in each of these areas:

- System management tools, such as the System Management Interface Tool (SMIT), Distributed SMIT, and Visual System Management (VSM), focus on the details of managing resources for individual systems.
- Network management tools, such as NetView, System Monitor, and Trouble Ticket/6000, focus on global management of system network resources based on Simple Network Management Protocol (SNMP).
- Performance management tools, such as Performance Toolbox for AIX (PTX), focus on detailed resource utilization at individual system nodes across the network.

Performance management tools have very special requirements: low overhead, variable data granularity, high-density data presentation, and high-bandwidth data transport and storage. At the same time, these tools have a common need to access configuration and network data managed by system and network management tools. For these tools to be used both separately and in concert, some degree of overlapping functionality is useful.

Performance Management Phases

Primary performance management tasks can be grouped into four major phases. By using the proper performance management tools in each of these phases, users can obtain optimal performance from the hardware and software they are developing or have purchased.

Component Design and Development

In the component design and development phase, software engineers must create high-quality programs within a specific time that provide functionality and performance at a given price level. To meet the time and cost constraints of customer demands, there are trade-offs between function and performance. Performance-conscious programmers generally focus on the micro-optimization of their individual components. To achieve this optimization of performance, they need detailed modeling, calibration, and measurement tools to visualize the resource utilization of their components in conjunction with the system.

System Integration

The system integration phase brings together component pieces of hardware, operating system, and applications to create a functioning system that can operate by itself or with other systems. At this time, software engineers should focus on the utilization of shared client/server system resources, such as CPU, disk, memory, and communication channels. These resources may require additional trade-offs and usage prioritizations between component pieces to achieve optimal overall system performance.

Software engineers need tools that enable them to see the big picture as well as to focus on micro sections of the system. Such tools enable system designers to create system performance profiles that can be used by component designers for improving component performance as well as by Information Services (IS) personnel in their pre-operations planning phase.

Pre-Operations

In the pre-operations phase, IS personnel design and layout systems and networks to meet the needs of their particular installation. Before deployment in their production environment, systems must be integrated so that the equipment and software will meet the performance capabilities of their planned systems and networks. IS personnel then set performance goals and service

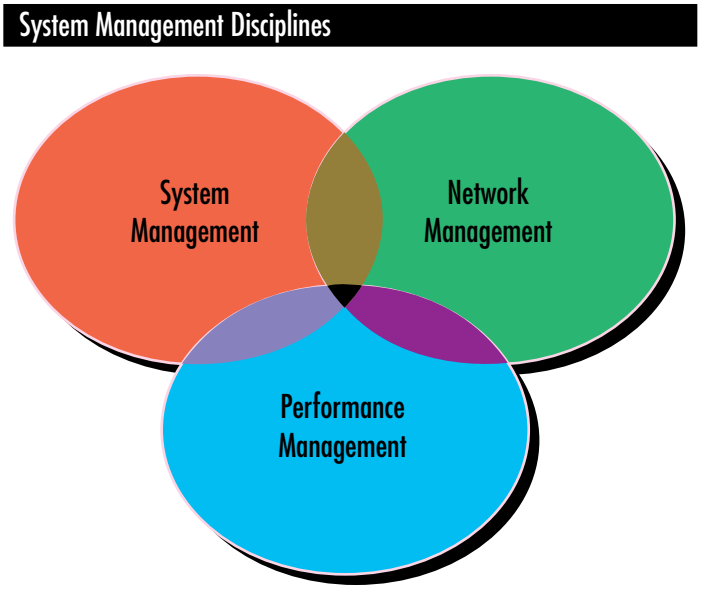


Figure 1. The interlocking relationship of management disciplines

levels that determine what will be considered normal or abnormal performance in the operations phase. Performance calibration tools such as PTX are key to understanding and setting parameters for service-level monitoring.

Operations

In the operations phase, the systems and networks have been deployed into the customer's production environment. System administrators must monitor key resource parameters to detect out-of-range performance, find persistent abnormalities, and fine-tune system resource utilization. System administrators must also be able to determine if performance and service-level goals are being met. They require tools to help diagnose and debug performance problems online. They must also collect long-term data for trend analysis and future capacity planning.

All phases of performance management involve performance calibration and benchmarking. To accomplish these tasks, system administrators need accurate instrumentation of systems and applications, and tools that can read, record, and present this data coherently. System administrators use this data to establish realistic performance goals, track actual performance to desired performance, and implement trend analysis and capacity planning. Recorded performance data is also key for the diagnosis, debugging, and post-mortem analysis of disastrous system failures.

PAIDE and the Agent Component of PTX

- Support for selected non-AIX platforms
- Server side recording of statistics to local file

Manager Component of PTX

- Enhanced recording facilities
- A recording analysis program
- A set of utility programs to post-process recording files
- A dedicated exception monitoring program

Figure 2. Enhancements over PTX 1.1

Performance Management Users

Performance management tools are used by performance analysts, programmers, system and network administrators, support personnel, and general users. Each user group may have special requirements for performance tools suited to their jobs. Most tools are designed with a special group in mind.

- **Performance analysts** characteristically have detailed knowledge of the inner workings of system hardware and software. After many years of experience and detailed study of performance data, they can usually spot problems by examining a few key indicators. Performance analysts need tools that provide detailed information about the inner workings of the system.
- **Programmers** typically focus first on functionality and consider performance aspects only after the coding is completed. This can be especially problematic if they rely heavily on system services without realizing the resource requirements of these services. Programmers need tools that show them if their programs are using system resources efficiently.
- **System and network administrators and support personnel** are primarily concerned with keeping the systems and networks up and running for their customers. They need forecasting tools to alert them before problems happen, diagnosis and debug tools to help them when problems occur, and recording and analysis tools to help them understand how to prevent these problems from occurring in the future.
- **End users** typically need some sensory feedback to inform them of the availability of system resources. They need tools such as alarm lights and resource usage meters to let them know that work is being done.

All groups measure performance management tools by the key criteria of response time, resource availability, and throughput. The Performance Toolbox has facilities to help each of these user groups address their key performance problems in the four phases of performance management.

Performance Tuning Model

A simple performance tuning model consists of six main steps:

1. Monitor key performance statistics.
2. Log and record statistics.
3. Analyze resource utilization statistics and identify bottlenecks.
4. Prescribe a remedy to a performance problem.
5. Set and adjust alert/alarm conditions and notification policies.
6. Make prescribed tuning adjustments and return to step 1.

With the proper performance tools, these steps can be repeated to obtain a fine-tuned environment that meets the desired performance goals. Each step in this model can be done by a variety of tools—manual, semi-automatic, and automatic. In an ideal "lights out" operation, this model would be totally automatic; tools would be preprogrammed to perform each step without human intervention. However, today's technology has not yet reached the point at which total automation occurs with high reliability and consistency. Therefore, it is very important to have semi-automatic tools that perform data collection, sorting, and preliminary analysis, while deferring to trained human operators on critical decisions.

Some tools can operate in a semi-automatic mode by monitoring key statistics and notifying a user when certain actions and decisions need to be made. For example, PTX can alert an operator to pathological performance behaviors to enable the operator to respond with the appropriate corrective actions. PTX also offers users the option of customizing their set of tools with a flexible and expandable toolbox format. Its powerful graphic presentation facility can also be dynami-

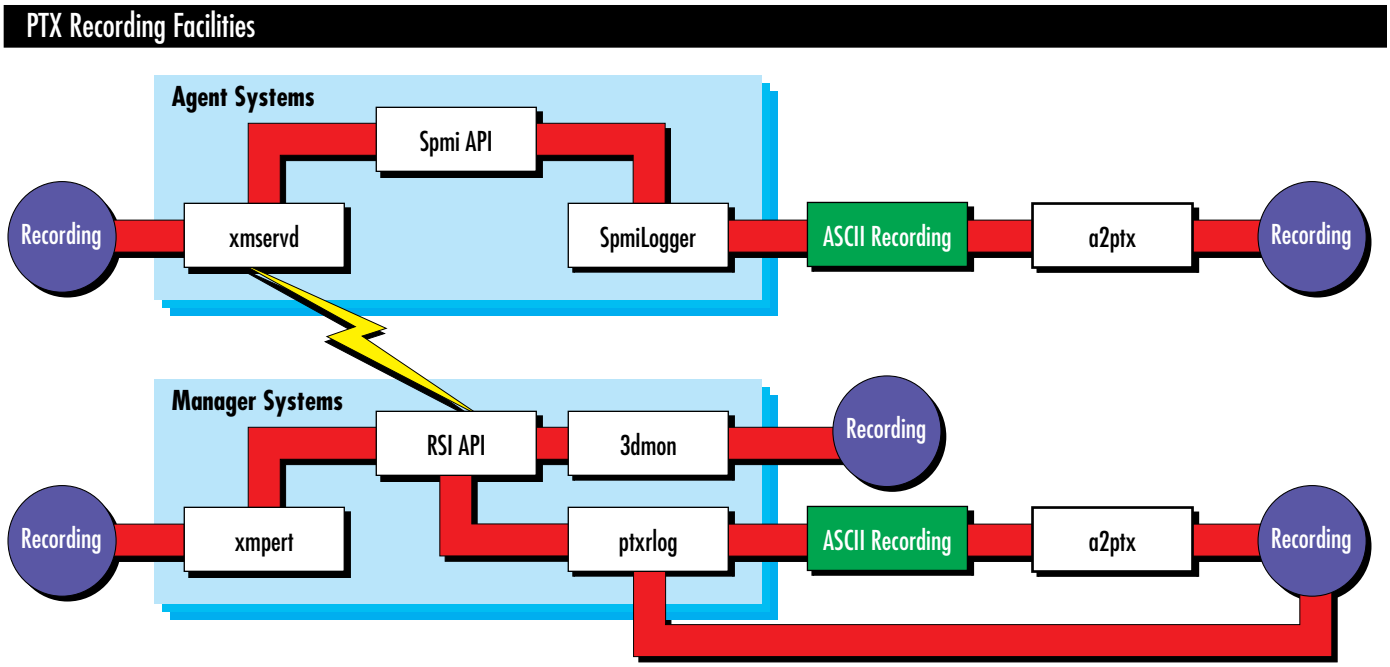


Figure 3. PTX recording facilities

cally customized for individual users. The Performance Toolbox Version 1.1, as described in the article "AIX Performance Toolbox/6000" (*AIXpert*, November 1993, pp. 24-28), provides a good basis and framework to do this. This article also provides a description of the AIX Performance Aide (PAIDE) product, which is the agent component of PTX.

Key enhancements described in the next section build upon and strengthen this tuning model in a distributed heterogeneous client/server environment.

Enhancements in PTX Version 1.2

Two versions of PTX are announced for shipment in the third quarter of 1994. Version 1.2 runs on AIX Version 3.2.5 and Version 2.1 runs on AIX Version 4.1. The two versions are almost identical; they differ only in the supported statistics necessitated by differences in the operating systems. Figure 2 shows the most important enhancements in PTX over Version 1.1.

Support for Selected Non-IBM Platforms

Beginning with Hewlett-Packard Series 9000/700 systems running HP/UX Version 9.03, PAIDE and the agent component of PTX contain everything necessary to install and run the agent on non-AIX systems. Support for HP 9000/700 will be included in the initial shipment of PTX and PAIDE Versions 1.2 and 2.1. Agents for Sun SPARCstations™ with SunOS 4.1.3 will follow shortly. The non-AIX agents will provide all the functionality of the AIX agents, except the ability to export their statistics to Simple Network Management Protocol (SNMP). A basic set of statistics is available across all agents. However, because of differences in hardware and software and varying levels of instrumentation, each platform requires a customized set of statistics to monitor its most important probing points.

Server-Side Recording

In response to customer requests, the `xmservd` daemon of PAIDE and the agent component of PTX has been enhanced to enable recording directly to a local file instead of sending packets over the network. A configuration file provides the following information to the `xmservd` daemon:

- When to start and stop recording
- Which statistics to record
- The sampling interval for each statistic
- How to create and how long to preserve recording files

The overhead of recording from `xmservd` is normally so insignificant that it makes continuous

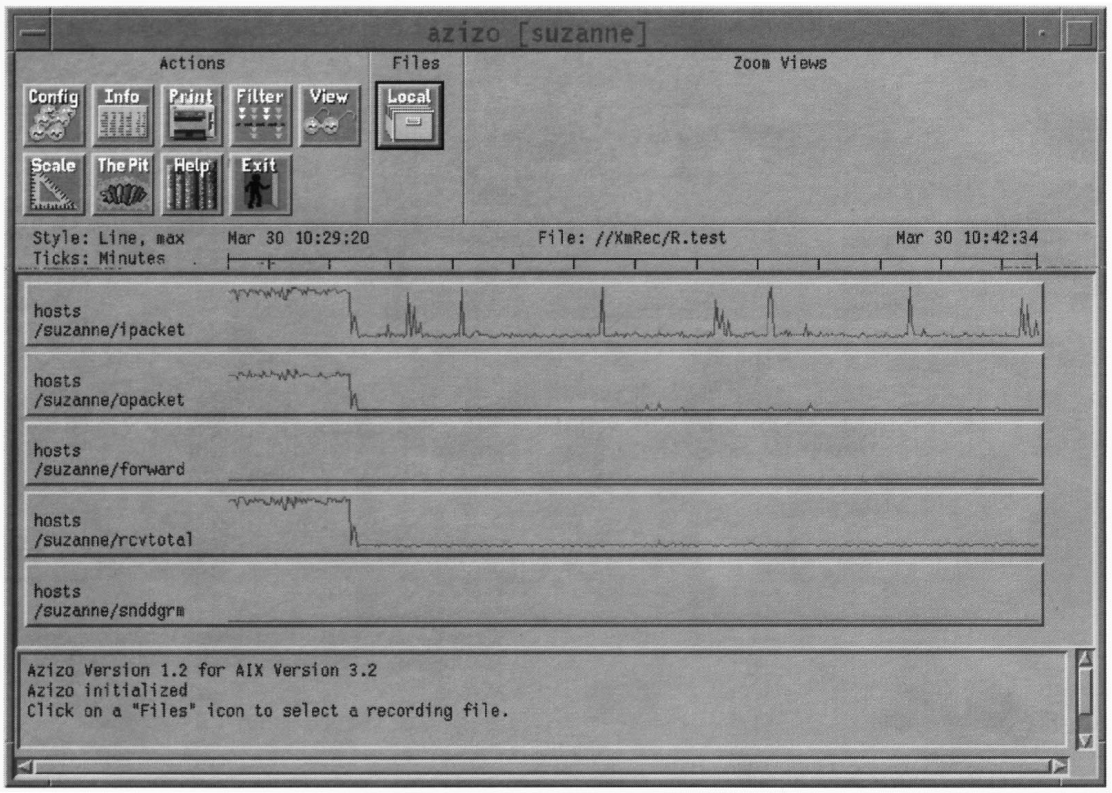


Figure 4. The azizo main window

recording feasible. With continuous recording, you always have the performance data for analyzing a performance problem that you did not detect while it occurred. The recordings produced by `xmservd` can be played back with the `xmperf` monitor and post-processed and analyzed with other programs of the PTX manager component.

Enhanced Manager Recording Facilities

The manager component of PTX has also been enhanced to provide more versatile recording environments. For example, you can now use PTX to record monitoring sessions from the `3dmon` monitor. Another improvement, `ptxrlog`, enables you to record from a remote system without having an active graphical monitor.

Figure 3 shows the various recording facilities of PTX Versions 1.2 and 2.1.

Recording Analysis Program

PTX Version 1.1 had only two ways to analyze a recording: play it back with `xmperf` or print out all details with the `xmtab` program. The new versions of PTX provide a new program—`azizo`–to analyze recordings. This program enables you to work with graphical or tabular views of the data and zoom-in on any part of a recording in any detail needed. The `azizo` program is designed to work in concert with the recording file utility programs described below.

Figure 4 shows the `azizo` main window. Figure 5 shows an example of `azizo`'s zoomed-in graphical window, which gives a partial view of the data in the recording file.

Recording File Utility Programs

With many more sources of recording files, a need arises for programs to split, merge, tabulate, and process recording files. For example, if the `xmservd` daemon has recorded activity on a client system and a server system, it may be interesting to play those recordings back simultaneously or analyze them together. The program `ptxmerge` can merge the two recordings into one file. With the `ptxsplit` program, you can choose from many options to split one recording file into several. You can even create recording files from ASCII input files so that data from other sources can be analyzed together with true recordings. Also, if you want to use other analysis tools, new PTX programs enable you to convert recordings to a variety of output formats for input to spreadsheet or presentation programs.

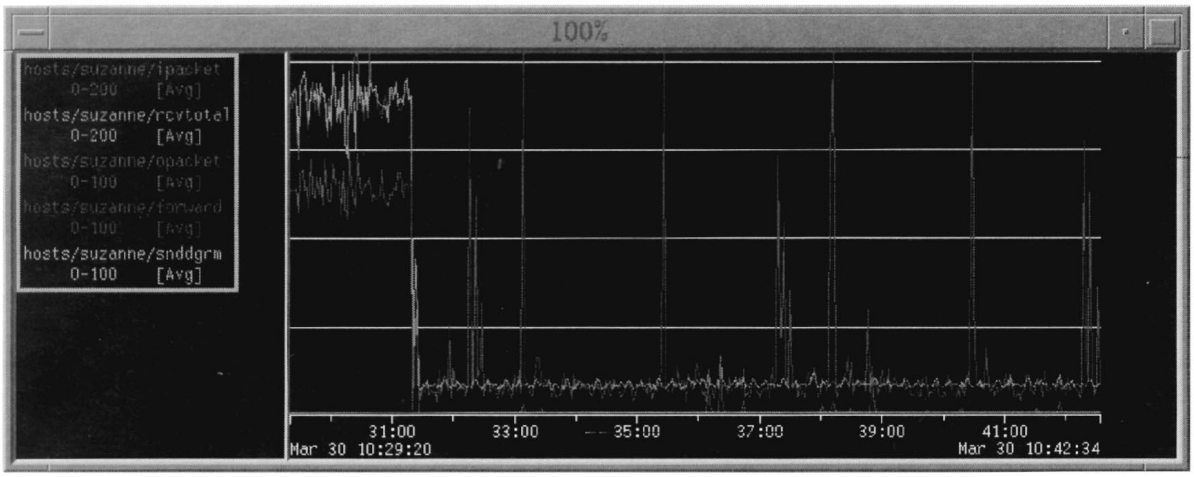


Figure 5. An azizo zoomed-in graph

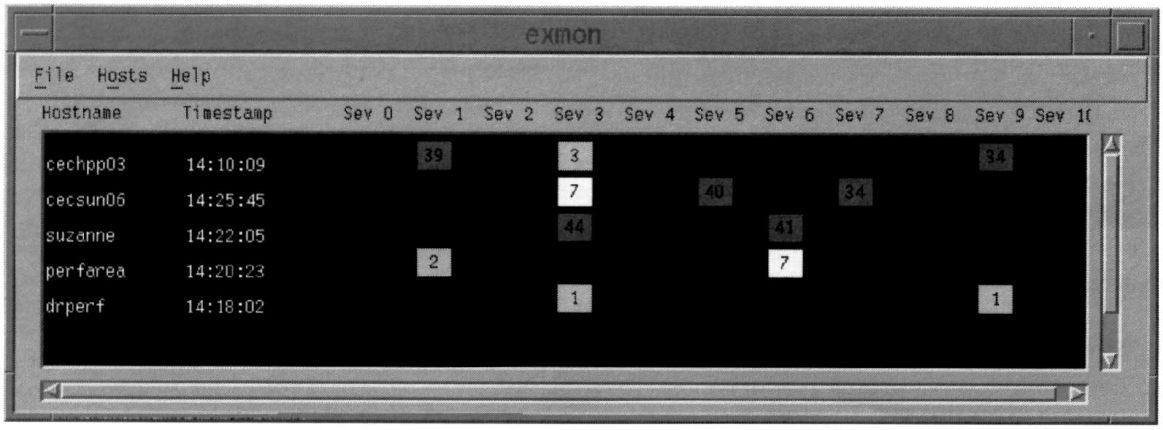


Figure 6. The exmon main window

Exception Monitoring Program

A new program called `exmon` monitors alarms generated by the filter daemons on remote systems. A matrix display of the alarms received brings the latest alarms to your attention and tracks alarm frequency. The `exmon` program also enables you to execute other programs to further analyze the performance on any system that generates an alarm. For example, you can invoke `3dmon` by clicking on a host name and selecting `3dmon` from a user-customizable menu. Figure 6 shows the matrix display of `exmon`.

Performance Management Trends and Directions

Performance tools cover the spectrum from manual to automatic operations, from character-based to live color graphic presentations, from single-node analysis to large distributed networks of heterogeneous machines, from novice to expert tools. These tools can be broadly viewed as first-, second-, or third-generation tools.

First-generation tools tend to be character-based, manually operated tools that only experts would use and understand. They usually present volumes of raw data about a single subsystem; this data typically must be decoded and interpreted by someone very knowledgeable in specific subsystems. Most first-generation tools do not work well, or at all, in a heterogeneous client/server environment. Their data and interfaces are not integrated or correlated across a family of applications. Typical UNIX tools such as `vmstat`, `netstat`, `iostat`, `sar`, and `ps` fit into this category.

Second-generation tools, such as the Performance Toolbox for AIX, tend to use live and interactive color graphic presentations that are easy to customize and use. A semi-automatic mode of operation relies on the pattern recogni-

tion skills of the user to learn to choose pathological performance patterns, characterize them, and then feed back that information to the system so that the system can later detect these patterns and automatically notify the user, or take some prescribed corrective action.

Second-generation tools are designed to operate in a heterogeneous client/server environment. These tools enable users to select the granularity and grouping of statistics they want to collect depending on the resources available. The data collected through program interfaces fits into more globally accepted syntax and semantic definitions so that it can be processed by a variety of applications. These tools usually provide data filtering and alarm facilities for interoperation between different applications.

Performance management tools should address the needs of a wide audience, from novice to expert.

Third-generation tools will be a more fully automated and integrated suite of tools that can be used by users with less detailed system and network knowledge and experience. Because the tools will incorporate more sophisticated analysis and control algorithms, the results will be more accurate. These tools will typically be more knowledge-based so they can automatically learn to recognize new pathological performance patterns. Once recognized, they can respond to correct those new performance problems by balancing available resources or by making detailed recommendations for new resources. Human intervention will be minimized.

Conclusions

Performance management tools are specialized tools that work with a suite of system and network management tools. They should address the needs of a wide audience, from novice to expert. Together, they should allow users 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. These tools should be easy to install, configure, and use. They should be flexible and expandable. Finally, they should be based on a common set of instrumentation and interoperate in a heterogeneous networked environment.

Key enhancements to the Performance Toolbox for AIX position it as an advanced second-generation performance management tool that can easily evolve into a key third-generation tool for AIX and other platforms.

James N. Chen, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Internet: jchen@perfmap.austin.ibm.com. Mr. Chen is a senior programmer in the Systems Management Architecture and Development area. He has been involved in AIX software development, technical marketing, support, software architecture, and performance tools since 1982. He has a BS in Electrical Engineering from Purdue University and an MS in Electrical Engineering from Rutgers University.

Niels Christiansen, Encee, Inc., 1015 Quail Park Drive, Austin, TX 78758. Internet: nchris@perfmap.austin.ibm.com. Mr. Christiansen worked in system engineering for IBM Denmark from 1965 to 1990. He is now president of Encee, Inc., a vendor to IBM's Systems Management Architecture and Development area. Mr. Christiansen currently works with design and implementation of performance tools. He received a BS-equivalent degree in Mechanical Engineering from the Copenhagen Teknikum College.