IBM's Magazine for AIX Developers September 1997

AIXpert

FOCUS ON 64-BIT ARCHITECTURE

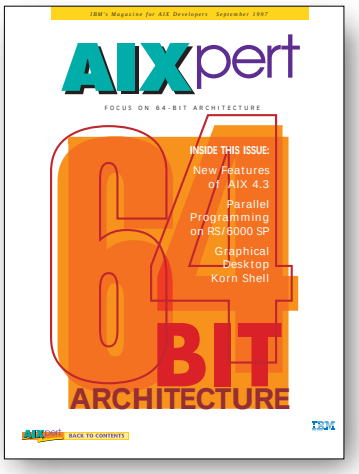
INSIDE THIS ISSUE:

64 BIT ARCHITECTURE

IBM

AIXpert DIGITAL VERSION IBM

TABLE OF CONTENTS



Click here to view a full-size version of this issue's cover.

Click here to go to a complete version of this issue for printing.

SEPTEMBER 1997

AIXpert BACK TO MAIN MENU

AIX Performance Hits a New (64-bit) Peak

COMMENTARY

In this issue, we try to pique your curiosity with a peek at what's coming in the anticipated new AIX release (coming soon to an RS/6000 near you), plus a collection of other useful AIX topics. Learn about the 64-bit features of the next AIX release in our keynote article. With a 64-bit capability, applications run faster and handle data more efficiently, and of course, you can still use applications based on the previous AIX releases. The article also addresses issues to consider when porting your applications to a 64-bit implementation. To prepare you for the new release, we've also included a schedule of training classes offered worldwide. Be sure to check when the classes will be offered near you.

This issue also brings you the first in a series of articles exploring parallel programming on RS/6000 Scalable POWERparallel (SP) systems. This article describes the different programming models and their features. Java aficionados will be interested in the article on IBM's optimizing native code Java compiler for AIX. It explains the differences between optimizing compilers and Just-in-Time (JIT) compilers and provides some performance data for different compiler benchmarks. Programmers and system administrators will both enjoy reading about portable programming via the graphical desktop Korn Shell, useful for all Common Desktop Environment (CDE) implementations. Programs written in this shell language can easily be migrated to C/Motif for performance improvements.

To help you ensure a quality end-product, we've also included an article on avoiding the release of flawed software through an effective configuration management scheme. This is a topic we're all concerned about.

Rounding out this issue is an article that shows you how to use AIX's terminal mirroring function to reduce problem determination time and improve training in a networked environment. And of course, we have the popular AIX question-and-answer section that includes "popular" questions that developers ask the technical support line. Check it out—perhaps they've answered a question you've been wondering about.

George Noren

George Noren, IBM Corporation, Internal Zip 1034, 11400 Burnet Road, Austin, TX 78758. Internet: geo@austin.ibm.com. Since joining IBM in September 1979, Mr. Noren has written hardware and software manuals, including AIX and RS/6000 manuals, and was a member of the InfoExplorer™ design team. He has worked as a system administrator for several AIX systems and is a Certified AIX System Administrator. He is currently Editor in Chief of the World Wide Web site for IBM's Solution Developer Program (www.developer.ibm.com) in addition to his work with AIXpert *Magazine. 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

Overview of AIX 4.3

By Ahmed Chibib

This article discusses the characteristics of the new 64-bit architecture of the RS/6000 and the new 64-bit capabilities of AIX 4.3. It also presents issues to consider when porting an application from 32-bit to 64-bit.[1]

The need for 64-bit processing is driven by the increasing demands of today's applications. Although there is no such thing as a "typical" 64-bit application, most need to address large amounts of storage. For example, as database engines grow in capacity, they must manage larger and larger amounts of data in memory buffer pools to meet their performance criteria. In addition, they need more processing power to avoid becoming CPU-bound. Data warehousing applications are another example of the need for increasing storage capacity and high performance.

In the technical application arena, greater processing power enables larger problems to be solved, and larger problems typically mean larger arrays of grid points or other mathematical representations of the physical problem being analyzed. For many technical applications, storage requirements grow exponentially with the dimensions of the problem.

Overview of the RS/6000 64-bit Architecture

Applications running on AIX 4.3 can now be 32-bit or 64-bit. When application developers create an application for AIX, they must chose the appropriate architecture, based on several criteria, including the type of application and the performance requirements. A 32-bit application executes in an environment with 4 GB of addressability (virtual address space). A 64-bit application executes in an environment with much larger addressability (over 16 billion gigabytes).

The 32-bit applications can be run on any RISC System/6000 system, but 64-bit applications can be run only on a 64-bit RISC System/6000 with AIX 4.3. Both 32-bit and 64-bit applications (and libraries) can be compiled and linked on both 32-bit and 64-bit systems and AIX 4.3.

The RS/6000's 64-bit architecture has many benefits, but the choice to develop for a 64-bit environment should be based on the type of application and its requirements. The issues to consider include the performance requirements, the type of objects and archive files to be used, how the execution environments differ between 32-bit and 64-bit, and tools that are available for 64-bit architecture.

[1] This article is adapted from the *AIX 64-bit Migration Guide*, which may be viewed in its entirety from the following Web site address: http:/www.developer.ibm.com/sdp/library/aix4.3/.

Performance

The 64-bit address space can dramatically improve the performance of applications that manipulate large amounts of data—multiple gigabytes or larger. This data can be within the application or obtained from files. If a 64-bit application can contain the data in its address space—either created in data structures or mapped into memory—there will be a performance gain compared to a 32-bit application where the data would not fit into the address space.

There are two reasons for this performance improvement. First, the system call overhead of reading and writing files can be avoided by mapping the files into memory. Second, 64-bit systems can support physical memories that are larger than the addressability of 32-bit applications, so 64-bit applications are needed to make full use of the physical memory available.

Although the same source code can be used to create both the 32-bit and 64-bit application, the 64-bit application will be the same size or larger than the 32-bit application. In addition, 64-bit applications often run slower than 32-bit applications, unless they use their larger 64-bit addressability to improve performance. For this reason, developers generally create 32-bit applications, unless 64-bit addressability is required by the application or it can be used to dramatically improve performance. The default mode for AIX development tools is to create 32-bit objects and applications.

64-bit Objects and Archive File Types

The 64-bit libraries and applications can be created only from 64-bit objects. A *64-bit object* is an object type (64-bit XCOFF format) created by a compiler or assembler in 64-bit mode; that is, the compiler or assembler has been requested to create 64-bit objects rather than 32-bit objects. The 32-bit XCOFF format was the only object type in all releases of AIX before AIX 4.3.

Developers cannot create an object or application using both 32-bit and 64-bit object files. While the system-provided archives and libraries contain both 32-bit and 64-bit object files, the linker selects the appropriate objects from the library based on the type of linking that is requested (32-bit or 64-bit) and creates an object or application of that type.

AIX has two archive file types. The first does not recognize 64-bit object files and cannot be larger than 2 GB. Prior to AIX 4.3, this was the only archive file type. The second archive file type recognizes both 32-bit and 64-bit object files and will work with files larger than 2 GB.

32-bit Versus 64-bit Execution Environments

Some differences between the 32-bit and 64-bit execution environments (or modes) include the following:

- All pointer types in 64-bit mode are 64 bits in size
- The C “long” type (and types derived from it) in 64-bit mode is 64 bits in size
- 64-bit applications can use 64-bit PowerPC instructions
- The size of a machine register is 64 bits in 64-bit mode

Theoretically, the maximum limit for 64-bit applications, their heaps, stacks, shared libraries, and loaded objects is millions of gigabytes. However, the practical limits are dependent on the file system limits, paging space sizes, and system resources available.

All C fundamental types, except “long” and pointer types, will be the same size in 32-bit and 64-bit compilation modes.

Tools for 64-bit Development

AIX supports the standard tools for building, examining, and debugging 64-bit applications. The `yacc`, `lex`, and `lint` tools work with source code for both 32-bit and 64-bit compilation.

Both 32-bit and 64-bit objects can be created using the C compiler and the assembler. Both 32-bit and 64-bit objects and applications can be created with the linker.

The following tools work with both 32-bit and 64-bit objects and applications: `make`, `ar`, `strip`, `dump`, `nm`, `prof`, `gprof`, `lorder`, `ranlib`, `size`, `strings`, and `sum`.

The dbx and xldb tools can debug both 32-bit and 64-bit applications.

The 64-bit Architecture of the PowerPC

For a computer system to be considered to have a true 64-bit architecture, it must handle data 64 bits in length; that is, a contiguous block of 64 bits (8 bytes) in memory must be defined as one of the elementary units that the CPU can handle. The instruction set must include instructions for moving a 64-bit datum, and arithmetic instructions for performing arithmetic operations on 64-bit integers.

The 64-bit architectures must also generate 64-bit addresses, both as effective addresses (the addresses generated and used by machine instructions) and as physical addresses (those that address the memory cards plugged into the machine memory slots). Individual processor implementations may generate shorter physical addresses, but the architecture must support up to 64 bit addresses.

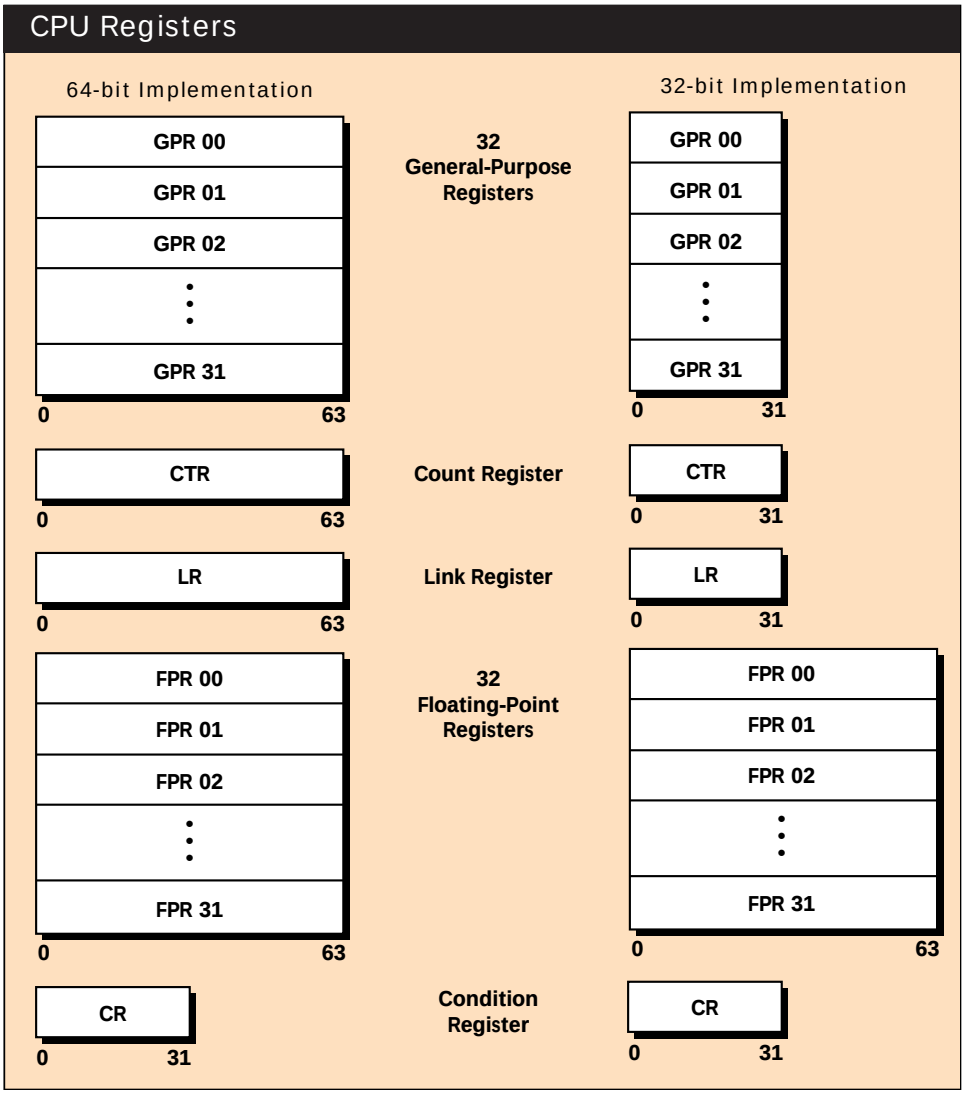


Figure 1. CPU registers for 32-bit and 64-bit architectures

PowerPC and Binary Compatibility

The IBM PowerPC has some important advantages compared to other processor architectures. From the beginning, it was designed as a 64-bit architecture, with 32-bit mode as a functional subset. Its 64-bit capability is *not* an adaptation of an existing 32-bit architecture. This design makes binary compatibility easier to maintain.

Binary compatibility with the current PowerPC processors is an important aspect of the 64-bit version of PowerPC. The 32-bit and 64-bit specifications have several differences. As shown in Figure 1, the number of General-Purpose Registers (GPRs) remains the same, but these registers are 64 bits long, not 32 bits long. A few other control registers move from 32 bits to 64 bits in length. Floating-point registers do not change in size, because they conform to industry standards for floating-point, which require 32-bit or 64-bit length data.

Why Choose 64-bit Architecture?

A 64-bit architecture has significant advantages. Very long integers can be used in computations. It can manage very large file systems. And it has the capability of addressing huge address spaces, both in virtual and physical memory.

Figure 2 shows the size of the address spaces that can be managed, as a function

of the length of the address generated by the CPU.

Complex applications such as large databases, large numeric applications, and multimedia environments are a natural for 64-bit applications because of their need to manage and operate on larger and larger data sets. Some advantages that are important for these applications include the following:

- Supports very large programs. As shown in Figure 3, a 64-bit processor can support a much wider address space than its 32-bit equivalent, which makes it possible to develop much larger and more complex applications. Today's 32-bit applications are limited to 4 GB.
- Can manage large volumes of data. The 64-bit architecture can manage very large amounts of data, possibly in real memory. This simplifies the task of the applications that typically handle large data sets, for example, those in multimedia, statistics, or database.
- Can handle larger files. A 64-bit environment can easily manage huge files, file systems, and databases without forcing the application to explicitly handle the situations in which data sets larger than 2 GB are needed. Today's most sophisticated applications already support this capability. AIX Version 4 itself supports files larger than 2 GB.
- Supports larger physical memory. The physical addresses generated by a 64-bit CPU are up to 64 bits in length. This eliminates the 4 GB limit in real memory of all 32-bit architectures.

64-bit Enabled AIX 4.3

AIX 4.3 is a universal upgrade for all RS/6000 systems currently running AIX Version 4.x, as well as future 32-bit and 64-bit systems.

Address Length	Flat Address Space
8 bit	256 bytes
16 bit	64 kilobytes
32 bit	4 gigabytes
52 bit	4,000 terabytes
64 bit	16,384,000 terabytes

Figure 2. Address space sizes

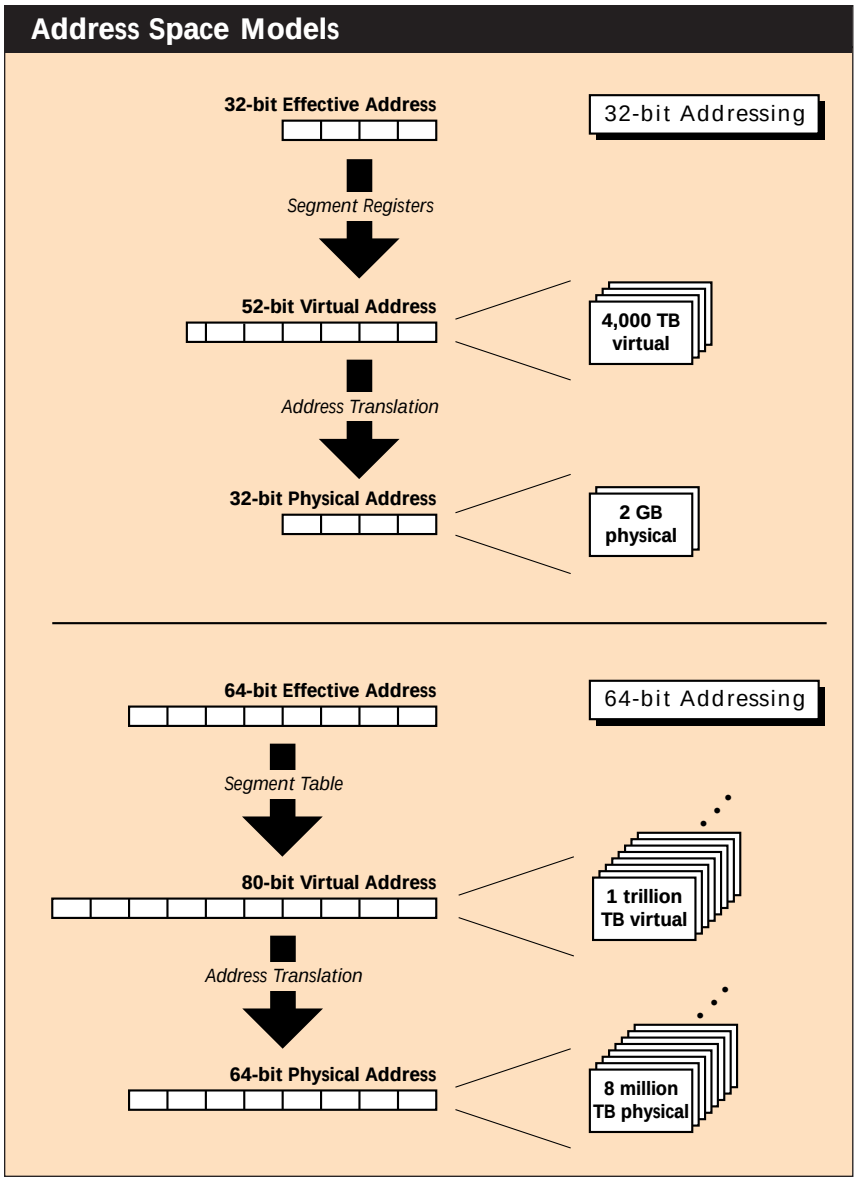


Figure 3. Address space models

End users should find 64-bit applications and 64-bit application support on AIX 4.3 completely transparent. They can run both 32-bit and 64-bit applications simultaneously. When they run 32-bit processes on 64-bit platforms, the execution is transparent and identical to executing on a 32-bit platform.

Platforms and Applications

Figure 4 summarizes the platforms on which 32-bit and 64-bit applications can run.

System Limits

As a result of the new 64-bit capabilities of AIX, system limits have been expanded as shown in Figure 5.

	32-bit Applications	64-bit Applications
32-bit Platforms (existing RS/6000s)	**32-bit applications run as they do on previous releases of AIX without modifications.** **Binary compatibility is maintained.**	**64-bit applications will fail to execute when run on any 32-bit system.**
64-bit RS/6000 Platforms	**Execution of 32-bit applications on 64-bit platforms should be transparent and identical to executing on a 32-bit platform.** **Binary compatibility is maintained.**	**64-bit applications should execute in a manner similar to an analogous 32-bit application.**

NOTE: Applications developed on AIX 4.3 are not backward compatible with previous releases of AIX.

Figure 4. Application platforms for 32-bit and 64-bit architectures

	Prior to AIX 4.3	AIX 4.3
File descriptors per process (both 32-bit and 64-bit processes)	**2,000**	**30,000**
System open file table	**200,000**	**1,000,000**
Number of shared memory IDs	**4,096**	**4,096**
Number of kernel threads per process (both 32-bit and 64-bit processes)	**512**	**32,768**
Number of processes	**131,072**	**131,072**
Number of threads per system	**262,143**	**262,143**

Figure 5. System limits

Figure 6 shows limits for 64-bit executables.

The File System

AIX 4.3 supports file sizes in excess of 2 GB. The 64-bit processes can open files of any size (in excess of 128 GB) without specifically indicating that they "understand" large files.

In the AIX 4.2 large file implementation, an application that is dealing with large files must have its file-size limit set to infinity. In the AIX 4.3 64-bit implementation, there are true 64-bit limits. Processes operating in 64-bit mode and those that are large-file enabled use the 64-bit limits. The 32-bit processes that are not large-file enabled will still see 32-bit limits, with limits exceeding 2 GB treated as infinity.

Shells

The `ksh`, `bsh`, and `csh` shells each have a built-in `ulimit` command. Prior to AIX 4.3, the maximum value of any `ulimit` that could be specified was a 32-bit value. In AIX 4.3, `ulimit` values as large as 64-bits can be specified in each shell. This is the only functional change made to each of the AIX 4.3 shells.

New Features of AIX 4.3

The new functionality in AIX 4.3 is not only available to 64-bit applications, but 32-bit applications can use it as well. The only exception is functionality that results directly from the larger address space.

The following sections describe the key new features available in AIX 4.3.

16 GB of real memory. This feature allows up to 16 GB of physical memory to be supported by AIX. This support is intended for Enterprise Server memory configurations. RS/6000 hardware systems with more than 4 GB of memory will become available in the future.

AIXwindows Version 11 Release 6. This includes all supported core X client programs, libraries, font server, fonts, and all client X internationalization support. This also updates the Motif libraries to Motif 2.1.0. X11R5 X extension load modules are not supported in the X11R6 X server. X11R5 `loadddx` load modules are not supported in the X11R6 X server.

Parameter	Limit
Executables	**$7*2^{56}$ bytes**
Sum of sizes of text data and BSS sections	**$7*2^{56}$ bytes**
Symbol values (generally address of objects)	**2^{64}**

Figure 6. Limits of 64-bit executables

ONC+. The ONC+ technology has been licensed from SunSoft and incorporated into AIX. ONC+ introduces CacheFS functionality:

- AIX headers C++ aware. All shipped header files in the Base System are C++ aware to support the xlC product.
- M:N Pthreads. The current 1:1 threads implementation has been replaced with an M:N version. This conforms with IEEE® POSIX® 1003.1c (threads). It provides increased performance for several types of database applications, as well as greater per-process limits for applications with large numbers of threads, such as databases.

  Existing AIX `Pthread` applications conform to POSIX 1003.1c Draft 7. AIX 4.3 supports both Draft 7 and Draft 10 in 32-bit mode. However, the 64-bit mode supports only Draft 10; therefore, if you are porting existing `Pthread` applications to 64-bit, you must convert to Draft 10.

Thread-Safe Libraries. `Non-thread-safe` and `thread-safe _r` libraries have been combined into one set of libraries, making thread-safety a default. For example:

AIX 4.2 includes: `libc.a (non-thread-safe)` and `libc_r.a (thread-safe)`
AIX 4.3 includes: `libc.a (thread-safe)`

Libraries, such as X11R6, which link with `libc.a` are not thread-safe by default;

they are thread aware. New libraries that are thread-safe include the following: `libbsd.a`, `libm.a`, `libmsaa.a`, `libs.a`, `libdes.a`, `libXext.a`, and `libXi.a`.

These thread-safe libraries provide a convenient programming model to exploit Symmetric Multiprocessors (SMPs) and to simplify the exploitation of threads by applications, middleware, and other Application Programming Interface (API) providers.

WebSM. This is an easy-to-use interface to AIX system management functions that enables administrators to remotely access them over the Internet. WebSM is the new primary interface for system management.

WebSM has the following characteristics:

- Accessible remotely via the Internet or intranets
- Accessible from any client platform that has Java 1.1 support or a Java 1.1-enabled Web browser
- Consistent with the Common Desktop Environment (CDE) desktop, and Microsoft Windows 95, Windows NT™ applications in user interface style
- Less administrative skill required than traditional UNIX system management applications
- Organization of system management administration in the form of "real world" objects in the user interface; for example, users, groups, devices, software packages/applications
- Full coverage of AIX system management functionality; enables administrators to perform all key AIX system management from the WebSM

Enhanced Online Documentation. In AIX 4.3 the online publications are HTML based. InfoExplorer has been replaced by any Web browser that can view, search, and hypertext link an HTML document .

NFS Version 3. NFS Version 3 has been extended to support HANFS hooks and daemons, removing the restriction of HANFS configurations with only two nodes. This allows a node to serve as backup for more than one other node.

IP Version 6. Internet Protocol Version 6 is the next generation of the Internet Protocol, which relieves IP address limitations and adds robust fail-safe routing, automatic configuration of IP addresses, as well as improved security and integrity.

The IP Version 6 implementation in AIX 4.3 protects customer investment in existing applications while introducing the next generation of IP. Customer applications that are written with the current IP (Internet Protocol Version 4) applications interface will work as-is (binary compatible).

Most AIX-provided Internet applications, such as name resolver, configuration, mail, `r-cmds` (`rlogin`, `rsh`), telnet, ftp, and tftp are upgraded to support IP Version 6. AIX 4.3 IP Version 6 provides the required infrastructure for serving all the Internet-ready boxes and also the devices to participate in the growing Internet. AIX IP Version 6 supports host requirements, but not IP forwarding.

IP Security on IP Versions 4 and 6. IPSec, a security protocol in the IP layer, provides security services to ensure packet authentication, integrity, access control, and confidentiality. A secure tunnel is established between the two systems to perform key exchange, message encryption, and message authentication.

AIX 4.3 supports new IP security protocols, Authentication Header (AH) for integrity and authentication, and Encapsulating Security Payload (ESP) for confidentiality. The AIX implementation of IPsec interoperates with other IPsec-enabled products, such as IBM FireWall.

A new key management mechanism is added to AIX 4.3 since keys must be generated and distributed for IPsecurity. A shared master key is manually distributed between systems at the tunnel endpoints where one system generates and exports a master key file and the other system uses this key file to import the master key definition. Session key protocols, both static for tunnel duration, and dynamic are also provided.

Address Space Layout

Segment Number	Use in 32-bit User Mode
0	kernel text
1	user text
2	process private data
3	available for user
4	shmat, mmap
5	shmat, mmap
6	shmat, mmap
7	shmat, mmap
8	shmat, mmap
9	shmat, mmap
0xA	shmat, mmap
0xB	shmat, mmap
0xC	shmat, mmap
0xD	shared lib text
0xE	shmat, mmap
0xF	shared lib data

Segment Number	Use in 64-bit User Mode
0	kernel
1	kernel
2	process private
3–0xC	shmat/mmap
0xD	loader use
0xE	shmat/mmap
0xF	loader use
0x10–0x6FFFFFFF	application text, data, BSS, heap
0x70000000–0x7FFFFFFF	default shmat/mmap
0x80000000–0x8FFFFFFF	private load
0x90000000–0x9FFFFFFF	shared library text and data
0xA0000000–0xEFFFFFFF	reserved for system use
0xF0000000–0xFFFFFFFF	application stack

Figure 7. Address space layout for 32-bit and 64-bit architectures

BEST-X. The BEST-X security product installs on top of AIX 4, and provides the features necessary to meet a European E3/F-B1 certification.

LDAP. LDAP (Lightweight Directory Access Protocol) is a directory service protocol that runs over TCP/IP. It can be used with stand-alone and other kinds of directory servers.

Address Spaces

Figure 7 shows the address space layout for both 32-bit and 64-bit applications. The new 64-bit user address space is substantially larger than the one for 32-bit applications. A 32-bit application will always see the 32-bit address space, regardless of whether it is running on 32-bit or 64-bit hardware, however a 64-bit application will see the entire 64-bit address space.

See *The AIX 64-bit Migration Guide* (http://www.developer.ibm.com/sdp/library/aix4.3/) for a detailed discussion of address space programming interfaces and the kernel's remapping routines.

AIX 4.3 provides a 60-bit effective address space (up to 2^{32} segments). Processes can access address spaces via `shmat()` or `mmap()`. These `shmat` and `mmap` segments are allocated starting at segment 0x70000000 if no address is specified. Segments are allocated in increasing order in the first available segment at or after segment number 0x70000000. The `shmat` and `mmap` segments are placed where requested if the segment number is available and less than segment number 0x80000000.

Explicitly loaded modules—using the `load()` system call—are loaded into separate segments starting at segment number

0x80000000. Segment numbers are allocated in increasing order in the first available segment number at or after segment number 0x80000000. Explicitly loaded modules are limited to segment numbers 0x80000000 to 0x8FFFFFFF.

Application Development Toolkit (ADT)

All development tools that process program source code can target the 64-bit execution environment while running on 32-bit RS/6000s. However, testing 64-bit executables requires 64-bit hardware because there is no simulation environment of the 64-bit ABI on 32-bit hardware. It is easy to port programs between 32-bit and 64-bit environments and have equivalent functionality. It is the data size issues that become the primary porting concern.

Exploiting 64-bit address space requires more substantive programming changes in most C applications. Compiler flags can be used as porting tools that warn about all programming constructs that may cause errors when code is compiled. Examples of such constructs include longs assigned or cast to `int`s, pointers assigned or cast to `int`s, and parameter mismatches.

The C for AIX compiler. The 64-bit implementation of the C front end has not changed the behavior of the compiler. The default compilation and assembly mode is 32-bit. The compiler will only change the behavior of code when compiling for 64-bit mode. The developer can change the default from 32-bit to 64-bit mode.

The 64-bit compiler can be invoked by passing a separate flag, `-q64` (`-qarch=ppc64`). The 64-bit binder is evoked by passing a separate flag (`-b64`).

Compiler Invocation. The new features of the compiler that enable 64-bit mode include the following:

- Predefined `__64BIT__macro` when invoked for 64-bit compilations
- `OBJECT_MODE` environment variable
- `-qarch` support for 64-bit suboption

__64BIT__ macro. When the compiler is invoked to compile for 64-bit mode, it defines the preprocessor `macro __64BIT__`. When it is invoked in 32-bit (default) mode, this macro is not defined. This variable can be tested via `#if defined(__64BIT__)` or `#ifdef __64BIT__` to select lines of code, such as `printf` statements, that are appropriate for 64- or 32-bit mode. This ability to choose execution mode of the final executable at compile time and the existence of the `__64BIT__` macro implies there is no need to test execution mode at runtime.

OBJECT_MODE Environment Variable. The C compiler obtains information from environment variables. These may be the `setlocale` variable which changes the language of messages, or the `_xlC_test_usrincpath` variable, which overrides the default place to look for system header files (`/usr/include`). This latter option makes it easy to test new header files without changing them in the default location. Neither of these environment variables have a competing command-line option that can contradict it.

A new environment variable called `OBJECT_MODE` replaces the default compilation mode of the compiler when it is used. If `OBJECT_MODE` does not exist, the compiler defaults to 32-bit mode, facilitating compatibility with systems such as AIX 4.3 installed on 32-bit machines, which do not support 64-bit.

Command-Line Options. Command-line options override the environment variable `OBJECT_MODE` when setting the compilation mode. When compiling for 64-bit mode, the compiler generates 64-bit instructions when dealing with 8-byte data, and produces files in the 64-bit XCOFF format. The compiler cannot generate 64-bit instructions that operate safely on 32-bit XCOFF, which implies that 64-bit and 32-bit objects cannot be mixed. An architecture (`ARCH`) compile-time flag (option) determines 64-bit instruction and XCOFF. This `ARCH` suboption completely and uniquely determines the instruction set and XCOFF mode.

Mixed Mode Compilation. When 32-bit and 64-bit compilation modes are mixed, XCOFF objects will not bind. This becomes obvious if the compile and link

occurred in one step, however, if the compile and link occurred in different steps, the developer may not be aware of this.

If an application was compiled and produced 64-bit objects, remember to link using the 64-bit mode (when linking using xlc), otherwise the objects will not link. Objects in mixed XCOFF mode will never link and must be recompiled completely, ensuring that all objects are compiled in the same mode.

Compiling and Linking for 64-bit Execution. The default compilation and assembly mode is 32-bit. The developer can change from the default 32-bit to 64-bit mode. In the compiler, an option such as `-q64` and/or `-qarch=ppc64` can be used to change the compilation mode.

Lint. `lint` has been modified to flag some common 32-bit and 64-bit porting problems. Using the `-t` flag enables these changes. For example, `lint -t` will find the following common potential problems:

- Functions not prototyped. Function prototypes allow the compiler and `lint` to flag mismatched parameters.
- Assignment of a long or a pointer to an `int`, which could cause truncation. Even assignments with an explicit cast will be flagged.
- Assignment of an `int` to a pointer. If the pointer is referenced, it may be invalid.
- Shift operations involving a long or pointer. Since the pointer size was increased to 64-bits, the result may no longer be valid.
- Masks using bitwise OR, AND or XOR involving longs or pointers. Again, since the pointer size has changed, the mask may no longer be sufficient.

The following are a few problems `lint -t` cannot find:

- Shared data must be the same size in both 32-bit and 64-bit. For example, the `utmp` structure in `/usr/include/utmp.h` is used to read and write data from `/etc/utmp` and its size must be consistent in both 32-bit and 64-bit modes.
- Unions that use longs or pointers may no longer work. The following two union examples work fine in a 32-bit environment but will not work in a 64-bit environment. Figure 8 shows two examples.

Example 1

```
union {
int *p; /* 32 bits / 64 bits */
int i;  /* 32 bits / 32 bits */
};
```

Example 2

```
union {
double d;  /* 64 bits / 64 bits */
long l[2]; /* 64 bits / 128 bits */
};
```

Figure 8. 32-bit union examples

Two other differences can affect programs. In 32-bit mode registers are only 32-bits wide, so long long's are passed in two general-purpose registers. In 64-bit mode, long long's are passed in only one (64-bit wide) general-purpose register.

In 32-bit mode when a function is passed floating-point arguments and it is not prototyped, each floating-point argument is placed in one floating-point register and in two general-purpose registers. In 64-bit mode, each floating-point argument is still placed in one floating-point register, but only one general-purpose register.

The Assembler. The assembler will continue to execute in 32-bit mode on 64-bit systems, but will also produce object files in the 64-bit XCOFF format for 64-bit execution.

Assembler Execution Mode. The architecture on which an assembly is intended to execute can be specified in two ways:

- Via the `-m` (for "machine") command flag, which sets the default target architecture and mnemonic set for the assembly. Figure 9 shows values for `-m` accepted by the assembler

- Via a `.machine` pseudo-op specifying one of the codes that is valid with `-m`. The `.machine` pseudo-op overrides any `-m` command-line flag. More than one `.machine` pseudo-op can appear in an assembly; that is, different parts of the assembly can be assembled for different target machines with different mnemonic sets

-m	Architecture and Implementation
ppc	**32-bit PowerPC; this is the default if `-m` is not specified**
pwr	**POWER architecture, POWER implementation**
ppc64	**64-bit PowerPC**
com	**POWER and PowerPC**
pwrx, pwr2	**POWER architecture, POWER2 implementation**
any	**POWER, or POWERPC**
601	**32-bit PowerPC, 601 implementation**
603	**32-bit PowerPC, 603 implementation**
604	**32-bit PowerPC, 604 implementation**

Figure 9. Assembler values

The Assembler Lexical Scanner. The assembler's lexical scanner now accepts integer values in a variety of bases that require up to 64 bits to represent. This allows for longer numeric fields; for example, as many as 16 hexadecimal digits can be required to specify a 64-bit mask field.

To avoid the problem of reading, writing, and checking (verifying) such long string of digits, the lexical scanner will now accept and ignore underscore characters ("_") within numeric fields. This allows numbers to be separated into groups, such as grouping by thousands (groups of 3) in decimal numbers.

Thus, instead of coding:

```
.long 4294967295
```

the user can now code:

```
.long 4_294_967_295
```

The Linker. The AIX linker supports the development of 64-bit applications, libraries, and kernel extensions. For 64-bit applications and libraries, the linker reads and writes XCOFF64 files while performing internal processing appropriate for 64-bit mode.

For AIX 4.3, the base kernel and kernel extensions are XCOFF32 files. Thus, a 64-bit kernel extension comprises XCOFF32 object files linked in 32-bit mode, using an export file that marks some symbols as 64-bit kernel services (Device drivers that support 64-bit processes do not need to be compiled or linked in any special way. They indicate their ability to handle 64-bit processes when they are configured.)

Linking in 32-bit mode works the same way on both AIX 4.3 and AIX 4.2.

Two new flags have been added, `-b32` and `-b64`, which will specify linking modes of 32-bit and 64-bit, respectively. These options may come from a stanza file, such as `/etc/xlC.cfg`, or from the command line. The new environment variable, `OBJECT_MODE`, can also be used to specify a mode.

- `-b32`: Specifies 32-bit linking mode. In this mode, all input object files must be XCOFF32 files, or an error is reported.
- `-b64`: Specifies 64-bit linking mode. In this mode, all input object files must be XCOFF64 files, or an error is reported.
- `OBJECT_MODE` environment variable. If neither the `-b32` nor `-b64` option is used, the `OBJECT_MODE` environment variable is examined to determine the linking mode. If the value of `OBJECT_MODE` is "64", 64-bit mode is used. If the value is "32_64", the

linker prints an error message and exits with a non-zero return code. Otherwise, 32-bit mode is used.

The Loader. The loader for 64-bit programs provides equivalent semantics to the 32-bit loader.

Other Application Development Utilities. For the following XCOFF-specific commands, `dump`, `nm`, `lorder`, `ranlib`, `size`, `strip`, a new flag, `X`, has been added. It requires an argument of either 32, 64, or 32_64. This flag indicates whether to recognize only 32-bit objects or only 64-bit objects (in addition to any non-object files, which are always valid), or both.

New Functions. These commands have been enhanced to provide the same functionality available for both 32-bit and 64-bit objects and executables. They will also understand both new and old archive file formats.

The `prof` and `gprof` commands have been expanded to support profiling of 64-bit executable programs. The same functionality is available for both 32-bit and 64-bit executables.

The `lint` command has a new flag to allow the user to specify whether the source code is intended for a 32-bit (default) or a 64-bit compilation, or a port from 32-bit to 64-bit.

C source code generated by the `lex` and `yacc` utilities will be able to compile correctly in either 32-bit or 64-bit compilations. The `string` and `sum` commands perform the same functions for both 32-bit and 64-bit XCOFF files.

All BOS commands recognize and support data values that might exceed 231 in a 64-bit environment.

Archive File Format. The AIX 4.3 `ar` command handles the archiving of 64-bit XCOFF object modules in addition to 32-bit XCOFF object modules. To support the two different formats of object files, the symbols of 64-bit objects are distinguishable from those of 32-bit objects.

There are two global symbol tables: one for 32-bit object symbols and one for 64-bit object symbols. The `ar` command recognizes each type of object file and stores its symbols in the appropriate table.

The `ar` command maintains compatibility with the previous archive file format by adding, deleting, reordering, and listing members without altering the format of an old archive file except in two cases:

- When the user explicitly requests conversion to the AIX 4.3 format by using the `-o` option
- When the user adds a 64-bit object to the archive (This requires conversion because a 64-bit object cannot be handled by the old-format archive.)

The two basic changes of the archive format are separate global symbol tables for 32-bit and 64-bit object symbols and the expanded size limits of a file format. The `ar` command handles both this new archive format (for 32- and 64-bit) while continuing to recognize the old format (32-bit only) for compatibility.

PTX. All PTX commands remain functionally compatible with their previous behavior. The commands on a 32-bit system look and act the same as they would have prior to AIX 4.3. The commands themselves all remain 32-bit executables, enabled to recognize 64-bit values where appropriate.

<<Author name. address. title, responsibilities and education.>>

IBM High Performance Compiler for Java

By Ven Seshadri

IBM has developed an optimizing native code compiler for Java. This article describes the differences between static and Just-In-Time compilation, the basic structure of the AIX compiler, some implementation highlights, performance results, and the current status of the system.

Most Java Virtual Machines contain a Just-In-Time (JIT) compiler to improve performance. JIT compilers turn Java™ bytecodes into native (object) code on-the-fly as they are loaded into the virtual machine. The virtual machine then executes the generated object code directly instead of interpreting bytecodes. Because this translation occurs at execution time, the speed requirements for compilation limit the performance optimization of a JIT compiler.

An optimizing native code compiler for Java compiles Java bytecodes directly into native (object) code just as compilers for C/C++, FORTRAN, or COBOL. Unlike JIT compilers, this static compilation occurs only once—before execution time. Thus, traditional resource-intensive optimization techniques (such as dataflow analysis or interprocedural optimization) can be applied to improve the performance of the generated code.

Since the static native code compiler adheres to the machine-independent bytecode specification, statically compiling a Java application to multiple platforms results in object code that produces identical results on each platform. However, some important distinctions exist between statically compiled Java and dynamically interpreted (or JIT-compiled) Java, which we discuss in the next section.

Early beta-level versions of the optimizing native code compiler of Java for both AIX® and Windows NT™ are freely available at IBM's alphaWorks™ Web site (http://www.alphaworks.ibm.com).

Static vs. Just-In-Time Compilation

The Java programming language and Application Programming Interfaces (APIs) contain several dynamic language features, the most important being dynamic bytecode loading and Release-to-Release Binary Compatibility (RRBC). These features require late (load-time) binding capabilities, which are easily supported within a virtual machine. A Just-In-Time compiler for Java supports these features simply by using the late binding facilities of the Java Virtual Machine.

A static compiler uses the bytecodes for a Java application to generate either a set of shared objects or Dynamic Link Libraries (DLLs), or a statically bound executable before the application executes. Although

Ven Seshadri

our current implementation does not support dynamic bytecode loading or RRBC, these features can be implemented in a statically compiled context with some additional overhead.

If the runtime system includes a JIT compiler, dynamic bytecode loading can be implemented within a static compilation framework. The JIT compiler would be invoked to create a DLL from the bytecodes being loaded. Then the runtime system can load this DLL like any other application DLL. For this to work correctly, the bytecodes (or equivalent information) for classes referenced directly and indirectly by the dynamically loaded class must be present at runtime. This allows the compiler to generate the appropriate references to instance data and method tables in those classes.

The RRBC problem in object-oriented languages like Java arises because instance fields and class methods are referenced from other classes indirectly, usually by base+offset addressing. A static or JIT compiler determines the layout of instance method tables and object instance data of a class at compile time. It also generates code in client classes of that class, which contains inline offsets into these structures.

If a class changes in a way that causes the layout of the instance method tables or instance data to change (for example, adding an instance method or deleting a field), all of its client classes must be recompiled. This recompilation is unnecessary for a JIT compiler because the layouts of instance data and instance method tables are determined at class load time, which is the same as JIT compile time. References to instance fields and methods of other classes in the bytecodes are by name. These are turned into base+offset addressing after the class is loaded.

A statically compiled framework offers several ways to support RRBC. One approach is to extend the system linker and loader to understand special relocations for offsets into instance data and instance method tables. The compiler would generate these special relocations in the object code instead of inline offsets, which would then cause the linker or loader to generate the appropriate inline offset. This approach would require support from various operating system facilities and also increase load time.

An optimizing native code compiler for Java compiles Java bytecodes directly into native (object) code.

Another approach is to use object-oriented runtime support systems, such as IBM's System Object Model (SOM) which provides language-neutral object services such as RRBC. This method would use pre-existing services, but these services tend to impose a noticeable performance penalty.

A third approach would be to generate all references to instance method tables and instance data through an extra level of indirection—the compiler would generate temporaries to hold the actual offsets used in base+offset addressing. The compiler would then generate DLL initialization code to set appropriate values in these temporaries. This approach would not require special support from the operating system, but would impose a runtime penalty with increased load time and an additional level of indirection to access method tables and instance data.

Basic System Structure for the AIX Compiler

Figure 1 shows the basic structure of the compiler. The input is a single Java bytecode (`.class`) file, which contains a single class in the application. A translator processes the bytecodes to produce an internal compiler Intermediate Language (IL) representation of the class. The common back end from IBM's XL family of compilers for the RS/6000™ is then used to turn this intermediate representation into an object module (`.o` file) that is linked with other object modules from the application and libraries to produce an executable program.

The libraries implement garbage collection, the Java APIs, and various system routines to support object creation, threads,

exception handling, and application startup and termination. The compiler will also accept Java source code as input; if so, it invokes the AIX Java Development Kit's Java source-to-bytecode compiler (javac) to first produce bytecodes.

Currently, the compiler has a simple command-line interface. However, it is easier to develop optimized native code by using the IBM VisualAge™ for Java application development tool, with extensions to support this compiler (beta availability for these extensions is expected in the future).

These extensions will provide a seamless integration between the Integrated Development Environment (IDE) and the compiler. They enhance the IDE "Export" Smartguide so that when the Java program is ready to be deployed, the developer selects the "Export" option from the IBM VisualAge for Java IDE. Instead of exporting the Java bytecode files, they export a compiled object, providing compile-time flags as appropriate. The Smartguide will transfer the Java bytecode file(s) seamlessly to a development system running the same operating system as the production environment, and invoke the IBM High Performance Compiler™ for Java for that platform.

Object and Class Internal Structure

An object reference acts as a pointer to some data in the heap. The data consists of the following:

- A pointer to a static data structure containing the metadata for the class of that object (one per class)
- Eight bytes to implement Java's per-object locks
- Instance data of the object

The class metadata contains the following types of information:

- Class initialization status
- Method table
- Field table
- Inner classes table
- Interface table
- Access flags for the class, methods, and fields

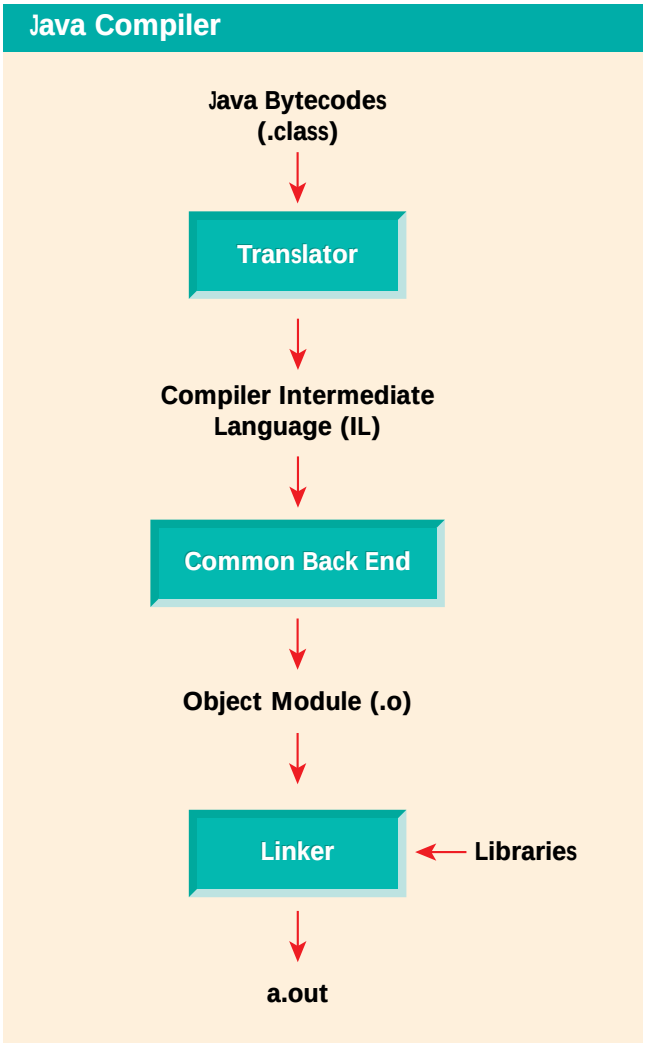


Figure 1. Basic structure of Java compiler

Much of the information contained in the class metadata supports the rich set of runtime facilities in Java, including interface method invocation, cast checking at runtime, reflection APIs, and the Java Native Interface (JNI).

All types of classes—plain, interface, and array—are represented using the same metadata format. Java requires a unique instance of `java.lang.Class` for each class in the application. The class metadata is the data for the unique instance of `java.lang.Class` for that class (thus the first twelve bytes of the class metadata for a class must correspond to the first twelve bytes of the data for an object—four bytes for the class metadata pointer and eight bytes for the lock information).

Garbage Collection

Use of the common back end is an important design decision that merits some discussion. By using the same back end that is used in IBM's C/C++, FORTRAN, COBOL, and other RS/6000 compilers, we can obtain the same high-quality, robust code optimization found in these other products. It also dictates that we use a conservative garbage collector instead of a nonconservative copying collector, since the common back end provides no special support for garbage collection (such as being able to distinguish pointers from nonpointers at runtime).

While this may appear to be a disadvantage, it is important to note that code generation for systems that support copying collection imposes runtime overhead that slows down user code. This overhead can be considerable. With some garbage collection, schemes such as generation scavenging, every pointer stored in the heap requires compare-and-branch code to determine if this location needs to be specially marked for garbage collection.

This overhead slows the execution of code that is not part of a garbage collection—simply to support garbage collection. Since time spent collecting garbage is typically far less than time spent running user code, our scheme optimizes generated user code at the expense of more costly garbage collection. We use the publicly available Boehm garbage collector, a conservative collector that has been ported to many platforms.

Code Generation Optimizations

Using the common back end from IBM's XL compilers immediately gives us a rich set of language-independent optimizations, such as instruction scheduling, common subexpression elimination, intramodule inlining, constant propagation, and global register allocation. We also perform some Java-specific optimizations in the translator.

Runtime checking code (null pointer, array bounds, and zero-divide checking) accounts for a significant amount of overhead in Java. The translator simulates the bytecode stack while compiling basic blocks to determine whether such checks can be eliminated. For example, in this way, most calls to constructors that immediately follow object creation can be performed without determining that the object reference passed to the constructor is not null.

The translator also emits direct calls to instance methods that are known to be final, or to members of a final class. The usual indirect call through the instance method table is unnecessary because final classes cannot be subclassed and final methods cannot be overridden. Therefore, at compile time the translator will know which method will be invoked.

Performance

Figure 2 shows the performance of our compiler on a set of publicly available language processing applications, including the following:

- Javac: Sun's Java 1.0.2 source-to-bytecode compiler
- Jacorb: Jacorb Version 0.5, a CORBA/IDL-to-Java translator, by Gerald Bose, Berlin University
- Toba: A Java-to-C translator, University of Arizona
- JavaLex: A lexical analyzer generator for Java, Princeton University
- JavaParser: A parser for Java, Sun Microsystems®, Inc. Generated automatically by Sun's Jack tool
- JavaCup: A LALR parser generator for Java, by Scott Hudson, Georgia Institute of Technology
- Jobe: A Java obfuscator, by Eron Jokipii

The benchmarks were run on an RS/6000 with a 133 MHz PowerPC™ 604 CPU, 96 MB of memory, and a 512 KB L2 cache.

Each benchmark was run using four different compilers/virtual machines:

- AIX Java Virtual Machine 1.0.2D (without JIT compiler)
- AIX Java Virtual Machine 1.0.2D with IBM JIT compiler 1.0+
- Our compiler with full runtime checking

- Our compiler with runtime checking turned off

Compiling with runtime checking turned off is unsafe since the generated code is not Java-compliant. However, we present the results below to show the overhead introduced by runtime checking.

Our compiler was used with the optimization flag (`-O`) turned on. As shown in Figure 2, compiling to native code provides a speedup between 4 and 17 times over interpreted code, and between 3 and 13 times over the JIT compiler. This family of applications tends to exercise Java's object-oriented features, such as instance method calls, object creation, and garbage collection. Note that in this set of benchmarks, turning off runtime checking does not provide a significant benefit. However, it makes a considerable difference for codes that spend a significant amount of time in tight loops manipulating arrays, such as scientific codes.

The javac benchmark measured above, using Sun's Java 1.0.2-level source-to-bytecode compiler, is provided in compiled form in the samples subdirectory on our Web site (www.alphaworks.ibm.com).

Status

One possible usage scenario for the compiler is on a performance-sensitive intranet server-side Java application where the code is ready to roll into production. Once the application is written and debugged, the compiler can be used on the server to create a high-performance static executable.

Another scenario is for an Internet Web server to automatically invoke the compiler to accelerate existing Java servlets. The first time a servlet is called, the Web server calls the compiler to create a static executable. With subsequent invocations, the Web server calls the previously created static executable. The portability of the servlet is retained at the Java bytecode level.

An early beta-level technology demonstration version of our compiler for AIX is available on IBM's alphaWorks Web site (www.alphaworks.ibm.com). At publication time for this article, it is a Java 1.0.2-level implementation with some missing features. The missing features include the `java.awt` and `java.applet` APIs, dynamic class loading, and Java-style Release-to-Release Binary Compatibility.

Compiler Performance

Benchmark	Interpreted	JIT	Compiled	Compiled (no check)
Javac	40.2s	21.1s	3.9s	3.7s
Jacorb	47.2s	34.3s	7.0s	6.8s
Toba	67.2s	51.7s	5.7s	5.5s
JavaLex	49.4s	38.9s	2.9s	2.8s
JavaParser	60.6s	20.1s	6.1s	4.8s
JavaCup	15.7s	11.9s	1.5s	1.5s
Jobe	18.1s	13.6s	4.6s	4.2s

Figure 2. Java benchmarks

References

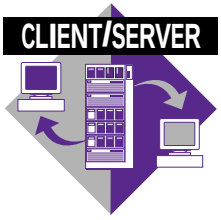
Boehm, H. J. "Space Efficient Conservative Garbage Collection." *Proceedings of the ACM SIGPLAN '93 Conference on Programming Language Design and Implementation*, June 1993.

Acknowledgments

The author would like to acknowledge the contributions of the following people to the first release of the High Performance Compiler for Java: Dick Attanasio, Tony Cocchi, Pat Gallop, Mark Mergen, Mauricio Serrano, Janice Shepherd, Steve Smith, Simon Tooke, and Dean Williams.

Ven Seshadri, IBM Canada Ltd, Software Solutions Division, 1150 Eglinton Ave. E., North York, Ontario, CANADA, M3C 1H7. E-mail: seshadri@vnet.ibm.com. Mr. Seshadri is an advisory software developer at IBM's Toronto Lab, and has been working on the design and implementation of IBM's High Performance Compiler for Java since 1996. Prior to that, he was responsible for implementing loop transformations in IBM's XL FORTRAN and XL High Performance FORTRAN compilers. Mr. Seshadri has a BASc in Engineering Science and an MASc in Electrical Engineering, both from the University of Toronto.

Parallel Programming Models on the IBM RS/6000 SP



By David Klepacki and Xianneng Shen

This article is the first in a series that will introduce parallel programming models on the RS/6000 SP system. It presents an overview of the various programming models and their basic features. Future articles will provide more detailed information about each programming model.

The IBM RS/6000 Scalable POWERparallel™ system is the large-scale server of the IBM RS/6000™ product line, which includes desktop workstations, group servers, departmental servers, enterprise servers, and large-scale servers. The RS/6000 SP™ system is a general-purpose parallel computer. It can operate in one of three modes:

- **Sequential/Batch mode**: More than one serial or Symmetric Multiprocessing (SMP) job can be run at the same time (Server consolidation is a good example of this mode).
- **Parallel mode**: More than one computing unit (uniprocessor or SMP) can be used at the same time for the same job.
- **Hybrid mode**: Both parallel jobs and sequential jobs can coexist on the same system.

The basic hardware components of the RS/6000 SP system are the computing units (nodes) and the interconnection network (switch). Each node is an RS/6000 server, which can be either a uniprocessor server or an SMP server. The systems are packaged into frames that come in two sizes (heights): 79 inch and 49 inch. The 79-inch frame has eight drawers and the 49-inch frame has four drawers.

In addition, the system has three types of nodes based on their physical size: uniprocessor thin nodes, uniprocessor wide nodes, and SMP high nodes. Either two thin nodes or one wide node fits into a single drawer; a high node requires two drawers.

RS/6000 SP Architecture

The RS/6000 SP is a share-nothing architecture; that is, each node has its own memory (RAM), hard drives, network connections, and operating system (AIX). Communication between nodes is carried out by sending and receiving messages over the network. This type of architecture allows high availability (engineered redundancy), flexibility (open choices of configuration), and scalability (the ability to incrementally increase performance).

In modern computer architecture taxonomy, the RS/6000 SP is also classified as a distributed-memory Multiple Instruction Multiple Data (MIMD) machine. MIMD simply means that different instruction streams of the same job that may operate

on different objects within the same dataspace can be executed simultaneously by more than one processor.

The switch network provides a low-latency and high-bandwidth communication path between the nodes. A Communication Subsystem (CSS) provides the common IP communication protocols as well as a high-speed proprietary protocol known as *user space*. Typically, technical applications employ the user-space protocol and commercial applications employ the IP communication protocols.

A complete copy of the AIX operating system runs on every node since each node is an RS/6000 server. A Parallel System Support Program (PSSP) runs on top of the AIX operating system on each node. The PSSP provides system management, high-availability infrastructure, the communication subsystem, and the virtual shared-disk system. The Parallel Environment (PE) software provides a parallel application development environment, which includes debugging and performance monitoring tools.

Parallel Programming Models

The RS/6000 SP supports three classes of parallel programming models: message-passing, data parallel, and shared memory.

Message-passing Mode

In a message-passing programming model, explicit message passing (send/receive) is used to communicate between nodes. This model provides optimal performance because it is congruent to the underlying architecture of the RS/6000 SP. In this model, an application developer makes calls to a message-passing library to invoke the various Interprocess Communication (IPC) routines within the source code.

The RS/6000 SP fully supports the Message Passing Interface (MPI), the industry-standard message-passing library. IBM strongly supports MPI standardization and has actively participated in the public MPI forum, which defines the MPI standard. The first standard definition of MPI (MPI-1) was published in May 1994.[1] Recently, the second standard definition (MPI-2) has been announced.[2] Standardization ensures portability of your applications across different parallel machines (although performance is not ensured).

The RS/6000 SP supports three classes of parallel programming models: message-passing, data parallel, and shared memory.

In addition to MPI, the RS/6000 SP supports other message-passing libraries, such as Message Passing Library (MPL), Parallel Virtual Machine (PVM), and PVMe. MPL, the first native message-passing library for the RS/6000 SP, is being phased out in support of the MPI standard.

PVM, a public domain environment developed by Oak Ridge National Lab[3], provides some features not found in either MPL or MPI-1. PVMe is an enhanced version of PVM that specifically exploits the technology of the RS/6000 SP switch hardware. Although PVM and PVMe are still widely used today, the new features specified in the MPI-2 standard will eventually cause these environments to phase out as well.

Data Parallel Programming Model

The data parallel programming model evolved from earlier parallel machine architectures known as Single Instruction Multiple Data (SIMD) machines. These machines simultaneously execute the same instruction stream on more than one processor, but operate on different objects within the same dataspace.

This programming style is very popular with the scientific and engineering communities, which are typically interested in solving mathematical problems on fixed spatial grids. Their computational technique is

[1] MPI Standard
[2] MPI-2 Standard
[3] PVM User Guide

known as *domain decomposition*, whereby the computational space is divided among the processors so that the mathematical operators can be applied to each of the disjoint subspaces in parallel. The data parallel programming model is the most natural model for this kind of computation.

The industry standard for data parallel programming in FORTRAN is High Performance FORTRAN (HPF). HPF is characterized by exclusive use of comment-based directives within the FORTRAN source code. These directives specify the distribution, layout, and alignment of the various data elements (mostly arrays) among the different processors.

The HPF compiler (or preprocessor) is responsible for the movement of data between the processors when needed. This alleviates the burden of explicitly managing the movement of data between the processors, as in the message-passing model. Therefore, the data parallel model is often referred to as an *implicit* method of programming. However, this ease-of-use with regard to the data parallel model does not come without a price on a MIMD architecture such as the RS/6000 SP.

Since the underlying architecture is MIMD and not SIMD, the data parallel programming model is not congruent to the underlying hardware; therefore, the performance will generally not be optimal. Despite this fact, the data parallel model still has advantages.

Data Parallel Model

One advantage of this model is the ability to easily change the layout of the data. In HPF, the layout of data can be easily altered with one or more `change all` commands of a text editor. For example, a one-dimensional distribution of a matrix, specified by (`BLOCK,*`) in a HPF directive, can be changed to a two-dimensional distribution among the processors by using (`BLOCK,BLOCK`) instead. The only changes needed are to the parameters inside the (relatively few) HPF directives.

On the other hand, altering the data layout in an MPI application typically would require substantial rewriting of code to accommodate the different buffer management and synchronization requirements. Therefore, the trade-off in performance by using HPF might be worthwhile for both the ease-of-use and flexibility of code changes it provides.

Besides the performance trade-off, the data parallel model suffers from other afflictions: the inability to accommodate dynamic load balancing and dynamic data structures. This should not come as a surprise, since the data parallel model is mainly interested in being applied to static domains as described above. However, newer physical models and computational techniques are forcing the need to address such issues. Although there are various research attempts to incorporate support for dynamic parallelism within HPF, none have come to fruition. In fact, the most recent HPF standard specification (HPF-2)[4] has removed some of the mandatory features dealing with dynamic redistribution, since none of the HPF compiler vendors could provide such functionality efficiently.

The industry standard for data parallel programming in FORTRAN is High Performance FORTRAN (HPF).

HPF Compilers

Today, several HPF compilers are available for the RS/6000 SP. IBM produces the xlHPF compiler; Applied Parallel Research, Inc. produces the xHPF preprocessor; and the Portland Group produces the pgHPF preprocessor. These third-party vendors supply preprocessors to retain the use of IBM's xlf compiler and all of the optimizations that come with using it.

For the C language, there is an ANSI® standard specification known as Data Parallel C (DPC) as well as a widely used (but not a standard) language known as High Performance C (HPC). HPC is public

[4] HPF-2 Standard Document

domain and attempts to mimic the features and functionality of HPF by using similar comment-based directives. All of these issues for HPF also apply to HPC.

Like HPF, DPC has its origins from the days of the SIMD architectures. It is a direct offshoot of the C language pioneered by the now defunct Thinking Machines Corporation. In fact, many applications written in C can run with minor alterations on the RS/6000 SP using a DPC compiler. Although IBM does not produce a DPC compiler, it is available for the RS/6000 SP via a third-party software vendor, Pacific-Sierra Research, Inc.

The ANSI standard DPC language is not based upon the exclusive use of comment-based directives. Rather, *shape distribution* is introduced, in which data distribution is specified in the source code by using left subscript variables. Although the syntax of this implementation is quite different, the issues involved in the data parallel model remain unchanged.

Shared Memory Model

The third type of programming model is the shared memory model, or more accurately the Virtual Shared Memory (VSM) model, since the present discussion is in the context of a distributed-memory parallel architecture such as the RS/6000 SP.

The VSM model is the ultimate extrapolation of parallel programming to a single virtual address space. Although the underlying hardware contains separate and distinct memory address spaces due to the multiplicity of individual RS/6000 server nodes, the VSM programmer can ignore this fact and assume a single (logical) address space. In fact, this programming model is basically the same model that is used when developing uniprocessor workstation applications; the only difference is the comment-based directives that identify independent parallel operations.

This model differs from a directive approach of a data parallel model such as HPF in one important way: the VSM model is not at all concerned with data distribution among the processors. The only required directives are those needed to identify independent operations within the source code, whether they are independent iterations of a loop or independent sections of code. The VSM programmer does not worry about how to partition and distribute the data among the processors as in HPF, nor about managing buffers and synchronization as with MPI.

VSM is usually the fastest and easiest model to produce a running parallel code. However, as before, since the underlying hardware is not shared memory, VSM will not yield optimal performing applications. The trade-offs involved are very similar to the trade-offs associated with the data parallel model.

> ***Virtual Shared Memory (VSM) is usually the fastest and easiest model to produce a running parallel code.***

The VSM environment must simulate a shared memory environment on top of the physically distributed memory environment. Again, this does not come without a price. This simulation requires time and resources in order to achieve the illusion of shared memory for the programmer. When a processor makes reference to a data element that is not local to its address space, the VSM system must perform the necessary operations to find and retrieve the valid data within the machine and transfer it to that processor. Furthermore, these operations are happening continuously throughout the lifetime of the job and on many processors simultaneously.

As you can see, these issues can become complex. In a future article, we will explore these issues in greater detail.

Note: A VSM environment is available today for the RS/6000 SP from a third-party vendor, Myrias Computing Technologies. Their VSM software environment is called the Parallel Application Management System (PAMS), which has been running on the RS/6000 SP since April 1997. Although the performance is not competitive with an equivalent message-passing application, the

performance is quite acceptable given the simplicity of its use. Furthermore, initial testing up to 64 nodes has indicated that PAMS provides a scalable shared memory environment for many types of applications. Lastly, PAMS runs on any node configuration, whether the nodes are thin, wide, or high, and automatically adjusts the load to compensate for the performance differences of the underlying hardware.

The next article in this series will take a closer look at the issues involved with MPI programming on the RS/6000 SP. It will also include a discussion of available development tools (both IBM and third party) and a look at the new features in MPI-2.

David Klepacki, *IBM Corporation, 522 South Road, Poughkeepsie, NY 12601. Dr. Klepacki has been working in IBM's POWERparallel Systems Group since its beginning in 1991 as a computational physicist and scientific applications specialist with emphasis on performance benchmarking. Today, in addition to his technical endeavors, he also manages the parallel software tools segment for the technical marketing branch of the RS/6000 Division. Dr. Klepacki's current interests include performance programming, scalable parallel algorithms, scalable I/O, and portable high-performance computing tools. He holds a PhD in Theoretical Nuclear Physics from Purdue University as well as an MS in Electrical Engineering from Syracuse University.*

Xianneng Shen, *IBM Corporation, 522 South Road, Poughkeepsie, NY 12601. E-mail: xshen@us.ibm.com. Dr. Shen is a senior marketing support representative in the RS/6000 Executive Briefing Center. He has a BS and an MS in Electrical Engineering from the University of Electronic Science and Technology of China, an MS in Computer Engineering from Syracuse University, and PhD in Electrical Engineering from Syracuse University*

Help Desk and Remote Training Terminal Mirroring

By Walter Lipp, Eddie Ho, and Bill Chesky

In a remote support and training environment, system administrators can provide direct assistance by taking control of the end user's terminal and keyboard. The terminal mirroring function of AIX can help shorten the time necessary for problem determination and with less confusion.

Terminal mirroring can assist help desk personnel to provide direct help via remote online coaching. It can also play a key role in new application training for users since the trainer and user have identical screen control and interactions. The administrator can login to the application server and take control of the end user's terminal when assistance is needed. This function, available in AIX 4.2.1., is part of the service aid filesets `bos.sysmgt.servaid`. Figure 1 shows a typical environment for a remote help desk.

Overview

Terminal mirroring is a function of the tty subsystem. The tty subsystem in AIX is a stream-based modular design conforming to the UNIX industry SVR4 standard. Each session represents a tty stream. The modular and layered architecture allows session capabilities to be added or modified without relinking. This allows for the dynamic modification of protocol characteristics when the session is initialized. Figure 2 shows the structure of the tty system.

The terminal mirroring function is implemented by inserting a special streams module into the streams of two tty sessions, allowing one of the two sessions to monitor the other. A daemon is also spawned that manages certain tasks related to the initialization and termination of the terminal mirroring environment. This function is an extension of the console mirroring capability available for Symmetric Multiprocessing (SMP) Micro Channel®-based systems.

Console mirroring is designed to enable the remote system administrator to monitor and/or take control of a console connected to serial port #1 via a modem connected to serial port #2. Figure 3 shows console mirroring.

Terminal mirroring extends this capability to any two logical terminal ports including native serial ports, multiport ttys, low function terminal (lft), and network terminals such as telnet, rlogin, xterm, and so on. The pair of terminal ports include the following:

- Target station: The tty stream whose data stream/applications are monitored by another user
- Monitoring station: The tty stream that monitors the data of another users' data stream or application

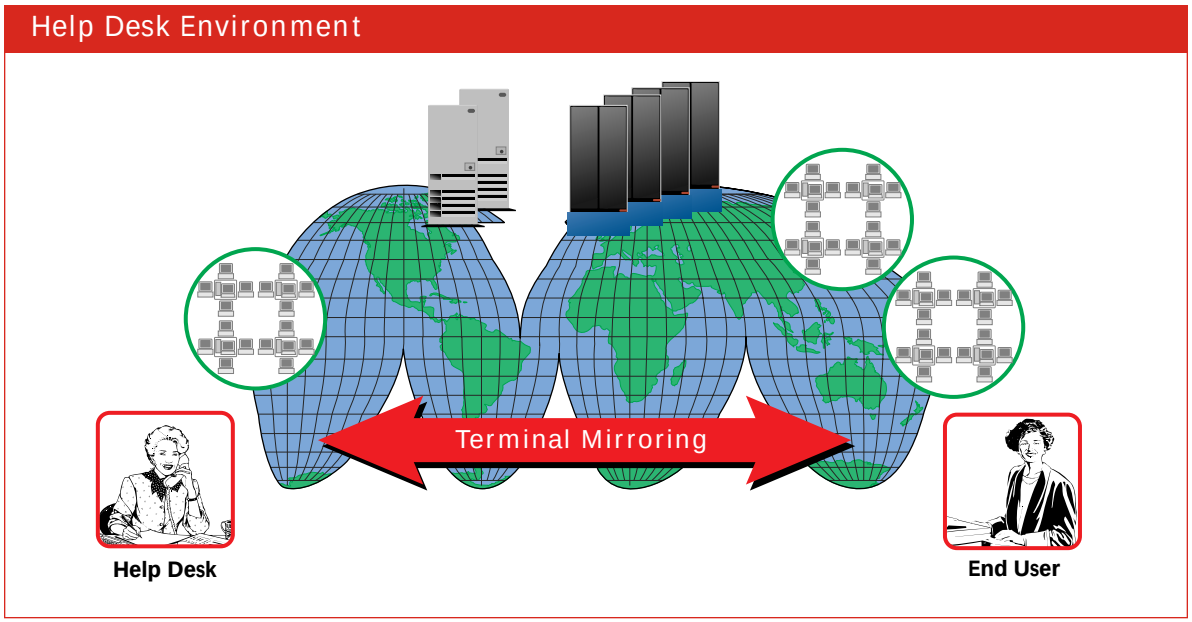


Figure 1. Typical remote help desk environment

Terminal Mirroring Environment

Once terminal mirroring is invoked, the monitoring station takes on the ID of the target station. Any keystrokes entered in the monitoring station's session are no longer seen by the device that was originally associated with that session. Instead, they are redirected to the device associated with the target station's session. Also, any output directed to the target station is routed to the monitoring station as well. Thus, users in both sessions see the keystrokes issued by either keyboard.

This rerouting of data is also true for non-keyboard I/O as well, such as `ioctl()` requests and escape sequences. For this reason, the terminal types of the two devices must be identical; otherwise, escape sequences will be misinterpreted. Both the target and monitoring station's sessions will be notified when terminal mirroring is invoked.

This function is part of the service aid filesets `bos.sysmgt.servaid`. The following summary describes the operating characteristics:

- Provides full access to the monitored station
- Works with any terminal style, including multiport adapters, modem, or network

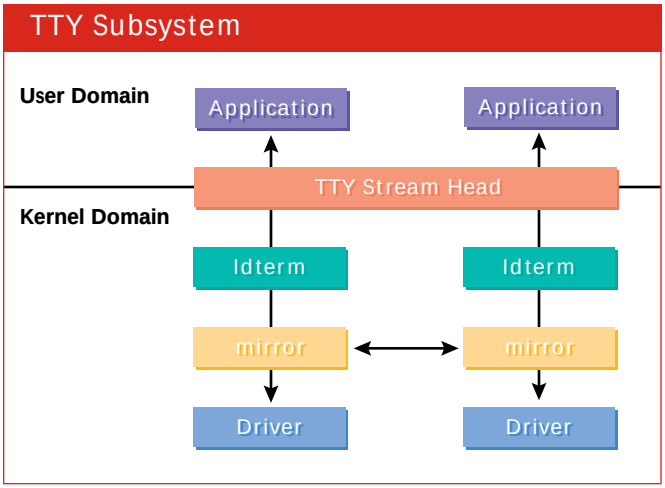


Figure 2. TTY system structure

- Requires the `TERM` environment variable to be the same in both stations; if not, the data and control sequence will not be translated properly
- Does not require the two stations to have the same baud rate and flow control characteristics to operate properly
- Compromises your administrative capabilities if the system console is used as the monitoring device, because console messages will be lost

- Use the `chdev -l ttyx-a fastcook=disable` command to disable the intelligence processing option `fastcook` of the session attribute when using the 128 port tty as a monitoring station

- Allows the maximum of one pair of station mapping per system

- For privacy, the monitored station will be notified with the following message:

```
portmir: Remote user connected,
mirroring active
```

- The session must have either an active `getty` process or a user logged in before the terminal can be mirrored

- A slight data loss may occur at the targeted station if there is outbound data from the host during the enable and disable of mirror function

- Only a character terminal can be mirrored; X-windows with graphics is not supported

Security Access

Root user has unlimited authority to monitor all stations—one at a time—with no special setup required.

Non-root users can also monitor other user stations. However, this requires the user to be logged into the target station to define a file in their home directory called `.mir`. This directory contains a list of all users who are allowed to monitor that user's sessions.

Configuration and Setup

Configuration and setup is handled by System Management Interface Tool (SMIT). The following shows the path:

```
smit -> Device -> Console -> Port Mirror
```

The fastpath is `smit portmir`.

Figure 4 shows the main panel for terminal mirroring.

Console Mirroring

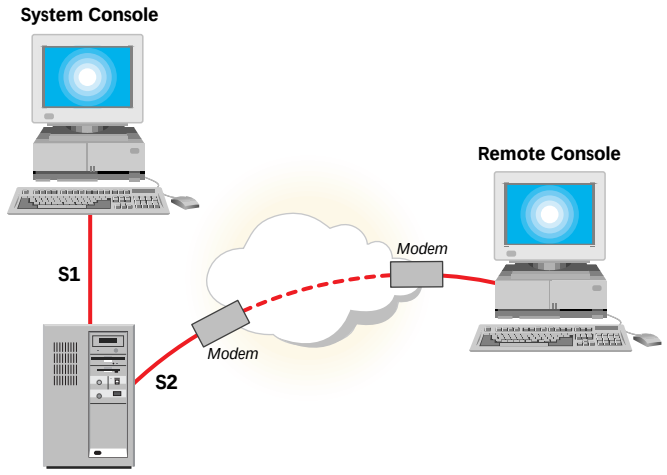


Figure 3. Console mirroring

Terminal Mirroring Components

The mirroring function has two major components:

- Port monitoring module: Responsible for data passthrough to the companion port monitoring module. It is pushed and initialized by the mirror daemon residing on the monitoring station

- Mirror daemon: Runs on the monitoring station and handles the following:

  — Setup of the port to be monitored by pushing the stream monitoring module in the targeted stream

  — Special I/O controls that are unique to the streams monitor

  — Session cleanup when mirror function is disabled

Command Descriptions

The `portmir` command allows the administrator to set up the mirror function instead of the SMIT interface.

Figure 5 shows flags associated with the portmir command.

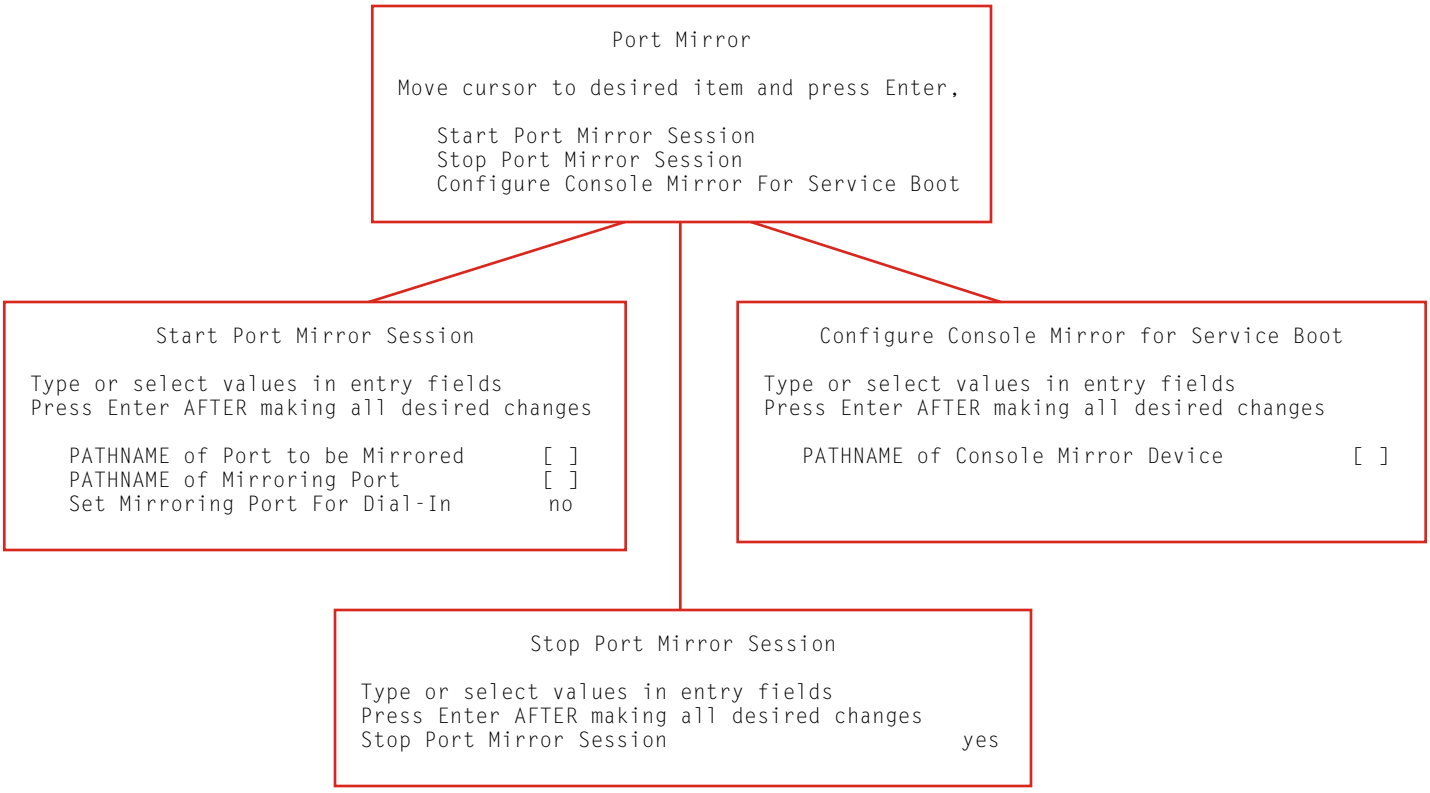


Figure 4. SMIT panels for terminal mirroring configuration

`-t <target port name>`	The station to be monitored
`-m <monitoring port name>`	Optional parameter; the default device is the current tty where the `portmir` command is issued
`-o`	Disable the terminal mirroring function
`-s mir_modem`	Service boot only. Requires `mir_modem` script to be setup for remote console mirroring. It is only used for compatibility mode with the SMP MCA system. This option can only be used when the machine is in the service mode
`-c`	Set default mirroring station for service-mode terminal mirroring
`-q`	Query current set by `-c` option
`-d mir_modem`	Set monitoring port for dial-in; requires the `mir_modem` script and is used for compatibility mode with the SMP MCA system

Figure 5. portmir command description

Examples

The following examples illustrate the use of the commands in Figure 5.

To mirror user 1 on tty1 (target) from user 2 on tty2 (monitor):

- User 1 must place user 2's login ID into `/u/user1/.mir`
- Issue this command from tty2:

```
>portmir -t tty1 -m tty2
```

Note: `-m` is optional because the default monitor is where the command is run.

To mirror tty1 (target) from user on tty2 (monitor) who is dialing in:

- Ensure that `/usr/share/modems/mir_modem` is linked to the correct modem communications script file
- Issue the `portmir` command with the `-d` option only from root user
- Issue this command:

```
>portmir -t tty1 -m tty2 -d mir_modem
```

To disable mirroring during the service boot, enter this command:

```
>portmir -c off
```

Remote Terminal Mirroring Using a Modem

It is also possible to invoke terminal mirroring using a modem. This feature facilitates true remote support by allowing help desk personnel to monitor the commands entered from an off-site system console as well as enter commands on the console directly while a customer observes.

Remote terminal mirroring is accomplished using a special chat script (designated as `mir_modem` in the usage statement), which initializes a modem connected to the monitoring station port for dial-in access.

On a newly installed AIX system, examples of these chat scripts for specific types of modems can be found in the directory `/usr/share/modems`. This is also the directory that the `portmir` command searches by default if the user does not specify the full path name for the chat script file and it cannot be found in the current directory. If it still is not found in `/usr/share/modems`, then the `portmir` command exits with an error.

The example chat scripts that can be found in this directory at installation time are as follows:

```
mir_modem.7851
mir_modem.usrobotics
```

In addition, another file `mir_modem.without.modem` is included. Since it will be discussed in more detail later, this file would most commonly come into play when using the `-s mir_modem` option.

Also contained in the `/usr/share/modems` directory is a file simply called `mir_modem`, which is a link to one of the other files (`mir_modem.usrobotics` at installation) mentioned above. This file may be used in place of the full path name to a chat script by changing the link to point to the appropriate file, then invoking the `portmir` command with `mir_modem` as the chat script file argument. Again, this would most commonly come into play when using the `-s mir_modem` option.

A user initiates a remote terminal mirroring session using the `-d` option. For example:

```
portmir -d mir_modem -t /dev/tty0
   -m /dev/tty1
```

This command causes the `portmir` command to parse the chat script contained in the file `mir_modem` and initialize the modem attached to the device `/dev/tty1` for dial-in access. At this point, a remote user would then dial into the modem, which would answer, establish connection, and begin the terminal mirroring session on the device `/dev/tty0`.

Service-mode Terminal Mirroring

Finally, terminal mirroring is also possible when booting a system in service mode, allowing help desk personnel to perform

service-mode diagnostics on a system remotely. In this mode, the target station is always the system console and the monitoring station is a device previously specified by the user.

Service-mode terminal mirroring is invoked using the following command:

```
portmir  -s mir_modem
```

Normally, a user would never invoke the `portmir` command in this manner. Rather, this invocation takes place behind the scenes when the system is booted in service mode. If the user were to attempt to execute the `portmir` command as above while in normal mode, the `portmir` command would exit with an error.

The user must take two steps while the system is running in normal mode in order to properly activate service-mode terminal mirroring for the next time that the system is booted in service mode:

1. Set the service-mode default monitoring station using the `-c` device option. For example, the command `portmir -c /dev/tty1` establishes `/dev/tty1` as the default monitoring station during service-mode terminal mirroring.

   If the default monitoring station has not been established, then service-mode terminal mirroring will not occur when booting in service mode.

   The value of this parameter can be checked using the command `portmir -q`.

   The value of this parameter can be set to `NULL`, which disables service-mode terminal mirroring, using the command `portmir -c off`.

2. Symbolically link the file `/usr/share/modems/mir_modem` to the correct `mir_modem` chat script file.

   If the monitoring station is local, the above file must be linked to `/usr/share/modems/mir_modem.without.modem`. If the monitoring station is remote, then the above file must be linked to the appropriate `mir_modem` chat script for the type of modem connected to the monitoring station.

Once these two tasks have been completed, the user simply reboots in service mode and terminal mirroring should begin automatically. In the case of remote terminal mirroring, just as with normal-mode terminal mirroring, the remote party must dial into the system and make a connection to the monitoring station's modem before the terminal mirroring session begins.

Conclusion

Terminal mirroring or port mirroring enhances the ability of system administrators to determine problems. It also provides application-level contact between the end user and help desk personnel. The result is less confusion and more immediate and direct support for users.

Eddie Ho, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. E-mail: eddieho@vnet.ibm.com. Mr. Ho is a programming consultant in the RS/6000 Executive Briefing Center. He has a BS in Computer Science from the University of Wisconsin and an MS in Computer Science from North Dakota State University.

Walter Lipp, IBM Corporation, 11400, Burnet Road, Austin, TX 78758. Mr. Lipp is a staff programmer in AIX Communications Development. He has a BS in Electronics Engineering Technology from DeVry Institute of Technology in Chicago.

Bill Chesky, EASE Software, 4108 Lewis Lane, Austin, TX 78756. Mr. Chesky is a computer consultant specializing in kernel design and development. He has a BA in Mathematics and Computer Science from Kalamazoo College.

Graphical Desktop Korn Shell

By George Kraft IV

The Graphical Desktop Korn Shell (DtKsh) is a featured part of the Common Desktop Environment (CDE). DtKsh provides a consistent and reliable graphical Motif shell language that is supported on all CDE-compliant systems.

Portability and pervasiveness are two important characteristics to consider when you are developing code. Using a programming language with a well-defined and stable Application Programming Interface (API) answers the need for portability. A programming language with a large, established installed base provides pervasiveness. Although Perl, Tcl/Tk, Common Gateway Interface (CGI), and Java have large installed bases, they are not suited for some projects. The reason for this is their inconsistent installation base due to their lack of a well-defined API or their rapidly changing API.

The Desktop Korn Shell (`DtKsh`) that comes with the Common Desktop Environment (CDE) is built on the ksh93 standard with X, Xt, Motif®, ToolTalk®, and CDE built-in APIs. Unlike Perl and Tcl/Tk, major vendors have built and supported `DtKsh` through the CDE initiative. Using `DtKsh`, desktop programmers can develop and/or prototype plug-and-play Graphical User Interface (GUI) applications that are compatible on all CDE-compliant systems without compilation. Although `DtKsh` applications are interpreted for portability, they can be easily migrated to Motif in C for performance.

Tcl/Tk can be ported to C with the aid of special Tcl/Tk libraries; however, programmers are as disadvantaged with the C Tcl/Tk libraries as they are with the Tcl/Tk shell, because of a not-so-standard Application Programming Interface. `DtKsh`, unlike Tcl/Tk, provides a well-established API set where the programmer's knowledge transcends from C to shell programming.

Desktop Korn Shell

- Based on Korn Shell
- New desktop built-in APIs
- Shipped with CDE-compliant systems
- Plug and play
- Machine-independent compatibility
- Migration path to C

Figure 1. The advantages of DtKsh

DtKsh Benefits

In AIX, `/bin/ksh` is an XPG4-compliant version of ksh88. CDE's `/usr/dt/bin/dtksh` on AIX is based on the newer ksh93 standard. Ksh93 now includes floating-point mathematics, associative arrays, new string operations, hierarchical variables, reference variables, developer-extendable APIs using attached shared libraries, and character class patterns.

George Kraft IV

Floating-point mathematics. Korn Shell variables can be type cast, or defined, to various aggregate data types. Floating-point mathematics is a new feature to Korn Shell that enables the assignment and operation of decimal values. The following example defines the floating-point variable PI, then assigns to it the decimal value of 3.1459.

```
typeset -F PI # define "PI" as a float
PI = 3.1459
```

Associative arrays. Instead of using positive integer indices, associative arrays allow elements of an array to be addressed using alphanumeric strings. The following example shows `SYSINFO` as an array containing information about an operating system. The associative `SYSINFO` array can be indexed with the alphanumeric string of `os` to find the string value of `AIX`.

```
typeset -A SYSINFO # define "SYSINFO"
                   # as an
                   # associative array
SYSINFO["os"]=AIX
```

New string operations. Six new string operations were introduced in ksh93. These new operations provide substringing and substitution of a string pattern with an alternate. Substringing permits extraction of a smaller string given an offset where to begin and possibly its length.

- A substring of a larger string can be extracted by length at a given starting point, or a substring can be taken by starting at the offset within the larger string and stopping at the end of the string. The following shows a substring of a given length:

  ```
  ${variable:offset:length}
  ```

- A substring of no particular length can be taken by just providing the offset.

  ```
  ${variable:offset}
  ```

String substitution of a character pattern can be performed for the first occurrence, repeated occurrence, at the beginning of the string (prefix), or at the end of the string (suffix).

- Substitute the first occurrence of a pattern with the alternate string:

  ```
  ${variable/pattern/string}
  ```

- Substitute all occurrences of a pattern with the alternate string:

  ```
  ${variable//pattern/string}
  ```

- Substitute the pattern prefix with the alternate string:

  ```
  ${variable/#pattern/string}
  ```

- Substitute the pattern suffix with the alternate string:

  ```
  ${variable/%pattern/string}
  ```

Hierarchical variables. Hierarchical variables, or compound names, enable C structure-like aggregate data types. This allows Korn Shell to store information in variables in an associative fashion. For example, if we had a box with the width of

```
#! /usr/dt/bin/dtksh

  main()
  {
    XtInitialize TOPLEVEL dtHello DtHello "$@"

    XmCreateMessageDialog HELLO $TOPLEVEL hello \
            dialogTitle:"DtHello" \
            messageString:"$(print "Hello\nWorld")"

    XmMessageBoxGetChild HELP $HELLO DIALOG_HELP_BUTTON
    XtUnmanageChild $HELP
    XmMessageBoxGetChild CANCEL $HELLO DIALOG_CANCEL_BUTTON
    XtUnmanageChild $CANCEL

    XtAddCallback $HELLO okCallback exit
    XtManageChild $HELLO
    XtMainLoop

   }

   main
```

Figure 2. DtKsh "Hello World" program

80 and height of 24, then we could store all that information in one hierarchical variable instead of separate and disjointed variables of storage. Each element of the compound name must be used before setting submembers.

```
BOX= # declare before assigning submembers
BOX.WIDTH = 80
BOX.HEIGHT = 24
```

Reference variables. Referencing allows a variable to point to the same value as another variable; both variables reference the same value as shown below:

```
# name reference
typeset -n FOO=BAR

FOO="Hello World"

# print "Hello World"
print ${BAR}
```

Desktop built-in commands. Korn Shell provides some standard X, Xt, Motif, POSIX® internationalization, and CDE C language APIs directly built into the shell. Direct access to these APIs from the shell provides a significant runtime performance improvement for `DtKsh` shell applications. Using the standard X and Motif APIs, with some semantic changes to the source, makes it possible for `DtKsh` shell scripts to be migrated to C and compiled.

POSIX internationalization. Korn Shell provides the shell equivalent of the C language POSIX internationalization APIs `catopen` and `catgets`. The internationalization APIs allow the shell program to change its message catalog depending on its language. Internationalized shell scripts enable multilingual support.

Character class patterns. Regular expressions in the shell are enhanced by predefining a set of character class patterns. Now we can easily match certain classes of characters by using the `[[:class:]]` notation where as class is defined as `alnu`, `alpha`, `cntrl`, `digit`, `graph`, `lower`, `upper`, `print`, `punct`, `space`, and `xdigit` as a specified class.

```
#include <stdlib.h>

#include <Xm/MessageB.h>

main(argc, argv)
        int argc;
        char **argv;

{

       Widget topLevelShell, hello, help, cancel;
       Arg xargs[10];
       int n;
       XmString title, greet;

topLevelShell = XtInitialize(argv[0], "DtHello",
       NULL, 0, &argc, argv);

hello = XmCreateMessageDialog(topLevelShell, "hello",
NULL, 0);

title = XmStringCreateSimple("DtHello");
greet = XmStringCreateLtoR("Hello\nWorld");

XtVaSetValues(hello,
      XmNdialogTitle,title,
      XmNmessageString,greet,
      NULL);

XmStringFree(title);
XmStringFree(greet);

help = XmMessageBoxGetChild(hello, XmDIALOG_HELP_BUTTON);

XtUnmanageChild(help);
cancel = XmMessageBoxGetChild(hello,
XmDIALOG_CANCEL_BUTTON);
XtUnmanageChild(cancel);

XtAddCallback(hello, XmNokCallback, exit, NULL);

XtManageChild(hello);

XtMainLoop();

}
```

Figure 3. C language "Hello World" program

```
# only print files that
# begin in upper case
print  [[:upper:]]*

# old way
print [A-Z]*
```

DtKsh 'Hello World' Source

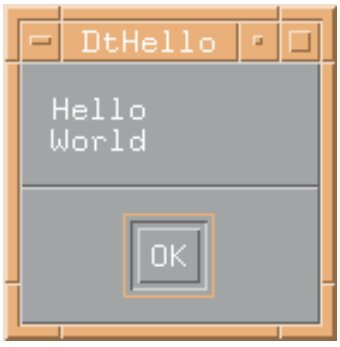
The familiar “Hello World” Motif application, shown in Figure 2, is actually written in `DtKsh` instead of C. Similar to C, we initialize the top-level shell widget, then start building the GUI application. Figure 2 shows a standard Motif message dialog using the familiar `XmCreateMessageDialog` API.

In `DtKsh`, handles to widgets can be retrieved, widgets can be managed and unmanaged, and callbacks can be created. Afterwards, the program enters into the Xt Intrinsic's main loop via `XtMainLoop` where it processes X protocol events. In this case, clicking on the “OK” button would be an event processed by the event loop.

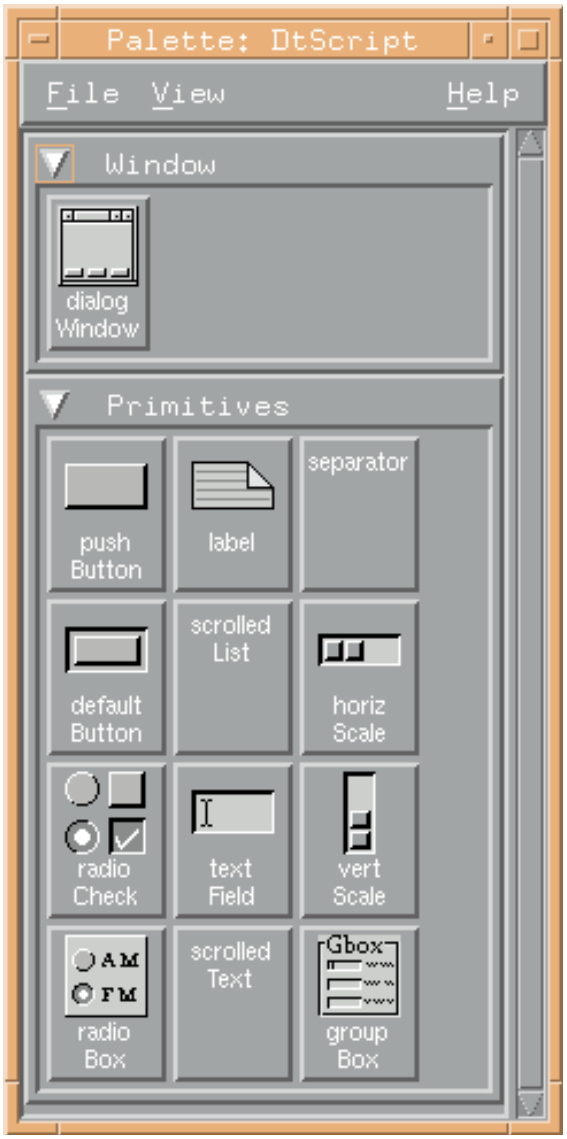
Motif “Hello World” Source

The Motif “Hello World” `DtKsh` application in Figure 2 can be easily ported to C with a few minor changes, shown in Figure 3. By adding some include files, defining some variables, adding some commas and semicolons, and sprucing up some arguments, we have a C program. The result is that `DtKsh` shell scripts make the same API calls as the C Motif application.

AIX provides some extra `DtKsh` help through a GUI builder. Developers can drag and drop widgets onto a canvas, then add some logic code to enable the application to do some work. Like any



DtKsh and Motif "Hello World" application



AIX's DtScript GUI Builder

GUI builder, the code is somewhat verbose; however, it is consistent and portable. AIX is the only version of UNIX that offers this feature.

User Extendable

Developers can create their own new APIs for `DtKsh` by creating glue layer libraries. Glue layer libraries enable `DtKsh` to be extended with built-ins for functions such as system management and networking. The performance advantage of using built-in functions rather than calling to an external command is that built-ins execute within the process of the shell script. Commands that are called externally must

create new resources in the operating system and run as separate processes.

DtKsh glue layer libraries pass arguments between a normal UNIX C library and the DtKsh shell, and they return a success or failure status. The following list provides a few rules for creating a glue layer:

- Name the function with a b_ prefix
- Function returns a 0 integer for success, or between 1 and 255 for failure
- Function should take argc and argv as input
- Link your glue layer libraries shared

```
#!/usr/dt/bin/dtksh

builtin -f ./libexample.o example # dynamically load shared
library

example hello world # call newly loaded builtin command
```

Figure 4. Example of DtKsh

```
int b_example(int argc, char *argv[])
{

      int i;

      for (i = 0; i < argc; i++) {
            printf("argv[%d] = %s\n", i, argv[i]);
      }

      return(0);

}
```

Figure 5. Example of C to shell glue

Figure 4 shows a DtKsh shell script that dynamically loads the "example" shared glue layer library. Once the glue layer library is loaded and the new built-in APIs are defined, then the script can make direct calls with arguments to the new built-in functions. In Figure 4, the example built-in is called with the hello world arguments.

By providing inline built-in functions, we can run scripts much faster because we are not relying on outside programs running as separate system processes. Figure 5 shows the C glue layer for the example built-in shared library. Following the rules outlined above, we prefix the example function with a b_, and we pass in an argument vector and its size. Then after the function has done its work, we return 0 for success and a positive integer for failure. DtKsh built-in functions can also act as procedures that pass environment variables in and out through its argument list. See *Desktop Korn Shell Graphical Programming*[1] by J. Stephen Pendergrast, Jr. (ISBN 0-201-63375-2) for more details on how to pass in and get back environment variables from built-in procedures.

Conclusion

We have seen that the Desktop Graphical Korn Shell provides programmers with the standard ksh93 baseline APIs with the addition of the X Window System®, Motif, and the Common Desktop Environment. Shell programmers can write portable shell scripts, prototype GUI shell scripts, and migrate GUI shell scripts to faster running C programs. DtKsh also provides programmers with the ability to extend the shell language with built-in shared libraries so that scripts can benefit from feature-rich libraries, such as those for configuration management.

George Kraft IV, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Kraft is an advisory software engineer for IBM's new Network Computer Division. He recently moved from IBM's RS/6000 Division where he worked on the AIX integration of the IBM Network Station. He has a BS in Computer Science and Mathematics from Purdue University.

[1] Also see the following Web site: http://www.aw.com/cp/pendergrast.html.

Effective Process Configuration Management on AIX

By Tani Haque

Organizations that produce mission-critical and business-critical software on AIX cannot afford to ignore the consequences of releasing a product that incorporates flawed software and runs the risk of recall. Such consequences can be devastating to the customer and lead to major crises when mission-critical and business-critical applications are involved. Process configuration management ensures that the final software release is complete and incorporates all the changes implemented during the life cycle.

Tertiary care hospitals—hospitals of the last resort as they are also called—know that they cannot afford to make a mistake because the patient's life is at stake. A similar scenario, albeit not quite as dramatic, also takes place in the typical software engineering organization producing embedded systems on an AIX® platform. If the embedded software is imperfect, it can potentially kill the life of the product in which it is incorporated. It also can leave the organization with no choice but to recall this product—a very costly embarrassment.

Recalling products becomes necessary in cases where poor software can cause serious damages.

For example, flawed software that is integrated into the inflight navigation system of a commercial aircraft can potentially cause a plane crash. Among many other crises that can occur, software bugs can potentially create havoc in banking transactions, leaving customers frustrated and banking operations halted. For business-critical applications, software defects can threaten the very competitiveness of the organization, causing development delays, productivity dips, and significant cost overruns—not to mention unleashing scores of unsatisfied end users and customers clamoring for their money back.

Why Use Process CM?

Organizations using AIX and other platforms are always looking for ways to achieve greater product quality, reliability, serviceability, and performance in their production of mission-critical applications. Process Configuration Management (CM) can help these organizations produce quality, embedded software applications.

Traditional configuration management primarily provides narrow version control

This article originally appeared in *CrossTalk, The Journal of Defense Software Engineering*, Volume 10 Number 4, April l997. It has been updated and is now published with permission of *CrossTalk*.

Tani Haque

functionality. This approach to CM is limited in scope because it does not allow process management and provides only unconnected islands of information to developers. Consequently, software engineers cannot bridge their efforts effectively or conduct cohesive development work in parallel and concurrently. Rather, they require automated process workflow and instant access to a centralized database of information about the progress of the development work, including the status of all software changes in the system.

Process configuration management provides the answers in distributed, heterogeneous environments. The advantage of process CM is that it combines process workflow with automation of all software changes across the life cycle—from development through maintenance.

Specifically, process CM automates the integration of version and change management on AIX and other platforms. It also considers the organizational, development, and CM processes of the organization. These processes typically cover all critical issues that impact the development organization, such as ways to achieve contractual requirements, defense standards, and other standards such as ISO 9000 or the Software Engineering Institute (SEI) certification. Other process examples include software engineering procedures, working practices for implementing process CM, and roles and responsibilities assigned to team members.

A process-based CM system can capture such processes. They can be combined to create a control plan, which will provide an important blueprint to effectively build embedded software in a cohesive development environment. The significance is that these processes can be duplicated and improved in the next project life cycle.

Tools of Choice

Currently, there is an array of CM tools from which to choose. The most appropriate ones for controlling the development of embedded applications on AIX are those that integrate change and version management while threading the processes underneath these functions. In a nutshell, these products automate all the CM functions, including defect tracking, version control, automatic building, software distribution, process workflow, and incident management. See Figure 1.

These process-driven CM solutions provide software engineers with the support they need to successfully implement parallel development, concurrent working, and distributed engineering. The result is an environment where software development is under control and ad hoc development is prevented.

Since these process CM products are based on client/server architecture, they can

Configuration Management Tools

Change tracking: Keeps a full audit trail of software change reports including design defects, software bugs, and change requests in the system. So, at any stage in the development cycle, software engineers know which change reports have been fixed and which ones are still pending.

Version management: Allows files to be checked in and out. It tracks the status of all revisions for each individual file and also maintains configuration threads (baselines) of files and their specific revisions. These files can be code or documents.

Automatic building: Allows software engineers to build different versions of the same product. It also enables the automatic control of derived files such as executables.

Software distribution: Enables developers to distribute software changes in parallel and concurrently across the development life cycle.

Process workflow: Allows software changes to be channeled automatically to the appropriate software development team member in the sequence of events of the development life cycle.

Incidents management: Allows software engineers to keep track of all the incidents raised in the system. This facilitates communication with users and customers via E-mail notifications about the status of these incidents.

Figure 1. Tools for configuration management

be used easily in both homogeneous and heterogeneous environments. These systems enable organizations to absorb workstations into their projects and to have a strategy in place for managing multiple flavors of cooperating UNIX®, OpenVMS, Windows, and Windows NT™ operating systems. This is a major advantage.

The relational databases of such process CM products include information about the application. They also provide control mechanisms to regulate the use of this information, together with the procedures governing product development. For example, this information provides details about the product design, development of software modules and associated documentation, building of configurations, issuing of product items, user-defined rules and procedures, delegation of tasks, and the objects created.

Process CM systems also make it possible to establish special databases to accommodate separate activities implemented by many development groups working on AIX. These databases can feature and control several applications developed in parallel and concurrently.

How to Implement Change Management

The following examples show how software developers and their managers can effectively implement many change management activities using a process CM system in their environments.

Creating Change Documents

Users can easily create change documents, such as defects or change requests. They can, for example, enter various attribute fields that will define the specifics of the change documents, such as severity or importance. Once these attributes have been entered, users can change the application part, which is affected by a change document. When all necessary information is entered, users can then save the change document in the system. This document is now registered and actioned to be processed or authorized by the next person in line in the life cycle of that type of change document.

Opening Change Documents List

At any point in time, software engineers can open and check the change document list. The process CM tool displays various categories that can be further filtered by querying on their attributes. This allows users to restrict the selection of change documents displayed on the screen. This capability is particularly important to managers because it provides high visibility on all change documents that are currently in the system. These managers can instantly assess and report on the status of a project.

This functionality is also critical to software engineers. They can then easily check the list of change documents that fall under their responsibility in a pending list associated with that user. This pending list also provides users with a clear picture of completed tasks and assignments that remain to be done. Ultimately, this increases the accountability of everyone involved in the project, because all team members know their roles and responsibilities during each phase of the life cycle.

> ***A process configuration management solution provides managers with many critical status reports showing the progress of the development effort.***

Reviewing Change Documents

A process CM system enables users to review, update, or delegate change documents. The system ensures that only authorized users can implement the changes, since an edit option is only available to users with responsibility to revise these documents. Other users on the team do not have access to this option.

Classifying Change Documents

A process CM system allows users to classify and query newly created or existing change documents according to their relationships to other change document types. These users also can relate application parts to change documents. In this situation, the

hierarchical structure of the application parts represents its functional breakdown. For example, a product might contain hardware and software components represented as two distinct application parts in the system. Each part would then have subordinate parts representing sibling functions.

With this logical structure, managers responsible for modifying the hardware application parts can easily identify revisions they need to implement. And the same holds true for those team members responsible for modifying the software components. Such capability is critical when building embedded software so that all application parts related to the software, hardware, firmware, and documentation can be clearly delineated. So, again, there is no confusion about the managers' responsibilities and which components need to be modified.

Implementing Changes

Software engineers can implement changes to specific files. They do this by requiring that a valid change document be related to an item before that item can be extracted. The result is efficient tracking and the ability to audit change requests while facilitating the implementation of configuration builds and baselines.

Actioning Change Requests

A process CM system allows software engineers to action or move objects, such as change requests from one state in the life cycle to the next state in the life cycle. Using the actioning process, software engineers can advance the change document to a designated state in the life cycle. The system's process workflow capability automatically delivers the change request to the individual(s) responsible for the next action in the life cycle. For example, a change request may be actioned for consideration by a Change Control Board (CCB) authorized for scheduling work. The CCB could action this type of change request so it would appear in the pending list for the designated developers. The developers would, in turn, action the change request to perhaps a "ready for test" state.

This process allows each change document to pass through its normal path of progression. The system will verify whether the user has the necessary privileges to action the change request form to its intended final life cycle. The system also promotes quality by requesting that the individual or CCB responsible for approving change requests determines that the work is performed correctly.

How to Implement Version Control

An appropriate process CM system for producing embedded software allows developers to implement version and revision control of their design and coding, workset management, build management, release control, maintenance and traceability, and process integration with change management. The version management function automates the versioning, building, release, and distribution of software across the life cycle. The following practical examples illustrate how such a function might be implemented with a process-based CM system.

Users can access a list showing the actual physical directory structure that contains the version objects. Project team members can refer to this physical directory structure to easily perform typical file operations in version management such as checking files in and out. They can also perform version management operations, such as extract (checkout) and return (checkin).

Software engineers using a process CM system can perform version management in parallel and concurrently. They do this through worksets that contain groups of objects. It is worth noting that a typical product may have a workset designated for functional changes, another for addressing maintenance, and yet another for customization.

By creating and maintaining worksets, users can concentrate on their activities and avoid interfering with other developers who are concurrently working on different aspects of the same application. This approach to version management significantly increases the productivity of the team and speeds the entire development effort. It also enables software engineers to deal with the complexities of producing large applications.

Before consolidating their work, users can compare their changes and make decisions about conflicting changes. This ensures that only wanted changes are merged and absorbed into the ongoing development effort. Under the process CM system, changes are merged by merging different worksets together and including them in the mainstream development workset.

Benefits of an Integrated Process CM Approach

A process configuration management solution, which integrates process workflow with the version and change management functions, provides managers with many critical status reports showing the progress of the development effort. Such deliverables may include traceability and management reports, change management reports, configuration management reports, among many other important metrics. This gives managers the ability to choose—at any given point in time—to redeploy staff and resources according to the changing needs of the project.

Software engineers can orchestrate their efforts harmoniously throughout the development life cycle. The result is an integrated approach that allows operations to flow smoothly. The flow of communication also greatly improves laterally and vertically, so everyone can be in sync to produce quality embedded software on AIX.

Finally, a proper CM system provides facilities for telecommuting users to implement configuration management from the Internet and other remote sites. These concurrent activities can be conducted under the umbrella of a set of common processes in a completely controlled and secure environment. This not only fosters cohesiveness in the development environment, but it also optimizes software engineering efforts across distributed, heterogeneous networks.

Conclusion

Today, organizations that produce mission-critical and business-critical software on AIX cannot afford to ignore the consequences of bringing to market a product that incorporates flawed software and risk the chance of recall. Such consequences can be devastating to the customer and lead to major crises when mission-critical and business-critical applications are concerned.

When software must be absolutely and positively right, process configuration management is the only way to go. It ensures that the final software release is complete and incorporates all the changes implemented during the life cycle.

Today's competitive market dictates the need to place the development of embedded software systems under active control of process CM. This puts the organization on a path to avoid serious software glitches that can lead to expensive product recalls and result in profit losses and unhappy customers. Call it an insurance policy, if you will. In today's complex development environment, process configuration management is mandatory. It's not just another piece of luxuryware.

10 Benefits of Process CM at a Glance

The unique benefits of process CM to the organization include:

- **Better management of software assets**
- **Effective parallel, concurrent, cross-platform, and Web developments**
- **Defined, repeatable, and improvable processes**
- **Higher software quality**
- **Proof to the customers that their requirements are being met**
- **Smooth process workflow**
- **Compliance with ISO 9000 and SEI certification**
- **Complete traceability of software changes**
- **Critical metrics such as CM reports and impact analysis**
- **High visibility over the entire project**

Tani Haque, SQL Software, Vienna, VA. E-mail: info@sql.com. Phone: 703-760-0448. Web: www.sql.com. Mr. Haque is CEO and founder of SQL Software, a leader in process configuration management. He has over 20 years of experience in software engineering. He created the first process engine even before the Software Engineering Institute established the Software Maturity Model, and he integrated this leading-edge technology into PCMS .Dimensions, a comprehensive process CM solution by SQL Software.

AIX Questions

The AIX Solution Provider Technical Support Group in Austin, Texas, supports software vendors who are developing or porting applications to AIX. This article is a compilation of questions that are frequently asked by vendors. The name of the responding Technical Support Group staff member appears after each response.

Where can I obtain information about the ANSI C standards?

You can find information about the ANSI C standards from the following URL:

http://www.ansi.org

—Jeff Simon

How can I measure performance of a context switch for threads and processes?

A *context switch* is the switching from one thread to another inside a process. The cost of such an action refers to a performance issue. AIX provides a tool called `vmstat` for monitoring such activities. The number (rate) of kernel-thread context switches can be found under the "cs" column, which is under the "faults" column.

A context switch consists of the following:

- Saving the machine state of the departing process
- Recalling the machine state of the selected process
- Mapping mapping (`u_area`) and other virtual space of the selected process
- Switching CPU to execute with the registers of the selected processes

When another process is given control of the CPU, the context, or working environment, the previous process must be saved and the context of the current process must be loaded. Since AIX has an efficient context switching procedure, each switch is inexpensive in terms of resources. The `vmstat -s` tool can be used to monitor CPU context switching (for example, dispatch of a new process). If there is any significant increase in context switches, you should do some further investigation.

A fileset called `bos.acct` contains `vmstat`. See InfoExplorer™ for additional information about `vmstat`.

—Jeff Simon

How can I determine if a deadlock on AIX has occurred?

Run the `dlock` subcommand to determine if the condition has occurred. A deadlock occurs when two or more threads are waiting indefinitely for an event that can only be caused by one of the waiting threads. When such a state is reached, these processes are considered deadlocked.

—Wade Carlin

Wade Carlin

Jeff Simon

How can I map a file system object into virtual memory using mmap()?

The `mmap()` function explicitly maps a file system object into virtual memory. Figure 1 shows an (unsupported) example.

—Hung Dinh

```
/*    Program shows how to use  function call mmap() to map a
      file system object into virtual memory     */

#include <sys/signal.h>
#include <sys/types.h>
#include <sys/mman.h>

#define SADDR   0x40000000

main()
 {
 int i;
 char *mmptr;

/* mmap function invocation  */

   mmptr = (char *)mmap((char *) SADDR,256 , PROT_READ|PROT_WRITE,
            MAP_ANONYMOUS|MAP_FIXED, -1, 0);

/* SADDR is the starting address of the mem region to be mapped
   256 is the # bytes of the region to be mapped

   PROT_READ   region can be read

   PROT_WRITE  region can be written

   MAP_ANONYMOUS specifies the file descriptor of the file system object
   to be mapped. If the MAP_ANONYMOUS flag is set, the fd parameter
   must be -1. MAP_ANONYMOUS means the creation of a new anonymous mem
   region that is initialized to all zeros

   MAP_FIXED specifies the mapped region be placed exactly at the address
   specified by the addr parm
*/

    if (mmptr == (char *) -1 )
       {
         perror("Problems with mmap ");
         exit(1);
       }

 /* write some data into the mem region to be mapped */

 for(i=0;i<256;i++)
 {
     *(mmptr+i) ='Y';
 }

 /* verify the data is written  */
 /* this should print out 256 'Y's as mapped by the above for loop */
 printf(" mapped data is  %s   \n", (mmptr));
 }
```

Figure 1. An mmap() function0 example

How do I get Common Desktop Environment (CDE) to read my .profile?

The `.dtprofile` file in your home directory controls whether your `.profile` is read. Uncomment (remove the leading #) on the line that contains `DTSOURCEPROFILE=true`.

—Wade Carlin

I want to use mutexes and condition variables across multiple processes, but cannot find the constant PTHREAD_PROCESS_SHARED.

The `pthread_condattr_setpshared()` function is present but not implemented on AIX at this time. It simply returns as shown in Figure 2.

The function `pthread_mutexattr_init()` is also present, but currently not implemented on AIX. The `pthread_mutexattr_t` is the attribute structure, and this function on AIX marks it as being initialized. There is no real initialization to be done because there are no attributes, as shown in Figure 3.

Shared mutexes and condition variables will be added in AIX 4.3. There are no plans to implement this functionality in AIX 4.2 or prior releases.

—David McCloud

What is the performance of different IPCs (semaphores, message queues, pipes)?

Hung Dinh

For any form of Interprocess Communications (IPC), the overhead is usually attributable to the time required to execute the system calls (`read, write, msgsnd`) and the time required to move the data between the processes. Message queues are faster than pipes and FIFOs, since the latter two techniques use the general `read` and `write` system calls. Message queues have their own system calls, which can be implemented efficiently. Since the semaphore calls do not transfer a message between the user process and the kernel, they are the most efficient in terms of speed. The following book details the performance aspect of different IPCs: *UNIX Network Programming* by W. Richard Stevens (ISBN 0-13-949876-1).

—Jeff Simon

I have a huge C++ non-threaded application that I need to port over to use threads; however, I would like to avoid recompiling the application.

For C++ programs, you must recompile the application. This is because applications that use threads must to be compiled with one of the thread-aware compiler invocations

```
int
pthread_condattr_setpshared (const pthread_condattr_t \\*attr,
                             int pshared)
 {
     return (ENOSYS);
 }
```

Figure 2. PTHREAD_PROCESS_SHARED

```
int
pthread_mutexattr_setpshared (const pthread_mutexattr_t    \\*attr,
                              int pshared)
{
      return (ENOSYS);
}
```

Figure 3. The pthread_mutexattr_t attribute structure

(`xlC_r`, `xlC_r4`) to bring in the proper flags (see `/etc/xlC.cfg`; for example, the `options` field, and the order of the libraries listed under `libraries2`). If you did not compile your application using one of the thread-aware compiler invocations, then you must recompile your code.

—Jeff Simon

How do I set up a user with the ability to remotely logon to an AIX machine without being prompted for a password?

The `$HOME/.rhosts` file defines which remote hosts (computers on a network) can invoke certain commands on the local host without supplying a password. See InfoExplorer for the `.rhosts` file format.

—Jeff Simon

What do I look for when the AIX system hangs?

System hangs inside the kernel are generally caused by high-priority (low-value) threads or missed events. A missed event is one that has posted or awakened and now has a thread sleeping on it. The thread sleeping on this event has somehow missed the post or wakeup.

—Wade Carlin

Compiled by Jeff Simon, IBM Corporation, 11400 Burnet Road, Austin, TX 78758.