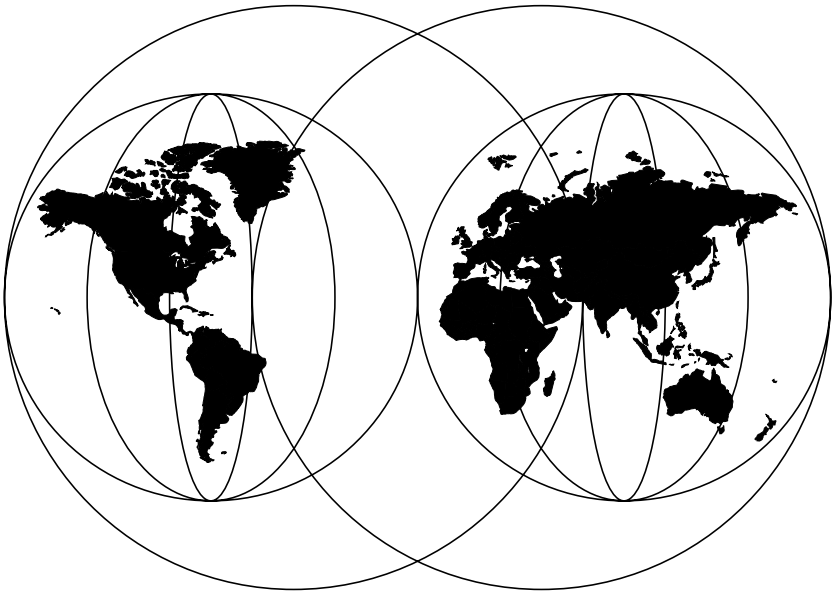
AIX 64-bit Performance in Focus

Andy Hoetzel, Rosa Fernandez, Motonobu Koh, Romuald Marshall
Dave Martin



International Technical Support Organization

http://www.redbooks.ibm.com

SG24-5103-00

SG24-5103-00

International Technical Support Organization

AIX 64-bit Performance in Focus

April 1998

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix F, “Special Notices” on page 179.

First Edition (April 1998)

This edition applies to AIX Version 4, Release 3.

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

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

Contents

Figures

Tables

Preface

This redbook focuses on 64-bit performance issues and AIX V4.3 performance enhancements. It describes in detail modifications made to the performance tools that support 64-bit AIX.

System and application performance tuning topics are covered, with an emphasis on large memory and file issues. Tuning for both 32-bit and 64-bit applications is described. Furthermore, 64-bit programming issues for new applications and the migration of existing 32-bit applications are described.

This redbook is intended to give Technical Sales Representatives, Technical Consultants, System Administrators and System Engineers an in-depth understanding of 64-bit AIX performance and tuning. Developers involved in writing 64-bit applications and migrating existing 32-bit applications to the 64-bit environment will find the chapter on migration very useful.

A new mindset will develop as large memory 64-bit machines become generally available, leading to unusual and unconventional programs that take full advantage of large memory and other increased resources. This redbook is intended to prepare Developers, System Engineers and others to handle the performance issues associated with such applications.

To benefit from this redbook, the reader should have a basic understanding of the hardware and software used in modern information systems. Detailed knowledge of specific hardware or software products is not assumed.

The Team That Wrote This Redbook

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

Andy Hoetzel is an International Technical Support Specialist for RS/6000 and AIX Performance at the International Technical Support Organization, Austin Center. He writes extensively and teaches IBM classes worldwide on all areas of AIX internals, performance and tuning. Andy holds a master of science degree in computer science from the University of Texas at El Paso. Before joining the ITSO, Andy worked in the AIX Competence Center in Munich, Germany as an AIX Technical Support Specialist.

Rosa Fernandez is an IT Specialist in France for RS/6000 at the West Area AIX Competence Center in Paris, France. She has six years of experience as a developer in CAD/CAM, device drivers and graphics real-time applications.

She joined IBM in 1990 and was in charge of the national technical support for CADAM products for two years. After four years as marketing and press manager for CATIA/CADAM Products, she joined the RS/6000 division. She holds a master of computer science from the University of Tours in France.

Motonobu Koh is an IT Specialist in Japan. He has six years of experience with AIX and is a certified Advanced Technical Expert RS/6000 AIX V4.2. His areas of expertise include performance measurement and tuning, capacity planning, and eNetwork Software.

Romuald Marshall is an IT Specialist in Singapore. He has nine years of experience with AIX. He holds a degree in Electrical and Electronic Engineering from the National University of Singapore. His areas of expertise include performance analysis and databases.

Dave Martin is an AIX Product Developer in the UK. He has six years of experience with the DirectTalk for AIX product. He holds a degree in Electrical and Electronic Engineering from The Queen's University of Belfast. His areas of expertise include performance measurement and tuning and capacity planning.

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

Marcus Brewer
International Technical Support Organization, Austin Center, Texas, USA

John Weiss
International Technical Support Organization, Austin Center, Texas, USA

Tara Campbell
International Technical Support Organization, Austin Center, Texas, USA

Steve Gardner
International Technical Support Organization, Austin Center, Texas, USA

Jasenn McNair
International Technical Support Organization, Austin Center, Texas, USA

Terry Rosser
International Technical Support Organization, Austin Center, Texas, USA

Janice Hawley
International Technical Support Organization, Austin Center, Texas, USA

Mathew Accapadi
IBM Austin

Boyd Murrah
IBM Austin

Ahmed Chibib
IBM Austin

Herman Dierks
IBM Austin

Ryan France
IBM Austin

Augie Mena III
IBM Austin

Stephen Nasypany
IBM Austin

Bret Olszewski
IBM Austin

Deanna Quigg
IBM Austin

David Sheffield
IBM Austin

Luc Smolders
IBM Austin

Chet Holliday
IBM Dallas

Harold Lee
IBM Dallas

Jhy-Chun Wang
IBM Poughkeepsie

Bill Oswald
IBM Rochester

Marcel Fryters
IBM Toronto

David Olshefski
IBM Watson Research Center

Heidemarie Hoetzel
IBM Germany

Comments Welcome

Your comments are important to us!

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

- Fax the evaluation form found in “ITSO Redbook Evaluation” on page 199 to the fax number shown on the form.
- Use the electronic evaluation form found on the Redbooks Web sites:

  For Internet users `http://www.redbooks.ibm.com`
  For IBM Intranet users `http://w3.itso.ibm.com`

- Send us a note at the following address:

  `redbook@us.ibm.com`

Chapter 1. Introduction to 64-bit Components

64-bit computing and 64-bit technology is a current topic of discussion between computer engineers and users. In certain areas of the computer industry, 64-bit technology is, truly, the state of the art. It is clear that 64 bits will typically give you more computing power than 16 bits or 32 bits. However, 64-bit advantages differ between environments. That is why we begin this redbook by defining the capabilities of IBM RS/6000 64-bit and 32-bit systems.

With the introduction of AIX V4.3 and the announcement of the RISC System/6000 model S70, IBM is providing *64-bit solutions* for its customers.

From the start, the PowerPC was designed as a 64-bit architecture (with 32-bit mode as a functional subset). The 64-bit capability is not an adaptation or a "remodelling" of an existing 32-bit architecture, as are some of its competitors.

This chapter describes the main 64-bit benefits for users and highlights the different components of the 64-bit RS/6000 solutions, such as:

- 64-bit machine architecture
- 64-bit operating system capabilities
- 64-bit application software

We continue by referring to the subsequent chapters that describe the considerations and procedures that must be taken to move your solution into the 64-bit dimension without any performance degradation.

1.1 64-bit Machine Architecture

All computers handle data in chunks of a fixed size. Originally, almost all parts of a computer system operated on data of the same size. These chunks of data were referred to as the *wordlength* of that specific machine. Since it governed many of the capabilities, the wordlength was a key machine characteristic. Systems were classified as "8-bit", "16-bit" or "32-bit", and there was a clear implication that larger was better.

Defining the size of a modern system is more difficult and does not necessarily have the same significance. Today’s designs tend to use different data sizes in different parts. For example, most of the RISC System/6000 Micro-Channel Architecture (MCA) platforms have a 128-bit memory bus width and are 32-bit machines. Also, most machines, such as all RISC System/6000s, handle floating-point data in 64-bit registers.

What is the size of a machine that has a 32-bit integer part, a 64-bit floating-point part, a 52-bit virtual address, and a 256-bit memory bus?

Since the smallest size, typically the integer part, is the limiting factor, this would be defined as a 32-bit machine. Although, in many aspects, this machine has parts that are as large or larger than its 64-bit counterparts.

1.1.1 Basic Concepts

From an operational point of view, an architecture is said to be 64-bit when:

- It can handle 64-bit-long data. In other words, a contiguous block of 64 bits (8 bytes) in memory is the elementary unit that the CPU can handle. This means that the instruction set includes instructions for moving 64-bit-long data and arithmetic instructions for performing arithmetic operations on 64-bit-long integers. Arithmetic instructions on floating-point numbers are already implemented with 64-bit-long instructions in 32-bit machines.
- It generates 64-bit-long addresses, both as *effective addresses* (the addresses generated and used by machine instructions) and as *physical addresses* (those used by the memory cards plugged into the machine memory slots). Individual processor implementations may generate shorter addresses, but the architecture must support up to 64-bit addresses.

Therefore, the two great advantages of a 64-bit architecture are the possibility of using long integers in computations and the capability of addressing large memory spaces.

Note that *virtual addresses* govern the amount of memory space addressing available to applications. They are not specified by the processor architecture. Since virtual addresses are handled by the AIX Virtual Memory Manager (VMM), the operating system defining its virtual memory implementation defines the virtual address limits and application capabilities for using the memory.

In the same manner, large file access is not specified by the processor architecture. For example, AIX V4.2 implements support on RS/6000 32-bit machines for large file access (beyond the 2 GB limit). If you want more details about AIX V4.2 large file support implementation, see the redbook *AIX Version 4.2 Differences Guide*, SG24-4807.

1.1.2 PowerPC: A 64-bit Architecture Design

The full 64-bit instruction set of the PowerPC architecture has been implemented on a new line of *RS64 PowerPC* processors which were

delivered with the RS/6000 S70 machine. Of course, the 32-bit instruction subset of the PowerPC architecture is still present, ensuring binary compatibility for any 32-bit applications that currently run on the PowerPC. Moreover, 32-bit applications and 64-bit applications can run concurrently on the same machine. Hardware 32-bit binary compatibility on PowerPC architecture does not rely on high-overhead emulation techniques and has required no development efforts.

This scenario lets customers make a smooth migration that satisfies all their environmental demands, whatever they may be. Ultimately, RS/6000 customers can run their established 32-bit information technology investments, at the same or better performance level, in the new 64-bit environment.

One of the main successes of the 64-bit computer system is its 32-bit compatibility and interoperability. At the PowerPC hardware level, 32-bit compatibility is guaranteed.

Table 1. 32-bit/64-bit Compatibility

	32-bit Applications	64-bit Applications
32-bit platforms	Binary compatibility	Fail to execute
64-bit platforms	Binary compatibility	The only platform to run 64-bit

The line of PowerPC 60*x*e processors (PowerPC 601, PowerPC 603e, PowerPC 604, PowerPC 604e, PowerPC 604e3) is still a 32-bit subset implementation of the 64-bit PowerPC architecture.

The RS64A PowerPC chip is the first full 64-bit PowerPC implementation.

For more information on the 64-bit architecture, see the redbook *AIX Version 4.3 Differences Guide*, SG24-2014.

1.1.3 Other Hardware Architectures

For the reasons explained above, the system architecture base for systems with large physical memory, or any memory above the 32-bit real address, is the Common Hardware Reference Platform (CHRP). AIX supports real memory addressing greater than 32 bits only on CHRP systems. This feature does not support the combination of Micro-Channel on top of CHRP (PCI-based) systems. Specifically, MCA DMA services remain a 32-bit subsystem.

1.1.4 Benefits of 64-bit PowerPC Architecture

The benefits of 64-bit PowerPC architecture can be summarized as follows:

- Extended-Precision integer arithmetic
- Access to larger executables
- Access to larger data
- Access to larger file datasets
- Access to larger physical memory
- Access to higher SMP server scalability

1.1.4.1 Extended-Precision Integer Arithmetic

The ability to use very long integers in computations is a feature that can be very helpful in specialized applications. Some applications need to deal with integers or bit strings larger than 32 bits. The 64-bit RS64 PowerPC does this more efficiently than 32-bit hardware.

For example, programs that perform data matrix manipulation can deal with large sets in potentially half as many references and logical/arithmetic operations as before. Programs that perform software operations on bit areas (graphics) in virtual storage can deal with twice as much data per operation as before.

1.1.4.2 Access to Larger Executables

The ability to use a very long integer as an offset address allows applications to handle, within the same executable, larger data areas such as `malloc` areas and initialized and uninitialized variables. This support means that it is possible to develop much larger and more complex applications than with today's 32-bit applications.

However, most current applications do not reach or even approach an instruction number limit. The most important application benefit will be the availability for a program to handle many more and much larger variables and constants.

1.1.4.3 Access to Larger Data

Another advantage of the 64-bit architecture is the capability of managing very large amounts of data, possibly in real memory. This simplifies the task of the applications that need to handle big data sets.

Examples are typically in the areas of multimedia, statistics, decision support and databases. Once again, most of the existing applications do not find 32-bit environments limiting, but the demand for 64-bit capability will increase in the future as more powerful and demanding applications are developed.

The ability to handle elementary data that is larger than 32-bit is also very desirable. However, many advanced applications still work on data smaller than 64-bit.

For instance, in the multimedia environment, sound requires 16-bit data and graphics are based on 24-bit (and sometimes 32-bit) data.

1.1.4.4 Access to Larger File Datasets

The ability to create and maintain very large file systems is increasingly important for many users. Today's applications tend to require larger and larger sets of data. Modern techniques, such as datamining, are based on fast access to all data. New and emerging applications areas, such as multimedia, often use and produce huge amounts of data. The 64-bit architecture allows efficient handling of large amounts of data.

As seen before, accessing large files (beyond the 2 GB limit) is an operating system issue that has been implemented since AIX V4.2. However, in some cases it is not sufficient to access such large files; users want to handle, in memory, the data coming from files. Moreover, when you are able to handle large data in memory, you increase the number of files you are able to manipulate.

That's why large files support and larger file datasets are usually issues mentioned as 64-bit issues, because they depend on the large memory support provided by the 64-bit architecture.

1.1.4.5 Access to Larger Physical Memory

As mentioned before, the physical addresses that a 64-bit CPU generates are up to 64 bits in length. Once again, this eliminates the 4 GB real memory limit that all 32-bit architectures have. Many of today's 32-bit systems don't even support as much as 4 GB of real memory and have much lower physical limits. Not many UNIX installations currently have that much memory, but, as for the previous items, application needs grow every year, and real memory greater than 2 GB will become more common in the future.

1.1.4.6 Access to Higher SMP Server Scalability

As seen before, 64-bit hardware expands addressability. Greater addressability allows for more memory and, consequently, for more processors.

64-bit technology lets you scale enterprise SMP servers to higher capacity performance.

1.2 64-bit Operating System Capabilities

Building a 64-bit machine around PowerPC processors is fairly straightforward, so there is no lack of capable hardware. However, to really exploit large size processors, the entire system has to support 64-bit. Designing an operating system for 64-bit is more challenging, especially in the UNIX open arena, where portability and standard conformance are key aspects.

Along with the introduction of 64-bit machines, AIX operating system enhancements exploit large memory, expose a standard-compliant 64-bit Application Programming Interface (API) to applications and middleware, and maintain binary compatibility with current 32-bit applications that are able to run concurrently.

Two distinct topics are included in the term *64-bit standards*. One is the latest step in the evolution of open systems standards, the Single UNIX Specification Version 2, also referred to as UNIX98. The second is a broad industry agreement on data size in 64-bit programs, referred to as the *LP64* data model, described in 5.1.1, “64-bit C Programming Specifications” on page 61.

UNIX98 includes 64-bit computing features, without actually defining it, and does not specify any dependency on 64-bit hardware. The specification defines a programming environment for large files that allows both 32-bit and 64-bit programs to have this capability. The specification also cleans up a few APIs that carried implications of 32-bit data types, allowing a 64-bit programming environment the same opportunity to have standard conformance as a 32-bit programming environment. The nature of the specification is such that a conforming 64-bit environment might exist on a system that supports only 64-bit programs, or on a system that supports other environments (such as 32-bit binary compatibility) in addition to 64-bit programs.

Consider the benefits of 64-bit computing for several customers, each with different information technology investments that are based on existing 32-bit system implementations. It becomes obvious that 64-bit computing will complement 32-bit computing in different ways for each customer, and each customer will exploit the various elements of 64-bit computing at different speeds.

For these reasons, the AIX performance tools changes described in Chapter 3, “Performance Tools Changes” on page 31, are minor and reflect IBM’s intent to preserve the general look and feel of the existing tools, yet expand

their scope to cover 64-bit applications. Therefore, customers can take advantage of a larger SMP provided by a 64-bit hardware architecture and continue to work with the same tools they used on their 32-bit RS/6000 machines.

Chapter 4, “Performance Tuning Considerations” on page 49, describes 64-bit performance tuning and gives the new parameters and main characteristics needed to handle large memory and files without any revolutionary concept introduction. AIX V4.3, since it is a 32-bit binary compatibility and interoperability operating system, allows 64-bit capabilities while it maintains all well-known tools and features used by AIX system administrators.

In these ways, AIX V4.3 gives IBM customers a smooth way to migrate the 64-bit environment capabilities with minimal user training and system management changes.

1.2.1 AIX V4.3 Key Messages

Users new to AIX V4.3 should understand the following key messages in order to avoid some of the misconceptions about the support of both 32-bit and 64-bit programs by AIX V4.3.

Note

The following is not a complete set of features in AIX V4.3, but rather those features relevant to 64-bit function and application migration.

Complete binary compatibility for 32-bit programs in AIX V4.3

- 32-bit programs run on the 64-bit PowerPC.
- It is not necessary to modify or recompile existing 32-bit programs.
- 32-bit programs will run on both 32-bit and 64-bit hardware, unchanged.
- There is 100 percent compatibility in 32-bit mode.
- There is no performance penalty (or gain) for 32-bit applications.

PowerPC is a 64-bit architecture

- 64-bit environment is a superset of 32-bit.
- 32- and 64-bit environments are native.
- There is no emulation whatsoever in either mode.

There are no barriers between 64-bit and 32-bit

- They are complementary, not competing, environments.
- AIX V4.3 enhances 32-bit systems.
- AIX V4.3 enables 64-bit systems.

- 64-bit environment is an upwards-compatible AIX addition.
- 64-bit is not just a replacement for existing 32-bit functions.
- AIX V4.3 provides two application environments on 64-bit systems.
- There is only one AIX product for both 32-bit and 64-bit platforms.
- There is no intention to force all applications to become 64-bit.

The 4 GB limit for system memory has been lifted

- 4 GB (2^{32}) was the limit for 32-bit PowerPC platforms.
- AIX V4.3 now supports memory > 4 GB.
- AIX V4.3 supports 16 GB memory in the RS/6000 S70 server.
- AIX will automatically extend to larger memory sizes when available.
- 32-bit programs are still limited to < 4 GB each (because there are only 16 x 256 MB segments available in the 32-bit AIX model).
- On the S70, a number of large 32-bit programs can *each* have up to 4 GB of real memory.
- In other words, a large 32-bit workload can make good use of any memory > 4 GB.
- 64-bit programs can access 2^{60} bytes (there are 2^{64} segments available).

Application address space can be > 4 GB

- On an S70 with AIX 4.3 with programs compiled for 64-bit.
- For applications that have outgrown 32-bit addressing.

It is not necessary to recompile for performance

- 32-bit apps will perform no differently on a 64-bit machine if recompiled as a 32-bit application, neglecting any compiler improvements.
- 32-bit apps recompiled to 64-bit may see very slight performance variation either way. See 2.6, “Performance of Migrated Programs” on page 23.
- Default compiler mode is to produce 32-bit executables.
- Simple flag (-q64) to produce 64-bit executables.

Kernel is 32-bit, with additional 64-bit extension on 64-bit platforms

- Kernel does not need to be 64-bit to support full 64-bit function.
- 32-bit Kernel maintains robustness and compatibility with 32-bit.
- Kernel extensions provide all the function required by 64-bit programs.
- New 64-bit Application Binary Interface (ABI).
- Binary compatibility for device drivers.
- VMM now supports 64-bit process address spaces.
- Where necessary, some system calls are modified for 64-bit arguments (for example, files and memory operations).
- There are some device driver implications.

Full co-existence of 32-bit and 64-bit processes

- On 64-bit machines 32- and 64-bit processes can be mixed.

- There is *no* performance penalty or compatibility issue.

Full interoperability for 32- and 64-bit processes

1. Process to Process
    - Can share files, memory, IPC resources, signal each other (but watch alignment of shared memory).
    - Can `exec()` each other transparently.
    - Both 32- and 64-bit can set process limits for the other type process.
2. 32-bit and 64-bit Infrastructure
    - Compiler remains 32 bit.
        - Runs on both 32-bit and 64-bit platforms.
        - Is able to produce both 32-bit and 64-bit executables.
    - Header files changed to support both environments.
    - Infrastructure of shared libraries enhanced.
        - Same pathnames in both environments.
        - Maintaining single source files, makefiles, is straightforward.
        - Tool to manage shared libraries defaults to 32-bit content.
3. 32-bit and 64-bit Kernel Support
    - Both 32- and 64-bit virtual address spaces supported.
    - VMM and dispatcher and others involved.
    - 32- and 64-bit have equal access to system services.
    - Default behavior always favors 32-bit compatibility.
    - Device drivers generally isolated by kernel from processes.

Can I run 64-bit applications on 32-bit machines?

- Obviously, a 64-bit program will *not* run since the 32-bit PowerPC lacks the 64-bit instructions. See Table 2.
- But *compiling* and *linking* of 64-bit programs on 32-bit machines is quite possible (if the relevant libraries and so forth are installed).

Table 2. Will It Work?

	32-bit platform	64-bit platform
Program compiled for 32-bit	OK, normal performance	OK, normal performance
Program compiled for 64-bit	Will not run. The 32-bit PowerPC processors and all RISC processors do not include any 64-bit instructions.(1)	Performance may be better or worse than 32-bit program.

(1) AIX displays an error message similar to:
`exec(): 0509-036 Cannot load program kt_64 because of the following errors:0509-032 Cannot run a 64-bit program on a 32-bit machine.`

AIX commands and tools

- Most elements of AIX remained 32-bit: they don't need 64-bit abilities.
- Most tools that need to support 64-bit do so (to handle 64-bit object formats, process contexts and process address spaces).
- All tools and commands will continue to support 32-bit.
- Compilers and linkers support 64-bit (note, this does not mean that they are 64-bit applications).

1.3 64-bit Application Software

As shown above, 64-bit computing is mainly about supporting and exploiting large memory. Therefore, a 64-bit machine running with 256 or 512 MB memory will have little advantage over a 32-bit counterpart.

The same is true for 64-bit applications. If your application will manage less than 2 GB of 32-bit data, porting it to be a 64-bit application will only increase the executable size and loader time. This could be a potential performance disadvantage and a wasted development investment.

More details about the benefits of porting your applications from the 32-bit to the 64-bit model and the necessary migrating efforts are discussed in Chapter 2, "Why Do I Need 64-bit?" on page 13. Moreover, Chapter 5, "Programming in 64-bit" on page 61, Chapter 6, "Migration Techniques" on page 97, and Chapter 7, "64-bit Interlanguage Calls" on page 125, describe the different programming techniques, source code migration tips and 32-bit/64-bit program interoperability.

The following list details the two main characteristics of 64-bit application development:

1. Development of 64-bit applications does not require 64-bit hardware.

   All development tools that process program source code are able to target the 64-bit execution environment while running on 32-bit hardware systems.

2. Testing of 64-bit executables requires 64-bit hardware.

   There is no simulation environment of the 64-bit ABI on 32-bit hardware. The equivalence of function between 32-bit and 64-bit environments allows programs to be easily ported with equivalent functionality, with data size issues being the primary porting concern.

1.4 Conclusion

It should be noted that 64-bit is just one way to address the need for fast access to large amounts of data. IBM mainframes, still the most powerful general purpose database engines available, are 32-bit machines dealing with data in memory through expanded storage. Massively parallel machines such as the IBM RS/6000 SP can support extremely large memory by using multiple, independent memory in parallel.

On the other hand, for the past two years, IBM AS/400s with OS/400 V3.6 have been 64-bit machines handling 64-bit database addressing. For example, DB2 for OS/400 can handle 1 TB of indexes.

This book covers the 64-bit capabilities introduced in AIX V4.3 to familiarize users with the 64-bit dimension.

1.5 References

The following are good sources for additional information.

- 64-Bit Announcement from THE ACHIEVER
- The RS/6000 64-bit Solution

  `http://www.rs6000.ibm.com/resource/technology/64bit6.html`

- AIX V4.3 Migration Guide

  `http://w3dev.austin.ibm.com/library/aix4.3/`

Chapter 2. Why Do I Need 64-bit?

In this chapter we take a look at why the industry and IBM are moving to a 64-bit software and hardware base and the sort of applications that benefit most. We describe the key benefits of the AIX V4.3 64-bit architecture and give examples of applications.

We attempt to answer the question "Do I Need a 64-bit Application?" and to debunk some myths about 64-bit systems and architecture.

We then take a look at the specific implementation of 64-bit function in IBM's AIX V4.3 on RS/6000 systems and discuss issues of compatibility and performance.

This chapter assumes some familiarity with standard 32- and 64-bit concepts as well as the basic PowerPC architecture and the specific implementation of 64-bit support in AIX V4.3.

As a quick refresher, remember that a 32-bit machine can address 2^{32} bytes of memory, which is a little more than 4,000,000,000 bytes or 4,096 MB or 4 GB. In discussing large address spaces, frequently-used quantities are the megabyte (MB) which is 2^{20} bytes or approximately 1,000,000 bytes, the gigabyte (GB), 2^{30} bytes or approximately 1000,000,000 bytes, and the terabyte (TB) which is 2^{40} bytes.

Virtual address spaces can, of course, be much larger than physical address spaces, and it will be no surprise that the AIX V4.3 64-bit virtual address space on 64-bit platforms is huge compared to that available to 32-bit applications.

With today's high-density memory technologies, the cost of 1 GB, 4 GB, or even 16 GB of memory is high, but not out of proportion to other hardware costs. So it is reasonable for users to want to purchase and use more memory than existing 32-bit systems can access.

Relative Costs

Costs shown below are relative to RS/6000 Model S70:

- Base RS/6000 Model S70 4-way CPU + 1 GB RAM = 100%
- Each additional 1 GB RAM = +15% (max 16 GB)
- Each additional 4-way CPU card = +35% (max 12-way)

2.1 IBM's RS/6000 PowerPC 64-bit Architecture

The PowerPC was defined, from the start, to have a 64-bit architecture. The 32-bit chips (for example, the PPC601 and PPC604) all implemented a *subset* of the 64-bit architecture. Now, with the 64-bit chip (RS64A), the full 64-bit architecture is implemented. The 64-bit instruction set is a *superset* of the PowerPC 32-bit chip's instruction set, with additional instructions for controlling and handling the wider 64-bit data types and registers.

The number of CPU registers (the basic storage cell where the CPU stores the data on which it performs its computations) remains the same, but these registers are now 64 bits long instead of 32 bits. A few other control registers also move from 32 to 64 bits in length.

64-bit data can now be moved or operated upon in one operation, rather than having to be broken by the programmer or compiler into two 32-bit parts as would have been the case with 32-bit processors.

Note

Note that the floating-point registers do not change in size since even the 32-bit PowerPC chips had 64-bit floating-point registers in order to conform with industry standards for floating-point, which require both 32- and 64-bit data lengths.

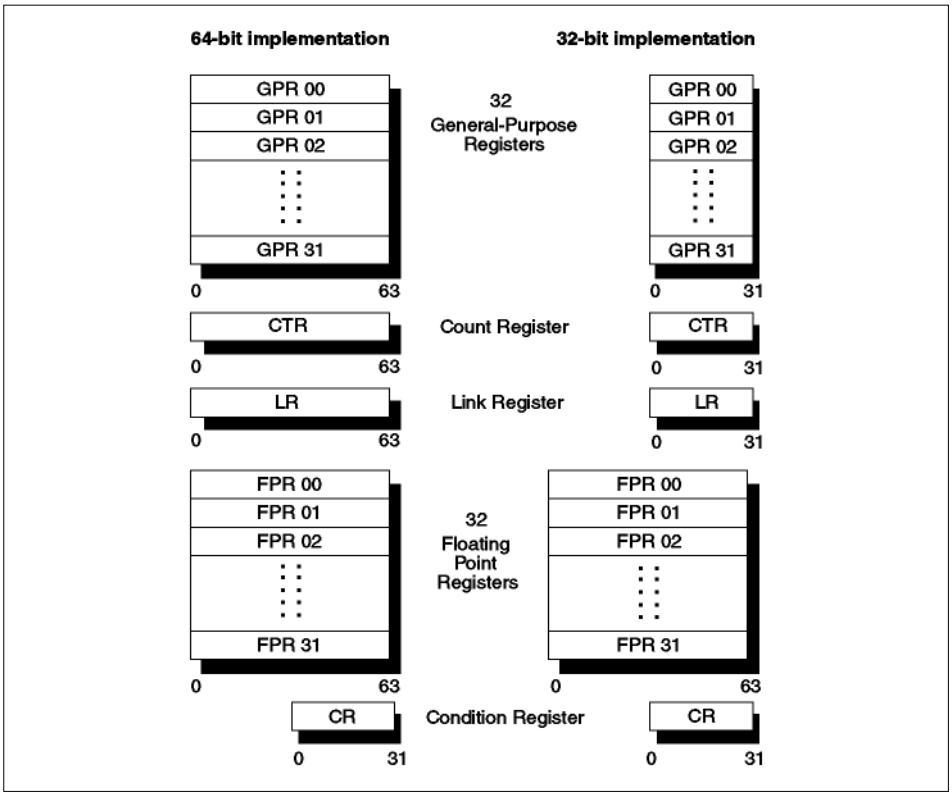


Figure 1. PowerPC's 64-bit Implementation vs. 32-bit Implementation

The type of operations which can benefit from the 64-bit chip's wider registers and busses include:

- Logical operations on 64-bit strings
- Shift operations on 64-bit registers
- Arithmetic operations and pointer calculations (on 64-bit integers)
- String (large data) moves or copies

When a 64-bit chip executes a 32-bit instruction, it uses half of the register width. This happens automatically, and the 32-bit application *does not know* and *does not need to know* that it is executing on a 64-bit processor. This is part of the "full binary compatibility" provided by AIX V4.3, but such compatibility is also inherent in the PowerPC architecture.

2.1.1 Full Binary Compatibility

With IBM's 64-bit PowerPC chip, AIX V4.3 will run existing 32-bit applications *unchanged* and without requiring a recompile.

This was the prime objective of AIX V4.3: full binary compatibility for 32-bit applications on 64-bit systems with *no modification or recompilation required.*

2.1.2 What 64-bit Architecture is Not

Rapid advances in chip technology have enabled very large and complex chips to be manufactured. This has made possible the move from 32-bit to 64-bit chips. 64-bit chips will typically be up to twice the area of 32-bit chips. This area increase is due to data buses ("highways") that are twice as wide as before, to the many registers that are twice as wide as before, and to the arithmetic and logical units which now need to be able to process data of twice the previous width.

Thus a 64-bit architecture enables the more efficient handling of 64-bit data types and the utilization of more physical memory. In itself, however, a 64-bit architecture does *not* improve performance.

The last sentence is important. Just in case you missed it, go back and reread it now. There is a widespread misconception that application performance automatically doubles when run on a 64-bit system.

Note

No 64-bit system, not even IBM's 64-bit PowerPC chip running AIX V4.3, will run most *existing* 32-bit applications any faster than an equivalent 32-bit system.

Only a very few *specialized* 32-bit applications, when significantly modified and then recompiled as 64-bit applications, will perform much better by taking full advantage of the 64-bit address space.

2.1.3 A Summary So Far

Obviously, this is not the full story, and in the following sections we will expand on the performance and compatibility implications of both 32-bit and 64-bit applications running on 64-bit systems.

Keep in mind, however, that a 64-bit system and architecture does *not* automatically increase performance. We will see, however, that there *are*

some types of applications and workloads which are very suited to 64-bit systems.

There are, at present, very few applications which have hit the 32-bit addressability barrier. But, those which have will benefit greatly from the 64-bit address space and from running as a 64-bit application.

2.2 The Move to 64-bit

Two main forces are driving the move from 32-bit to 64-bit architectures. In brief:

Engineering, scientific and business

Some applications in these areas have an increasing and real need for an address space greater than 32 bits in size, in other words, greater than 4 GB, and hence a need to become real 64-bit applications. Section 2.3, “Users in Engineering, Science and Business” on page 17 takes a closer look at these applications.

Existing 32-bit systems

These are becoming short of resources (for example, memory) which limits the overall performance and throughput of heavy users of 32-bit applications. We take a detailed look at this aspect in 2.4, “Users of Existing 32-bit Systems” on page 19.

2.3 Users in Engineering, Science and Business

This group of users will be the leaders in using new 64-bit applications because 64-bit systems will provide them with the tools to tackle the two most pressing problems in this area: the need to be able to handle very large files and the need to handle ever more complex and heavy-duty computational problems.

Large Datasets

The sizes of datasets that engineering, scientific and business users require to handle are frequently larger than will fit in the memory of a 32-bit machine.

Memory mapping of large files and databases is a common technique which aims to avoid constant disk I/O by keeping the whole file resident in memory, where it can be read and modified at the maximum speed of the processor.

If a file is too large for a 32-bit workspace, special programming steps need to be taken to access the data in smaller amounts which will reside in memory. This will invariably give less-than-optimum performance and may excessively

complicate the programs. Many users will, instead, rely on AIX's paging function to bring the data in from disk when required, but this is also at some considerable performance penalty.

Computation needs
There are two aspects to this need.

1. The first aspect is about program complexity. Some computations often require access to huge amounts of memory, for example, for simulations which make use of very large arrays. Again, the 32-bit physical memory addressing sets a limit which is too low for many heavy-duty computational users.

   As in the case of large files, complex programming work-arounds are possible to operate on large data with small memory. It would be much easier if there was sufficient memory in the first place to enable problems to be solved without having to handle programming limitations at the same time. As a beneficial side-effect, performance would almost certainly increase also.

2. The second aspect concerns application throughput. Increasingly, SMP systems (for example, the 12-way S70) are being used along with parallel programming algorithms to handle scientific and other problems. In the case of 32-bit systems, this means sharing no more than 4 GB of memory (and often restricted to 2 GB or less) with up to 12 very fast processors. This is almost certainly a situation in which the processors would not each have sufficient memory to operate at their maximum capability.

   A 64-bit system is able to provide the necessary memory and I/O resource per processor to enable SMP systems to scale well and to provide the bulk computation needs of the engineering, scientific and business communities. The RS/6000 Model S70 provides up to 16 GB of memory.

To get an idea of the types of applications that will benefit from 64-bit treatment, we now look at a selection.

2.3.1 Typical 64-bit Applications

There is no typical 64-bit application! The common goal, however, is to exploit the benefits of the 64-bit architecture. The typical engineering, scientific or business application which needs a 64-bit system will display some of the characteristics described below. There will be considerable overlap in the characteristics of 64-bit applications. These examples attempt to highlight the key areas in which 64-bit applications will have the greatest impact:

Decision Support Large databases, compute intensive. Definition: Use of data to identify ways in which a business

can change its ways to become more efficient and competitive. Two examples are:

Data Warehousing — Medium to large databases requiring significant amount of memory. SQL user data selection.

Data Mining — Compute-intensive with medium to large databases, requiring significant amounts of memory. Large flat files or data warehouse databases.

Internet-based Apps.	Large number of users; significant processing required.
e-business Apps.	Large databases, many applications, significant processing.
Huge Web Servers	High throughput required, large file caches, many simultaneous users, significant processing, much I/O.
Multimedia Servers	High throughput required, using large file caches in the large amounts of physical memory available.Very high I/O.
Numerically Intensive	Scientific and engineering compute and memory intensive, including computations using very large arrays, sparse arrays.
General Databases	For greatly increased performance, it will be possible to hold databases completely in memory. These applications will range from the large data warehouses to smaller databases requiring near-real-time response.
Large Array Operations	In-memory compute-intensive simulations, for example, nuclear physics, crash simulations, aerodynamic simulations.

2.4 Users of Existing 32-bit Systems

For existing 32-bit multi-user systems, performance is often restricted, not by the processor capacity, but by disk I/O capacity (bandwidth). The I/O is caused by paging due to insufficient memory resource available to the users. When paging takes place, performance decreases considerably because disk accesses are much slower than memory accesses.

The availability of additional memory to such a system could greatly reduce or stop paging, with consequent very significant performance gains, enabling much more effective and efficient use of the processing resource.

2.4.1 Heavy-Duty 32-bit Application Users

This defines the second category of users who will benefit from 64-bit systems.

For such users, when using 64-bit machines, the operating system will use the large amount of memory at its disposal to provide more resource to each individual 32-bit application. The applications will not use 64-bit functions directly, but AIX will make use of the huge address space on behalf of the applications.

Some of this type of 32-bit application are also good candidates for migration to 64-bit. However in many cases these 32-bit applications will gain most simply by allocating to them the memory resource that will enable them to function without excessive paging.

Some examples follow:

Internet-based Apps.	Large number of users; significant processing required.
e-business Apps.	Large databases, many applications, significant processing.
Huge Web Servers	High throughput required, large file caches, many simultaneous users, significant processing, much I/O.
Multimedia Servers	High throughput required, using large file caches in the large amounts of physical memory available.Very high I/O.
General Servers	Server consolidation for ease of administration and cost-performance.
Real-time Applications	Requiring responses within milliseconds. Some transaction-processing systems, voice-recognition, and voice-response systems are examples.

For users with large install bases and heavy-duty 32-bit applications on RS/6000 systems, the need for 64-bit hardware capability arises from the need to fully utilize today’s powerful processors. 32-bit RS/6000 systems have powerful processors but are limited by AIX to a maximum of 2 GB

memory. When shared among a large number of applications, this may not be sufficient to enable the processor(s) to operate near to full capacity, without becoming I/O bound due to paging.

Excessive Paging

If AIX allocates more virtual memory than it has physical memory, it effectively has to extend the memory size by grabbing space in the paging volume. This disk-based memory is paged-in to physical memory when an application needs to access it. Naturally, disk I/O is 1000's of times slower than direct accesses to memory, so when any application needs to access the contents of memory which has been paged-out to disk, its performance suffers.

With 2^{64} addresses available, 64-bit RS/6000 systems have a very significantly larger address space: this enables more physical memory to be installed and used, and many more virtual memory segments to be addressed before the onset of paging.

With a 64-bit physical address bus, in theory up to 2^{64} bytes of memory could be addressed. In practice, because of technology and cost limitations, it will be less than this. At present, the IBM RS/6000 Model S70 64-bit server enables 16 GB of memory, which is significantly more than in any existing 32-bit system.

Thus, such a system could host many more memory-hungry 32-bit applications, than could any 32-bit system. Hence, 64-bit systems can potentially run many more concurrent, large, 32-bit applications on the same RS/6000 system than could a 32-bit system. This gives the customer advantages in terms of higher system utilization, and reduced system administration, due to the need to maintain a smaller total number of RS/6000 systems.

In this way some users of 32-bit applications will find that a 64-bit machine is a very suitable platform for their applications, even though those applications will remain compiled as 32-bit applications.

This, indeed, was one of the objectives of 64-bit support in AIX: to provide more resource for large-scale 32-bit solutions.

2.5 AIX 64-bit Architecture Benefits

The following are the key benefits of the AIX V4.3 64-bit architecture on which this redbook is focussing for discussion of performance and tuning in these areas:

64-bit data types Specialized applications can benefit from the extra precision and performance of the 64-bit integer hardware. But probably the main use will be by most other 64-bit applications which will use 64-bit integer arithmetic to manipulate 64-bit address pointers. Additionally, logical (bit mask) operations and shift operations will be able to be performed on 64-bit data, benefiting some application types. Note that even with 32-bit PowerPC hardware, the floating-point hardware has always been to 64-bit precision, so floating-point hardware precision has not changed.

Access to large files Data warehousing, scientific and multimedia applications frequently need very large files and filesystems. The previous support in AIX 4.2 is extended to include 64-bit applications. Large files are now easily handled by the naturally large 64-bit data types.

Huge address space Specialized applications can exploit both the large physical memory (16 GB = 2^{34}) and huge virtual memory (2^{80}) available. Many scientific applications will be able to be programmed more simply and perform significantly better. The segments for data and stack are now huge, and memory mapping is significantly improved.

Larger executables (large text) Specialized applications may utilize the possibility of executables > 2 GB. in size.

To some extent, all the above can be summed up in two words: *convenience* and *performance.* In almost every case the rationale for going to a 64-bit application will be to gain performance by utilizing larger memory spaces, mapping larger files, and so on.

A desirable by-product of the 64-bit address space, however, will make the programming of such large applications less complex and more convenient. The programmer will far less frequently hit artificial limits. Far larger data objects can be addressed, a larger range of integers can easily be used,

larger strings and numbers can be used for logical and shift operations, and large files can be easily handled using the naturally larger 64-bit pointers.

No longer will it be necessary to artificially partition data in order to operate on it in amounts that can fit into the relatively small amount of memory available to 32-bit systems.

2.6 Performance of Migrated Programs

Earlier we claimed that most 32-bit applications will not benefit from being compiled and run as a 64-bit application, and that only a small number of specialized programs will benefit from the full 64-bit treatment.

So then, what *will* be the performance difference (better or worse) if we were to take our favorite 32-bit application and re-compile it as a 64-bit application?

There is no reason for a *large* performance difference between a 32-bit program and the same program re-compiled as a 64-bit program. There are, however, a number of reasons for *small* performance variations. The total effect of these variations will be very application-dependent.

Summarized below are a number of these factors which will cause performance to vary:

64-bit program performance may be *better* due to:

- Compiler optimizations to better sequence 64-bit instructions where possible.
- Code re-structure to make it more amenable to 64-bit optimization.

64-bit program performance may be *poorer* due to:

- Larger footprint, both text and data, causing longer load times.
- The use of 64-bit pointers that need to be translated when making calls into the 32-bit kernel.
- Program load times (affects `fork(), exec(), exit()`) are slower for 64-bit programs.

2.6.1 Variable Performance

Taking each of the above factors in turn, here is some additional detail:

- There is no evidence of any general gain yet in the area of compiler optimization. However if your 32-bit application makes use of long long (64-bit) data types, some improvement should be possible due to the

compiler being able to use one 64-bit instruction instead of two 32-bit instruction.

- Re-designed code, which, for example, concatenated 32-bit bit fields to enable handling 64-bit fields in one go, could allow the compiler to generate 64-bit instructions and thus handle the data in approximately half the number of operations.
- The larger footprint (size of executable and data) is explained by 32-bit data types which have become 64-bit (for example, pointers) and thus occupy more space, and by the larger amount of space required for padding in some structures to correctly align some datatypes.
- For 64-bit pointer operations, although the actual instructions, both 32-bit and 64-bit, proceed at the same speed, there is a small additional overhead in bus bandwidth when 64-bit datatypes are transferred to and from memory.
- A simple test of program load times showed the 64-bit program to take longer than the 32-bit program load. For the effect to become significant, an exceptionally high rate of loads would have to be performed.
- Another simple test program, using `getpid()` calls, which issued system calls at a very high rate showed a little bit poorer performance when run as a 64-bit application. The `getpid()` runs only a small number of instructions: many system calls would average 10 to 100 times as many instructions, and thus the overhead of the system call would be distributed more thinly, the lower the call rate became. A figure close to one percent on average would be expected. The difference is explained by the need for 64-bit processes making system calls, to go via a 64-bit library interface routine `lib64.a` which handle reformatting of the data and mapping of 64-bit addresses to the kernel's 32-bit address map. When a 64-bit AIX kernel is implemented in the future, the effect will likely be reversed.

Individually, the performance effect of most of these items are almost negligible. Taken together they could however add up to a small performance degradation. But it is very unlikely that all the factors are present at the same time.

Neglecting potential performance improvements via compiler or re-designing the application, it appears that for 32-bit applications that are re-compiled as 64-bit applications, performance should be comparable.

2.7 The Necessity for 64-bit Applications

Let's look at some indicators of the future for 64-bit operating systems and applications.

2.7.1 Inventory of AIX 64-bit Programs

At present, there does not appear to be a need for a large number of 64-bit applications or a great rush to migrate them. A good illustration of the need can be gained by looking at the following URL in the Internet:

```
http://www.rs6000.ibm.com/software/Apps/LPPmap.html
```

On this web site called "The AIX V4 IBM Licensed Program Products RoadMap", IBM lists all its AIX Licensed Program Products (LPP's) and indicates their compatibility with AIX 4.3 in terms of:

- Whether the LPP functions with AIX 4.3 or not.
- Whether the LPP exploits new function (64-bit addressing) of AIX 4.3.

At present (March 1998), this list shows that out of some hundreds of IBM LPP's, *only DB2* claims to exploit AIX V4.3 functions. Obviously the accuracy and currency of this list is only as good as the efforts of those who maintain it, and it is likely to be somewhat down-level, but nevertheless reasonably accurate.

There already exist other applications like Oracle Version 8 which is a 64-bit server, with support for both 32-bit and 64-bit applications.

It appears that the *real* need for the 64-bit address space is only for a very few applications, and that the conversion rate is slow at present.

We can expect to see many more LPP's become 64-bit in time, as more and more applications begin to hit the limits of the 32-bit address space.

2.7.2 A PC-Based Illustration

Think of the PC marketplace: when Intel introduced the first 32-bit processor (the 80386) in 1985, 32-bit applications did not exist, nor did a 32-bit operating system! Both IBM and Microsoft gradually introduced 32-bit function into their OS/2 and Windows operating systems. Even now, 13 years later, we find that some parts of these operating systems are still 16-bit.

2.8 64-bit Hardware

At the time of writing, our experience of 64-bit systems is based on the RS/6000 7017-S70 server which briefly provides, per system:

- 4 to 12-way SMP processing capability (64-bit 125 MHz RS64A cpu's)
- Up to 16 GB memory
- Up to 60 media/disk bays
- Up to 56 PCI adapter slots

There are some S70 hardware-related performance issues that are worth discussing briefly. Note that each of these affect the performance of both 32-bit programs and 64-bit programs.

2.8.1 S70 Memory Optimization

The S70 memory sub-system uses dual ports to communicate with the processors. Performance will be optimized if memory is balanced with the same amount of memory on each of the two ports.

Memory must be installed in groups of 4 identical cards. For example Feature Code 4173 comprises four 256 MB cards, totalling 1 GB.

One Feature Code 4173 would be functional, but would not give optimum performance. A second Feature Code 4173 should be installed on the other memory port to properly balance the system.

The S70 feature codes provide memory in increments of 0.5, 1, 2 and 4 GB.

Also Note

There is a limit of a total of 20 memory slots, therefore plans for memory upgrades should take future needs into account so that optimal performance is achieved, without limiting future memory expansion.

2.8.2 S70 I/O Address Space

The S70 implements I/O address space as 2 GB starting at the first 2 GB boundary. The remainder of physical memory starts at the 4 GB boundary. Any I/O access in a system with greater than 2 GB of memory (for 32-bit adapters) requires a layer of mapping to make the devices think that the addresses are still in the 4 GB address space, that is, addressable by 32 bits.

The I/O mapping is only required when more than 2 GB of memory is installed. Thus applications (32-bit or 64-bit) will suffer a slight performance loss if S70 system memory is upgraded from 2 GB or less, to greater than 2 GB.

2.8.3 S70 Floating-Point Performance

The RS64A chip's floating-point performance may not match your expectation of a 64-bit PowerPC chip. The design point for the RS64A was for commercial server performance and capability.

The RS/6000 Model S70 is the only platform currently using the RS64A chip, and it is marketed as a commercial server system.

Future versions of the RS64A chip are intended to have enhanced floating-point performance, which will better support scientific and engineering applications, without hurting commercial and server performance.

Consequently, if your application is numeric-intensive (Engineering/Scientific), you may wish to consider either SP technology or stand-alone servers, in either case using SMP technology or the P2SC (Power2 Single Chip).

2.9 Summary and Recommendation

Owners of existing 32-bit applications should think carefully before re-compiling them as 64-bit applications.

- Simply recompiling will not change the function of the program to use any more address space.
- Re-compiling as a 64-bit application can in some cases cause a small performance reduction (Section 2.6, “Performance of Migrated Programs” on page 23).

32-bit applications which would benefit from breaking the 32-bit address space limitation should be considered separately:

- There may be significant programming effort required to convert these to efficient 64-bit programs.
- Once converted, however, maintenance should be easier due to the less restricted, simpler, 64-bit programming environment.

We Recommend
Consider carefully whether you need a 64-bit application. Obviously some people have already decided that they do not need 64-bit yet.

Bearing in mind the performance effects of migration to 64-bit (see 2.6, “Performance of Migrated Programs” on page 23), there is no clear performance advantage for most existing 32-bit programs.

The correct choice should generally be to use a 32-bit application unless 64-bit addressability is required by the application or can be used to dramatically improve its performance.

By writing a 32-bit application carefully and obeying the migration guidelines (see Chapter 6, “Migration Techniques” on page 97), you can have a well-performing application, and at the same time be positioned to easily migrate to 64-bit, if or when you need to in the future.

2.10 References

AIX Version 4.3 Differences Guide, SC24-2014.

Optimization and Tuning Guide for Fortran, C, and C++, SC09-1705.

The PowerPC Architecture: A Specification for a New Family of RISC processors, 2nd Ed, Morgan Kaufmann Publishers, Inc.

2.10.1 Web Pages

There are a large number of Internet Web pages which contain relevant information. These are grouped into two categories for convenience:

1. Publicly Accessible Web Pages
2. IBM Business Partner Pages

Only pages labelled "[www]" are freely accessible. Other pages will be accessible by IBM internal users only.

Key
"[www]" means www page with public access.

"[sdp]" means page requiring SDP (IBM Solution Developer Program) password access, which is generally free. To register, visit:
`http://w3dev.austin.ibm.com/welcome/w_home.html`

"[ibm]" means w3 IBM internal page with general IBM access

Numbered references are the top-level pages: the following un-numbered references will be found within those top-level pages.

2.10.1.1 Publicly Accessible Web Pages

RS/6000 Systems Hardware

- [www] RS/6000 Enterprise Server Model S70

  `http://www.rs6000.ibm.com/hardware/enterprise/s70.html`

- [www] PowerPC Hardware Architecture Reference

  `http://www.rs6000.ibm.com/resource/technology/chrp/index.html`

RS/6000 Tech ReSource

1. [www] RS/6000 Tech ReSource [Papers] Top Level

   `http://www.rs6000.ibm.com/resource/technology`

- [www] The RS/6000 64-bit Solution [White Paper]

  `http://www.rs6000.ibm.com/resource/technology/64bit6.html`

  `http://www.rs6000.ibm.com/resource/technology/rswp64bt.pdf`

- [www] New Dimensions in Scalability & Performance

  `http://www.rs6000.ibm.com/resource/technology/aix43ppr.html`

  `http://www.rs6000.ibm.com/resource/technology/rsaix43.pdf`

- [www] PowerPC Hardware Architecture Reference (Full text)

  `http://www.rs6000.ibm.com/resource/technology/chrp/index.html`

- [www] Beyond the 32-bit Barrier (In "RS/6000 Results" Magazine)

  `http://www.rs6000.ibm.com/resource/results/archive/november/cver_stry.htm`

IBM AIX V4 LPPs

- [www] The AIX V4 IBM Licensed Program Products Roadmap

  `http://www.rs6000.ibm.com/software/Apps/LPPmap.html`

InfoExplorer in HTML

- [www] AIX System and Related Product Documentation [InfoExplorer]

  `http://www2.austin.ibm.com/doc_link/en_US/a_doc_lib/aixgen/`

- [www] 64-bit Updates to Device Driver Information

  `http://www2.austin.ibm.com/doc_link/en_US/a_doc_lib/aixgen/devdr/overview.htm`

ITSO and Redbooks

- [www] Redbooks Home Page (full text of some older books)

  `http://www.redbooks.ibm.com/homepage.html`

- [www] RS/6000 Redbooks Collection (full text, older books)

  `http://www.rs6000.ibm.com/resource/aix_resource/Pubs/redbooks`

2.10.1.2 IBM Business Partner Pages

- [ibm] ISV's RS/6000 Information [RS/6000 Level]

  `http://w3isv.austin.ibm.com/info/rs6kinfo.html`

- [ibm] AIX 4.3 Migration Guide (25Jun97, old)

http://w3isv.austin.ibm.com/info/mguide/index.html

- [sdp] AIX 4.3 Migration Guide (currently being updated)

  http://www.developer.ibm.com/library/aix4.3/aix43_revision.html

- [ibm] AIX 4.3 migration guide (0.4.4 1/98, HTML and PostScript)

  http://w3dev.austin.ibm.com/library/aix4.3/ (HTML)

  http://w3dev.austin.ibm.com/library/aix4.3/aix43dl.html (PostScript)

- [sdp] AIXpert 9/97 Focus on 64-bit Architecture

  http://www.developer.ibm.com/library/aixpert/cd/97SEPTEM/S97TOC.PDF

  http://w3dev.austin.ibm.com/library/aixpert/cd/97SEPTEM/S97TOC.PDF

- [sdp] AIX Version 4.3 Release Notes

  http://www.developer.ibm.com/sdp/library/aix4.3notes/index.html

  http://w3dev.austin.ibm.com/sdp/library/aix4.3notes/index.html

Chapter 3. Performance Tools Changes

A number of things are addressed in this chapter. First is the change in the packaging of the performance tools. The second thing addressed is the new enhancements that have been made to the tools. In some cases, the changes take the form of new parameters or flags. In others, it is in the format of the output reports. Occasionally, when there is something else of interest worth mentioning, it is documented.

As you will see, the performance tools on AIX have not changed very much since AIX V4.2. Since these tools have been quite extensively covered in the redbook titled *RS/6000 Performance Tools in Focus*, SG24-4989, they are not readdressed here. In any case, most of the modifications to the tools were made to enable them to operate on both 64-bit and 32-bit environments. These changes do not modify the functional operation of the commands or tools.

3.1 Packaging Changes

In earlier releases of AIX V4, some of the AIX-specific performance tools, such as `bf` and `fdpr`, were packaged and sold as a separately licensed program product outside the Base Operating System (BOS). AIX users who wanted to collect and analyze performance on their systems were constrained to the standard UNIX tools. These tools are great in helping users determine at the first level what the resources holding back system performance were. To go anywhere beyond that, such as to analyze the performance of a specific program and tune it, they either had to invest some money on additional tools or work their way through by means of trial and error.

Some of these tools are now bundled together with AIX V4.3 as the fileset `perfagent.tools 2.2.30.0`. The fileset, which comes with no additional charge, has to be explicitly installed if you wish to use the tools contained within. The tools provided are:

- bf (Bigfoot) and bfrpt
- fdpr (Feedback Directed Program Restructuring)
- filemon
- fileplace
- genkex
- genkld
- genld
- lockstat

- netpmon
- rmss
- stem
- stripnm
- svmon
- syscalls
- tprof

A brief description of some of the tools and their functions can be found in the file `/usr/lpp/perfagent/README.perfagent.tools`.

As these analysis tools are closely tied to the kernel due to the use of VMM and other data structures, they should be of great help when optimizing performance on AIX. These tools also greatly assist performance groups and support centers in debugging customer performance problems.

3.2 Summary of Changes to the Tools

The following tables give indications of changes that were made to the performance tools in the process of migrating them from the 32-bit AIX to the 64-bit AIX. Generally, the tools were enabled for 64-bit support. Other changes worthy of note, if any, are discussed in the later pages of this chapter.

The first column of the table specifies the name of the tool. The second gives a very brief comment, and the third column indicates if the tool is discussed in this chapter by pointing to the relevant section.

Table 3 shows a summary of changes for AIX-specific tools.

Table 3. Summary of Changes for the Performance Toolbox for AIX V4.3 Tools

Tool	Comments	Section
bf and bfrpt	No 64-bit support	3.3.1 on page 34
filemon	Enabled for 64-bit	3.3.2 on page 35
fileplace	Enabled for 64-bit	3.3.3 on page 36
fdpr	Enabled for 64-bit	Not discussed
genld	Enabled for 64-bit	3.3.4 on page 36
genkld	Enabled for 64-bit	3.3.5 on page 37
genkex	Enabled for 64-bit	Not discussed

Tool	Comments	Section
lockstat	No 64-bit support	Not discussed
netpmon	Enabled for 64-bit	Not discussed
rmss	Enabled for 64-bit	3.3.6 on page 37
stem	No 64-bit support	Not discussed
stripnm	Enabled for 64-bit	3.3.7 on page 38
svmon	Enabled for 64-bit	3.3.8 on page 38
syscalls	No 64-bit support	3.3.9 on page 38
tprof	Enabled for 64-bit	3.3.10 on page 38
perfmgr	No change	Not discussed
perfagent.server	No change	3.3.11 on page 39

Table 4 shows a summary of changes for standard UNIX tools.

Table 4. Summary of Changes for the Standard (UNIX) Tools

Tool	Comments	Section
vmstat	No change	Not discussed
iostat	No change	Not discussed
sar	No change	Not discussed
ps	No change	Not discussed
pstat	No change	Not discussed
netstat	Expanded to support IPv6	Not discussed
nfsstat	No change	Not discussed
no	New parameters added	3.4.1 on page 39
nfso	New parameter added	3.4.2 on page 41
nice	No change	Not discussed
renice	No change	Not discussed
prof	Enabled for 64-bit	3.4.3 on page 41
gprof	No change	Not discussed

Table 5 shows a summary of changes for a variety of tools that deal with performance issues.

Table 5. Summary of Changes for the Other Tools

Tool	Comments	Section
lslv	No change	Not discussed
trace and trcpt	Enable for 64-bit	3.5.1 on page 42
Performance Diagnostic Tool	No change	Not discussed
perfpmr	No change	3.5.2 on page 44
cpu_state	No change	3.5.3 on page 44
bindprocessor	No change	Not discussed
schedtune	No change	Not discussed
vmtune	New parameter added	3.5.4 on page 44
xgprof	No change	Not discussed
Program Visualizer	No change	3.5.5 on page 45
utld	No change	Not discussed
Xprofiler	Not 64-bit enabled	3.5.6 on page 46

3.3 Performance Toolbox for AIX V4.3

Performance Toolbox for AIX V4.3 is the name given to a suite of AIX-specific performance tools. It is made up of `perfagent.tools`, `perfagent.server`, and `perfmgr`. The following pages document the changes that have taken place with some of the tools from this suite.

3.3.1 The bf and bfrpt Commands

The `bf` (bigfoot) command monitors memory usage of applications. It can be run using executable programs without recompilation and can provide memory footprints for processes, system components, and subroutines. It is possible for `bf` to identify page sharing between processes and subroutines within the processes.

The `bfrpt` command is the post-processing utility which can produce graphical and tabular reports.

The bf and bfrpt commands are currently undergoing modification work to adapt them to fit in the 64-bit environment. No change in functionality is expected.

However, due to the changes in the 64-bit architecture, changes such as the increase in the number of segments, segment ID, and number of frames, the format of the output may change.

3.3.2 The filemon Command

The filemon command collects and presents trace data on the various layers of file system utilization, including the logical file system virtual memory segments, logical volume manager (LVM), and physical disk layers. Data can be collected on all the layers or on specific layers by specifying the -O option. The default is to collect data on the VM segments, LVM, and physical layers. Both summary and detailed reports are generated. This command now supports 64-bit addresses.

At the time of this writing, there was a defect outstanding on the Detailed File Stats section of the output for reads and writes. You are advised to be on the lookout for the latest PTFs for perfagent.tools if you need the read and write time information.

The report segment shows INF and NaNQ instead of figures.

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

FILE: /tmp/big/xaa  volume: /dev/biglv (/tmp/big)  inode: 17
opens:1
total bytes xfrd:104861696
reads:25601(0 errs)
  read sizes (bytes):avg  4096.0 min    4096 max    4096 sdev     0.0
  read times (msec):avg    NaNQ min     INF max   0.000 sdev    NaNQ

FILE: .  volume: /dev/hd1 (/home)  inode: 56
opens:13206
total bytes xfrd:104857600
writes:25600(0 errs)
  write sizes (bytes):avg  4096.0 min    4096 max    4096 sdev     0.0
  write times (msec):avg    NaNQ min     INF max   0.000 sdev    NaNQ
lseeks:13205
```

3.3.3 The fileplace Command

The `fileplace` command displays the placement of a file's blocks within a logical volume or within one or more physical volumes. This command expects an argument containing the name of the file to examine.

This command has been modified in AIX V4.2 to support reporting for files larger than 2 GB. However, In AIX V4.3, running the command `#/usr/bin/fileplace fname` against a file that is 2 GB (2,147,483,648 bytes) or larger may return the error fname: `Value too large to be stored in data type`.

Otherwise, there is no change in the observed output.

3.3.4 The genld Command

The `genld` command extracts a list of loaded objects for each process currently running on the system. It has not changed much from the version on the previous release of AIX, except that it now supports 64-bit addresses. It displays information for 32-bit and 64-bit loaded objects for each process running on the system. Thus, a 32-bit process such as `init` would generate the following output:

```
Proc_pid:        1      Proc_name: init
                                        10000000  init
```

while a 64-bit one such as `shlap` would cause the following to be reported:

```
Proc_pid:     3188      Proc_name: shlap
                                      100000000  shlap
```

The `shlap` command is a part of the `bos.64bit` fileset. On systems with hardware that supports 64-bit, such as the S70, the installation of this fileset will enable the AIX operating system to execute 64-bit applications.

Notice that the address value has one more digit for the 64-bit process (`shlap`) than for the 32-bit one (`init`).

Note

At the time of this writing, `genld` fails to output addresses, paths, and object names for 32- or 64-bit libraries for 32/64-bit processes. An APAR (IX74290) is being worked on to address this defect.

3.3.5 The genkld Command

The `genkld` command extracts the list of shared libraries and shared objects currently loaded onto the system and displays the address, size, and path name for each object on the list. It has been modified to include 64-bit support.

An excerpt of the report created when the `genkld` command is run is shown below to illustrate this.

```
Virtual Address Size File

d04e0980 16a0d /usr/lib/libpthreads.a/shr_xpg5.o
d04e0980 16a0d /usr/lib/libpthreads.a/shr_xpg5.o
d01df000 acd /sbin/comp.uext/
:
:
:
d0000520 192355 /usr/lib/libc.a/shr.o
9000000002bc980 29aa7 /usr/lib/libpthreads.a/shr_xpg5_64.o
9000000002bc980 29aa7 /usr/lib/libpthreads.a/shr_xpg5_64.o
9000000002ba2c0 1402 /usr/lib/libcrypt.a/shr_64.o
900000000000520 2b9851 /usr/lib/libc.a/shr_64.o
9000000002ba2c0 1402 /usr/lib/libcrypt.a/shr_64.o
900000000000520 2b9851 /usr/lib/libc.a/shr_64.o
```

The last six lines in this case report 64-bit shared library modules. The repeated entries indicate they were loaded more than once.

3.3.6 The rmss Command

The `rmss` command simulates a system with various sizes of real memory, without having to extract and replace memory boards. By running an application at several memory sizes and collecting performance statistics, one can determine the memory needed to run an application with acceptable performance.

The `-f` option, which defines the final memory size that `rmss` can simulate, no longer takes a value of 4 MB. The range of values acceptable are 8 MB to the total physical memory available on the system. The same is true with the `-c` option. Specifying a value smaller than 4 MB to the command would result in the following message:

```
# Simulated memory size must be between 8 and 4096 Mb.
```

The performance of the `rmss` command will decrease as real memory increases. This is because `rmss` goes through the free list from the page

frame to the table and marks those required to decrease the amount of real memory available, as used. As real memory increases, the time taken to mark these frame by frame will increase as well.

3.3.7 The stripnm Command

The `stripnm` command displays the symbol information of a specified object file. It is concerned with the archive file format as well as the XCOFF format.

The `stripnm` command looks at the XCOFF format of the file on which it is to process and determines from there if it is 32-bit or 64-bit.

3.3.8 The svmon Command

The `svmon` command is an AIX-specific tool that provides global, process-level, and segment-level reporting of memory use.

Currently, it breaks when run against 64-bit processes. It does not display the "lib data" and "shared library text" working segment information.

Also, comparing the output of "`vmstat 2 10`" against "`svmon -G -i 2 10`", it is found that the `fre` value of `vmstat` no longer correlates with the `free` value from `svmon`.

3.3.9 The syscalls Command

The `syscalls` command is used to trace system calls for all or specified processes on a system. The tool is currently undergoing some changes, and you would encounter the following error message when you attempt to run it:

```
You must first configure the stem kex.
```

This tool will not support 64-bit applications.

3.3.10 The tprof Command

The `tprof` command is an AIX-specific profile that is based on the trace facility. By using this profiler, you can identify the heaviest processes on the machine, the heaviest subroutines in your programs, and the time spent in each subroutine. You should use the `-g` flag when compiling your source if you are using an AIX C compiler to get the most out of this profiler.

```
# xlc -g foo.c -o foo
```

The implementation of `tprof` on AIX V4.3 has been modified to allow operation on both 64-bit and 32-bit executables. It does not differ functionally from that the release found in AIX 4.2.1 or earlier.

The tprof command looks at the XCOFF format of the file on which it is to process and determines from there if it is 32-bit or 64-bit.

Note

Sometimes, especially on SMP machines, because of the way things have been designed, the kernel masks its interrupts when running certain processes. As a result, tprof would end up reporting less CPU ticks than had actually taken place.

3.3.11 The perfagent.server Tool

No additional features have been added to perfagent for AIX V4.3. Its libraries, such as libSpmi and libarm, will remain as 32-bit libraries. This means two things. First, 64-bit applications cannot compile or link with libSpmi because mixed-mode compilation is not supported.

Second, response time monitoring using libarm for a 64-bit application is not supported.

When installing the perfagent.server fileset on your system, you should also install APAR IX68740 to fix certain known problems.

3.4 Standard Performance Tools

Standard tools covers the tools that follow have been a part of the standard AIX package for many years. Again, the following pages document changes that have been made to them in the process of porting them over to AIX V4.3.

3.4.1 The no Command

The no command is used to configure network attributes. It sets or displays current network attributes in the currently running kernel and therefore must be run again after each reboot.

A number of new parameters have been added to this command in AIX V4.3, and they are briefly documented below. Most of them are related to the newly introduced IPv6. Parameters beginning with *ndp* refer to neighbor discovery. You can obtain further details on neighbor discovery from the *RFC1970: Neighbor Discovery for IP Version 6 (IPv6)*.

ndpqsize Number of packets to hold waiting on completion of a NDP entry. Its default value is 50 packets.

ndpt_keep	The time, in half seconds, to keep an NDP entry. The default value is 120 or 60 seconds.
ndpt_reachable	The time, in half seconds, to test if an NDP entry is still valid. The default is 30 or 15 seconds.
ndpt_retrans	The time, in half seconds, to wait before retransmitting an NDP request. The default value is 1 or half a second.
ndpt_probe	The time, in half seconds, to delay before sending their first NDP probe. Its default is 5 units or 2.5 seconds.
ndpt_down	The time, in half seconds, to hold down a NDP entry before deleting it. Its default value is 3 units or 1.5 seconds.
ndp_umaxtries	The maximum number of Unicast NDP packets to send. Its default value is 3.
ndp_mmaxtries	The maximum number of Multicast NDP packets to send. Its default value is 3.
ip6srcrouteforward	Specifies whether the system forwards source-routed IPv6 packets. The default value of 1 allows the forwarding of source-routed packets. A value of 0 causes all source-routed packets that are not at their destinations to be discarded.
ip6_defttl	This is the default hop count to use for IPv6 packets if no other hop count is specified. Since hop counts are normally specified in most situations, this value is not used much. The default value for this parameter is 64 hops.
ip6_prune	This tells how often to check the IPv6 routing table for expired routes. Its default is 2 seconds.
inet_stack_size	This parameter lets you configure the inet interrupt stack table size. The only time you would normally need to use this is if you are running with unoptimized debug kernel and/or netinet. To change the value, it must be set in the rc.net file because changing it on the fly has no effect.

You will probably not get any significant performance gains from changing these parameters for anything except, perhaps, in very specialized

environments. The command for changing any of the no parameters takes the form:

```
# no -o <parameter>=<value>
```

3.4.2 The nfso Command

The nfso command can be used to configure NFS attributes. It sets or displays network options in the currently running kernel. A new parameter, nfs_rfc1323, has been added to this command in AIX V4.3. This parameter allows users to enable rfc1323 functionality for NFS without having to modify any network options (no) parameters.

It is turned off by default. To turn it on, you will need to enter:

```
# nfso -o nfs_rfc1323=1
```

This option can be used to enable very large TCP window size negotiation to occur between systems. If using the TCP transport between NFS client and server, and both systems support it, this allows the systems to negotiate a TCP window size in a way that will allow more data to be "in-flight" between the client and server.

3.4.3 The prof Command

The prof command interprets profile data collected by the monitor subroutine for the executable file (a.out by default). It reads the symbol table in the object file program and correlates it with the profile file (mon.out by default). The prof command displays, for each external text symbol, the percentage of execution time spent between the address of that symbol and the address of the next, the number of times that function was called, and the average number of milliseconds per call.

A few minor reporting problems were observed with prof at the time of this writing. First, the memory addresses are not represented properly. Programs compiled in 32-bit mode get their addresses of functions returned as zeroes.

```
# prof -x
Address Name                %Time     Seconds     Cumsecs  #Calls
msec/call
       0 .main                 0.0        0.00        0.00       1      0.
       0 .free                 0.0        0.00        0.00       2      0.
       0 .free_y               0.0        0.00        0.00       2      0.
       0 ._findbuf             0.0        0.00        0.00       1      0.
:
:
```

If compiled in 64-bit, the address is returned as 1100000000 if the `-o` (octal) flag is used with prof and as 900000 if the `-x` (hex) flag is used instead.

```
# prof -o
 Address Name                %Time     Seconds     Cumsecs  #Calls
msec/call
       1 .main                 0.0        0.00        0.00       1      0.
1100000000 .printf               0.0        0.00        0.00       1      0.
1100000000 ._doprnt              0.0        0.00        0.00       1      0.
1100000000 .free                 0.0        0.00        0.00       2      0.
1100000000 .free_y               0.0        0.00        0.00       2      0.
:
:
```

A second problem observed is that executing `prof` with the `-v` flag does not send the graphic version of the profile to the standard output, but instead causes a core dump.

```
# prof -v
Illegal instruction(coredump)
```

3.5 Other Tools

Some of the other performance or performance-related tools are discussed here.

3.5.1 The trace Facility

The AIX trace facility captures a sequential flow of time-stamped system events that can be monitored. To capture a meaningful trace, you would normally specify trace hooks to it.

There are two types of trace hooks, namely the generic, and the non-generic. Trace data is formatted with trace templates, which are found on the system in /etc/trcfmt.

A number of functional changes have been made to the trace subsystem to support 64-bit tracing. The changes are as follows:

- The `trcgen` and `trcgenk` system calls take a 64-bit buffer address.
- Up to two 64-bit data words can be traced with a non-generic hook.
- Two new formatting codes are provided to format signed and unsigned 64-bit decimal numbers. Thus, 64-bit data may be traced using the generic trace interfaces, which have not changed.

- To preserve binary compatibility, no changes have been made to generic trace hooks. However, this means that the data word associated with a generic hook is 32 bits only.
- Two new formatting codes, D8 and U8, are provided to format signed and unsigned 64-bit decimal numbers. These formatting codes operate on data in the variable length generic trace buffer as well as on the non-generic entry's data words. Thus, 64-bit data may be traced using the generic trace interfaces.

In addition to this, programming interfaces have also been defined for 64-bit trace.

The following prototype statements have been added to `/usr/include/sys/trcmacros.h`. These are under the `#else` of a "`#ifdef _NO_PROTO`" statement, with the unprototyped calls kept under `_NO_PROTO`:

```
extern void utrchook(unsigned int hkwd, unsigned int d1, unsigned int d2,
unsigned int d3, unsigned int d4, unsigned int d5);

extern void trcgen(int chan, unsigned int hkwd, unsigned int dword,
unsigned int len, char * buf);

extern void trcgent(int chan, unsigned int hkwd, unsigned int dword,
unsigned int len, char * buf);

extern int trcon(int chan);
extern int trcoff(int chan);
extern int trcstart(char * str);
extern int trcstop(int chan);
```

This applies to applications only, not to kernel and kernel extension code. This may cause compiler warnings when compiling some applications on AIX V4.3.

The new `TRCHK64L1`, `TRCHK64L1T`, `TRCHK64L2`, and `TRCHK64L2T` macros can be used to trace one or two 64-bit values to channel 0, non-generic.

The new `$L1` and `$L2` macros are used to access the first or second 64-bit value traced in non-generic entries.

The `trace` daemon will not change. Users will have to output the 64-bit data through the variable-length trace buffer provided by the `trcgen` and `trcgent` system calls, or use the `TRCHK64`.

3.5.2 The PerfPMR Package

The PerfPMR package is a collection of scripts which gathers performance and configuration information in order to help IBM analyze a possible AIX performance defect. It is not intended to be used as a tool for general performance monitoring or analysis.

However, many customers are curious as to how `perfpmr` gathers its data. The following is a listing of the exact commands and flags the PerfPMR scripts use to monitor and record system performance:

```
ps -elk
ps gv
vmstat -s
netstat -v
netstat -m
netstat -rs
netstat -s
sar -A
vmstat
iostat
netstat
nfsstat -csnr
trace
tprof -k -s
filemon -O all -v
netpmon -O all
```

3.5.3 The cpu_state Command

The `cpu_state` command controls and lists which processors on a multiprocessor system will be active when the system is next started.

The `cpu_state` command is not supported on the S70. When the command `cpu_state -l` is run, the following message results:

```
1731-007: Command not supported on this platform
```

3.5.4 The vmtune Command

The `vmtune` command can be used to modify the VMM parameters that control the behavior of the memory-management subsystem.

A new parameter has been added to this command, namely `lrubucket`. This parameter was added to `vmtune` as part of the page-replacement changes made for large real memory machines. The default value is 512 MB, and

there is really no reason to change that, though IBM wanted to provide the mechanism via `vmtune` just in case.

The VMM's page replacement strategy is a pseudo least-recently-used algorithm. It simply looks at each frame in the system to ascertain if it had been referenced recently. If it has, the VMM clears the reference bit and keeps looking. The next time around, if the frame has not been referenced, the reference bit will still be clear, and the VMM will page out that page (or just take it if it's not dirty).

With large real-memory machines, it takes a long time to make this second pass and successfully free up a frame. Therefore, when the VMM runs out of frames, it takes a long time to satisfy the request for a new one.

The solution is to divide large-memory systems into smaller buckets. Then the page-replacement routine makes two passes per bucket before continuing on to the next bucket. This means less latency when a frame is needed.

The `lrubucket` parameter is the size (in 4K pages) of this bucket. So with the default value of 512 MB, the latency to get a free frame should always be similar to a system with 512 MB of total memory, no matter how much real memory there is.

3.5.5 The Program Visualizer

The Program Visualizer (PV) for AIX, is a tool that provides graphical, animated views of trace data. It can display the behavior of a target program and the system underlying it concurrently, allowing you to observe the application and system behavior as it unfolds over time. The layers displayed include the program itself, user-level libraries, the operating system, and the hardware.

The version of PV available still stands at 0.8.3. Besides some bug fixes, there have been no major changes made to it over the last year. PV, as it is distributed now, will not work with traces from a shared-memory multiprocessor because there is no facility to show behavior by CPU. As such, its usefulness is considerable diminished if you want to run it on a multiprocessor machine.

If you would like to follow the developments of this tool, you can do so by accessing its World Wide Web site at `http://www.research.ibm.com/pv`. If you are an IBM internal user, you can also access the internal PV FORUM.

3.5.6 The Xprofiler Tool

Xprofiler is a GUI-based tool that helps you analyze your application's performance. You can profile both serial and parallel applications with this product, though it would probably be more useful profiling parallel applications. When you run a serial application, a single profile data file is generated, while a parallel application produces multiple profile data files.

Like `gprof`, Xprofiler lets you analyze CPU (busy) usage only. It does not give you other information such as CPU idle, I/O, or communication.

To use Xprofiler, you application must be compiled with the `-pg` option. For example, if you want to profile a program called foo.c, you would:

```
# xlc -pg -o foo foo.c
```

If you need to, you could compile and link it in two separate steps, as follows:

```
# xlc -pg -c foo.c
# xlc -pg -o foo foo.o
```

Notice that when you compile and link separately, you must use the `-pg` option with both the compile and the link commands.

The Xprofiler GUI provides the capability of viewing included functions. Note, however, that your application must also be compiled with the `-g` option in order for Xprofiler to display the included functions. The `-g` option is also required for source statement profiling.

Important note

The `-pg` option is not a combination of the `-p` and the `-g` compiling options. The man pages for `xlc` list the meanings of the three flags, `-g`, `-p` and `-pg`, as follows:

`-g`	Produces information for the debugger.
`-p`	Generates profiling support code.
`-pg`	Generates profiling support code including BSD profiling support.

When you run a program compiled with the `-pg` option, a file called "gmon.out" is created in the local directory, if the application is a serial one. For a parallel application, a separate gmon.out file is written for each task running in the application. The output files are suffixed with the task ID to prevent them from overwriting each other—for instance, `gmon.out.0`, `gmon.out.1`, and so on.

In order to get a complete picture of your parallel application's performance, you must indicate all of its gmon.out files when you load the application into

Xprofiler. When you specify more than one `gmon.out` file, Xprofiler shows you the sum of the profile information contained in each file.

It is possible to generate the `gmon.out` files in one system and then to transfer it to another to run Xprofiler on it. However, you must make certain that all the libraries used in the source and destination machines are at identical software levels. Otherwise, the resultant profiles would be rendered meaningless.

The Xprofiler is currently packaged as part of the Parallel Environment for AIX Version 2.3 and not available as a separate, stand-alone product.

Note

If an application is compiled in real 64-bit mode, the `gmon.out` file created when this application is run will be in the new 64-bit output file format. Xprofiler is not yet able to read this format.

3.6 References

The following are good sources for additional information.

- *RS/6000 Performance Tools in Focus*, SG24-4989-00
- *PE Operation and Use Vol. 2 - Tools Reference,* SC28-1980
- RFC1970: Neighbor Discovery for IP Version 6 (IPv6)

Chapter 4. Performance Tuning Considerations

Though the architecture has changed, with AIX V4.3 being 64-bit, the methodology for performance tuning, nevertheless, remains very much unchanged. It is still a matter of resource management and proper system parameter setting.

Chapter 2, “Why Do I Need 64-bit?” on page 13, went into some detail about the categories of applications that could benefit from a 64-bit environment. The common tie that binds all these applications is that they are driven by their need for large addresses. In this chapter, we look at the performance tuning considerations for those types of applications.

For good methodologies on performance tuning, the reader is advised to refer to the *AIX Versions 3.2 and 4 Performance Tuning Guide*, SC23-2365.

4.1 CPU Tuning Considerations

The 64-bit AIX V4.3 does not provide much in enhancement by way of CPU performance tuning. However, there are some issues worth mentioning considering the nature of the loads that a machine running it may be subjected to.

4.1.1 The schedtune Command

Below, we look at some of the schedtune parameters that could be of particular interest in AIX V4.3 environments. The schedtune command can be used to set the parameters for CPU scheduling and VMM processing. The command can only be executed by the root user, and changes made by this tool last only until the next reboot of the system. Executing the schedtune command with no options shows the current parameter values:

```
# /usr/samples/kernel/schedtune
      THRASH           SUSP        FORK             SCHED
-h    -p    -m       -w    -e      -f       -d       -r        -t
SYS   PROC  MULTI    WAIT  GRACE   TICKS    SCHED_D  SCHED_R   TIMESLICE
 6    4     2        1      2      10       16       16         1
```

- **multi**

  A large machine running the 64-bit AIX V4.3 would probably be used to handle a large or multiuser-type environment rather than a workstation-type workload. Thus, it is reasonable to expect that the number of active user processes at any given time would be more than two.

The `m`, or multi, parameter, determines a lower limit for the number of active processes that are allowed to stay active when the system is thrashing. These are the processes that are runable and waiting for page I/O. Processes that are waiting for events and processes suspended are not considered active. Nor is the wait process considered active. Also excluded from the count are kernel processes and processes with fixed priority values less than 60, pinned memory, or awaiting events because no process in these categories is ever eligible for suspension.

The default value specified for this limit is 2. It ensures that at least two user processes are always able to be active when the system is thrashing. Increasing this to some larger value may result in better performance. Since there is no clear guideline on what an optimum value of `m` should be, you will probably have to experiment with different values of `m` to find your ideal figure.

```
# /usr/samples/kernel/schedtune -m 10
```

This parameter, however, is ignored by the system if the memory load control is not turned on. Thus, there is no sense in tuning the value of `m` unless the value of `h` is not zero.

Note

Thrashing occurs when memory is over-committed.

When a page fault occurs, the referenced page must be paged in and, on average, one or more pages must be paged out. AIX attempts to steal real memory from pages that are unlikely to be referenced in the near future via the page replacement algorithm.

At some level of competition for memory, no pages are good candidates for paging out to disk because they will all be reused in the near future by the active set of processes. When this happens, continuous paging in and out occurs. This condition is called thrashing.

Thrashing results in incessant I/O to the paging disk and causes each process to encounter a page fault almost as soon as it is dispatched, with the result that none of the processes make any significant progress.

At a certain point in time when this happens, the system will begin to suspend jobs so as to attempt to recover from the thrashing state. The CPU will keep suspending jobs until it reaches the value specified by the `-m` option.

- **timeslice**

  The default CPU timeslice is 10 ms. This is the longest possible time that a process or thread can be in control before it is replaced by another process or thread. The `timeslice` parameter value of 1 corresponds to this, and in most cases, you would leave this parameter unchanged.

  However, if the applications you run on your machines predominantly use threads with the SCHED_RR scheduling policy, such as do certain database applications, you may want to consider increasing this value so that it is 30 ms or more.

  ```
  # /usr/samples/kernel/schedtune -t 3
  ```

Note

This parameter only applies to threads with the SCHED_RR scheduling policy. That is to say, the variable time slices only affect fixed priority threads.

4.2 Memory Tuning Considerations

One of the main benefits of the 64-bit AIX V4.3 is it expanded addressability. What this translates to, among other things, is the ability to support more physical memory, up to 16 GB in the case of the new S70 machine.

4.2.1 Paging Space

A side effect of using large amounts of memory is that there should be a large amount of paging space made available to back it up. At the very least, the disk space allocated for paging space should be equivalent to the size of real memory. Of course, the actual amount of paging space to be defined on any system varies and will depend on the load imposed on it.

It follows then that paging space will be very huge. For example, if there is 16 GB of real memory available on a system, at least 16 GB of disk space has to be put aside for paging. The default hd6 created in the rootvg would, in most cases, be far from sufficient, and it will be necessary to expand the hd6 paging space. Care must be taken when expanding hd6 to ensure that it is one contiguous whole and not fragmented since fragmentation would impact performance negatively.

Ideally, there should be several paging spaces of roughly equal sizes created, each on a separate physical disk drive. You should also attempt to create

these paging spaces on relatively lightly loaded physical volumes so as to avoid causing any of these drives to become bottlenecks.

4.2.2 Modifying VMM with vmtune

The vmtune command is used to modify the VMM parameters that control the behavior of the memory-management subsystem. The vmtune command can only be invoked successfully by the root user, and any changes made to its parameters will only remain in effect until the next reboot. Running vmtune with any parameters gives you the current settings:

```
# /usr/samples/kernel/vmtune
vmtune:  current values:
-p       -P        -r          -R         -f       -F       -N        -W
minperm  maxperm minpgahead maxpgahead minfree maxfree pd_npages maxrandwrt
3067    12268       2           8        119      127     524288        0

-M      -w      -k      -c        -b         -B           -u        -l
maxpin npswarn npskill numclust numfsbufs  hd_pbuf_cnt lvm_bufcnt lrubucket
  13088    1024     256       1      93          64          9     131072

number of valid memory pages = 16360    maxperm=75.0% of real memory
maximum pinable=80.0% of real memory    minperm=18.7% of real memory
number of file memory pages = 2492      numperm=15.2% of real memory
```

- **numclust**

  Since you have large amounts of memory, you should probably do some tuning so that when the syncd runs, there won't be a huge amount of I/O that gets flushed to disk. One of the things you should consider is turning on the write-behind options using the vmtune command. This increases performance by asynchronously writing modified pages in memory to disk rather than waiting for syncd to do the flushing.

  Sequential write-behind initiates I/O for pages if the VMM detects that writing is sequential. The file system divides each file into clusters of four dirty pages of 4 KB each. These 16 KB clusters are not written to disk until the program begins to write the next 16 KB cluster. At this point, the file system forces the four dirty pages to be written to disk. By spreading out the I/O over time instead of waiting for syncd, it prevents I/O bottlenecks from taking place. A benefit derived from the clustering is that file fragmentation is diminished.

  If it is envisaged that there would be sequential writes of very large files, it may benefit performance by boosting the numclust value to an even higher figure. Any integer greater than 0 is valid, and the default is 1 cluster. Care

must be taken when changing this parameter to ensure that the devices used on the machine support fast writes.

To turn on sequential write-behind:

```
# /usr/samples/kernel/vmtune -c 2
```

- **maxrandwrt**

  Another type of write-behind supported by the `vmtune` command is the random write-behind. This option can be used to specify the threshold (in 4 KB pages) for random writes to be accumulated in memory before the pages are written to disk. This threshold is on a per-file basis. You may also want to consider turning on random write behind. To turn on random write-behind, try the following value:

  ```
  # /usr/samples/kernel/vmtune -W 128
  ```

  It should be noted that not every application would benefit in performance from write-behind. In the case of database index creation, it is actually beneficial to disable the write-behind before the creation activity. Write-behind can than be re-enabled after the indexes have been created.

- **maxperm**

  If it is intended for the system to serve as an NFS file server, the large bulk of its memory would be dedicated to storing persistent file pages rather than working segments. Thus, it would help performance to push up the `maxperm` value to take up as much of the memory as possible. The following command would do just that:

  ```
  # /usr/samples/kernel/vmtune -P 100
  ```

  The converse is true if the system is to be used for numerically intensive computations or in some other application where working segments form the dominant part of the virtual memory. In such a situation, the `minperm` and `maxperm` values should be lowered. Fifty percent would be a good start.

- **maxpgahead**

  If large files are going to be read into memory often, the `maxpgahead` should be increased from its default value of 8. The new value could be any power of 2 value because the algorithm for reading ahead keeps doubling the pages read. The flag to modify `maxpgahead` is `-R`. So, to set the value to 16, you would enter:

  ```
  # /usr/samples/kernel/vmtune -R 16
  ```

Note

When `maxpgahead` is changed, `maxfree` will have to be adjusted to accommodate the change. The `maxfree` value should be greater than `minfree` by at least 8 or by the `maxpgahead` value, whichever is greater.

4.2.3 Binding Processes to Processors

The `bindprocessor` command is used to bind or unbind all the threads of a process to a processor. For thread-based programs, especially those with many threads, you should avoid binding the process to a specific processor. Binding it to a processor will result in the process running as though on a uniprocessor machine, taking absolutely no advantage of the multiprocessor capability of an SMP machine. Thus, unless you have a very good reason for it, you should be careful not to bind such processes to a specific processor.

This is something that you normally should not do.

4.3 I/O Tuning Considerations

The expanded addressability of the 64-bit AIX V4.3 means that larger files can be read and written than was possible with earlier releases of AIX. There are some possible tuning steps that can be taken to get the most out of a system running this version of AIX.

4.3.1 Logical Volume Striping

For an application accessing very large files on a system with large memory, the primary bottleneck would be the disk I/O. Files have to be read from the disk to memory in order to be processed and then written from memory to disk once the work on it is completed. A great deal of time would have to be spent waiting on I/O in these instances.

In such a situation, it seems prudent to implement logical volume striping. With the data in a logical volume spread evenly across several disks, the application accessing the large files is able to utilize the I/O capacity of a few drives in parallel.

It would be worth your while to keep some of the following in mind when tuning for a system for lots of sequential I/O over striped logical volume.

- The stripe unit size should be set to a large value, such as 64 or 128 KB.

- The max_coalesce value should be set to equal the stripe size. The max_coalesce value is the largest request size (in terms of data transmitted) that the device driver attempts to transfer to or from a logical disk in a single operation.
- The minpgahead should be left at its default of 2, while maxpgahead can be raised to 16 times the number of disk drives, resulting in the reading of one stripe unit from each of the disk for each read ahead.
- I/O requests should be made in blocks the size of maxpgahead.
- If the logical volume will occupy physical drives that are connected to two or more disk adapters, the I/O buffers used should be allocated on 64-byte boundaries. 64-byte-aligned I/O buffers should be used so as to avoid having the LVM serialize the I/Os to the different disks. The following code would yield a 64-byte-aligned buffer pointer:

```
char *buffer;
buffer = malloc(MAXBLKSIZE+64);
buffer = ((int)buffer 64) & ~0x3f;
```

4.3.2 Modifying I/O with vmtune

Some of the parameters provided by vmtune can help in I/O performance tuning. Of interest is the write-behind facility which has been addressed in an earlier section, “Memory Tuning Considerations” on page 51.

Turning on and tuning these parameters (numclust and maxrandwrt) could result in a reduced I/O bottleneck because now, writes to disk to not have to wait on the syncd daemon, but rather, can be spread more evenly over time. There will also be less file fragmentation because dirty pages are clustered first before being written to disk.

The other vmtune parameters that can be tuned for I/O performance are listed below. It is important to bear in mind, though, that using large files does not necessarily warrant any tuning of these parameters. The decision to tune them will depend very much on what type of I/Os are occurring to the files. The size of the I/O, whether it is raw or journaled file system I/O, and the rate at which the I/O is taking place are important considerations.

- **numfsbufs**

  This parameter specifies the number of file system buf structs. Buf structs are defined in /usr/include/sys/buf.h.

  When doing writes, each write buffer will have an associated buffer header as described by the struct buf. This header describes the contents of the buffer.

Increasing this value will help write performance for very large writes size on devices that support very fast writes. A filesystem will have to be unmounted and then mounted again after changing this parameter in order for it to take effect.

```
# /usr/samples/kernel/vmtune -M 100
```

The default value for this parameter is 93 for AIX V4. Each filesystem gets two pages worth of buf structs. Since two pages is 8192 bytes and since `sizeof(struct buf)` is about 88, the ratio is around 93. 8192/88=93. What the value of `numfsbufs` should be is based on how many simultaneous I/Os you would be doing to a single filesystem.

Usually, though, this figure will be left unchanged, unless your application is issuing very large writes (many megabytes at a time) to fast I/O devices such as HIPPI.

- **lvm_bufcnt**

  This parameter specifies the number of LVM buffers for raw physical I/Os. If the striped logical volumes are on raw logical volumes and writes larger than 1.125 MB are being done to these striped raw logical volumes, increasing this parameter might increase throughput of the write activity.

  ```
  # /usr/samples/kernel/vmtune -u 16
  ```

  The 1.125 MB figure comes about because the default value of lvm_bufcnt is 9, and the maximum size the LVM can handle in one write is 128 KB. (9 buffers * 128 KB equals 1.125 MB.)

- **hd_pbuf_cnt**

  This attribute controls the number of pbufs available to the LVM device driver. Pbufs are pinned memory buffers used to hold I/O requests.

  In AIX V4, a single pbuf is used for each sequential I/O request regardless of the number of pages in that I/O. The default allows you to have a queue of at least 16 I/Os to each disk, which is quite a lot. So, it is often not a bottleneck. However, if you have a RAID array which combines a lot of physical disks into one hdisk, you may need to increase it.

4.3.3 Working with the jfslog

The jfslog is written to every time when:

- A file is created.
- A file is deleted.
- An application opens a file with `O_SYNC` (in which case, every write causes a corresponding jfslog write);

- A `sync` or an `fsync` is done (in which case, one or more JFS transactions may occur for each file that has outstanding I/Os).

Two things can be done to avoid the jfslog from becoming a performance bottleneck. The first is to increase the size of the jfslog, and the second is to create more than one jfslog per volume group.

4.3.3.1 Increasing the JFS Log Size

With the increased filesystem sizes available to current versions of AIX, it is not unimaginable that the amount of concurrent transactions within the filesystem would increase. When this happens, there is a possibility of a lot of writes taking place in the jfslog.

By default, a jfslog file is created the size of one logical partition in a volume group. When the amounts of writes to jfslog increases beyond a threshold so that there is not enough time to commit these logs, any further writes will be suspended. These pending writes will remain so until all the outstanding writes are committed and the meta data and jfslog are in sync with each other.

If the jfslog is made bigger than the default, I/O can continue to proceed because the jfslog wrap threshold would not be reached as easily.

The steps taken to increase the jfslog would be as follows:

1. Backup the file system.
2. Create the log logical volume. Suppose you wish to make the jfslog size two logical partitions now, instead of one.

   ```
   # mklv -t jfslog -y LVname VGname 2 PVname
   ```

   where `LVname` is the name of the jfslog logical volume, `VGname` is the name of the volume group on which it is to reside, and `PVname` is the hdisk name on which the jfslog is to be located.
3. When the jfslog logical volume has been created, it has to be formatted:

   ```
   # /usr/sbin/logform /dev/LVname
   ```
4. The next step is to modify the affected filesystem or filesystems and the logical volume control block (LVCB).

   ```
   # chfs -a log=/dev/LVname /filesystemname
   ```
5. Finally, unmount and then mount the affected file system so that this new jfslog logical volume can start being used.

   ```
   # unmount /filesystemname; mount /filesystemname
   ```

4.3.3.2 Creating More Than One jfslog for a Volume Group

By default, one jfslog is created per volume group containing journaled filesystems. Sometimes, because of heavy transactions taking place in more than one file system in this volume group, it may be beneficial from a performance standpoint to create more jfslogs so that there would be less sharing, and thus less resource contention, for them.

The steps outlined in the earlier section can be used to create these additional jfslogs. Where possible, the jfslogs should be created on disks of relatively low activity so as to free the disk resources to focus on the logging activity.

4.4 Network Tuning Considerations

64-bit enablement does not generally change the way performance tuning of the network is viewed. However, a side effect of using 64-bit hardware is that more physical memory usage is supported. The increased memory could mean that the number of processes running on the system could be increased. When this happens, it may be necessary to review the default settings of some of the network parameters.

4.4.1 Modifying the Path MTU Settings

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

AIX supports a path MTU discovery algorithm as described in RFC1191. By default, this option is turned off. You can turn it on by using the `no` parameters `tcp_pmtu_discover` and `udp_pmtu_discover`.

```
# no -o tcp_pmtu_discover=1
# no -o udp_pmtu_discover=1
```

Since routes can change dynamically, the path MTU value for a path may also change over time. Decreases are automatically checked at a frequency specified by the `pmtu_default_age`, the default of which is 10 minutes. Increases, on the other hand, are checked at the frequency specified by the `pmtu_rediscover_interval`, which is set to a default of 30 minutes.

Turning on the tcp_pmtu_discover and udp_pmtu_discover parameters would probably be beneficial, from a performance viewpoint, for machines that support large MTUs and that sit on large bandwidth networks.

Note

A cautionary note should be provided about tuning this parameter, though. If your network comprises many smaller machines, such as PCs with small MTUs, you could end up running your network with a very small path MTU, thereby impacting your network performance negatively. In such a case, it would be prudent to weigh the performance degradation due to fragmentation against that due to small MTU sizes and decide on what whether or not to change the default setting.

It may also be worthwhile considering organizing your network physically along the capabilities of the machines in the network, putting components that can manage large MTUs in one network and the others in another. That way, the smaller machines do not pace the MTU sizing.

4.4.2 Modifying NFS Performance with nfso

A new parameter, nfs_rfc1323, is provided with the nfso command in AIX V4.3.

As described in 3.4.2, “The nfso Command” on page 41, this option is used to enable very large TCP window size negotiation to occur between systems. It increases the throughput potential between client and server. The option requires that both client and server have the following:

- TCP support
- MTU discovery
- Socket buffers large enough to handle the new MTU size

Some tests carried out internally have shown that setting nfs_rfc1323=1 improves performance when running over ATM with a large MTU size (59 KB for MCA, and 64 KB for PCI). Generally, applications that do large sequential reads and writes over TCP NFS between machines connected via a high-speed media, such as the ATM and the SP switch, should benefit from turning this parameter on. The MTUs of these are generally greater than 32 KB.

This parameter works independently of the no command’s rfc1323 option. Thus, if you can choose to have the optimization on just for NFS, do so.

4.5 References

The following are good sources for additional information.

- *AIX Version 4.3 Differences Guide,* SC24-2014
- *AIX Performance Tuning Guide: Versions 3.2 and 4*, SC23-2365
- *RS/6000 Performance Tools in Focus*, SG24-4989
- *EMEA Power Academy 1997: Fine Tuning AIX with vmtune and schedtune*

Chapter 5. Programming in 64-bit

While the large process space frees programmers from concerns about memory constraints, porting in general to 64-bit will present a modest technical effort which can rapidly worsen in the face of poor coding practices.

This chapter discusses 64-bit technical and programming issues in designing 64-bit applications.

If you want more details about AIX V4.2 large file support implementation, refer to the *AIX Version 4.2 Differences Guide*, SG24-4807.

5.1 64-bit Programming Specifications

To take advantage of a 64-bit hardware and operating system, you should be able to create 64-bit processes. Since AIX is UNIX standard compliant (X OPEN, XPG, ANSI, POSIX), 64-bit processes are often restricted to C programs: almost all UNIX implementations are written in C.

However, as seen in Chapter 2, “Why Do I Need 64-bit?” on page 13, one of the main audience for 64-bit programming applications are scientists and engineers involved in large numeric computations. Their favorite programming language is still FORTRAN.

Because of these reasons, 64-bit programming specifications are given for C language and FORTRAN language.

In this section, techniques for writing efficient C and FORTRAN programs are not discussed because they are extensively described in the book titled *Optimization and Tuning Guide for Fortran, C, and C++,* SC09-1705. Reading this book will be a great benefit for programmers.

If you want more details about AIX V4.2 large file support implementation, see the *AIX Version 4.2 Differences Guide*, SG24-4807, redbook.

5.1.1 64-bit C Programming Specifications

To enable 64-bit C programming, the C language standard has been extended to include 64-bit capabilities. This paragraph illustrates 64-bit features in the AIX C compiler.

5.1.1.1 Defining a New Standard: LP64

Although there are a number of choices regarding 64-bit processes, the choice that was settled upon by the Aspen working group, formed by X OPEN

and a consortium of hardware vendors, is called *LP64*, short for Long-Pointer 64. This is commonly called the 4/8/8 model, which stands for the Integer/Long/Pointer type sizes.

The benefit of this configuration is that it provides 64-bit addressing with 8-byte pointers, large object support (8-byte longs) and backward compatibility (4-byte ints). Other alternatives include an 8/8/8 solution called ILP64 and the LLP64 which was rejected.

In choosing LP64, we are allowing most int types in programs (which most people use anyway) to remain int types.

Depending on the coding style and function of the application, the number of cases where users expect the size of ints to be the same as pointers and long types will vary from program to program. Overall, this is a small percentage. The LP64 choice has been made to avoid migration efforts on many programs: there should be a migration need for a few instances of actual code changes.

Most of the other unexpected behavior only occur at the limits of numerical specifications where we expect an even smaller number of cases.

Currently, all IBM compilers are basically what is called ILP32 + long long. They use the ILP32 model and include a long long type. The following table, Table 6 on page 63, illustrates the type size for the different models.

In this table, ANSI fixed-size types are included and highlighted with (2). ANSI introduced the two set of types: One is the signed fixed-size integral types; the other is the unsigned fixed-size integral types. All AIX users are encouraged to use these ANSI types through the header `<inttypes.h>` rather than __int8/16/32/64 types highlighted with (3).

The use of __int8/16/32/64 are strictly used for Microsoft portability and compatibility with VisualAge C for Intel products. *They should not be used for new coding*. They do not have the same sign behavior in default char mode (command line option -qchar=signed/unsigned). *We highly recommend using ANSI fixed-size types for further compatibility*.

Table 6. ILP32, LP64 and C for AIX Compiler Model Type Size

Datatype	ILP32 (size in bit)	LP64 (size in bit)	C for AIX (32-bit / 64-bit)
char	8	8	Implemented
short	16	16	Implemented
int	32	32	Implemented
long	**32**	**64**	Implemented
long long	Not Defined	Not Defined	64 / 64
pointer	**32**	**64**	Implemented
float	32	32	Implemented
double	64	64	Implemented
long double	Not Defined	Not Defined	64 or 128 / 64 or 128(1)
int8_t uint8_t	Not Defined	Not Defined	Fixed-width type(2) 8 bits (1 byte)
int16_t uint16_t	Not Defined	Not Defined	Fixed-width type(2) 16 bits (2 bytes)
int32_t uint32_t	Not Defined	Not Defined	Fixed-width type(2) 32 bits (4 bytes)
int64_t uint64_t	Not Defined	Not Defined	Fixed-width type(2) 64 bits (8 bytes)
__int8 __uint8	Not Defined	Not Defined	Fixed-width type(3) 1 byte
__int16 __uint16	Not Defined	Not Defined	Fixed-width type(3) 2 bytes
__int32 __uint32	Not Defined	Not Defined	Fixed-width type(3) 4 bytes
__int64 __unit64	Not Defined	Not Defined	Fixed-width type(3) 8 bytes

(1) Depend on the setting of the long double option. By default, the size is 8.
(2) ANSI fixed-size types.
(3) Should not be used; use ANSI types instead.

5.1.1.2 Structure Alignment

The LP64 specification changes the size and alignment of certain structure elements, thus effecting the size of the structure itself. In general, all

structures but the simplest will have to be checked for size and alignment dependencies.

Structures with long integers and pointers will at least double in size under 64-bit mode, depending on the alignment. Sharing data structures between 32- and 64-bit processes is no longer possible, unless the structure is devoid of pointer and long types. Unions that attempt to share long and int types, or overlay pointers onto int types, will now be aligned differently or be corrupted.

The `-qalign` suboptions specify what aggregate alignment rules the compiler uses for file compilation. Use this option to specify the maximum alignment to be used when mapping a class-type object, either for the whole source program or for specific parts. You can choose six type suboptions. The default value is `-qalign=full`.

power The compiler uses the RISC System/6000 alignment rules.

full The compiler uses the RISC System/6000 alignment rules. The power option is the same as full.

mac68k The compiler uses the Macintosh alignment rules.

twobyte The compiler uses the Macintosh alignment rules. The mac68k option is the same as twobyte.

packed The compiler uses the packed alignment rules.

natural The compiler maps structure members to their natural boundaries. This has the same effect as the power suboption, except that it also applies alignment rules to doubles and long doubles that are not the first member of a structure or union.

If you use the `-qalign` option more than once on the command line, the last alignment rule specified applies to the file.

RISC System/6000 Alignment Rules (power, full, natural)

On the RISC System/6000 system, an aggregate is aligned according to its most strictly aligned member. Within aggregates, members are aligned according to their type. Table 7 summarizes size and alignment information for each type.

Table 7. RISC System/6000 Alignment Rules

Type	Alignment (ILP32)	Alignment (LP64)
char	byte aligned	byte aligned
short	half word aligned	half word aligned
int	word aligned	word aligned

Type	Alignment (ILP32)	Alignment (LP64)
long	**word aligned**	**double word aligned**
long long int	double word aligned	double word aligned
pointer	**word aligned**	**double word aligned**
float	word aligned	word aligned
double	double word aligned if `-qalign=natural`. Otherwise, word aligned	double word aligned if `-qalign=natural`. Otherwise, word aligned
long double	long double word aligned if `-qalign=natural`. Otherwise, word aligned	long double word aligned if `-qalign=natural`. Otherwise, word aligned

The first element in a structure will be aligned according to the strictest alignment of its members. Because of the potential padding introduced by the member alignment, structure alignment will not be exactly the same as in the 32-bit mode. This is especially important for arrays of structures which contain pointer or long types. The member alignment will change forcing each structure element of the array to start at a double word boundary.

Example:

```
#include <stdio.h>
void main(void) {
  struct li{
      long la;
      int  ia;
  } li;

  struct lii{
      long la;
      int  ia;
      int  ib;
  } lii;

  struct ili{
      int  ia;
      long la;
      int  ib;
  } ili;

  printf("length li  = %d\n",sizeof(li));
  printf("length lii = %d\n",sizeof(lii));
  printf("length ili = %d\n",sizeof(ili));
```

```
}
```

This sample program output will be different of 32-bit and 64-bit mode.

32-bit mode:

```
length li  = 8
length lii = 12
length ili = 12
```

64-bit mode:

```
length li  = 16
length lii = 16
length ili = 24
```

In 32-bit mode, both integers and long integers are four bytes, and struct size is four bytes times number of members. In 64-bit, each element of the array must be double word aligned. Thus the size of the struct `li` is 16 bytes (8 byte long + 4 byte int + 4 byte pad). The struct `lii` and `ili` are the same element, but have a different order. Because of the padding, the struct `ili` is expanded 24 bytes (4 byte int + 4 byte pad + 8 byte long + 4 byte int + 4 byte pad).

One should be careful when using an array of structures. The proper alignment will save system resource and have better performance. Keep in mind that many types of subroutines delivered have an alias for long or pointer. For example, the `size_t` type is defined as unsigned long.

Macintosh and Twobyte Alignment Rules

All unions and structures are half word aligned regardless of their members. Within the aggregate, members are aligned according to their type. Therefore, both IPL32 and LP64 use the same alignment rules. The table below summarizes alignment information for each type.

Table 8. Macintosh and Twobyte Alignment Rules

Type	Alignment (ILP32,LP64)	Size (IPL32)	Size (LP64)
char	byte aligned	byte	byte
short	half word aligned	half word	half word
int	half word aligned	word	word
long	half word aligned	**word**	**double word**
long long int	half word aligned	double word	double word
pointer	half word aligned	**word**	**double word**

Type	Alignment (ILP32,LP64)	Size (IPL32)	Size (LP64)
float	half word aligned	word	word
double	half word aligned	double word	double word
long double	half word aligned	double word	double word
long double with `-qlongdouble` or `-qlbdl128` option	half word aligned	quadruple word	quadruple word

The following sample program output shows the struct size difference only comes from the size of the data types.

32-bit mode:

```
length li  = 8
length lii = 12
length ili = 12
```

64-bit mode:

```
length li  = 12
length lii = 16
length ili = 16
```

Packed Alignment Rules

All structures are byte-aligned regardless of their members. All members are also byte-aligned. (Bit fields are byte-aligned, but bit-field members are not.) Using packed alignment rules, the struct size only depends on the data size.

The following sample program output also shows the struct size is equal to the total size of the members.

32-bit mode:

```
length li  = 8
length lii = 12
length ili = 12
```

64-bit mode:

```
length li  = 12
length lii = 16
length ili = 16
```

5.1.1.3 Bit Fields

Structure bit fields in 32-bit mode are limited to 32 bits, and can be of type signed int, unsigned int, or plain int. Bit fields are packed into a word. Adjacent bit fields that cross a word boundary will start at a storage unit which is a word aligned, half word in the mac68k and two byte alignment, and byte in the packed alignment.

In 64-bit mode, bit fields are limited to at most 64-bits in size and can be of type signed int, unsigned int, plain int, long, and unsigned long. If it is of type long, or unsigned long, the maximum size is 64 bits and is double word aligned. The alignment will remain the same in all other alignment modes. This means that adjacent declarations of bit fields which used to take two consecutive words in 32-bit mode can now be contained in a simple long declaration in 64-bit mode. Since long bit fields were not permitted before in 32-bit mode, this is usually not a portability problem.

Example:

```
struct bit_fields {
    unsigned a : 7;  /* total  7 bits */
    unsigned b : 6;  /* total 13 bits */
    unsigned c : 2;  /* total 15 bits */
    unsigned d : 8;  /* total 23 bits */
    unsigned e : 4;  /* total 27 bits */
    unsigned f : 4;  /* total 31 bits */
    unsigned g : 2;  /* total 33 bits */
};
```

In 32-bit mode, elements a to f are packed in a single word with 1 bit of padding. Element g is placed in the next consecutive word. In 64-bit mode, however, all the elements of the structure are packed into a double word.

5.1.1.4 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. This change from the default 32-bit to 64-bit mode is under user control.

64-bit printf Support

The `printf` subroutine format string for a 64-bit integer is different than the string used for a 32-bit integer. Programs that do these conversions must be careful to use the proper format specifier.

You should also consider the maximum figure size of `long` and `unsigned long` type. The `ULONG_MAX` (18,446,744,073,709,551,615) is twenty digits long.

Example:

```
i#include <stdio.h>
void main(void) {
    printf("LONG_MAX( d) = %d\n",LONG_MAX);
    printf("LONG_MAX( x) = %x\n",LONG_MAX);
    printf("LONG_MAX(lu) = %lu\n",LONG_MAX);
    printf("LONG_MAX(lx) = %lx\n",LONG_MAX);
}
```

Output:

```
LONG_MAX( d) = -1
LONG_MAX( x) = ffffffff
LONG_MAX(lu) = 9223372036854775807
LONG_MAX(lx) = 7fffffffffffffff
```

64-bit Iconv Support

The `iconv_open` subroutine uses the `LOCPATH` environment variable to search for a converter whose name is of the form:

iconv/FromCodeSet_ToCodeSet

The `FromCodeSet` string represents the sender's code set, and the `ToCodeSet` string represents the receiver's code set. The underscore character separates the two strings.

Note

All `setuid` and `setgid` programs will ignore the `LOCPATH` environment variable.

Since the `iconv` converter is a loadable object module, a different object is required when running in the 64-bit environment. In the 64-bit environment, the `iconv_open` routine will use the `LOCPATH` environment variable to search for a converter whose name is in the form:

iconv/FromCodeSet_ToCodeSet__64

The `iconv` library will automatically choose whether to load the standard converter object or the 64-bit converter object. If the `iconv_open` subroutine does not find the converter, it uses the FromCodeSet and ToCodeSet name pair to search for a file that defines a table-driven conversion. Many conversion tables are only for the 32-bit environment. The code conversion

table is created by the genxlt command. The iconvTable converter uses the LOCPATH environment variable to search for a file whose name is in the form:

iconvTable/FromCodeSet_ToCodeSet

The genxlt command is included in the fileset bos.loc.adt.iconv. If the genxlt command failed because no code was set, iconv_open will return error 22 (EINVAL). We recommend generating a code set conversion table for 64-bit in advance, for better application performance. See the genxlt command in the AIX Version 4.3 Commands Reference for more information.

Compiler Invocation

The features added to the compiler to enable 64-bit mode include:

- Predefined __64BIT__ macro when invoked for 64-bit compilations
- OBJECT_MODE environment variable
- -qarch support for 64bit suboption

The compiler can be invoked for 64-bit or 32-bit mode by setting an environment variable, or by using a command line option to set the compilation mode of the compiler.

The compiler evaluates competing options for compilation mode in the following order:

1. OBJECT_MODE setting
2. Configuration file
3. Command line

The above list forms the user-chosen compilation mode. This choice must be evaluated with the arch/tune setting to decide the actual compilation mode. The interaction of the above list of user-chosen compilation mode and the arch/tune setting is described in 5.1.3, “64-bit Object Format Specification” on page 77

__64BIT__ Macro

When the compiler is invoked to compile for 64-bit mode, the preprocessor macro __64BIT__ is defined. 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

The C compiler obtains information from environment variables. These may be the *setlocale* variable which changes the language of messages, or the *_xlC_test_usrinc* path variable which overrides the default place to look for system header files (/usr/include).

This latter option facilitates easy testing of new header files without having to change 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 has been introduced that, when used, replaces the default compilation mode of the compiler.

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 and produces files in the 64-bit XCOFF format. The compiler cannot generate 64-bit instructions that operate safely on 32-bit XCOFF. This implies that 64-bit and 32-bit objects can not be mixed.

5.1.2 64-bit FORTRAN Programming Specifications

FORTRAN (FORmula TRANSlation) is a high-level programming language primarily designed for applications involving numeric computations. FORTRAN, the language of choice for scientists and engineers, is now celebrating its 40th birthday. Due to its ease of use and intuitive structure, it is also the language of choice for novice programmers in universities.

Due to these reasons, we know a lot about FORTRAN as FORTRAN 77 standard and we still ignore the new capabilities of the Fortran 90 standard. In this section the main characteristics of Fortran 90 are presented so 64-bit capabilities can be discussed. FORTRAN examples are written in Fortran 90.

5.1.2.1 Fortran 90 Standard

The Fortran 90 language standard supports the FORTRAN 77 language standard. Fortran 90 adds a wealth of new features to FORTRAN 77. Compared to FORTRAN 77, Fortran 90 may offer an improved method or feature for performing a given task.

The following outlines some of the key features that Fortran 90 brings to the FORTRAN 77 language:

- Fortran 90 free source form

In addition to the fixed source form format (defined in FORTRAN 77), Fortran 90 defines a free source form format.

- Parameterized data types

  Although the length specification for a data type (for example, INTEGER*4) is a common industry extension, Fortran 90 provides facilities for specifying the precision and range of non-character intrinsic data type and the character sets available for the character data type.

- Derived types

  A derived type is a user-defined type whose components are of intrinsic type and/or other derived types.

- Array enhancements

  With Fortran 90, you can specify arrays expressions and assignments. An array section, a portion of a whole array, can be used as an array. Array constructors offer a concise syntax for specifying the values of an array.

- Pointers

  Pointers refer to memory addresses instead of values. Pointers provide the means for creating linked lists and dynamic arrays.

- Dynamic behavior

  Storage is not set aside for pointers targets and allocatable arrays at compile time. The ALLOCATE/DEALLOCATE statement lets users control the storage usage at run time. You can also use pointer assignment to alter the storage space with the pointer.

- Control construct enhancements

  The CASE construct provides a syntax for selecting, at most, one of a number of statement blocks for execution. The DO statement with no control clause and the DO WHILE construct offer increased versatility.

- New intrinsic procedures

  Fortran 90 brings dozens of new intrinsic procedures to FORTRAN.

- Procedure enhancements

  Fortran 90 introduces many features that make the use of procedures easier.

- Modules

  Modules provide the means for data encapsulation and the operations that apply to the data. A module is a non-executable program unit that can contain data object declarations, derived-type definitions, procedures and procedures interfaces.

The following examples of array programming code illustrates how powerful Fortran 90 features are:

- FORTRAN 77 source

```
REAL A(7,8), B(7,8), C(7,8)
DO 10 I=1,7
    DO 20 J=1,8
        C(I,J) = A(I,J)+B(I,J)
    20 CONTINUE
10 CONTINUE
```

- Fortran 90 source

```
REAL ,DIMENSION(7,8):: A, B, C
C=A+B ! Add the Two arrays
```

5.1.2.2 64-bit FORTRAN Integer Capabilities

As shown in Table 6 on page 63, ILP32 implementation does not implement 64-bit integer computations for C language. Only the optional feature long long in C for AIX compiler allows 64-bit integer computations.

However, in for FORTRAN, standard specification defines 64-bit integer capabilities. This section illustrates the different level of 64-bit size proposed by Fortran 90 (FORTRAN 77) standard.

FORTRAN Data Types

The Fortran 90 specification defines two categories of data types: *intrinsic types* and *derived types*. The derived types are user-defined data types, this feature is described in 5.1.2.1, “Fortran 90 Standard” on page 71. The intrinsic types, including their operations, are predefined and are always accessible. There are two classes of intrinsic data types:

- Numeric (also known as arithmetic): integer, real, complex and byte
- Nonnumeric: character, logical and byte

The Fortran 90 specification requires:

- Default *integer* and default *real* occupy one numeric storage unit.
- Double precision occupies two numeric storage units.

Since FORTRAN on PowerPC uses 32- and 64-IEEE floating-point formats, a numeric storage unit is 32 bits. Hence, INTEGER is 32 bits.

FORTRAN Data Types

Unlike C's long data type, FORTRAN's default INTEGER can not grow from 32 to 64 bits.

FORTRAN default INTEGER is 32 bits, even in the 64-bit environment.

FORTRAN specification defines 64-bit integer capabilities, even for a 32-bit environment.

64-bit Integer Computations

FORTRAN variables that are 64-bit integers can be defined using INTEGER(8).

A compiler option can force the FORTRAN for AIX compiler to default integers to 64 bits; this would then be out of conformance with Fortran 90. The `-qintsize` compiler option is used to change the default integer (and logical) to 2, 4 (the default) or 8 bytes. This option is intended to allow you to port programs unchanged from systems that have different default sizes for data.

For example, you might need `-qintsize=2` for programs written for a 16-bit microprocessor or `-qintsize=8` for programs for a Cray computer.

The XL Fortran V3 compiler, released in December 1993, already supports 64-bit integer arithmetic in 32-bit environments. Defining INTEGER(8), you can compute values between -9,223,372,036,854,775,808 (-2^{63}) and 9,223,372,036,854,775,807 ($2^{63}-1$).

5.1.2.3 XL Fortran V5.1 Capabilities

In addition to all of the functions available in XL Fortran for AIX Version 4.1, XL Fortran for AIX Version 5.1 provides the following features:

- SMP support
- 64-bit support
- Asynchronous I/O
- Debug memory routines

Although we are focusing on FORTRAN's 64-bit capabilities, it is important to mention that XL Fortran for AIX Version 5.1 is the first IBM FORTRAN compiler exploiting the RS/6000 Symmetric Multi-Processing (SMP) architecture. It supports both automatic parallelization of a FORTRAN program and explicit parallelization (through a set of directives that you can use to parallelize selected portions of your program). It also implements multithreaded programming capabilities and asynchronous I/O.

Though FORTRAN is a 40 year-old programming language, the continuing evolution of its specification gives FORTRAN many attractive features, especially for heavy SMP use and performance benefits.

For Further Reading

Please refer to these technical articles:

- XL Fortran Compiler for IBM SMP Systems (*AIXpert Magazine*, December 1997).
- Multithreaded Programming in XL Fortran Version 5 (*AIXpert Magazine*, December 1997).

5.1.2.4 XL Fortran 64-bit Support

FORTRAN compiled for 64-bit execution does the following:

- Extends the address part of a Fortran 90 POINTER to 64 bits: POINTER*8.
- Leaves the Fortran 90 *default integer* at 32 bits.
- Allows maximum array bounds to grow from +/– 2^{31} to +/– 2^{63}.

The IBM FORTRAN compiler implementing 64-bit features for AIX V4.3 and higher is referenced as XL Fortran Version 5.1. The 64-bit support includes:

- The *OBJECT_MODE* environment variable that enables you to obtain a 64-bit development environment.
- The `-q64` compiler option, to indicate the 64-bit compilation bit mode, and when combined with the `-qarch` option, determine the target machines on which the 64-bit executable will run.
- The `-q32` compiler option, to enable 32-bit compilation bit mode support in a 64-bit environment.
- The `-qwarn64` compiler option, to help port code from a 32-bit environment to a 64-bit environment.

OBJECT_MODE, `-q64`, `-q32`, and `-qarch` are explained in 5.1.3, “64-bit Object Format Specification” on page 77.

The `-qwarn64` option is discussed in 6.3.5, “The Compiler Option -qwarn64” on page 108.

Note

Note that high optimization level (`-O4` and `-qhot`), interprocedural analysis (`-qipa`), and SMP features (`-qsmp`) are not supported for 64-bit codes with XL Fortran V5.1.

5.1.2.5 Program Enlarged Limits

These are the new enlarged limits for a FORTRAN program compiled in 64-bit mode:

- The default integer and default real size is four bytes.
- The default integer pointer size is eight bytes.
- The maximum array size increases to the size of one segment (approximately 2^{40} bytes).
- The maximum dimension bound range is extended to [-2^{63}, 2^{63}-1].
- The maximum array constants have not been extended and will remain the same as the maximum in 32-bit mode. Therefore, arrays with a size greater than 2^{31}-1 cannot be initialized.
- The maximum iteration count for array constructor implied DO loops increases to 2^{31}-1.
- The maximum character variable length extends to the size of one segment (approximately 2^{40} bytes).
- The maximum character length of character literals remains the same as in 32-bit mode.
- The LOC intrinsic returns an INTEGER(8) value.

5.1.2.6 64-bit FORTRAN Migration Issues

The impact on FORTRAN applications should be very minimal. Unlike C pointers, Fortran 90 POINTER can not be manipulated arithmetically. The length of a POINTER is unspecified in the standardization specification; therefore, FORTRAN programs can not make assumptions about pointer size. Moreover, FORTRAN programs can not use EQUIVALENCE with anything containing a pointer.

However, programs that use pre-Fortran 90 integers pointers (a FORTRAN extension) may require changes:

- Such a pointer is defined by the language as a scalar variable of type INTEGER(4). This will have to change to INTEGER(8).
- Integer pointers can be manipulated arithmetically like C pointers and such arithmetic may not be portable from 32 to 64 bits.

A legal Fortran 90 program should not contain anything that depends on the length of a storage address, and the default integer length remains unchanged. It should be possible to run such programs in 64-bit address spaces by recompiling.

5.1.3 64-bit Object Format Specification

To support larger executables that can be fit within a 64-bit address space, a new object format has been created to meet the requirements of 64-bit executables. This section explains the meaning of the XCOFF object format and the AIX V4.3 format change due to XCOFF.

5.1.3.1 Definition

The eXtended Common Object File Format (XCOFF) is the object format for AIX. XCOFF combines the standard common object file format COFF (defined by AT&T) with the Table Of Contents (TOC) module format concept, which provides for dynamic linking and replacement of units within an object file.

One of the major interests in XCOFF is in its ability to dynamically resolve references to shared libraries and other external objects. COFF, on the other hand, can only resolve references statically.

XCOFF is the formal definition of machine-image object and executable files:

- These XCOFF objects are produced by language processors (assemblers and compilers).
- The binder combines individual object files into an XCOFF executable file.
- The system loader reads an XCOFF executable file to create an executable memory image of a program.
- The symbolic debugger reads an XCOFF executable file to provide symbolic access to functions and variables of an executable memory image.

In AIX V4.3, XCOFF has been extended to provide for 64-bit environment, object files and executable files. Now, there are two XCOFF formats: 32-bit XCOFF format (referred as XCOFF32) and 64-bit XCOFF format (referred as XCOFF64).

For a detailed description of these formats, see *AIX Version 4 Files Reference for AIX*, SC 23-2512 and *AIX V4.3 Files Reference for AIX*, SC23-4168.

Notice that the f_magic field, at offset 0 in the XCOFF object, defines the integer known as magic number. It specifies the target machine and environment of the object file. For XCOFF32, the only valid value is *0x01DF* (0737 Octal). For XCOFF64, the only valid value is *0x01EF* (0757 Octal) referred to as *U803XTOCMAGIC*.

Example:

```
# od -x -N2 a.out.32
0000000  01df
0000002
# od -x -N2 a.out.64
0000000  01ef
0000002
```

5.1.3.2 XCOFF Format Implications

There are different implications in the XCOFF format change of which users should be aware:

- Only AIX V4.3 and above can generate and load 64-bit XCOFF objects.

> **Important Note**
>
> As mentioned in 1.3, “64-bit Application Software” on page 10, if 64-bit application development does not require 64-bit hardware, AIX V4.3 should be installed on all RS/6000 32-bit development machines.

- When compiling for 64-bit mode, compilers generate 64-bit instructions and produce files in the 64-bit XCOFF format. The binder binds only 64-bit objects to create 64-bit executables. Note that objects bound together, statically or shared, must all be of the same object format. This means that *64-bit and 32-bit objects can not be mixed*.

> **64-bit Compilation Mode**
>
> The generation of 64-bit instructions and 64-bit XCOFF is called the 64-bit compilation mode.

- The following scenarios are not permitted, and will fail to load and/or execute:
  - A 64-bit object that references symbols which cannot be satisfied by 64-bit libraries.
  - A 64-bit executable that has references to symbols from a shared library with no 64-bit-capable version.

- A 32-bit executable that has references to symbols from a shared library with no 32-bit-capable version.
- A 64-bit executable that attempts to explicitly load a 32-bit module.
- A 32-bit executable that attempts to explicitly load a 64-bit module.
- Attempts to run 64-bit applications on 32-bit platforms.

- 64-bit mode support in compilers is mainly provided through the two new compiler options, `-q64` and `-q32`, used in conjunction with the compiler option `-qarch`. This combination determines the bit mode and instruction set for the target architecture. The `-q32` and `-q64` options take precedence over the setting of the `-qarch` option. Conflicts between the `-q32` and `-q64` options are resolved by the last option wins rule. Setting `-qarch=com` will ensure future compatibility for applications, whereas the `rs64a` and `auto` settings will be more system dependent**.**

  Examples:

  1. Using 32-bit compilation mode and targeting the 601 architecture:

     ```
     -qarch=601 -q32
     ```

  2. Now keep the same compilation mode, but alter the target to RS64A:

     ```
     -qarch=601 -q32 -qarch=rs64a
     ```

     Notice that the last setting for `-qarch` wins.

  3. Now keep the same target, but alter the compilation mode to 64-bit:

     ```
     -qarch=601 -q32 -qarch=rs64a -q64
     ```

     Notice that specifying `-q64` overrides the earlier instance of `-q32`.

Table 9. Settings for -qarch with -q32 or -q64

Compilation Mode	-qarch Setting
-q32	-qarch= <all_settings> except -qarch=ppc64
-q64	-qarch= auto if compiling on a 64-bit systems -qtune=rs64a
-q64	-qarch=com, -qarch=ppc, -qarch=rs64a -qtune=rs64a
-q64	overrides a conflicting setting for -qarch -qarch=601, -qarch=603, -qarch=604 -qarch=pwrx, -qarch=pwr2, -qarch=pwr, -qarch=p2sc, -qarch=pwr2s

- In compilers that do not support 64-bit, such as Pascal and PL1X, use of the `-q32` and/or the `-q64` option will cause the following warning:

```
1501-055 Option -q32, -q64 is not recognized and is ignored
```

- A new archives file type appears in AIX V4.3 and above. The big archives format, ar_big, is the default file format and supports both 32-bit and 64-bit object files. It will work with files larger than two GB. The previous one, referred to as the small archives format, can be used to create archives that are recognized on version of AIX older than 4.3. It can not be larger than two GB.

  To specify which kind of objects files the `ar` command should examine, a new flag, `-X`, has been added. With `-X64`, `ar` processes only 64-bit object files. With `-X32`, the default, `ar` processes 32-bit object files (and ignores 64-bit objects). With `-X32_64`, both 32-bit and 64-bit object files are processed. The behavior can also be changed by setting the OBJECT_MODE environment variable. See 5.1.3.3, “AIX V4.3 Default Mode” on page 81. If a 64-bit object is added to a small format archives, `ar` first converts it to the big format, unless `-g` is specified. By default, `ar` only handles 32-bit object files; any 64-bit object files in an archive are silently ignored.

  In almost all cases, the `-g` flag physically positions the archive members in the order in which they are logically linked. The resulting archives are always written in the small format, so this flag can be used to convert a big-format archive to a small-format archive. Archives that contain 64-bit XCOFF objects cannot be created in or converted to the small format.

Note

The `ar` command does support two `x` flags:

- `-x` which extracts the named files by copying them into the current directory.
- `-X` which specifies the type of object files `ar` should examine**.**

Note that *uppercase X* and *lowercase x* are two `ar` flags with different meanings; don’t be confused.

- Like the `ar` command, there are other AIX utilities which use XCOFF objects. They have been expanded with the same new flag, `-X`. These utilities are:

  dump Dumps selected parts of an object file. The `-X {32|64|32_64}` mode specifies the type of object file dump should examine.

lorder Finds the best order for member files in an object library. The `-X {32|64|32_64}` mode specifies the type of object file `lorder` should examine.

nm Displays information about symbols in object files, executable files, and object-file libraries.

ranlib Converts archive libraries to random libraries**.**

strip Reduces the size of an XCOFF object file by removing information used by the binder and symbolic debug program.

size Displays the section sizes of the XCOFF object files.

- 64-bit mode support in AIX linker is mainly provided through the two new linker options, `-b64` and `-b32`, coming from the compiler stanza file (/etc/xlC.cfg or /etc/xlf.cfg). In `-b32` mode, all input object files must be XCOFF32 files or an error is reported. Only XCOFF32 archives members are processed. XCOFF64 archive members are ignored. In `-b64` mode, all input object files must be XCOFF64 files or an error is reported. Only XCOFF64 archive members are processed. XCOFF32 archive members are ignored.

  If both `-b32` and `-b64` options are specified, the last option specified wins (same behavior as `-q32`/`-q64`). The compiler driver automatically and quietly generates the correct options to call the binder or the correct assembler option (`-a32` or `-a64`). Therefore, the user does not need to specify them.

- The `make` command doesn't discriminate XCOFF object formats; it only discriminates on the timestamp of files. The one case where this can cause a problem is when you try to add 32-bit and 64-bit objects with the same name to an archive. Running `make` in 32-bit mode, then 64-bit mode will not update the second object. The `make` command only checks the timestamp of the first object it finds with the correct name.

5.1.3.3 AIX V4.3 Default Mode

As shown previously, setting the type for XCOFF objects is an important issue because of the consequences it has on the system. To set the compiler's default mode to 64-bit will not maintain behavior compatibility with AIX V4.3 installed on a 32-bit machine. On the other hand, to set default mode to 32-bit will be painful to 64-bit users on a 64-bit machine.

OBJECT_MODE Definition

For these reasons, AIX V4.3 defines an environment variable called *OBJECT_MODE* which replaces the default of compilers if the environment variable exists**.** If OBJECT_MODE does not exist, then compilers will have a

default and it will be 32-bit mode; this facilitates compatibility with systems that do not have OBJECT_MODE set (such as AIX V4.3 installed on a 32-bit machine).

OBJECT MODE can be set to one of three values:

1. OBJECT_MODE=32

   This sets the mode of utilities to generate and/or use 32-bit objects.

2. OBJECT_MODE=64

   This sets the mode of utilities to generate and/or use 64-bit objects.

3. OBJECT_MODE=32_64

   This sets the mode of utilities to accept both 32- and 64-bit objects.

   Compilers, AIX binder and AIX loader will not function in this mode and will generate the error message:

   ```
   1501-054 OBJECT_MODE=32_64 is for mixed-mode and is not a valid
   setting for the compiler.
   ```

   Utilities accepting this OBJECT_MODE setting are: `ar`, `dump`, `nm`, `lorder`, `ranlib`, `size`, and `strip`. These utilities have been discussed in 5.1.3.2, “XCOFF Format Implications” on page 78.

Mixed-Mode Definition

In AIX V4.3, *32-bit XCOFF objects* and *64-bit XCOFF objects* are *supported both and at the same time* only by a limited number of utilities. This capability of supporting both object types is called *mixed-mode*.

Utilities supporting mixed-mode are: `ar`, `dump`, `nm`, `lorder`, `ranlib`, `size`, `strip`.

They have been expanded with a new flag, `-X`. The valid settings for this flag are 32, 64 and 32_64.

Compilers, AIX binder and AIX loader are not mixed-mode eligible.

If OBJECT_MODE is not set, the default mode for all utilities is set to 32-bit. If OBJECT_MODE is set to anything else, this error message is issued:

```
1501-055 OBJECT_MODE setting is not recognized and is not a valid
setting for the compiler.
```

OBJECT_MODE and Default Bit Mode

Between compiler options, compiler configuration files and source file statements and, OBJECT_MODE environment variable, a priority policy has been defined.

As mentioned earlier, in AIX V4.3, the last one wins rule is applied on the following evaluation order (from lowest to highest priority):

1. OBJECT_MODE setting
2. Compilers configuration file:

   `/etc/xlC` and `/etc/xlf`
3. Command Line:

   `-q32`, `-q64` or `-a32`, `-a64` or `-b32`, `-b64` or `-X32`, `-X64`, `-X32_64`
4. Source file:

   Source file priority is defined to be the highest, although since AIX V4.3, there is no more `arch` suboption to the pragma directive (see “Bit Mode Setting and Pragma Directive” on page 84).

Concerning OBJECT_MODE setting and compiler options, Table 10 on page 83 shows the default bit mode and options that are set for each setting of the OBJECT_MODE environment variable.

Table 10. Default Compiler Bit Mode Determined by the OBJECT_MODE Setting

OBJECT_MODE Setting	Default Bit Mode	Default Option Set
Unset	32-bit	-q32
32	32-bit	-q32
64	64-bit	-q64
32_64	Not permitted	n/a

When you mix 32-bit and 64-bit compilation modes, you may not know if your object is XCOFF32 or XCOFF64 format. In other words, if you compiled and produced 64-bit objects, you need to remember to link these objects using the 64-bit mode, otherwise the objects will not link. Similarly, 32-bit objects must be linked using 32-bit mode. The only solution is to recompile completely making sure that all objects will be in the same mode, as explained in the following note.

Important Note

The use of *OBJECT_MODE* to determine the default bit mode can cause serious problems if a user is unaware of the setting of OBJECT_MODE. For example, the user may not be aware that OBJECT_MODE has been set to *64* and may unexpectedly obtain *64-bit object files*.

We strongly urge users to be aware of the setting of OBJECT_MODE at all times and to set OBJECT_MODE themselves to ensure that the compiler is invoked for the correct bit mode.

These are some examples of mixed-mode utilities:

1. To list all files in lib.a, whether 32-bit, 64-bit, or non-objects, enter:

   ```
   ar -X32_64 -t -v lib.a
   ```

2. To dump the object file headers from only 64-bit objects in lib.a, enter:

   ```
   dump -X64 -o lib.a
   ```

3. To display symbol of all 64-bit objects in libc.a, ignoring all 32-bit objects:

   ```
   nm -X64 /usr/lib/libc.a
   ```

4. To remove both the 32-bit and 64-bit symbol tables from lib.a, enter:

   ```
   strip -X 32_64 lib.a
   ```

5. To compile in 64-bit mode when you have an 32-bit environment:

   ```
   .
   export OBJECT_MODE=32
   .
   xlf -q64 myprog.f
   ```

   Notice that `-q64` overrides OBJECT_MODE setting.

Bit Mode Setting and Pragma Directive

A potential conflict exists if you compile a file with the following line in the source file:

```
#pragma options arch=601
main()
{
.....
}
```

If this file was compiled with the command line option:

```
xlc -q64 or xlc -qarch=ppc64
```

the resolution rule (the last one wins) will cause the source file to be compiled in 32-bit mode but the stanza file, xlc, will proceed to call the linker or the assembler in 64-bit mode, and they will fail.

For this reason, C for AIX compiler will not implement the `arch` option to the pragma directive to prevent a change of architecture after the compiler has been invoked.

5.1.4 64-bit Performance Benefits

The 64-bit address space can be used to dramatically improve the performance of applications that manipulate large amounts of data. This data can either be created within the application or obtained from files. Generally, the performance gain comes from the fact that the 64-bit application can contain the data in its address space (either created in data structures or mapped into memory), where the data would not fit into a 32-bit address space. Thus, the data would need to be multiple GBs in size or larger to show this benefit.

5.1.4.1 Program Size

If the same source code is used to create a 32-bit and a 64-bit application, the 64-bit application will usually be larger than the 32-bit application. The 64-bit application is not likely to run faster than the 32-bit application unless it makes use of the larger 64-bit addressability. Because most C programs are pointer-intensive, a 64-bit application will be almost twice as large, depending on how many pointers and longs are declared. A C++ program's text and data usage is almost always twice the size, due to the large number of pointers it uses under the covers to implement virtual function tables, objects, templates, and so on. The following example shows the size difference of compiling the same code (see 5.1.1.2, “Structure Alignment” on page 63) with the `-q64` option and without.

Example:

```
-rwxr-xr-x   1 root     staff       4191 Feb 26 09:59 sample32
-rwxr-xr-x   1 root     staff       4608 Feb 26 09:59 sample64
```

Generally, the correct choice is to create a 32-bit application, unless 64-bit addressability is required by the application or can be used to dramatically improve its performance.

5.1.4.2 Include Files

Many include files have pointers and structures in them, and their inclusion in 64-bit mode will change the size of your data section even if the application does not use structures and pointers explicitly.

5.1.4.3 Using Large Data

Programs with large data spaces require a large amount of paging space. For example, if a program with a 3 GB address space tries to access every page in its address space, the system must have 3 GB of paging space. The operating system page-space monitor terminates processes when paging space runs low.

64-bit applications may require paging I/O tuning in compensation for the large data handling benefit.

5.1.4.4 Using Large Memory Address

One of the 64-bit application benefits is mapping large data into real memory to access it without disk I/O. You can handle more than 2 GB data in real memory. On a 64-bit system, the hardware provides a continuous range of virtual memory addresses, from 0x0000 0000 0000 0000 0000 to 0xFFFF FFFF FFFF FFFF FFFF, for accessing data. The total addressable space is more than 1 TB. However, maximum hardware physical memory size is still 16 GB (as for the IBM RS/6000 Model S70). To prevent paging, it is important to keep memory use to no more than the physical amount.

5.1.4.5 Using Extended System Limits

System limits have been expanded on 64-bit AIX. This does not have a direct effect on performance, but offers new and expanded application solutions. For example, some network server type applications, using many socket file descriptors, will benefit from being able to handle more than 2,000 sessions.

Table 11. 64-Bit System Limits

Reference	Old Limit	New Limit
File descriptors per process[*1]	2,000	32,768
System open file table	200,000	1,000,000
Number of shared memory IDs	4096	65,536
Number of kernel threads per process[*1]	512	32,768
Number of processes	131,072	Not expanded
Number of threads per system	262,143	Not expanded
Size of Shared or Mapped Files	256 MB	2 GB
Executables		$7*2^{56}$ bytes[*2]

[*1] both 32-bit and 64-bit processes
[*2] $7*2^{56}$ bytes = 458,752 TB
[*3] 2^{64} = 16,777,216 T

Reference	Old Limit	New Limit
Sum of sizes of text, data and BSS sections		$7*2^{56}$ bytes[*2]
Symbol values (generally address of objects)		2^{64} [*3]

[*1] both 32-bit and 64-bit processes
[*2] $7*2^{56}$ bytes = 458,752 TB
[*3] 2^{64} = 16,777,216 T

5.2 Cache Line Size and Performance

The unit of access in the cache is called a line. In SMP environments, it is possible for two processes to reference two different portions of data that fall in the same cache line because they lie close to each other in memory. In a case where a process on processor 1 changes a value in cache, the cache consistency logic will invalidate the other processor's cache line, causing a cache miss when another value is accessed, even though the two processes were not sharing any data. This is called false sharing. False sharing increases cache misses and bus traffic, further reducing SMP throughput and scaling.

One solution for false sharing is to allocate padding between data.

Example:

```
struct critical_data{
long long_type_data;
char pad[124];
} critical_data[DATA_SIZE];
```

This sample is based on a data cache line of 128 bytes and the long type is 32-bit in length. You can see the size of data cache line with `_system_configuration.dcache_block` in sys/systemcfg.h. When migrating to 64-bit, you have to check the padding size for the right size. This size may not affect for application behavior but can affect application performance.

5.3 Data Sharing with 32-bit and 64-bit Applications

These samples show how to access large files and memory between coexisting 32-bit and 64-bit applications.

5.3.1 Creating Large Files

The open subroutine allows the creation of a file larger than 2 GB, if the _LARGE_FILES flag is defined. It is possible to create and open large files

greater than 2 GB in 32-bit applications. You can also can use the open64 or create64 subroutines. The following sample program works in 32-bit mode and 64-bit mode.

```
#include <stdio.h>
#include <fcntl.h>
#include <sys/errno.h>

#define ROOP_256  (16 * 1024 * 1024) /* 16 MB */
#define ROOP_3GB  (4 * 3)      /* total  3 GB */

void main(void) {
  int   fp64;
  int   seg_no;
  int   counter;
  char  data[17];

  if((fp64 = open64("/large_data/test.data", O_RDWR)) < 0 ) {
      printf("open64 error %d\n", errno);
      exit();
  }
  for (seg_no = 0; seg_no < ROOP_3GB; seg_no ++) {
      sprintf(data,"*%02d-256 MB area*",seg_no);/* 16 bytes data */
      printf("%s\n",data);
      for (counter = 0; counter < ROOP_256; counter ++) {
          if (write(fp64, data, 16) < 0) {
              printf("fprintf error %d\n",errno);
              close(fp64);
              exit();
          }
      }
  }
  close(fp64);
}
```

This sample program creates a 3 GB data file. It would be the same if written as part of a 32-bit application, the only difference being in the use of the open64 subroutine instead of the open subroutine (or fopen64 instead of fopen). The open64 subroutine is equivalent to the open subroutine, except that the O_LARGEFILE flag is set in the open file description associated with the returned file descriptor. The open64 subroutine does not return EOVERFLOW even if file size exceeds OFF_MAX (2^{32}-1 bytes or 2 GB -1). Instead, the file offset limit of DEV_OFF_MAX (2^{40} -1 bytes or 1 TB -1) is used.

The i-node mechanism limitations of AIX V4 prevent a file from growing beyond 68,589,453,312 bytes (64 GB - 124 MB).

Remember that files larger than 2 GB can only be created in a large file-enabled JFS. When creating the JFS, the bf option must be set to `bf=true`.

Using large files may make disk tuning an even more important performance consideration.

5.3.2 Using Large Memory Area

This sample program is reading 3 GB data from a large file and copies it to twelve shared memory segments.

```
#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#define  TEST_FILE  "/large_data/test.data"
#define  SHMEM_SIZE  256*1024*1024  /* 256 MB */

void main(void) {
  int     mem_id[16];
  char   *fp64_ptr;
  char   *sh_data[16];
  int     fd64;
  int     i, j;
  off64_t eof;

  if((fd64 = open64(TEST_FILE, O_RDWR)) < 0 ) {
      printf("open error %d\n", errno);
      exit();
  }
  eof = lseek64(fd64, 0, SEEK_END);
  printf("Data size = %x\n", eof);
  lseek64(fd64, 0, SEEK_SET);

  for (i = 0; i < 12 ; i++) {
      mem_id[i] = shmget((key_t)(0x20000 + i),SHMEM_SIZE,IPC_CREAT |
0x0666);
      if (mem_id[i] < 0) {
          printf("shmget[%d] error %d\n",i, errno);
          exit();
      }
      if ((sh_data[i] = shmat(mem_id[i],0,0)) == (char *)-1L) {
          printf("shmat[%d] error %d\n",i, errno);
          exit();
      }
```

```
        printf("shmat = %p ",sh_data[i]);
        read(fd64, sh_data[i], SHMEM_SIZE);
        printf("sh_data[%02d] = ",i);
        for (j = 0; j < 16; j++) {
            printf("%c",*(sh_data[i] + j) );
        }
        printf("\n");
    }
    close(fd64);
    for (i = 0; i < 12 ; i++) {
        shmdt(sh_data[i]);
    }
}
```

The program's output will be as follows:

```
Data size = c0000000
shmat = 700000000000000 sh_data[00] = *00-256 MB area*
shmat = 700000010000000 sh_data[01] = *01-256 MB area*
shmat = 700000020000000 sh_data[02] = *02-256 MB area*
shmat = 700000030000000 sh_data[03] = *03-256 MB area*
shmat = 700000040000000 sh_data[04] = *04-256 MB area*
shmat = 700000050000000 sh_data[05] = *05-256 MB area*
shmat = 700000060000000 sh_data[06] = *06-256 MB area*
shmat = 700000070000000 sh_data[07] = *07-256 MB area*
shmat = 700000080000000 sh_data[08] = *08-256 MB area*
shmat = 700000090000000 sh_data[09] = *09-256 MB area*
shmat = 7000000a0000000 sh_data[10] = *10-256 MB area*
shmat = 7000000b0000000 sh_data[11] = *11-256 MB area*
```

The following `ipcs` command shows shared memory information:

```
# ipcs -mb
IPC status from /dev/mem as of Wed Feb 25 16:30:36 CST 1998
T     ID     KEY        MODE       OWNER    GROUP     SEGSZ
Shared Memory:
m      0 0x0d076951 --rw-rw-rw-     root   system      1440
m15503361 0x00020000 -----r--rw-     root   system 268435456
m  94210 0x00020001 -----r--rw-     root   system 268435456
m  90115 0x00020002 -----r--rw-     root   system 268435456
m  73732 0x00020003 -----r--rw-     root   system 268435456
m  61445 0x00020004 -----r--rw-     root   system 268435456
m  61446 0x00020005 -----r--rw-     root   system 268435456
m  61447 0x00020006 -----r--rw-     root   system 268435456
m  61448 0x00020007 -----r--rw-     root   system 268435456
m  61449 0x00020008 -----r--rw-     root   system 268435456
m  61450 0x00020009 -----r--rw-     root   system 268435456
m  61451 0x0002000a -----r--rw-     root   system 268435456
```

```
m  61452 0x0002000b -----r--rw-     root   system 268435456
```

You can see that there are twelve shared memory IDs; each is 256 MB, with a 3 GB memory area from virtual address 0x0700 0000 0000 0000 to 0x0700 0000 BFFF FFFF.

You can also read data from memory using a 64-bit pointer:

```
for (i = 0; i < 12 ; i++) {
      read(fd64, sh_data[0] + (long)SHMEM_SIZE * (long)i, SHMEM_SIZE);
  }
```

The `read` subroutine mechanism does the following:

1. Assures that the FileDescriptor parameter is valid and that the process has read permissions. The subroutine then gets the file table entry specified by the FileDescriptor parameter.
2. Sets a flag in the file to indicate a read operation is in progress. This locks other processes out of the file during the operation.
3. Converts the offset byte value and the value of the NBytes variables into a block address.
4. Transfers the contents of the identified block into a storage buffer.
5. Copies the contents of the storage buffer into the area designated by the Buffer variable.
6. Updates the current offset according to the number of bytes actually read. Resetting the offset assures that the data is read in sequence by the next read process.
7. Deducts the number of bytes read from the total specified in the NByte variable.
8. Loops until the number of bytes to be read is satisfied.
9. Returns the total number of bytes read.

The cycle completes when the file to be read is empty, the number of bytes requested is met, or a reading error is encountered during the process. Errors can occur while the file is being read from disk or in copying the data to the system file space. That is why the maximum data the `read` subroutine can handle at any time is 2 GB -1. Thus, the sample program cannot be modified as follows:

```
read(fd64, sh_data[0], SHMEM_SIZE * 12);
```

If you do this, the `read` subroutine will return an error with EFAULT (bad address), because the conversion of the offset byte value into a block address will fail. Remember that the AIX kernel is still 32-bit.

During the read operation, the i-node is locked. No other processes are allowed to modify the contents of the file while a read is in progress. However the file is unlocked immediately on completion of the read operation. If another process changes the file between two read operations, the resulting data is different, but the integrity of the data structure is maintained. That is why the read operations are not managed in a strict parallelism, although they are executed on a SMP system. You have to consider the size of the data to be read, especially if it is referred to by many processes.

The `write` subroutine functions much like the read subroutine. The byte offset for the write operation is found in the system file table's current offset.

Sometimes when you write to a file it does not contain a block corresponding to the byte offset, resulting from the write process. When this happens, the `write` subroutine allocates a new block. This new block is added to the i-node information that defines the file. If adding the new block produces an indirect block position (`i_rindirect`), the subroutine allocates more than one block when a file moves from direct to indirect geometry.

During the write operation, the i-node is also locked. No other processes are allowed to modify the contents of the file while a write is in progress. However the file is unlocked immediately on completion of the write operation. If another process changes the file between two write operations, the resulting data is different, but the integrity of the data structure is maintained.

It is not guaranteed that all shared memory addresses are continuous. It is better to specify the address with the `shmat` subroutine.

```
if ((sh_data[i] = shmat(mem_id[i]
                     ,(const void *)(0x7F0000000000000L + 0x10000000L * i)
                       ,0))  == (char *)-1L) {
/* Do error handling */
}
if (sh_data[i] != (char *)(0x7F0000000000000L + 0x10000000L * i) {
/* Do something recovery */
}
```

The segment number for `shmap`/`mmap` is from 0x7000 0000 to 0x7FFF FFFF. You should ensure that the offset value is suffixed as a long integer type, otherwise you will see an unsuffixed number problem. This problem is discussed in 6.3.3, “Code and Data Analysis” on page 103.

5.3.3 Read Data with 32-bit Application

The following sample program has five threads, one is the *initial* thread, and the other four threads write shared memory data to files. Reading shared

memory created by a 64-bit application is no different from 32-bit applications.

```
#include <pthread.h>
#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#define  SHMEM_SIZE  256*1024*1024  /* 256 MB */
#define  SH_MAX      12
#define  MAX_NAME    64
void *read_th(void *);
int   mem_id[SH_MAX];
char *sh_data[SH_MAX];

void main(void) {
    pthread_attr_t  attr;
    pthread_t       th_read[SH_MAX];
    int             i,j,rc;

    if ((rc = pthread_attr_init(&attr)) != 0) {
        printf("pthread_attr_init error, rc = %d\n", rc);
        exit();
    }
    rc = pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
    if (rc != 0) {
        printf("pthread_setdetachstate error, rc = %d\n", rc);
        exit();
    }
    for (i = 0; i < SH_MAX; i +=3) {
        if ((rc = pthread_create(&th_read[i],&attr,read_th,(void*)i))!=0) {
            printf("pthread_create(sig_th) error, errno = %d\n" , errno);
        }
    }
    for (i = 0; i < SH_MAX; i +=3 )  {
        pthread_join(th_read[i], NULL);
    }
    for (i = 0; i < SH_MAX; i +=3 )  {
        printf("shmat = %p ",sh_data[i]);
        printf("sh_data[%02d] = ",i);
        for (j = 0; j < 16; j++) {
          printf("%c",*(sh_data[i] + j) );
        }
        printf("\n");
        shmdt(sh_data[i]);
    }
```

```
    pthread_attr_destroy(&attr);
    printf("Normal end\n");
    exit();
}                                   /* End of Main                        */

void  *read_th(void *i)  {
    extern int   mem_id[SH_MAX];
    extern char *sh_data[SH_MAX];
    int          th_no;
    int          fd;
    char         fname[MAX_NAME +1];

    th_no = (int)i;
    bzero(fname, MAX_NAME +1);
    sprintf(fname, "/save_data/test.data%02d", th_no);
    if((fd = open(fname, O_RDWR | O_CREAT)) < 0 ) {
      printf("open error %s\n", fname);
      pthread_exit(NULL);
    }
    mem_id[th_no] = shmget((key_t)(0x20000 +
(int)th_no),SHMEM_SIZE,IPC_EXCL);
    if (mem_id[th_no] < 0) {
        printf("shmget[%d] error \n",th_no);
        pthread_exit(NULL);
    }
    if ((sh_data[th_no] = shmat(mem_id[th_no],0,0)) == (char *)-1L) {
        printf("shmat[%d] error %d\n",th_no, errno);
        pthread_exit(NULL);
    }
    printf("thread = %d\n",th_no);
    printf("thread = %d\n",th_no);
    if (write(fd, sh_data[th_no], SHMEM_SIZE)  < 0) {
        printf("write error\n");
    }
    close(fd);
    pthread_exit(NULL);
}                       /* End of read_th     */
```

The output of this program is as follows:

```
thread = 0
thread = 6
thread = 3
thread = 9
shmat = 30000000 sh_data[00] = *00-256 MB area*
shmat = 40000000 sh_data[03] = *03-256 MB area*
shmat = 50000000 sh_data[06] = *06-256 MB area*
```

```
shmat = 60000000 sh_data[09] = *09-256 MB area*
Normal end
```

The order of the threads is sometime different because of the thread scheduling mechanism. The sample output shows that the third thread executed before the second thread. If you want to ensure strict ordering, use a synchronized mechanism, such as the `pthread_cond_wait` subroutine.

Notice the value of shared memory address is four bytes in length. This sample has a 1 GB memory address from 0x3000 0000 to 0x6FFF FFFF.

The sample has some critical performance issues. It will manage logical parallel write operations, but not physical disk I/O operations, because all files are in the same file system. The file layout will also impact performance.

5.4 References

The following are good sources for additional information.

- The RS/6000 64-Bit Solution

  `http://www.rs6000.ibm.com/resource/technology/64bit6.html`

- AIX V4.3 Migration Guide

  `http://w3dev.austin.ibm.com/library/aix4.3/`

- *AIX Version 4 General Programming Concepts: Writing and Debugging Programs*, SC23-2533

Chapter 6. Migration Techniques

This chapter explains how to migrate your 32-bit application to 64-bit.

The most serious challenge comes from 32-bit applications that store internal data structures directly into their programs. Although this issue is simple to solve, code that makes assumptions about the size of native integer storage must be fixed. For example, code that casts or converts memory pointers to integers must be checked and corrected. While less prevalent, code must be checked to insure that any shifting and masking operations that manipulate long integers still work properly with a 64-bit long.

One significant migration task is dealing with input and output file dependencies. This is relevant when you migrate an application that is in the middle of a pipeline of applications, where each application reads the previous application's output as input, and then passes its output to the next application in the pipe. Before migrating one of these applications to 64-bit, you must verify that the output will not produce values outside of the 32-bit range. Typically, once an application is ported to 64-bit, all downstream applications (that is, any application that depends on output from the ported application) have to be ported to 64-bit.

Porting to 64-bit involves well-known technical issues that average programmers can master.

The migration process has three steps:

1. Component analysis
2. Port code to 64-bit environment
3. Package 64-bit solution

The first step is to decide which component of your application should be used on 64-bit. You have already seen how to determine this in Chapter 2, "Why Do I Need 64-bit?" on page 13. The second step is to modify the program such that it has the same behavior on 64-bit. The last step is to incorporate additional functions or new application limitation for 64-bit.

6.1 Migration to AIX V4.3

When planning to migrate an application to 64-bit, you must also consider migration to the AIX V4.3 operating system.

6.1.1 AIX V4.3 Changes

There are many changes from AIX V4.2 to AIX V4.3, and some of them may affect your code.

- The *time_t type* has changed from long type in AIX 4.2 to int type in AIX 4.3.
- MB_CUR_MAX has changed from int to size_t in AIX 4.3.
- Calling `setlocale()` in 64-bit mode will not find the user-defined locale file.
- Make doesn't discriminate on object formats but on time stamps of files.
- The new variable name int64 is type-defined in inttypes.h.
- New header file predefined types that are based on long.

See *"Operating System-specific issues that may affect your code"* in the */usr/vac/README.C* file, or Appendix E.2, “README: C for AIX” on page 172 for more information.

6.2 32-bit to 64-bit Application Migration Planning

There are two reasons for performance improvements using 64-bit application. The system call overhead of reading and writing files can be avoided by mapping the files into memory. Moreover, 64-bit systems can support a physical memory that is larger than the addressability of 32-bit applications, so 64-bit applications are needed to make full use of the physical memory available.

6.2.1 Limitation

There is no way to create an object or application using 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.

6.2.2 32-bit and 64-bit Application

There are two archive file types in AIX. The first one does not recognize 64-bit object files and cannot be larger than 2 GB. The second archive file type recognizes both 32-bit and 64-bit object files and will work with files larger than 2 GB. The second type can be produced using AIX V4.3 on any of the 32-bit processor models, but can be run on the 64-bit processor models.

6.2.3 32-bit Binary Compatibility on 64-bit Hardware

Kernel exploitation of hardware binary compatibility also helps AIX V4.3 provide binary compatibility for well-behaved 32-bit device drivers and kernel

extensions, running on 64-bit systems with 32-bit application workloads. There are a few situations where an existing device driver or kernel extension must be enhanced to handle direct interaction with a 64-bit application, but this can be done in ways that do not impact its compatibility for 32-bit applications.

Under the RS/6000 philosophy of not forcing unnecessary recompilation to 64-bit, the large majority of application programs are likely to remain 32-bit and only a small number of key programs scale up to exploit the performance benefits of a 64-bit address space. 32-bit binary compatibility in AIX V4.3, on both 32-bit and 64-bit systems, is an important ingredient for building the smooth evolutionary road to 64-bit computing.

6.2.4 64-bit and 32-bit Interoperability Requirements

32-bit binary compatibility on 64-bit systems alone may not be sufficient. It is not enough to just say that 32-bit programs will continue to run on the same system where 64-bit programs also run. Serious investment protection means being able to leave 32-bit programs alone, even when another program they cooperate with is changed to exploit a 64-bit address space.

A simple example of interoperability is provided by the sophisticated interaction between processes in a transaction processing environment. Client processes provide end user transaction function, calling on the database management processes for reading or updating database objects. The database manager program may become a 64-bit application process in order to expand its buffer space into very large memory. The client programs that call the database manager should continue to run as 32-bit programs, with binary compatibility, and no requirement to recompile because the database manager changed. The operating system's role in this example is to provide the infrastructure by which this goal is easily achieved by the middleware applications.

The open systems standards do not define interoperability requirements between 32-bit and 64-bit programming environments. The extensive interoperability requirements between 32-bit and 64-bit programming environments are seen as equals, differentiated only by their address space size and data model size. The corresponding model for 32-bit programming is ILP32, that is, the types int, long, and pointer are each 32-bit quantities. The model for 64-bit, LP64 data model, is different only in its size of long and pointer. These extensive interoperability requirements makes it a design requirement in AIX V4.3, so they cooperate and interoperate as well as homogeneous processes.

64-bit and 32-bit interoperability refers to the two programming environments that AIX V4.3 provides on 64-bit systems, where 64-bit and 32-bit processes coexist, interact, and share common resources. This interoperability is actually present at several levels:

- They can share system resources, files, shared memory, and Inter-Process Communication (IPC) resources, and can signal each other.
- They can execute each other by using the exec subroutine, without concern or knowledge that the process being created will be 32-bit or 64-bit.
- They can set process resource limits that will apply to a process of the opposite type. The extensions in this area, for 32-bit processes to set 64-bit limits, are modeled upon comparable extensions provided for large file support in a 32-bit environment.

6.2.5 System Limits

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

6.2.6 The 64-bit Migration Procedure

The following list shows how to migrate to 64-bit. This procedure does not mean application tuning or extending functions; just the steps for the same behavior on 64-bit mode are explained. You may have to change code for using expanded limits after this procedure.

1. Code and data analysis

   Examine all types to determine whether the types should be 32-bit or 64-bit. For system types, the type will be the appropriate size for use with library/system calls. For user-defined types, 32-bit types should be defined based upon int or unsigned int or some system type that is 32 bits long in 64-bit mode. For user-defined types, 64-bit types should be defined based upon long or unsigned long or some system type that is 64 bits long.

2. Modify data types

   Change all types to the chosen type. When doing so, examine all arithmetic calculations to make sure that expansion and truncation of data values is done appropriately. Make sure that no assumption is made that pointer values will fit into integer types.

3. Verify other program's output usage

Make sure that all output produced is contained in the 32-bit range. If this is not possible, then any other application using this data needs to be ported to 64-bit or at least be made 64-bit aware.

4. Remove dependency location

   Remove any dependencies on the locations (relative and absolute) of the application text, application data, application heap, application stack value, errno, tok_of_stack structure, shared library data, shared library text, and SVC tables.

5. Bad address usage

   64-bit processes may now receive the signal SIGSEGV (segmentation violation) rather than an error of type EFAULT (errno 14 Bad address) when passing a bad address to a system call. Any dependency on system call protection of bad address usage should be removed.

6. Debug and test

   Test 64-bit code and confirm as same behavior. If you see any difference, debug the code and go to step 1.

6.3 64-bit Corresponding Commands

The commands `yacc`, `lex`, and `lint` work with source code destined for both 32-bit and 64-bit objects. The commands `make`, `ar`, `strip`, `dump`, `nm`, `prof`, `gprof`, `ld`, `ranlib`, `size`, `strings`, and `sum` work with both 32-bit and 64-bit objects and applications. The commands `dbx` and `xldb` allow the debugging of both 32-bit and 64-bit applications.

6.3.1 New Crash Commands

The following are new commands for the crash debugger.

sr64
Usage: `sr64 [-p pid] [-l limit] [-n] [start_esid [end_esid]]`

The command `sr64` displays esids (effective segment IDs, also known as segment register number) and their corresponding segvals (segment register contents). In absence of the `-p` flag, `sr64` will use the current process. It will list all entries in the `adspace` unless a starting esid (and possibly ending esid) is given.

It will stop listing if the number of entries specified by the `-l` flag have been printed. Since an `adspace_t` (address space mapping struct) holds 16 entries (contents of all segment registers), each line will consist of an esid, its

corresponding value, and the three subsequent values following it in the adspace_t.

This is done to optimize screen usage and to keep consistency with the kernel debugger. The -n flag also prints the uadnodes for the displayed data. The data structure that represents the mapping from a 64-bit address to an srval is a tree. the tree has radix-4 (that is, each non-leaf node may have up to 16 children). The leafs of the tree contain adspace_t's (that is, the srvals for 16 segments). In general, the tree is unbalanced, with the maximum number of nodes between the root and a leaf equal to 8 (for a 2^{60} byte address space). struct uadnode is the declaration for a node in the tree (uadnode entries are not included in the count when limiting the data via -l).

Example:

```
# crash
> p |grep sample
 43 a    2ba6   3fdc   2ba6     0     0    42  sample
> sr64 -p 43 |pg
    SegReg Num :   Values (* = segment register is allocated)
    0x000000000:  *0x60000000   0x007fffff  *0x60038f4e   0x007fffff
    0x000000004:   0x007fffff   0x007fffff   0x007fffff   0x007fffff
    ......
    0x080020010:   0x007fffff   0x007fffff   0x007fffff   0x007fffff
    0x080020014:  *0x60038cae   0x007fffff   0x007fffff   0x007fffff
    ......
    0x090000000:  *0x60010184   0x007fffff   0x007fffff   0x007fffff
    0x090000004:   0x007fffff   0x007fffff   0x007fffff   0x007fffff
    ......
    0x0fffffff0:  *0x60074c1d  *0x60004f41  *0x60014f45  *0x60040f10
    0x0fffffff4:  *0x60040f50  *0x60030f4c  *0x6004cf13  *0x6002cf4b
```

segst64

Usage: segst64 [-p pid] [-l limit] [-s segflag[:value][,segflag[:value]]...] [-n] [start_esid [end_esid]]

The command segst64 displays segstate info. The segstate structure is used by the shared memory services to mark the usage of allocated segments. The segstate for the current process is displayed unless the -p flag is given. All of the segstate entries are displayed unless limited by the -l flag or the starting and possibly ending esid parameters.

Example:

```
> segst64 -p 43 |pg
    ESID       segstate segflag     num_segs fno/shmp/srval/nsegs
    SR00000003 [ 0]     AVAILABLE   00000000 0000000a
```

```
SR0000000d [ 1]     OTHER       00000001 00000001
SR0000000e [ 2]     AVAILABLE   00000000 00000001
SR0000000f [ 3]     OTHER       00000001 00000001
SR00000010 [ 4]     TEXT        00000001 00000001
SR00000011 [ 5]     WORKING     00000001 00000000
SR00000012 [ 6]     WORKING     00000001 00000000
......
SR00000038 [12]     WORKING     00000001 00000000
SR00000039 [13]     WORKING     00000001 00000000
SR0000003a [14]     AVAILABLE   00000000 8001ffda
SR80020014 [15]     WORKING     00000001 00000000
SR80020015 [ 0]     AVAILABLE   00000000 0ffdffea

SR8fffffff [ 1]     WORKING     00000001 00000000
```

6.3.2 Header File Changes

The header file `types.h` includes three new types for easy migration of header files:`__long32_t`, `__ulong32_t`, and `ptr64`. When an application is compiled in 32-bit mode, `__long32_t` is of type `long`, and `__ulong32_t` is of type `ulong`. For applications compiled in 64-bit mode, `__long32_t` is of type `int`, and `__ulong32_t` is of type `uint`. Thus, the new types `__long32_t` and `__ulong32_t` remain size invariant at four bytes.

Shipped header files that can be used by both 32-bit and 64-bit applications and contain structures for entities that must remain size invariant between 32-bit and 64-bit, must make these two modifications:

- Replace all fields of type `long` with type `__long32_t` or some other 32-bit type. Similarly replace all fields of type `ulong` with type `__ulong32_t` or some other 32-bit type.
- Replace all pointers of type `int` or some other 32-bit type. This change only applies if you can not let a 64-bit application use a 64-bit pointer for this field, which should not be the case for most structures.

6.3.3 Code and Data Analysis

In 64-bit mode, the size of long and pointer types is changed. You have to check application behavior, especially if the logic depends on data size.

long, int

The types long and int are not interchangeable. The C long type (and types derived from it) in 64-bit mode is 64 bits in size. You should consider all types related to the `long` and `unsigned long` types. For example, `size_t`, used in many subroutines, is typedef as unsigned long.

Unsuffixed Number

A number like 4294967295 (`UINT_MAX`), when parsed by the compiler, will be a typed `signed long` in 32-bit mode. This is appropriate since `signed long` is four bytes and mixing `long` with `int` is usually allowed in 32-bit mode.

In 64-bit mode, this same number will choose `signed long`, which is eight bytes. This causes some operations, such as comparing the `sizeof(4294967295)`, to return 8. The fix for the above case is to write the number as 4294967295U. This will allow the compiler to pick unsigned `int`.

Unsuffixed constants are more likely to become 64-bit long if they are in hexadecimal. When porting code, this needs to be remembered. All constants that have the potential of impacting constant assignment should be explicitly suffixed. Using suffixes and explicit types with all numbers will save the application from unexpected behavior.

Example:

```
#include <stdio.h>
void main(void) {
  long l = LONG_MAX;
  printf("size(2147483647)  = %d\n",sizeof(2147483647));
  printf("size(2147483648)  = %d\n",sizeof(2147483648));
  printf("size(4294967295U) = %d\n",sizeof(4294967295U));
  printf("size(-1)  = %d\n",sizeof(-1));
  printf("size(-1L) = %d\n",sizeof(-1L));
  printf("LONG_MAX = %d\n",l);
}
```

32-bit

```
size(2147483647)  = 4
size(2147483648)  = 4
size(4294967295U) = 4
size(-1)  = 4
size(-1L) = 4
LONG_MAX = 2147483647
```

64-bit

```
size(2147483647)  = 4
size(2147483648)  = 8
size(4294967295U) = 4
size(-1)  = 4
size(-1L) = 8
LONG_MAX = -1
```

The sample 64-bit output LONG_MAX is not really -1. The reason for the -1 is that the printf subroutine handles it as an integer. Keep in mind that the C language limit (sys/limits.h) has also changed.

```
#ifdef __64BIT__
#define LONG_MAX     (9223372036854775807)
#define LONG_MIN     (-LONG_MAX - 1)
#define ULONG_MAX    (18446744073709551615)
#else /* __64BIT__ */
#define LONG_MAX        INT_MAX
#define LONG_MIN        INT_MIN
#define ULONG_MAX    (UINT_MAX)
#endif /* __64BIT__ */
```

In 64-bit mode, (LONG_MAX == (int)LONG_MAX) returns a negative value.

pointer, int

In 32-bit mode, int, long and pointer types have the same size and can be freely assigned into each other. In extended mode, integer and long types can be assigned into pointer types and vice versa with only a warning. In ANSI mode, assignment between integral and pointer types will generate a service-level message.

In 64-bit mode, all pointer types are 64 bits in size. Exchanging pointers and int types can cause segmentation faults, and passing pointers to a function expecting an int type will result in truncation.

The following is an example of an incorrect assignment:

```
int i;
int *p;
i = (int)p;
```

Using cast makes the problem harder to detect. The warning message will disappear but the problem still exists. The following steps should be taken:

- Remove any assumption that a pointer type can fit in a C integer type (or types derived from an integer).
- Remove any assumption that a C long type can fit in a C integer type (or types derived from an integer).
- Remove any assumption about the number of bits in a C long type object when bit shifting or doing bitwise operations.
- Remove any assumption that a C integer can be passed to an unprototyped long or pointer parameter.

- Remove any assumption that an unprototype function can return a pointer or long.

6.3.4 The New lint Option -t

The command `lint -t` checks for problematic assignments when porting from 32- to 64-bit. The `lint -t` command 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

  This type of assignment 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.

Example:

```
 +1  #include <stdio.h>
 +2  void main(void) {
 +3    int    foo_i;
 +4    long   foo_l;
 +5    int   *foo_pt;
 +6
 +7    foo_l  = boo(1);
 +8    foo_l  = foo_l << 1;
 +9    foo_l  = 0xFFFFFFFF;
+10    foo_l  = (foo_l & 0xFFFFFFFF);
+11    foo_l  = LONG_MAX;
+12    foo_l  = (long)foo_i;
+13    foo_i  = (int) &foo_l;
+14    foo_pt = (int *)foo_i;
+15  }
+16  long boo(long boo_l) {
+17    return(boo_l);
```

```
+18  }
# lint -t sample.c
"sample.c", line 7: warning: conversion from "int" may lose accuracy
"sample.c", line 8: warning: Left Shift involving a "long"
"sample.c", line 10: warning: bitwise " AND " involving a "long"
"sample.c", line 12: warning: conversion from "int" may lose accuracy
"sample.c", line 13: warning: conversion from "PTR long" may lose accuracy
"sample.c", line 16: error: illegal redeclaration of boo
"sample.c", line 17: warning: conversion from "long" may lose accuracy
```

This sample will be compiled with no error with the cc command. Notice line 9,11 and 14 are also affected for 64-bit porting. There are a few other problems that `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.

  1. Example:

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

  2. Example:

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

There are two other differences that can affect programs:

- In 32-bit mode, long long's are passed in two general purpose registers while 64-bit mode long long's are passed in only one general purpose register.
- In 32-bit mode, when a function is passed a floating-point argument(s) 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.

6.3.5 The Compiler Option -qwarn64

The compiler option, `-qwarn64`, checks for possible long-to-integer truncation. This option functions in either 32- or 64-bit compiler modes. In 32-bit mode, it functions as a preview aid to discover possible 32- to 64-bit migration problems.

Informational messages are displayed where data conversion may cause problems. In 64-bit compiler mode, the `-qwarn64` option checks the following common problems:

- Truncation due to explicit or implicit conversion of long types into int types.
- Unexpected results due to explicit or implicit conversion of int types into long types.
- Invalid memory references due to explicit conversion by cast operations of pointer types into types.
- Invalid memory references due to explicit conversion by cast operations of int types into pointer types.
- Problems due to explicit or implicit conversion of constants into long types.
- Problems due to explicit or implicit conversion by cast operations of constants into pointer types.
- Conflicts with pragma options arch in source files and on the command line.

Example:

```
# /usr/vac/bin/cc sample.c -qwarn64
"sample.c", line 9.12: 1506-743 (I) 64-bit portability: possible change of
result through conversion of int type into long type.
"sample.c", line 12.12: 1506-743 (I) 64-bit portability: possible change of
result through conversion of int type into long type.
"sample.c", line 13.12: 1506-744 (I) 64-bit portability: possible
truncation of pointer through conversion of pointer type into int type.
"sample.c", line 14.12: 1506-745 (I) 64-bit portability: possible incorrect
pointer through conversion of int type into pointer.
```

This sample used the same code as `lint -t`. The `-qwarn64` option also checked unsuffixed numbers and conversions of `int` type into `pointer`. Lines 9 and 14 cannot be checked correctly with only the `lint -t` command.

Using 64-bit compiler mode invokes an even more detailed portability check. The last portability problem of the sample code in line 11 is also checked.

Example:

```
# /usr/vac/bin/cc sample.c -qwarn64 -q64
```

```
"sample.c", line 7.15: 1506-743 (I) 64-bit portability: possible change of
result through conversion of int type into long type.
"sample.c", line 9.12: 1506-743 (I) 64-bit portability: possible change of
result through conversion of int type into long type.
"sample.c", line 10.21: 1506-743 (I) 64-bit portability: possible change of
result through conversion of int type into long type.
"sample.c", line 11.12: 1506-748 (I) 64-bit portability: constant which
will overflow in 32-bit mode may select unsigned long int or long int in
64-bit mode
"sample.c", line 12.12: 1506-743 (I) 64-bit portability: possible change of
result through conversion of int type into long type.
"sample.c", line 13.12: 1506-744 (I) 64-bit portability: possible
truncation of pointer through conversion of pointer type into int type.
"sample.c", line 14.12: 1506-745 (I) 64-bit portability: possible incorrect
pointer through conversion of int type into pointer.
"sample.c", line 16.6: 1506-343 (S) Redeclaration of boo differs from
previous declaration on line 7 of "sample.c".
"sample.c", line 16.6: 1506-050 (I) Return type "long" in redeclaration is
not compatible with the previous return type "int".
```

Note

The `-qwarn64` option does not check shift operation. However, using both `lint -t` command and `-qwarn64` compiler options, you can perform extensive code analysis.

6.3.6 Superseded Libraries

Some libraries that have been superseded or deprecated for 32-bit applications are not being provided to 64-bit applications, so their APIs will be missing in 64-bit execution mode.

6.3.7 64-bit to 32-bit Data Reformatting

The 64-bit interface makes it appear to the kernel that the system service request came from a 32-bit program. To this end, any data passed across the interface in either direction must be adjusted in width to match the size expected.

See the *AIX Version 4.3 Difference Guide*, SG24-2014 for more information.

6.3.8 APIs Not Moved to 64-bit

The libraries and APIs listed below will not be supported in the 64-bit environment. Note that this section is to cover user-level APIs. Kernel, kernel extension and device driver APIs are not discussed in this section.

Note

If an API set is not listed, assume that it supports dual versions.

The following libraries do not have 64-bit versions:

lib300.a	Obsolete ASCII graphing library
lib300s.a	Obsolete ASCII graphing library
lib4014.a	Obsolete ASCII graphing library
lib450.a	Obsolete ASCII graphing library
libIN.a	Interactive systems library from old RT
libPW.a	Obsolete programmer's workbench library
libcur.a	Obsolete IBM-invented curses extensions
libplot.a	Obsolete ASCII graphing library
libbsd.a	Nonstandard BSD APIs (others are in libc.a)
libasl.a	ASCII interface library for SMIT
libvsm.a	GUI library for VSM
libi18n.a	Layout object library for internationalization
libnck.a	NCS library
libnetls_shr.a	Obsolete licence management library
libtermcap.a	terminfo interface library

All CDE libraries

The following API sets do not have 64-bit versions:

- All of the back level X11 compatibility versions
- All of the back level libcurses versions

The list of services that will not be made available to 64-bit applications until a later release include:

- Support for creating executables and libraries larger than 256 MB
- Support for more than one stack segment
- Support for page-level `shmat` granularity for 64-bit processes
- Printer device drivers (working with 64-bit processes)
- OpenGL

6.3.9 Application-Specific Porting Issues

Some types applications need specific consideration.

6.3.9.1 Asynchronous I/O

Asynchronous I/O (AIO) has been enhanced to support 64-bit enabled applications. On 64-bit platforms, both 32-bit and 64-bit AIO can occur simultaneously.

The struct aiocb, fundamental data structure associated with all asynchronous I/O operation, has changed. The element of this struct, aio_return, is now defined as ssize_t. Previously, it was defined as an int.

AIO supports large files by default. An application compiled in 64-bit mode can do AIO to a large file without any additional #defines or special opening of those files.

6.3.9.2 X11R6/Motif 2.1

AIX 4.3 supports X11R6 and Motif 2.1, providing a thread-safe stack and 64-bit libraries to Motif applications. 64-bit versions of the system libraries are provided in order to build 64-bit Motif or X programs.

In AIX 4.3, the X Server is 32-bit. The 32-bit and 64-bit libraries connect to the X Server by formatting the data into the appropriate data structures which the X Server understands. There are no problems in displaying either 32-bit or 64-bit applications on a 32-bit X Server.

See *AIX Version 4.3 Differences Guide*, SG24-2014 to get more information about X11R6.

6.3.9.3 Locales (ILS)

Locale objects are loadable objects. AIX 4.3 provides locale support for 64-bit applications. 64-bit input methods are supported for C locales only. Also, 32-bit loadable objects can not be used in 64-bit environments and vice versa.

AIX 4.3 provides dual objects for locales. 64-bit objects have __64 appended to the normal name. The setlocale subroutine is responsible for loading the correct object. From a normal user perspective however, locale support will function as expected. For example, if the user is using the "en_US" locale, then The setlocale subroutine will automatically load the 32-bit en_US locale object when running 32-bit applications and will load the 64-bit en_US locale object when running a 64-bit application.

A new command, locale64, has been added. Additionally, the locale command can be used with the "-O 64" option added to it to exec the locale64 command. The locale64 command has the same functionality as the standard locale command. However, it is compiled as 64-bit application so that locale64 (or locale -O 64) can be used to query locale status for 64-bit applications. In general, locale data should be the same in both 32-bit and 64-bit applications.

Note

64-bit applications will use different locale objects than 32-bit applications.

User or vendor-created 32-bit locale objects will not work in 64-bit applications unless they recompile and reship their locales using AIX V4.3. The `localedef` command has been modified in AIX V4.3 to create the 64-bit locale object at the same time the 32 bit locale object is created. For example, `localedef -i foo.src foo` will create locale objects `foo` and `foo__64` on AIX V4.3, where `foo` is the 32-bit locale object, and `foo__64` is the 64-bit locale object.

6.4 Thread Considerations

AIX V4.3 does not distinguish the 64-bit execution environment by making new functionality exclusively available to 64-bit applications, except functionality from having a large address space. In general, all new functionality is available to both 32-bit and 64-bit processes.

6.4.1 Thread Programming and Performance

Multithreaded programming is not only intended to improve application response time. Threads share the file descriptor table, signal handlers, and memory. Using multi-threads instead of many processes will require less system resources and lead to better performance. Also, creating a thread requires less overhead than creating process with the `fork` subroutine. Creating a thread only requires the allocation of the thread's private data area, usually 64 KB, and two system calls. Creating a process is far more expensive, because the entire parent process addressing space is duplicated.

6.4.2 New Pthreads

The current 1:1 threads implementation has been replaced with an M:N version. This provides increased performance for several types of database applications, as well as greater per-process limits for applications with large number of threads, such as databases.

Existing AIX pthread applications are based on industry standard POSIX 1003.1c Draft 7. AIX V4.3 supports both Draft 7 and Draft 10 in 32-bit mode. However, only Draft 10 is supported in 64-bit mode. Porting pthread

applications to 64-bit will have to convert to Draft 10. Default pthread is Draft 10. To invoke to compile with Draft 7 pthread, use libpthreads_compat.a.

```
xlc_r -o sample sample.c -D_AIX_PTHREADS_D7 -lpthreads_compat
```

It is impossible to use this with the `-q64` option, because Draft 7 does not support 64-bit mode. If you use it anyway, the loader will return an error message.

```
ld: 0711-345 Use the -bloadmap or -bnoquiet option to obtain more
information.:
```

The command `genkld` will show threads-shared modules.

```
# genkld |grep thread
        Virtual Address     Size               File
        d02020f8            14f9        /usr/lib/libpthreads_compat.a/shr.o
        d021c980           16a0d         /usr/lib/libpthreads.a/shr_xpg5.o
        d021a5e8            1082         /usr/lib/libpthreads.a/shr_comm.o
        d0204100           15992         /usr/lib/libpthreads.a/shr.o
       9000000002ba980    29aa7        /usr/lib/libpthreads.a/shr_xpg5_64.o
        900000000000520   2b9851         /usr/lib/threads/libc.a/shr_64.o
```

The libpthreads_compat.a/shr.o is 32-bit Draft 7-shared modules. The libpthreads.a/shr.o is a 32-bit Draft 7 LOADONLY-shared module for binary compatibility. The shr_xpg5.o is 32-bit Draft 10, and shr_xpg5_64.o is 64-bit Draft 10. The /usr/lib/thread/libc.a is a symbolic linked to /usr/ccs/lib/libc.a.

6.4.3 Thread-Safe Libraries

Non-thread-safe and thread-safe _r libraries have been combined into one set of libraries, thereby turning thread-safety on by default. On AIX V4.3, both libc.a and libc_r.a are symbolic links to the same library, /usr/ccs/lib/libc.a. The `cc_r` and `xlc_r` compiler command still works, but using the `cc` or `xlc` command with `-D_THREAD_SAFE` and `-lpthreads` also produces thread-safe applications.

```
xlc_r -o samp samp.c
xlc -o samp samp.c -D_THREAD_SAFE -lpthreads
```

In AIX V4.3, these commands have the same meaning.

6.4.4 Porting Issues

When planning to migrate a multithreaded application to 64-bit, you also have to consider migration to POSIX threads Draft 10.

6.4.4.1 No More Implicit Default Attribute

Some default attributes are changed. For example, PTHREAD_MUTEX_DEFAULT is not defined in Draft 10. Applications should not reference the following variables on Draft 10 environment;

```
pthread_attr_default
pthread_condattr_default
pthread_mutexattr_default
pthread_set_mutexattr_default_np
```

The default detach state is also changed. In Draft 10 the default is joinable (same as PTHREAD_CREATE_UNDETACHED) instead of detachable. All threads should be clearly initialized.

Example:

```
pthread_attr_t  attr;      /* thread attribute       */
pthread_t       tid;       /* thrtead ID             */

pthread_attr_init(&attr);  /* initialize an attribute */
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
pthread_create(&tid, &attr , thread_main, arg );

/* Do something  */

pthread_attr_destroy(&attr);/* destory thread attribute */
```

The thread created with the NULL attribute parameter will show a different behavior.

6.4.4.2 Parameters Types

Some of subroutine parameter types are changed. See the include file /usr/include/pthread.h, to check parameter types.

6.4.4.3 Removed pthread_yield() Subroutine

The pthread_yield subroutine that forced the calling thread to relinquish use of its processor and to wait in the run queue before it is scheduled again, is removed in Draft 10.

The pthread_yield subroutine may be used for programs implementing their own high-level locking services, instead of using the standard locking services (mutexes) provided in the threads library. For example, a database product may already use a set of internally-defined services and use this subroutine. The following example shows custom high-level routines independent of the thread library using the pthread_yield subroutine along with the _check_lock subroutine.

Example:

```
#include <sys/atomic_op.h>       /* for locking primitives       */
#define SUCCESS          0
#define FAILURE         -1
#define LOCK_FREE        0
#define LOCK_TAKEN       1
typdef struct {
        atomic_p         lock;   /* lock word                    */
        tid_t            owner;  /* identifies the lock owner    */
        ...              /* implementation dependent fields      */
} my_mutex_t;
...
int my_mutex_lock(my_mutex_t *mutex) {
tid_t   self;   /* caller's identifier */
        /* Perform various checks:is mutex a valid pointer?      */
        /*                        has the mutex been initialized? */
        ...
        /* test that the caller does not have the mutex          */
        self = thread_self();
        if (mutex->owner == self)
                return FAILURE;
        /* Perform a test-and-set primitive in a loop.           */
        /* In this implementation, yield the processor if failure.*/
        /* Other solutions include: spin (continuously check);   */
        /*         or yield after a fixed number of checks.      */
        while (!<_check_lock>(&mutex->lock, LOCK_FREE, LOCK_TAKEN))
                pthread_yield();    /* give up CPU               */
        mutex->owner = self;
        return SUCCESS;
} /* end of my_mutex_lock */
```

Now, you might modify your code to use the standard locking service, mutexes, insted of your own high-level locking services.

Example:

```
/* the initial thread */
pthread_mutex_t mutex;
int i;
...
pthread_mutex_init(&mutex, NULL);    /* creates the mutex      */
for (i = 0; i < num_req; i++)        /* loop to create threads */
        pthread_create(th + i, NULL, rtn, &mutex);
...                                  /* waits end of session   */
pthread_mutex_destroy(&mutex);       /* destroys the mutex     */
...
/* the request handling thread */
```

```
...                                 /* waits for a request    */
pthread_mutex_lock(&db_mutex);      /* locks the database     */
...                                 /* handles the request    */
pthread_mutex_unlock(&db_mutex);    /* unlocks the database   */
...
```

For more details about locking services, refer to *AIX Version 4.3 General Programming Concepts: Writing and Debugging Programs*, SC23-4128.

6.4.5 Threads Scheduling

Each thread has its own set of scheduling parameters. These parameters can be set using the thread attributes object before the thread's creation.

6.4.5.1 Scheduling Policy and Priority

The scheduling policy can be set when creating a thread by setting the schedpolicy attribute of the thread attributes object. The `pthread_attr_setschedpolicy` subroutine sets the scheduling policy to one of the three previously-defined scheduling policies. The current value of the schedpolicy attribute of a thread attributes object can be obtained by the `pthread_attr_getschedpolicy` subroutine.

The scheduling priority can be set at the thread's creation time by setting the schedparam attribute of the thread attributes object. The `pthread_attr_setschedparam` subroutine sets the value of the schedparam attribute, copying the value of the specified structure. The `pthread_attr_getschedparam` subroutine gets the schedparam attribute.

The current schedpolicy and schedparam attributes of a thread are returned by the `pthread_getschedparam` subroutine. These attributes can be set by calling the `pthread_setschedparam` subroutine. If the target thread is currently running on a processor, the new scheduling policy and priority will be implemented the next time the thread is scheduled. If the target thread is not running, it may be scheduled immediately at the end of the subroutine call.

The setting of scheduling policy and priority is also influenced by the contention scope of threads. Using the FIFO or the RR policy may not always be allowed.

6.4.5.2 Contention Scope

The threads library defines two possible contention scopes:

PTHREAD_SCOPE_PROCESS

Process (or local) contention scope. Specifies that the thread will be scheduled against all other local contention scope threads in the process.

PTHREAD_SCOPE_SYSTEM

System (or global) contention scope. Specifies that the thread will be scheduled against all other threads in the system.

The contention scope can only be set before thread creation by setting the contention-scope attribute of a thread attributes object. The `pthread_attr_setscope` subroutine sets the value of the attribute; the `pthread_attr_getscope` returns it.

The contention scope is only meaningful in a mixed-scope M:N library implementation. A single-scope 1:1 library implementation, as it was so far in AIX V4, always returns an error when trying to set the contention-scope attribute to PTHREAD_SCOPE_PROCESS, because all threads have system contention scope. Local user thread can set any scheduling policy and priority, within the valid range of values. However, two threads that have the same scheduling policy and priority but have different contention scopes will not be scheduled in the same way. Threads having process contention scope are executed by kernel threads whose scheduling parameters are set by the library.

Note

On the 1:1 library environment, setting scheduling policy always affects system performance.

6.4.5.3 Process Contention Scope

Applications should use the default scheduling policy, unless a specific application requires the use of a fixed-priority scheduling policy. Process contention scope has more opportunity to be used safely for multithreaded programs.

Using the RR policy ensures that all threads having the same priority level will be scheduled equally, regardless of their activity. This can be useful in programs where threads have to read sensors or write actuators.

Using the FIFO policy should be done with great care, even though process contention scope is implemented. A thread running with FIFO policy runs to completion, unless it is blocked by some calls, such as performing input and output operations.

See "Threads Scheduling" in *AIX Version 4 General Programming Concepts: Writing and Debugging Programs*, SC23-2533, for more information.

6.4.6 Thread Data Size

In 64-bit mode, thread stack minimal size and thread-specific data is changed to double size.

```
#ifdef __64BIT__
#define PTHREAD_SPECIFIC_DATA  2*PAGESIZE /* per-thread data */
#else
#define PTHREAD_SPECIFIC_DATA  PAGESIZE   /* per-thread data */
#endif /* __64_BIT__ */

#ifdef __64BIT__
#define PTHREAD_STACK_MIN     ( 2 * PAGESIZE * 2)
#else   /* __64BIT__ */
#define PTHREAD_STACK_MIN     ( PAGESIZE * 2)
#endif /* __64_BIT__ */
```

6.4.7 64-bit Thread Benefits

Threads do not share register files, stacks, thread-specific data, or heap-allocated storage. Creating many threads may cause a contention for memory addresses. Using 64-bit mode, it is possible to assign one segment size (256 MB) for each thread even if you create more than 12 threads.

Example:

```
#include <pthread.h>
#include <stdio.h>
#define  ROOP              40
#define  THREAD_STACK_MAX (256 * 1024 *1024)

void  *th_func(void *dummy)  {
    sleep(60);
    pthread_exit(NULL);
}
void  main(void) {
    pthread_attr_t  p_attr_t;
    pthread_t       func_th[PTHREAD_THREADS_MAX];
    char            dummy;
    int             counter;

    pthread_attr_init(&p_attr_t);
    pthread_attr_setdetachstate(&p_attr_t, PTHREAD_CREATE_JOINABLE);
    pthread_attr_setstacksize(&p_attr_t, THREAD_STACK_MAX);
```

```
    for ( counter=0; counter < ROOP; counter ++)  {
        pthread_create(&func_th[counter], &p_attr_t, th_func, NULL);
    }

    for ( counter=0; counter < ROOP; counter ++)  {
        pthread_join(func_th[counter], NULL);
    }
    pthread_attr_destroy(&p_attr_t);
}
```

This program creates 40 threads, and each thread has a 256 MB stack segment. Thus, the process has 41 threads (including the initial thread) and needs more than 10 GB address space. To enable this large address space, you have to use the `-bmaxdata` flag with the `xlc` or `ld` command. To link this program that will have more than 40 segments assigned, the following command could be used:

```
/usr/vac/bin/xlc_r sample.c -q64 -bmaxdata:0x290000000
```

The number 0x290000000 is the number of bytes, in hexadecimal format, equal to 41 segments each of 256 MB . The last segment is for the initial thread, and does not use 256 MB. So, it is possible to use `-bmaxdata:0x281000000`. If you forget the size of this initial thread and request only `-bmaxdata:0x280000000`, your application will fail with an error message.

6.5 Device Drivers Considerations

Device drivers provide application programming interfaces. The APIs can be accessed from 32-bit or 64-bit programs. Since the AIX kernel executes in 32-bit mode, the structures passed to device drivers (and kernel extensions in general) from 64-bit user programs have to be interpreted differently than the structures originating in 32-bit user programs. This refers to the long and pointer elements of the structures.

6.5.1 Device Drivers and 64-bit Application Support

The AIX system call remapping approach reallocates and reformats most of the system call structures when they originate in the 64-bit programs. As a result, kernel services (including extensions) see them as if they originate in the 32-bit programs. In a few cases, the system call structures are not explicit and therefore the system call remapping methodology cannot perform remapping and reformatting. The `ioctl` system call is the most obvious example of this, in that the third argument data type is undefined.

6.5.2 Changes to ioctl()

For AIX V4.3, the kernel guarantees that the arg parameter received by a device driver is always a 32-bit value. For 64-bit applications the kernel will remap the address to a 32-bit address. Device drivers that support 64-bit embedded pointers need to notify the kernel of this by setting the DEV_64BIT define for the d_opts flag passed to the devswadd() call from the config entry point of the device driver. For example:

```
devsw_struct.d_opts = DEV_MPSAFE | DEV_64BIT;
devswadd(devno,&devsw_struct);
```

A 64-bit application can use a device driver unmodified for 64-bit by using a new system call ioctl32(). This is available in both 32-bit and 64-bit mode.

If a device driver calls the devswadd kernel services to add a device to the device switch table, the device driver must indicate its ability to support 64-bit processes. This can be done in one of two ways:

1. By passing a new option (DEV_64BIT) in the d_opts field of devsw structure to the devswadd kernel service.
2. By issuing a SYS_64BIT command to sysconfig.

If the device driver does not indicate this capability, the kernel will fail the ioctl system call from 64-bit processes. This explicit indication of 64-bit process support is necessary so that structures that originated from 64-bit programs are not passed to device drivers that are not aware or capable of supporting 64-bit processes.

6.5.3 PCI d_map Changes

The Peripheral Component Interface (PCI) bus architecture allows for PCI devices that can drive a fully qualified 64-bit address. It is possible on a 64-bit PCI address bus with a single address cycle, or on a 32-bit address PCI bus using the Dual Address Cycle (DAC) protocol to issue the two halves of the 64-bit address. Even if a device is 64-bit capable, the parent host bridge must support 64-bit addressing before the function can be enabled.

6.5.4 Common Problems with PCI Device Drivers in 64-bit

The following is a list of common problems that PCI device drivers may encounter when running on a 64-bit system, even though they run without problems on 32-bit systems.

1. Not specifying mappable DMA space requirements.

Symptoms DMA_NORES return codes from d_map_page and/or d_map_list. Degraded performance due to resource pool being too small.

Description Prior to 64-bit systems, there was no real resource tied to a DMA mapping. So a device driver could have infinitely many outstanding d_map_page and d_map_list mappings. However, that is not the case with 64-bit systems, since there are resources tied to each mapping. The DMA_MAXMIN_* flags on the d_map_init call indicate to the DMA services how much space to reserve for this device.

2. Not calling d_unmap_page and d_unmap_list.

   Symptoms DMA_NORES return codes from d_map_page and/or d_map_list due to exhausting the resource pool.

   Description Prior to 64-bit systems, the d_unmap_page and d_unmap_list services were basically no-ops since there were no resources to unmap. However, there are resources tied to each mapping for 64-bit systems.

3. Calling d_unmap_page or d_unmap_list with incorrect parameters.

   Symptoms System crashes, unpredictable behavior.

   Description This is the type of problem that could have been undetected until running for the first time on a 64-bit system. The d_unmap_page call takes the address of the bus address parameter, just as the d_map_page call did. A common mistake is to pass the bus address parameter by value rather than by reference. Also, it is important that the bus_list parameter used on the d_unmap_list correctly reflects the number of list entries and addresses to unmap as previously mapped by a d_map_list call.

4. Improper use of DMA_READ/DMA_WRITE flags on d_map_list or d_map_page.

   Symptoms PCI target aborts with I/O failures.

   Description Prior to 64-bit systems, a device driver could have gotten away with setting up a DMA mapping as DMA_WRITE (indicating the memory will only be read by the device) and then having his device perform a DMA write to that memory. On 64-bit systems, that will result in a PCI target abort due to page access violations. If a memory page is to ever be written by a device, the flag setting must be DMA_READ on the mapping call.

5. Improper serialization.

Symptoms System crash (system asserts detecting re-entry).

Description All calls to the mapping and unmapping services for a single handle (d_handle) must be serialized. This could have gone undetected on 32-bit systems, since there were no resources being allocated or freed during the map and unmap calls.

6. Improper handling of DMA return codes.

   Symptoms Unpredictable system behavior, system crashes.

   Description The DMA_NORES return code from d_map_page and d_map_list is one that is only possible on 64-bit systems. This condition does not necessarily indicate an error, only that there are not enough resources currently available to map this entire request. The device driver must detect and handle this condition properly whether it be to unmap any partial mapping, or proceed with the partial I/O and re-issue the remainder later. The DMA_NORES returned by d_map_page does not require a corresponding d_unmap_page call, since the smallest granularity of mapping is a single page.The corresponding unmap service (d_unmap_page or d_unmap_list) must NOT be called when the DMA_NOACC return code is received, since no mapping was performed.

6.5.5 Network Device Drivers

AIX Common Data Link Interface (CDLI) architecture defines the ndd_ctl function in the Network Device Driver (NDD) structure. This function is used to implement user ioctls. User processes allocate sockets in the AF_NDD family, and then issue ioctls on the socket descriptor. The AIX sockets subsystem processes these ioctls. It copies the user space data to the kernel buffers and then passes the kernel space buffer to the ndd_ctl function. If the ioctls are invoked from 64-bit processes, the sockets subsystem uses appropriate mechanisms (such as copyin64 and copyout64) to copy the user space data to the kernel buffers.

Since the ndd_ctl interface uses kernel buffers, there is no special consideration with respect to the address of this buffer. However, if this buffer points to a structure that contains long or pointer elements, then NDDs need to interpret these two data types differently depending upon the calling process execution mode. For example, if the calling process is 64-bit, the long type must be interpreted as a 8-byte field (long long in 32 bit mode); similarly, the pointer type must be interpreted as a 64-bit pointer and needs to be used in the copyin64 or copyout64 operations.

To maintain binary compatibility with prior AIX releases, existing NDDs continue to work when used from 32-bit processes. To account for the new 64-bit data types, NDDs explicitly declare their 64-bit capability using a new `NDD_64BIT` flag. The sockets subsystem does not pass ioctls from 64-bit processes unless this flag is set.

6.5.6 Streams Modules and Drivers

The streams framework is enhanced so that streams system calls can be used from 64-bit and 32-bit user processes. The exported system calls `getmsg`, `putmsg`, `getpmsg` and `putpmsg` are being remapped consistent with AIX 64-bit remapping methodology. These system calls use the `strbuf` structure.

```
struct strbuf {
      int       maxlen; /* max buffer length */
      int len   len;    /* length of data    */
      char     *buf;    /* pointer to buffer */
  };
```

The buffer may contain a structure that is not explicitly defined by the system calls. This structure in turn may contain elements that have different lengths in the two execution modes. To account for this, the streams modules and drivers need to interpret the structures based on the mode of the calling process. Also, the streams framework needs to make sure that the streams modules and drivers are aware of the 64-bit data types to guard against passing the data structures to the ones that are not capable of handling them. All the existing streams modules and drivers fall in this category, unless they are enhanced to support 64-bit user processes.

6.6 References

The following are good sources for additional information.

- AIX V4.3 Migration Guide

  `http://w3dev.austin.ibm.com/library/aix4.3/`

- *AIX Version 4 General Programming Concepts: Writing and Debugging Programs*, SC23-2533

Chapter 7. 64-bit Interlanguage Calls

A significant number of applications use C and FORTRAN together, by calling each other or sharing files. Such applications are among the early candidates for porting to 64-bit for their abilities to handle more complex mathematical calculations.

For these reasons, this chapter will focus on FORTRAN and C interlanguage calls. However, programmers who use other languages will also benefit from the following content, which can be considered as a guideline for many other interlanguage calls.

7.1 Mixing FORTRAN and C: Programming Techniques

This section does not cover all the programming techniques but it discusses the programming techniques that are new issues in a 64-bit environment.

7.1.1 Conventions for XL Fortran and C External Names

You should follow these recommendations when writing C for AIX or FORTRAN for AIX code to call functions written in other languages:

- Names that begin with "_" are reserved for the names of library routines (naming convention for the C language library). Do not use "_" as the first character of a FORTRAN external name or for C external names.
- Also avoid the dollar character ($) as the first character in identifiers, to prevent conflict with the naming conventions for the C language library. The `-qdollar` option allows you to use the "$" character in identifiers.
- Programs and symbolic names in FORTRAN are interpreted as all lowercase by default. For example, FORTRAN does not distinguish between *PROFIT* and *profit*. It is the same variable by default. Use all lowercase procedure names to simplify calling the procedure from C or from FORTRAN.
- Avoid using long identifier names. The maximum number of significant characters in identifiers is 250 characters. They are recognized by both languages but carry a performance penalty.
- Names for module procedures are formed by concatenating "__" (two underscores), the module name, "_MOD_", and the name of the module procedure. For example, module procedure MYPROC in module MYMOD has the external names:

  ```
  __mymod_MOD_myproc
  ```

Notice that names with "__" (two underscores) or an underscore followed by a uppercase letter, are reserved in all contexts. For all these reasons, we strongly recommend not to use the "$" and "_" characters in identifiers to avoid naming conflicts in your programs.

FORTRAN for AIX provides you the `-qextname` option, which adds an underscore to the end of the FORTRAN names. This avoids conflicts between FORTRAN and non-FORTRAN function names. In this case, you should use an underscore as the last character of any non-FORTRAN procedure that you want to call from FORTRAN. Let's see an example without `-qextname` option:

- <myinterlanguage_prog.c file>

```
/* We need to declare all external FORTRAN procedure.
 */
extern void myaddprog (int*,int*,int*);

/* this is my main program written in C
 * it calls a FORTRAN program to do computations.
 */
main{}
{
int a,b,c;
call myaddprog (&a,&b,&c);
printf ("c, %d, is equal to a+b, %d + %d\n", c,a,b);
}
```

- <mycallee_prog.f file>

```
subroutine myaddprog(d,e,f)
integer*4 d,e,f
logical*2 ok
f=d+e
call myverif(d,e,f,ok)
if (ok .eq. .true.) write(6,*) 'You know what? I''m happy'
end
```

- <mycallee_prog.c file>

```
/* this is a C program called by FORTRAN. */
extern void myverif (int* i, int* j, int* k, unsigned short int* valid)
{
/* this is my subroutine written in C
 * it is called by a FORTRAN program.
 */
int l;
l= *i + *j;
if (l == *k) valid=1;
else valid=0;
```

```
}
```

Now, the same example is written for using the `-qextname` option. The programs below show how to call programs that are written in other languages:

- <myinterlanguage_prog.c file>

```
/* We need to declare all external FORTRAN procedure.
 */
extern void myaddprog_(int*,int*,int*);

/* this is my main program written in C
 * it calls a FORTRAN program to do computations.
 */
main{}
{
int a,b,c;
call myaddprog_(&a,&b,&c);
printf ("c, %d, is equal to a+b, %d + %d\n", c,a,b);
}
```

- <mycallee_prog.f file>

```
subroutine myaddprog(d,e,f)
integer*4 d,e,f
logical*2 ok
f=d+e
call myverif_(d,e,f,ok)
if (ok .eq. .true.) write(6,*) 'You know what? I''m happy'
end
```

- <mycallee_prog.c file>

```
/* this is a C program called by FORTRAN.
 */
extern void myverif (int* i, int* j, int* k, unsigned short int* valid)
{
/* this is my subroutine written in C
 * it is called by a FORTRAN program.
 */
int l;
l= *i + *j;
if (l == *k) valid=1;
else valid=0;
}
```

If porting an application or if your application encounters naming conflicts like these, you can use the `-brename` linker option to rename the symbol:

```
xlf90 -brename:<oldname>,<new_name> myinterlanguage_prog.f
```

However, our recommendation is to use only lowercase names, though you may still want the external procedure names to use both upper- and lowercase. You can use the `-U` option or the *@PROCESS MIXED* directive:

```
@process mixed
   external C_Func ! With MIXED, we can call C_Func, not just c_func.
   integer aBc, ABC ! With MIXED, these are different variables.
   common /xYz/ aBc ! xYz and XYZ are external names that
   common /XYZ/ ABC ! are visible during linking.
end
```

7.1.2 Passing Data between FORTRAN and C Languages

FORTRAN uses only variable addresses when it passes an argument between FORTRAN routines: FORTRAN default is call-by-reference. FORTRAN language specification does not allow the call-by-value.

Because of XL Fortran's call-by-reference conventions, even scalar values from another language must be passed as the address of the value rather than the value itself.

For example, a C function passing an integer value "X" to FORTRAN must pass "&X". On the other hand, a C function called by FORTRAN must define its arguments as pointers to the corresponding type (see the program example in 7.1.1, "Conventions for XL Fortran and C External Names" on page 125).

Call-by-Reference Parameters

If you write C functions that:

- You want to call from a FORTRAN program, declare all parameters as pointers.
- Calls a program written in FORTRAN, then all arguments must be pointers or scalars with the address operator.

Table 12 on page 129 shows the corresponding types in FORTRAN and C.

Table 12. Corresponding Data Type in FORTRAN and C

XL Fortran Data Types	C Data Types	Alignment3 2/64bit (min, max)
INTEGER(1) BYTE	signed char	(1/1, 1/1)
INTEGER(2)	signed short	(1/1, 2/2)
INTEGER(4)	signed int	(1/1, 4/4)
INTEGER(8)	signed long long	(1/1, 4/8)
REAL or REAL(4)	float	(1/1, 4/4)
REAL(8) or DOUBLE PRECISION	double	(1/1, 8/8)
REAL(16)	long double (with `-qldbl128`)	1/1,16/16
COMPLEX or COMPLEX(4)	structure of 2 floats	(1/1,4/4)
COMPLEX(8) or DOUBLE COMPLEX	structure of 2 doubles	(1/1,8/8)
COMPLEX(16)	structure of 2 long doubles	1/1,16/16 [1]
LOGICAL(1)	unsigned char	(1/1, 1/1)
LOGICAL(2)	unsigned short	(1/1, 2/2)
LOGICAL(4)	unsigned int	(1/1, 4/4)
LOGICAL(8)	unsigned long long	(1/1, 4/8)
CHARACTER	char	(1/1,1/1)
CHARACTER(n)	char[n]	(1/1,1/1)
POINTER	void [1]	1/1,4/8
Array	array	see [2]
Sequence derived type	structure (with -qalign=packed)	see [3]

[1] Double and long double structure members are treated differently depending on where they are in the structure. If they are the first member, double and 8-byte long doubles are doubleword-aligned, while 16-byte long doubles are quadword-aligned. Floats are singleword-aligned. All are singleword-aligned if they are not the first member.
[2] The alignment of an array is the same as the alignment of its element type.
[3] The alignment of the beginning of a structure or a union is the maximum alignment of any of its members. Each member within the structure or union must be placed at its proper alignment as defined by the table above, which may require implicit internal padding, depedning on the previous member.

Notice that:

1 means byte alignment
2 means half word alignment
4 means word alignment
8 means double word alignment
16 means quad word alignment

7.1.2.1 Passing Characters and Strings

In C, character strings are stored as arrays of the type char. These arrays are delimited by a "\0" character.

In FORTRAN, the only character type is CHARACTER, which is stored as a set of contiguous bytes, one character per byte. All character variables and expressions have a length that is determined at compile time. If FORTRAN passes a string argument to another routine, it adds a hidden argument giving the length to the end of the argument list. This FORTRAN hidden length argument must be declared explicitly in C.

On the other hand, C code should not assume a null terminator in FORTRAN-passed strings; the supplied and declared length should be always used. Use the `strncat`, `strncpm`, and `strncpy` functions of the C runtime library. These functions are described in the *AIX Version 4 Technical Reference, Volumes 1 and 2: Base Operating System and Extensions*, SC23-2614 and SC23-1615.

You can use the `-qnullterm` option of the compiler to automatically add the NULL character to certain arguments. This does not change the length of the dummy argument as defined by the XL Fortran calling convention. In the following example, there are two calls to the same C function, with and without the option.

Example:

```
@PROCESS NONULLTERM
SUBROUTINE CALL_C_1
    CHARACTER*9, PARAMETER :: HOME = "/home/rosa"
! Call the libc routine mkdir() to create some directories.
    CALL mkdir ("/home/rosa/testfiles\0", %val(448))
! Call the libc routine unlink() to remove a file
    CALL unlink (HOME // "/.hushlogin" // CHAR(0))
END SUBROUTINE


@PROCESS NULLTERM
SUBROUTINE CALL_C_2
    CHARACTER*9, PARAMETER :: HOME = "/home/koh"
```

```
! Call the libc routine mkdir() to create some directories.
    CALL mkdir ("/home/koh/testfiles", %val(448))
! Call the libc routine unlink() to remove a file
    CALL unlink (HOME // "/.hushlogin")
END SUBROUTINE
```

7.1.2.2 Passing Arrays

A C array does not need to be passed as pointer: C passes arrays to others C routines by reference, like FORTRAN's call-by-reference default behavior. So, a C array can be passed to FORTRAN without the "&" operator. Keep in mind that although C passes individual scalar array elements by value, it passes arrays by reference.

However, C stores array elements in row major order (array elements in the same row occupy adjacent memory locations). FORTRAN stores array elements in ascending storage units in column-major order (array elements in the same column occupy adjacent memory locations). Table 13 on page 131 shows how a two-dimensional array declared by A[3][2] in C and by A(3,2) in FORTRAN is stored:

Table 13. Storage of a Two-Dimensional Array with C and FORTRAN

Storage Unit	C Element Name	FORTRAN Element Name
Lowest	A[0][0]	A(1,1)
	A[0][1]	A(1,2)
	A[1][0]	A(2,1)
	A[1][1]	A(2,2)
	A[2][0]	A(3,1)
Highest	A[2][1]	A(3,2)

In general, for a multi-dimensional array, if you list the elements for the array in the order they are laid out in memory, a row-major array will be such that the right-most index varies fastest, while a column-major array will be such that the left-most index varies fastest.

For example, the FORTRAN array reference A(X,Y,Z) must be expressed in C as a[Z-1][Y-1][X-1].

7.1.2.3 Passing Pointers

A C function passing a pointer value P to FORTRAN so that FORTRAN can use it as integer POINTER must declare it as `void **p`.

7.1.3 Passing Arguments by Value

To call subprograms written in languages other than FORTRAN (for example, AIX operating system routines), the actual arguments may need to be passed by a method different from the default method call-by-reference used by FORTRAN.

7.1.3.1 Call-by-Value Definition

C routines, including those in system libraries such as libc.a, require arguments to be passed by value instead of by reference.

The default passing method can be changed by using the %VAL and %REF built-in functions in the argument list of a CALL statement or function reference.

%REF Passes an argument by reference (the default FORTRAN call-by-reference method) except that it suppresses the extra length argument strings.

%VAL Passes an argument by value (that is, the called subprogram receives an argument that has the same value as the actual argument, but any change to this argument does not affect the actual argument). Please notice that %VAL can not be used with actual arguments that are array entities, procedure names, or character expressions of length greater than one byte.

Passing arguments with %REF is referred as call-by-reference and passing arguments with %VAL is referred as call-by-value.

Call-by-Value Parameters

If you write a FORTRAN program that calls a C subroutine that expects its arguments as values, you need to use the built-in function %VAL.

- The called function will receive a copy of the value passed to it.
- The arguments by value are stored inside the available General Registers (GPRs) and the floating-point registers (FPRs). The GPRs and FPRs available for argument passing are specified in two fixed lists: R3-R10 and FP1-FP13.
- Remaining arguments are passed in storage on the stack.

The book *AIX Version 4 Assembler Language Reference*, SC23-2642 describes the following details of the subroutine linkage convention:

- Register usage (general-purpose, floating-point, and special-purpose registers)
- Stack
- The calling routine's responsibilities
- The called routine's responsibilities

As seen previously in 7.1.2, “Passing Data between FORTRAN and C Languages” on page 128, alignment depends on data type. Since %VAL causes the argument value to be passed, alignment within the user stack will happen. The alignment will not be the same in both a 32-bit environment and a 64-bit environment.

%VAL causes the actual argument to be passed as 32-bit or 64-bit intermediate values depending on compilation mode.

In 32-bit Mode

If the actual argument is:

- An integer or logical that is shorter than 32 bits, it is sign-extended to a 32-bit value.
- An integer or logical that is longer than 32 bits, it is passed as two 32-bit intermediate values.
- Of type real or complex, it is passed as multiple 64-bit intermediate values.
- Of type sequence derived type, it is passed as multiple 32-bit intermediate values.

Byte named constants and variables are passed as if they were INTEGER(1). If the actual argument is a CHARACTER(1), it is padded on the left with zeros to a 32-bit value, regardless of whether the `-qctyplss` compiler option is specified.

In 64-bit Mode

If the actual argument is:

- An integer or logical that is shorter than 64 bits, it is sign-extended to a 64-bit value.
- Of type real or complex, it is passed as multiple 64-bit intermediate values.
- Of type sequence derived type, it is passed as multiple 32-bit intermediate values.

Byte named constants and variables are passed as if they were INTEGER(1). If the actual argument is a CHARACTER(1), it is padded on the left with zeros to a 64-bit value, regardless of whether `-qctyplss` compiler option is specified.

7.1.3.2 Linkage Convention for Argument by Value Passing

The following refers to call-by-value, as in C or as in FORTRAN when %VAL is used. For purposes of their appearance in the list, arguments are classified as floating-point values or non-floating-point values:

In a 32-bit Environment

- Each INTEGER(8) and LOGICAL(8) argument requires two words.
- Any other non-floating-point scalar argument of intrinsic type requires one word and appears in that word exactly as it would appear in a GPR. It is right-justified, if language semantics specify, and is word-aligned.
- Each single-precision (REAL(4)) value occupies one word, each double-precision (REAL(8)) value occupies two successive words in the list, and each extended-precision (REAL(16)) value occupies four successive words in the list.
- A COMPLEX value occupies twice as many words as a REAL value with the same kind type parameter.
- In FORTRAN and C, structure values appear in successive words as they would anywhere in storage, satisfying all appropriate alignment requirements. Structures are aligned to a fullword and occupy (sizeof(struct X)+3)/4 fullwords with any padding at the end. A structure that is smaller than a word, is left-justified within its word or register. Larger structures can occupy multiple registers, and may be passed partly in storage and partly in registers.

In a 64-bit Environment

- All non-floating-point values require one doubleword that is doubleword-aligned.
- Each single-precision (REAL(4)) value AND each double precision (REAL(8)) value occupies one doubleword in the list, and each extended-precision (REAL(16)) value occupies two successive doublewords in the list.
- A COMPLEX value occupies twice as many doublewords as a REAL value with the same kind type parameter.
- In FORTRAN and C, structure values appear in successive words as they would anywhere in storage, satisfying all appropriate alignment requirements. Structures are aligned to a doubleword and occupy (sizeof(struct X)+7)/8 fullwords with any padding at the end. A structure that is smaller than a word is left-justified within its word or register. Larger structures can occupy multiple registers, and may be passed partly in storage and partly in registers.

This is a FORTRAN example of a call-by-value to a C function, which gives the storage mapping of the PARM AREA for 32-bit and 64-bit environment.

```
integer(2) court1
integer(4) entier1, entier2
real(4) flottant1
real(8) double1, double2
character car1
complex(8) cx1
call cfuncval (%val(entier1), %val(entier2), %val(double1),
     %val(flottant1), %val(car1), %val(double2),
     %val(court1), %val(cx1))
```

In Figure 2 on page 137, we can see that:

- A parameter is guaranteed to be mapped only if its address occupied.
- Data with less than fullword-alignment is copied into high-order bytes. Because the function in the example is prototyped, the mapping of parameters c and s1 is right-justified.
- The parameter list is a conceptually contiguous piece of storage containing a list of words. For efficiency, the first eight words of the list are not actually stored in the space reserved for them, but passed in GPR3-GPR10. Furthermore, the first 13 floating-point value parameter values are not passed in GPRs, but are passed in FPR1-FPR13. In all cases, parameters beyond the first eight words of the list are also stored in the space reserved for them.
- If the called procedure intends to treat the parameter list as a contiguous piece of storage (for example, if the address of a parameter is occupied in C), the parameter registers are stored in the space reserved for them in the stack.
- A register image is stored on the stack.

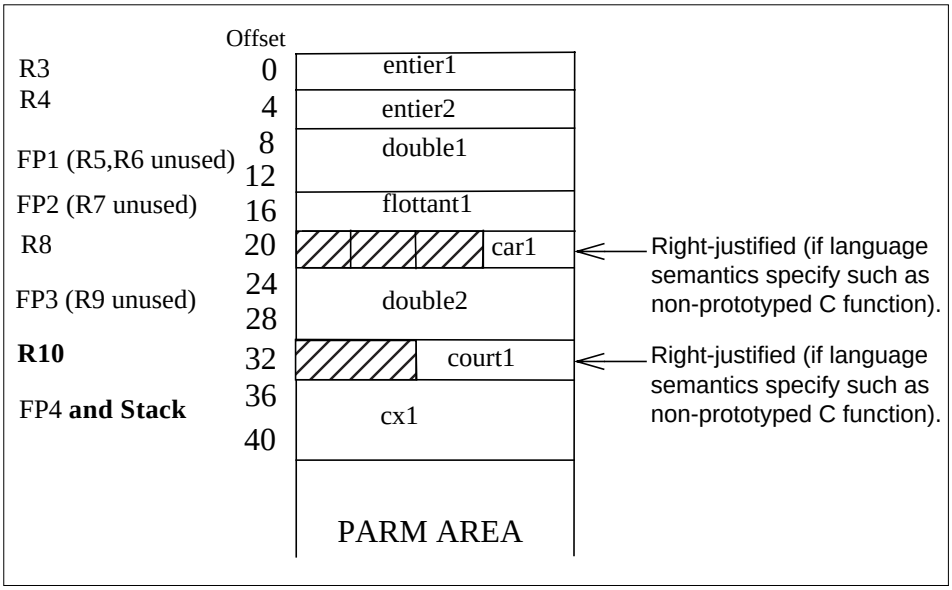


Figure 2. Storage Mapping of Parm Area in 32-bit Environments

As we can see in Figure 3 on page 138, the parm area (part of the user stack) is bigger in 64-bit than in 32-bit. We do not recommend programs to read the parm area by a direct mapping, due to these alignment differences between the two compilation modes.

Also notice that using 32-bit integers in a 64-bit environment does waste some space for alignment on subroutine call.

Comparing both 32-bit and 64-bit parm areas, you can see that this subroutine call will have better performance in a 32-bit than in a 64-bit environment. Developers should remember these issues when deciding to migrate their application to 64-bit.

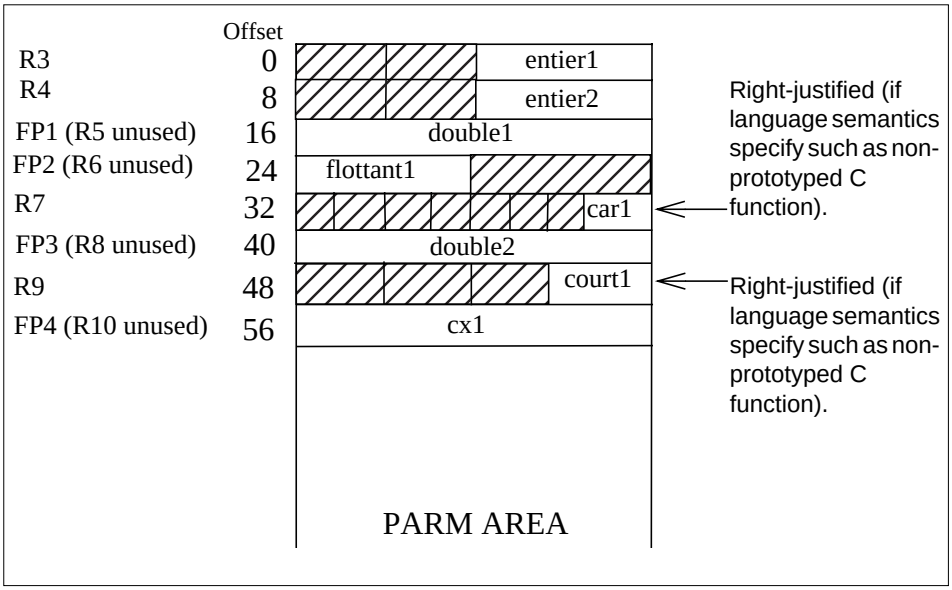


Figure 3. Storage Mapping of Parm Area in 64-bit Environments

7.2 Type Encoding and Checking

Detecting errors before a program is run is a key objective of the C for AIX compiler. Runtime errors are hard to find, and many are caused by mismatching subroutine interfaces or conflicting data definitions.

The C for AIX compiler and XL Fortran use a scheme for early detection that encodes information about all external symbols (data and programs). If the `-qextchk` option has been specified, this information about external symbols is checked at bind or load time for consistency.

Our Recommendation

Interlanguage applications can be hard to achieve due to the different passing argument techniques and rules. Therefore, we recommend you use `-qextchk` in the debugging step. For the final executable production, we recommend not using it, because it enlarges object size.

To store type information in the object file so the linker can detect mismatches, use the `-qextchk` compiler option.

Appendix A. Some 64-bit Mathematics Hints and Tips

Building machines that can handle large data areas like files and memory of more than four GB is not an obvious design. It also is more difficult for the user, who must manage large system resources or develop large programs, to keep from getting confused with such a wide range of data addresses.

If you are used to simple conversions such as 0xF is '1111' is 15, you are probably less familiar with the translation of 2^{52} or 0x6FFFFFF, which are very basic values inside the AIX 64-bit dimension. What is the value in the base 10 mathematical model? Can you name the number? This appendix provides some different hints and tips useful for the main conversions in the AIX environment. These will help illustrate how huge the AIX 64-bit dimension can be.

A.1 Hints and Tips for 32-bit Mathematics

This appendix is useful for users who are not familiar with hexadecimal and binary writing. The basic notions are explained here. If you are comfortable with 32-bit mathematics, proceed directly to Appendix A.2, "Hints and Tips for 64-bit Mathematics" on page 143.

A.1.1 Values and Representation of Binaries

This section defines the basic notion of a base value and how to convert an hexadecimal value to a decimal value.

The Basic Key Table

First, we manipulate four digits in the three main references. These are basic and common values (see Table 14 on page 139).

Table 14. Binary/Hexadecimal Values (from 0 to 15)

Base 16	Base 10	Base 2
0	0	0 0 0 0
1	1	0 0 0 1
2	2	0 0 1 0
3	3	0 0 1 1
4	4	0 1 0 0
5	5	0 1 0 1

Base 16	Base 10	Base 2
6	6	0 1 1 0
7	7	0 1 1 1
8	8	1 0 0 0
9	9	1 0 0 1
A	10	1 0 1 0
B	11	1 0 1 1
C	12	1 1 0 0
D	13	1 1 0 1
E	14	1 1 1 0
F	15	1 1 1 1

The following is the conversion table for the next 16 values, from 16 to 31 (see Table 15 on page 140).

Table 15. Binary/Hexadecimal Values (from 16 to 31)

Base 16	Base 10	Base 2
10	16	1 0 0 0 0
11	17	1 0 0 0 1
12	18	1 0 0 1 0
13	19	1 0 0 1 1
14	20	1 0 1 0 0
15	21	1 0 1 0 1
16	22	1 0 1 1 0
17	23	1 0 1 1 1
18	24	1 1 0 0 0
19	25	1 1 0 0 1
1A	26	1 1 0 1 0
1B	27	1 1 0 1 1
1C	28	1 1 1 0 0

Base 16	Base 10	Base 2
1D	30	1 1 1 0 1
1E	31	1 1 1 1 0
1F	32	1 1 1 1 1

This last table shows us that the number written in binary can be very long. For that reason, we remember only the first 16 values and we will use different ways to write and to convert them.

The Hexadecimal Representation for Binary Values

As the binary writing becomes difficult after the value 15, the usual convention in computer science, is to represent large binary values in hexadecimal base. In this case, where "1 1 1 1" means "F", we group binary digits by four.

Let’s see some examples of this representation:

1111 1111 represented by FF
1010 1000 represented by A8
1100 1110 represented by CE
1100 1011 1000 0100 represented by CB 84

This last example shows us hexadecimal values grouped by two digits which represent four bits each. "CB" represents the value of one byte.

Using this notation, we are able to write large binary values very quickly. However, we still have problems telling how much it represents since we normally count in decimal.

The Necessary Hexadecimal/Binary Conversions

Even if we can understand at first sight that "CD" is greater than "A8", we still don’t really know how much it is. To make this conversion easily, we have to convert the base value to the decimal system using the mathematical definition of base value.

- Base value definition

  This is the rule to convert a value in a different base.

Base Value Definition

Each digit of a number has to be multiplied by the base power of its ranking (ranking begins at 0).

$digit_n...digit_4\ digit_3\ digit_2\ digit_1$

Written in base, W represents the value of:

$(digit_n \times W^{n-1})+...+(digit_4 \times W^3)+(digit_3 \times W^2)+(digit_2 \times W^1)+(digit_1 \times W^0)$

with $W^0 = 1$.

Examples:

(base 10) 124 represents (1x100)+(2x10)+(4x1) = 1x102 + 2x101 + 4x100
(base 16) FCE represents 15x162 + 12x161 + 14x160
(base 2) 1 1001 represents 1x25+ 1x24+ 1x20

- Value capacity of bytes

  The binary conversion is by far the most-used method, because computers still manipulate everything in bits. Let's apply the previous rule and see how much a byte can represent:

  "1111 1111" represents $2^7+2^6+2^5+2^4+2^3+2^2+2^1+2^0= 255$

  If we add 1 to this value, the result is 256. That is "1 0000 0000" coded with 9 bits and we see that 256 is 2^8.

  If we use these 8 bits to code an address, we can have from address "0000 0000" to address "1111 1111", that is 256 (2^8) addresses.

  We have just defined the second basic computer rule which is:

Byte Value Capacity

With n bits, you can represent 2n addresses or values between 0 to 2n-1.

A.1.2 The AIX 32-bit Values

Depending on different choices that the AIX operating system uses to manage the memory (and therefore, the bits), some typical binary values are very well-known by the AIX system administrators' community.

These are the most well-known values:

1. 32-bit architecture limits the addressing capacity at four GB.

   Four GB is the value of 2^{32}.

2. One byte can address 256 locations.

   256 is the value of 2^8.

3. The AIX memory is organized with 4 K pages.

   4 K (4096) is the value of 2^{12}.

   The byte offset within an AIX memory page is coded with 12 bits.

4. The AIX memory segments are 256 MB long: they contain 65536 pages each.

   65536 is the value of 2^{16}.

   The page offset within an AIX memory segment is coded with 16 bits.

5. Inside AIX memory 32-bit architecture, four GB are addressed by 16 segments of 256 MB (4 GB / 0.25 GB = 4 x 4 = 16).

   16 is the value of 2^4.

   An AIX segment number is coded with 4 bits.

A.2 Hints and Tips for 64-bit Mathematics

The 64-bit architecture and the capabilities provided by 64-bit data excites people. But some are unaware that managing this huge data/address dimension will oblige us to review our usual ways of working. One of them is our pocket calculator.

As we have seen previously, manipulating 32-bit data/address with the help of some rules can be easily achieved. If you need to know the value of 2^{26}, your calculator will give you the precise answer.

```
67,108,864
```

However, if you need to know the value of 2^{27}, most calculators only give approximate answers:

```
1.342177e+08 --> 134,217,7??
```

Moreover, if you're new to the AIX 64-bit dimension like most people today, you'll like to know how many bytes VMM can address with four Giga segments of 256 MB each. You'll know how many segments are available for your user data stack (segment numbers from 0x10 to 0x6FFFFFFF). Your pocket calculator won't help you precisely manage such huge hexadecimal values.

With some algebra rules, it is easy to manipulate, without any calculator, 64-bit values.

A.2.1 Useful 64-bit Notations

These are some mathematical notions concerning calculation of huge values.

Algebra Rules of Power Numbers

We use Giga, Mega and Kilo quite often. A few of us remember that 1 K is 2^{10}, 1 M is 2^{20} and 1 G is 2^{30}.

To manipulate numbers such as these, which are conveniently expressed as powers of two, two simple algebraic rules are all that is needed:

Two Rules to Know

These are the two rules to remember:

1. $2^a \times 2^b = 2^{a+b}$ or $16^a \times 16^b = 16^{a+b}$
2. $(2^a)^b = 2^{a \times b}$ or $(16^a)^b = 16^{a \times b}$

Some Useful High Values

To use these rules, it helps to memorize the major powers of two, as:

Table 16. Huge Values

Terminology	Power of 2	Power of 16	Power of 10
1 KB (kilobyte)	2^{10}	4×16^2	$1024 \neq 10^3$!
1 MB (megabyte)	2^{20}	16^5	$\cong 10^6$
1 GB (gigabyte)	2^{30}	4×16^7	$\cong 10^{19}$
1 TB (terabyte)	2^{40}	16^{10}	$\cong 10^{12}$
1 PB (petabyte)	2^{50}	4×16^{12}	$\cong 10^{15}$
1 EB (exabyte)	2^{60}	16^{15}	$\cong 10^{18}$

Note that the values to power of 10 are approximate values. That's why, for manipulating huge values, we only have to remember the values of power of two.

This is the table that will help you:

Table 17. Powers of Two to Know

Power of 2	Decimal Values	Terminology
2^2	4	
2^3	8	

Power of 2	Decimal Values	Terminology
2^4	16	
2^5	32	
2^6	64	
2^7	128	
2^8	256	
2^9	512	
2^{10}	1024	1 KB
2^{20}	1,048,576	1 MB
2^{30}	1,073,741,824	1 GB
2^{40}	1,099,511,627,776	1 TB
2^{50}	11,258,990,684,262	1 PB
2^{60}	1,152,921,504,606,846,976	1 EB

A.2.2 AIX 64-bit Examples

Let's look at some examples demonstrating how easy it is to manipulate huge values with the concepts discussed above.

How Much Memory Can VMM Handle

In AIX 64-bit address implementation, segment numbers are stored in a 32-bit field. So, we would like to know how many segments and how much memory VMM can handle.

1. How many segments can VMM handle? Four Giga segments:

   2^{32} segments = 2^2 x 2^{30} = 2^2 Giga = 4 Giga

2. How much memory can VMM handle?

   1 EB =

   4 Giga segments =

   4 Giga x 256 MB =

   4 x 2^{30} x 256 x 2^{20} = 4 x 256 x 2^{50} =

   2^2 x 2^8 x 2^{50} = 2^{60} = 1 exabyte of memory!

How Many Segments Can the Application Text Section Use

The AIX 64-bit environment defines that application text, data, BSS, and the heap will be stored from segment number 0x10 to segment number 0x6fffffff. So, let's see how many segments are available between 0x0 and 0x6fffffff.

1. We read 0x6fffffff as 0x6 fff ffff.
2. fff ffff represents 16^7 segment numbers.

   $16^7 = (2^4)^7 = 2^{28} = 2^8 \times 2^{20} = 2^8$ Mega segments.
3. Since we have 0x6 fff ffff, it represents:

   6×2^8 Mega segments=

   6 x 256 Mega segments.

If you have an odd number (like 7), you should keep the result as: 7 x 256. You should not compute the result. Keeping it like this is useful if we want to do further calculations such as how much memory (6 x 256 Mega) segments represent:

6 x 256 Mega segments =

$6 \times 256 \times 2^{20}$ x 256 MB =

$6 \times 2^8 \times 2^{20} \times 2^8 \times 2^{20}$ bytes =

$6 \times 2^{16} \times 2^{40}$ bytes =

$6 \times 2^6 \times 2^{50}$ bytes =

6 x 64 petabytes =

384 petabytes or 384,000 terabytes of memory!

A.2.3 A Fast Way to Read 64-bit Values

When you see your first 64-bit hexadecimal values, your eyes could get lost reading 16 digits. We are still not used to read 16 hexadecimal digits. They are not easy to read and not easy to write, for example:

```
0x26FFE5A97410BC43
```

At first sight, you can hardly say how much it represents except that it is a huge value. This is fact that everyone knows, due to the 64-bit environment. However, there is a way that you can get a first idea about how much this number could represent.

Reading 64-bit Hexadecimal Values

If you look at the powers of 16, also see Table 16 on page 144, we have previously said that these values were not very helpful because they can be

computed with values to the power of 2. That is true except that three of them have an interesting feature such as:

- 1 megabyte is 16^5
- 1 terabyte is 16^{10}
- 1 exabyte is 16^{15}

Let's group our previous 64-bit hexadecimal value by five digits. Now, at the first sight, we can say that we have:

0x2 6FFE5 A9741 0BC43

2 EXABYTE+<tag1>+ 5 TERABYTE+<tag2>+ 1 MEGABYTE+<tag3>

In this way, you can not get the real decimal values: we are using EB, TB, GB, and MB which are not powers of ten. However, it gives you an idea if the number or quantity you are regarding is an average of Exa or Tera or Mega.

Table 18. KB, MB, GB, TB, PB, EB Decimal Values

Terminology	Decimal Value
1 KB	1,024
1 MB	1,048,576
1 GB	1,073,741,824
1 TB	1,099,511,627,776
1 PB	11,258,990,684,262
1 EB	1,152,921,504,606,846,976

A.2.4 A Useful Calculator

Since most calculators will be little help like `xcalc`, there still is the AIX desktop application "Calculator" within Desktop_Apps under Application Manager.

This calculator can do all the calculations previously described inside this appendix.

However, to get `dtcalc` running, you should have an AIX graphic display available running the AIX Common Desktop. It is not always the case: 64-bit SMP machines rarely have a graphic console.

Appendix B. Process Address Space Layout

This appendix describes how the Virtual Memory Manager (VMM) handles memory for 32-bit and 64-bit process execution. With this address space layout description, 32-bit and 64-bit application's capabilities and limits, such as large program support, number of shared memory segments and large file support, are easily explained.

B.1 Terminology

VMM defines different elementary units of memory differentiated by the type of their content. This classification and associated terminology remains the same for 64-bit process address layout.

Text segment The text segment contains the object code of the process (executable instructions). It is shared read-only by all processes executing the same program.

Library segment This segment contains instructions that perform functions that will be used by many processes.

User block Small and fixed-size memory space where user information is stored and maintained by the kernel (not accessible by user process).

Kernel stack Small and fixed-size memory space allocated from the kernel's memory space that serves to store local variable and subroutine return information when a process makes a call to a kernel routine. The kernel maintains kernel stack space for each process.

User stack Memory space used by processes to store process private functions/subroutines calls and associated information (subroutine return code, arguments list).

User data Memory space used by processes to store data. Data are stored into csect sections, depending on their type:

Heap section: dynamically allocated memory space (`malloc` calls).

BSS section: un-initialized variables.

Data section: initialized variables and constants.

B.2 32-Bit Process Address Space Layout

This part describes how the Virtual Memory Manager (VMM) handles memory for 32-bit process execution.

B.2.1 32-bit Address Space Layout

The layout described in Table 19 on page 150 is the segment register usage convention for 32-bit processes.

Table 19. 32-bit Address Space Implementation

Segment Number	Use	Size	Available Number
0x0	Kernel	256 MB	1
0x1	Process text	256 MB	1
0x2	Process private	256 MB	1
0x3 0xC	shmat/mmap (Attached with explicit compiler options)	2.56 GB	10
0xD	Shared library text	256 MB	1
0xE	shmat/mmap	256 MB	1
0xF	Shared library data	256 MB	1

The kernel and shared-library text segment are shared by all processes and have restricted access when executing in the user protection domain.

The process text segment is held by segment 1.

The process private segment, containing user block, kernel stack, user stack and user data regions, is held by segment 2. We have mentioned user block and kernel stack, small and fixed-size areas, because we cannot ignore their existence. But no user process or tunable system parameters can modify these areas dedicated to the system. For that reason, we can now ignore them and say that segment 2 is mainly used for user stack and user data regions.

The segment registers 3 through 12 may be loaded (using interfaces such as `shmat` or `mmap`) to provide access to files or shared memory data.

These conventions place constraints on the size of the user data and the user stack regions, given that they share a single 256 MB segment: segment 2 (see figure Figure 4 on page 151).

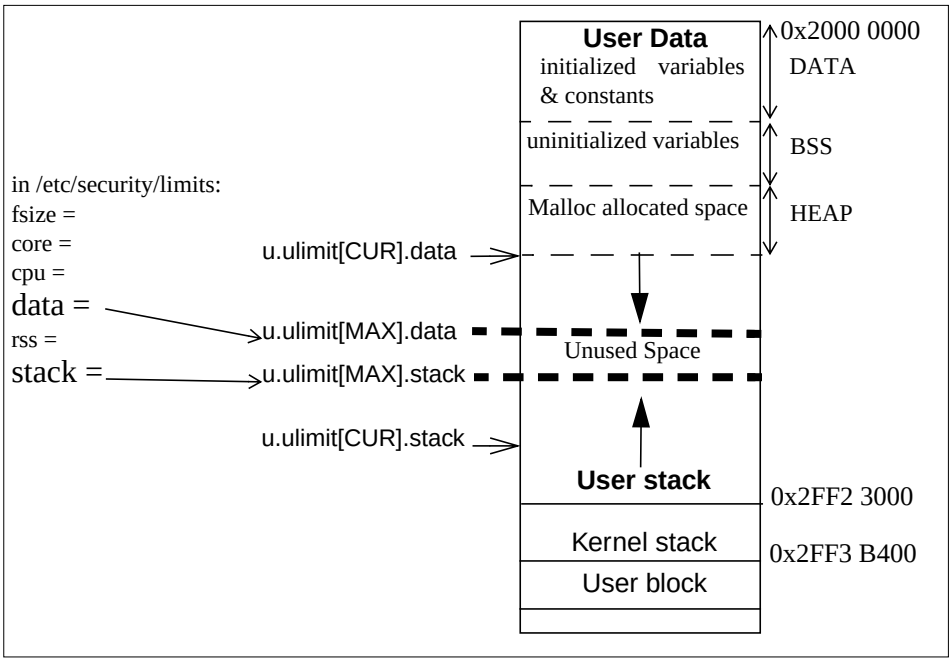


Figure 4. Process Private Segment (Segment 2)

B.2.2 32-bit Large Address Space Layout

A large program address space model is supported to handle programs with requirements for large amounts of uninitialized data. In this model, the process user data is relocated to begin at segment 3 for as much data as the program needs to a maximum of eight segments. The program can therefore have up to two GB of data.

Other aspects of the program address space remain unchanged. The user stack, kernel stack and u-block continue to reside in segment 2. As a result of this organizational scheme, the user stack is still limited by the size of segment 2 (see Figure 5 on page 152).

Note that this model still does not address the need for programs with executables greater than one segment in size (more than 256 MB of executable instructions). This is not an ordinary programming issue (excessive program complexity) and does not normally impact programs. To

enable the large address-space model, use the `-bmaxdata` flag of the `ld` command.

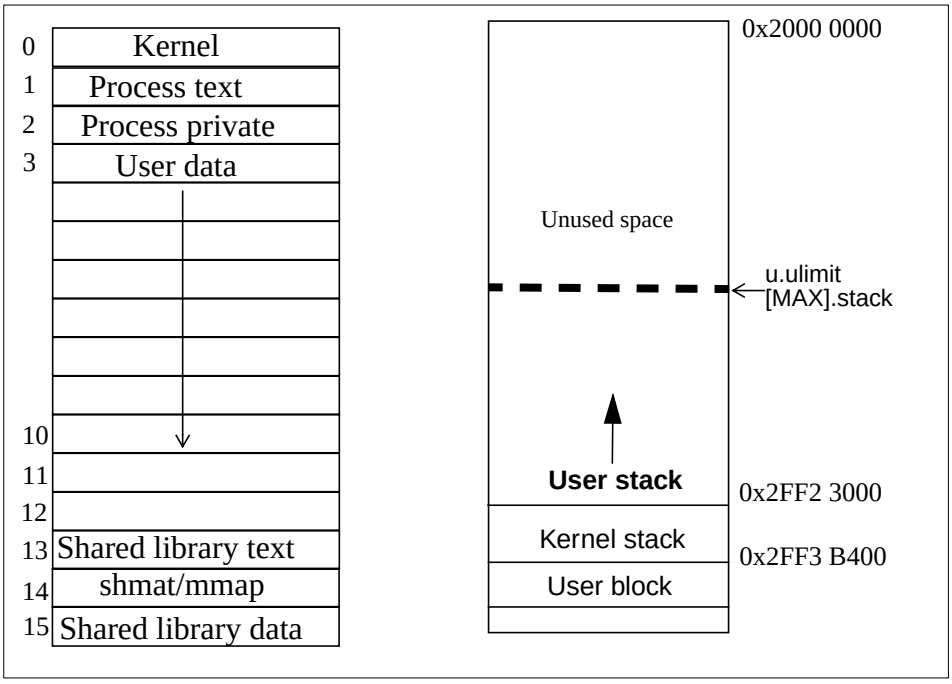


Figure 5. Process Private Segment (Segment 2) for Large Programs

For example, to link a program that will have the maximum eight segments reserved to it, the following command line could be used:

```
cc sample.o -bmaxdata:0x80000000
```

The number 0x80000000 is the number of bytes, in hexadecimal format, equal to eight 256 MB segments. Although larger numbers can be used, they are ignored because a maximum of eight segments can be reserved. The value following the `-bmaxdata` flag can also be specified in decimal or octal format. Use the `-bmaxdata` option only if the program needs very large data areas.

The maxdata value is stored in the `o_maxdata` field of the XCOFF header. The Extended Common Object File Format (XCOFF) is the object file format for AIX.

After placing the program's initialized and uninitialized data in segment 3 and beyond, the system computes the break value. The break value defines the

end of process' static data and the beginning of its dynamically allocatable data. Using the `malloc`, `brk`, `sbrk` subroutine, the process is free to move the break value toward the end of the segment identified by the maxdata field in the a.out header file.

For example, if the value specified in the maxdata field in the a.out header file is 0x80000000, then the maximum break value is up to the end of segment 10 (0xAFFFFFFF). The `brk` subroutine extends the break across segment boundaries but not beyond the point specified in the maxdata field.

Using the following command, you can do the following:

1. Patch large programs to use large data without relinking them:

   ```
   /usr/bin/echo '\0200\0\0\0' | /bin/dd of=<executable_file_name> bs=4
   count=1 seek=19 conv=notrunc
   ```

2. Patch program to use small address space model:

   ```
   /usr/bin/echo '' | /bin/dd of=<executable_file_name> bs=4 count=1
   seek=19 conv=notrunc
   ```

B.2.3 Paging Space Considerations for Large Programs

Programs with large data spaces require a large amount of paging space. For example, if a program tries to access every page in its address space, the system must have two GB of paging space. The operating system page-space monitor terminates processes when paging space runs low.

AIX uses two modes for paging space allocation: the late and the early paging space allocation. These policies can dramatically reduce or increase your paging space size requirement. This issue becomes important when your program handles large data areas like the 64-bit address capability. Therefore, AIX paging space policies will be discussed in Appendix B.3.2, "Large Data and Paging Space Allocation Policies" on page 158.

B.2.4 32-bit Address Space Layout and Inter-Process Communication

Prior to AIX 4.2, AIX used the segment registers from 0x3 to 0xC for shared memory regions, only 10 could be simultaneously attached to the process. Then, in AIX V4.2, a new segment (segment 0xE) became available for process use. This allows applications running with AIX V4.2 to attach 11 shared memory segments.

A single shared memory region, whatever its size may be, always consumed a 256 MB region of its address space. The rest of the address space not used within this 256 MB region is left practically inaccessible to the application.

This caused problems for application developers who wanted to use the total address space with shared memory regions of various sizes. It has been a concern for middleware developers, such as database vendors, who develop their software with common code and port it to various platforms.

In AIX V4.2, an extended `shmat` capability is available. If an environment variable *EXTSHM=ON* is defined, then processes executing in that environment will be able to create and attach more than 11 shared memory segments. The segments can be of size from one byte to 256 MB. The process can attach these segments to the address space for the size of the segment. Another segment could be attached at the end of the first one in the same 256 MB region. The address at which a process can attach will be at page boundaries, that is, a multiple of *SHMLBA_EXTSHM* bytes.

This new constant, *SHMLBA_EXTSHM*, is defined to be 0x1000 (page boundaries) for applications that need this value. The old constant, *SHMLBA,* is treated as 0x1000 internally, if the environment variable is set. Otherwise, the value of *SHMLBA* is interpreted to be 0x10000000 (segment boundary).

The maximum address space available for shared memory with or without the environment variable and for memory mapping is 2.75 GB. An additional segment register, 0xE, is available for user data, so the address space is 0x30000000 to 0xE0000000. However, a 256 MB region starting from 0xD0000000 is used for shared libraries and is therefore unavailable for shared memory regions or *mmapped* regions.

There are some restrictions on the use of the extended `shmat` feature. These shared memory regions can not be used as I/O buffers where the unpinning of the buffer occurs in an interrupt handler. The restrictions on the use are the same as that of `mmap` buffers.

However, the smaller region sizes are not supported for mapping files. Regardless of whether *EXTSHM=ON* or not, mapping a file will consume at least 256 MB of address space.

The *SHM_SIZE* `shmctl` command is not supported for segments created with *EXTSHM=ON*. The system call, `shmctl()`, is used by a process to remove, modify the attributes or obtain information about a shared memory region. When the command value is *SHM_SIZE*, the size of the shared memory region is to be resized. This feature is not supported with the page segment boundary for shared memory segments.

The environment variable provides the option of executing an application either with the additional functionality of attaching more than 11 segments

when *EXTSHM=ON* (with no need to recompile the application), or the higher-performance access to 11 or fewer segments when the environment variable is not set.

To take advantage of the new shared functionality, all the processes should have the new environment variable set in their environment. However, sharing of memory regions between processes with and without the new environment variable is supported.

When a shared memory region is created with the environment variable set:

- A process using the new environment variable will be able to attach the region using the new mechanism.
- A process using the old method (without the environment variable defined) will be able to attach to the region in the old method. This will consume a 256 MB area of the address space irrespective of the size of the shared memory region.

When a shared memory region is created without the environment variable set:

- A process using the new environment variable will be able to attach the shared memory region, but in the old mode consuming a 256 MB area of the address space, irrespective of the size of the region.
- A process not using the new environment variable will be able to attach to the region in the old fashion consuming a 256 MB area of the address space, irrespective of the size of the region.

B.3 64-Bit Process Address Space Layout

This part describes how the Virtual Memory Manager (VMM) handles memory for 64-bit execution programs.

B.3.1 64-bit Address Space Layout

The layout described in Table 20 on page 157 and Table 21 on page 158 is the segment register usage convention for 64-bit processes.

Terminology like text segment, user block and other terms are defined in Appendix B.1, “Terminology” on page 149. Also, the 64-bit address space layout description, described in Appendix B.2, “32-Bit Process Address Space Layout” on page 150, gives the changes to the 32-bit address space layout. To understand what follows, a look at both of these sections is useful.

The kernel section, previously held by segment 0, is now handled by two segments, 0 and 1.

The process text section, previously held by segment 1 and therefore given the program text section size limit of 256 MB, is now handled by segments 10-0x6FFFFFFF (384 000 TB), but it is still limited to 256 MB.

The user stack area, previously held by segment 2, is relocated to the segments 0xF0000000-0xFFFFFFFF (64 000 TB). The user stack area is still limited to 256 MB. Note that user block and kernel stack remain in segment 2 (process private).

The user data area, previously held by segment 2 or segments 3-C, is now handled with the text section by segments 10-0x6FFFFFFF.

Segments 0x80000000-0x8FFFFFFF may be used to load private copies of shared libraries.

Table 20. 64-bit Address Space Layout (Part 1)

Segment Number	Use	Size	Available Number
0x0000 0000	Kernel	256 MB	1
0x0000 0001	Kernel	256 MB	1
0x0000 0002	Process private	256 MB	1
0x0000 0003 0x0000 000C	shmat/mmap (Fixed addresses to be specified within 64-bit applications)	2.56 GB	10
0x0000 000D	Loader use	256 MB	1
0x0000 000E	shmat/mmap (idem)	256 MB	1
0x0000 000F	Loader use	256 MB	1
0x0000 0010 0x1000 0000 0x2000 0000 0x3000 0000 0x4000 0000 0x5000 0000 0x6FFF FFFF	Application text, application user data (BSS, Data, Heap)	384 000 TB	6 x 256 x 2^{20} segments

Table 21. 64-bit Address Space Layout (Part 2)

Segment Number	Use	Size	Available Number
0x7000 0000 0x7FFF FFFF	Default shmap/mmap	64 000 TB	256 x 2^{20} segments
0x8000 0000 0x8FFF FFFF	Private load	64 000 TB	256 x 2^{20} segments
0x9000 0000 0x9FFF FFFF	Shared library text and data	64 000 TB	256 x 2^{20} segments
0xA000 0000 0xB000 0000 0xC000 0000 0xD000 0000 0xEFFF FFFF	Reserved for system use Reserved for system use Reserved for system use	256 000 TB	2^{30} segments
0xF000 0000 0xFFFF FFFF	Application user stack	64 000 TB	256 x 2^{20} segments

B.3.2 Large Data and Paging Space Allocation Policies

AIX uses two modes for paging space allocation. The setting of the *PSALLOC* environment variable determines the paging space allocation mode.

The default mechanism is the late paging space allocation algorithm. The user can switch to an early paging space mode by setting the value of *PSALLOC* environment variable to early:

```
export PSALLOC=early
```

If the *PSALLOC* environment variable is not set, is set to null, or is set to any value other than early, the system uses the default late allocation algorithm.

The default late allocation algorithm for memory and paging space allocation assists in the efficient use of disk resources and supports applications of customers who want to take advantage a sparse allocation algorithm for resource management.

The late allocation algorithm does not reserve paging space when pages are touched. Under late allocation algorithm, a paging slot is allocated to a page of virtual memory only when that page is first read from or written into. That is the first time that the page's content is of interest to the executing program.

Some programs allocate large amounts of virtual memory and then use only a fraction of the memory. Many programs exploit late allocation by allocating virtual memory address ranges for maximum-sized data structures such as sparse vectors or matrixes as data structures, and then only using as much of the structure as the situation requires.

The pages of the virtual memory address range that are never accessed never require real-memory frames or paging space slots. This technique does involve some degree of risk because it is possible to overcommit resources. If all the programs running in a machine happened to encounter maximum-size situations simultaneously, paging space might be exhausted. Some programs might not be able to continue to completion.

The early allocation algorithm is intended for use in installations where this situation is likely, or where the cost of failure to complete is intolerably high. This algorithm causes the appropriate number of paging space slots to be allocated at the time the virtual memory address range is allocated, for example, with the `malloc` command. If there are not enough paging space slots to support the `malloc`, an error code is set.

Though the normal recommendation for paging space size is at least twice the size of the system's real memory (up to a memory size of 256 MB), the recommendation for systems that use *PSALLOC=early* is at least four times real memory size. As an example, at one time the AIX Windows server required 250 MB of paging space when run with early allocation policy.

For memories larger than 256 MB, and when using the default allocation, we recommend:

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

As an example, a machine with 2048 MB memory should have 2752 MB paging space size. A machine with 16 GB memory should have 20672 MB paging space size with a late paging space algorithm (more than 20 GB of disk space).

The paging space required for any application depends on how the application is written and how it is run. Certain applications can use extreme amounts of paging space if they are run in early allocation mode: for 64-bit applications, this could be a very expensive execution mode.

Because a 64-bit application's data size can grow to almost twice that of its 32-bit foot print, more paging space is needed. An application that had a virtual memory size of 512 MB on a 32-bit system will be closer to 1 GB when migrated to 64-bit.

The early paging space allocation for 64-bit applications or 32-bit large programs, should be chosen after the user has analyzed the virtual storage requirements of the machine workload and allocated paging space to accommodate them.

B.3.3 64-bit Address Space Layout and Inter-Process Communication

64-bit applications should explicitly specify a segment address from 3 to C to allocate `shmat/mmap` areas. By default, the system will allocate another segment address pool to a 64-bit application. So, interoperability between 64-bit and 32-bit application through memory shared areas must be explicitly designed and written within the 64-bit application source code:

- If no fixed address is specified, then allocation takes place from the `shmat/mmap` pool at ESIDs (Effective Segment Identifiers) 0x7000 0000 to 0x7FFF FFFF.
- If a fixed address is specified by a user, then the allocation will be allowed as long as the ESID is less than 0x8000 0000 and is not the ESID 0-2, 13 or 15.

Appendix C. A Sample Thread Program

This sample creates five threads. One is for the signal handler, and the other four threads are for writing shared-memory data to files. Reading shared memory created by a 64-bit application is no different than with a 32-bit application.

```
#include <pthread.h>
#include <stdio.h>
#include <errno.h>
#include <signal.h>
#include <fcntl.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define  SHMEM_SIZE  256*1024*1024  /* 256 MB */
#define  SH_MAX      12
#define  MAX_NAME    64

void *sig_th(void *);
void *read_th(void *);

int   mem_id[SH_MAX];
char *sh_data[SH_MAX];

void main(void) {
    pthread_attr_t  attr, sig_attr;
    pthread_t       th_read[SH_MAX];
    pthread_t       th_sig;
    int             i,j,rc;

    if ((rc = pthread_attr_init(&attr)) != 0) {
        printf("pthread_attr_init error, rc = %d\n", rc);
        exit();
    }
    rc = pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
    if (rc != 0) {
        printf("pthread_setdetachstate error, rc = %d\n", rc);
        exit();
    }
    if ((rc = pthread_attr_init(&sig_attr)) != 0) {
        printf("pthread_attr_init error, rc = %d\n", rc);
        exit();
    }
    rc = pthread_attr_setdetachstate(&sig_attr, PTHREAD_CREATE_DETACHED);
    if (rc != 0) {
        printf("pthread_setdetachstate error, rc = %d\n", rc);
```

```
        exit();
    }
    rc = pthread_create(&th_sig, &sig_attr, sig_th,(void *)NULL );
    if (rc != 0) {
        printf("pthread_create(sig_th) error, errno = %d\n" , errno);
        exit();
    }
    for (i = 0; i < SH_MAX; i +=3) {
       if ((rc=pthread_create(&th_read[i],&attr,read_th,(void *)i))!= 0) {
            printf("pthread_create(read_th) error, errno = %d\n" , errno);
        }
    }
    for (i = 0; i < SH_MAX; i +=3 )  {
        pthread_join(th_read[i], NULL);
    }
    for (i = 0; i < SH_MAX; i +=3 )  {
        printf("shmat = %p ",sh_data[i]);
        printf("sh_data[%02d] = ",i);
        for (j = 0; j < 16; j++) {
          printf("%c",*(sh_data[i] + j) );
        }
        printf("\n");
        shmdt(sh_data[i]);
    }
    pthread_attr_destroy(&attr);
    printf("Normal end\n");
    exit();
}                                   /* End of Main                      */

void *sig_th(void *dummy ) {
    sigset_t set;
    int      sig;
    int      rc;

    sigemptyset(&set);
    sigaddset(&set, SIGINT);
    sigaddset(&set, SIGUSR1);

    rc = sigthreadmask(SIG_BLOCK, &set, NULL);
    if (rc != 0) {
        printf("sigthreadmask error\n");
        pthread_exit(NULL);
    }
    while(1) {
        rc = sigwait(&set, &sig);
        if (rc != 0) {
            printf("sigwait error\n");
```

```
            pthread_exit(NULL);
        }
        switch(sig) {
            case SIGINT:
                printf("sig_handler recieve SIGINT\n");
                sigdelset(&set, SIGINT);
                rc = sigthreadmask(SIG_BLOCK, &set, NULL);
                if (rc != 0) {
                    printf("sigthreadmask error\n");
                    pthread_exit(NULL);
                }
                break;
            case SIGUSR1:
                printf("sig_handler recieve SIGUSR1\n");
                break;
            default:
                printf("sig_handler recieve SIG=>%d", sig);
                break;
        }               /* End of witch(sig)  */
    }                   /* End of while(1)    */
    pthread_exit(NULL);
}                     /* End of sig_th      */

void  *read_th(void *i)  {
    extern int   mem_id[SH_MAX];
    extern char *sh_data[SH_MAX];
    int          th_no;
    int          fd;
    char         fname[MAX_NAME +1];

    th_no = (int)i;
    bzero(fname, MAX_NAME +1);
    sprintf(fname, "/save_data/test.data%02d", th_no);
    if((fd = open(fname, O_RDWR | O_CREAT)) < 0 ) {
      printf("open error %s\n", fname);
      pthread_exit(NULL);
    }
    mem_id[th_no] = shmget((key_t)(0x20000 +
(int)th_no),SHMEM_SIZE,IPC_EXCL);
    if (mem_id[th_no] < 0) {
        printf("shmget[%d] error \n",th_no);
        pthread_exit(NULL);
    }
    if ((sh_data[th_no] = shmat(mem_id[th_no],0,0)) == (char *)-1L) {
        printf("shmat[%d] error \n",th_no);
        pthread_exit(NULL);
    }
```

```
    printf("thread = %d\n",th_no);
    if (write(fd, sh_data[th_no], SHMEM_SIZE)  < 0) {
        printf("write error\n");
    }
    close(fd);
    pthread_exit(NULL);
}                       /* End of read_th     */
```

This program will show output as follows:

```
thread = 0
thread = 6
thread = 3
thread = 9
shmat = 30000000 sh_data[00] = *00-256 MB area*
shmat = 40000000 sh_data[03] = *03-256 MB area*
shmat = 50000000 sh_data[06] = *06-256 MB area*
shmat = 60000000 sh_data[09] = *09-256 MB area*
Normal end
```

Appendix D. AIX Linker/Loader

In this appendix, the most important parameters controlling the linker/loader are described. You should be aware of some of these because they control the executable's capabilities, such as extending the user stack beyond segment 2 or defining the maximum segment number your application is able to attach.

If you need more information about user stack definition, see Appendix B.2, "32-Bit Process Address Space Layout" on page 150 and Appendix B.3, "64-Bit Process Address Space Layout" on page 155.

Table 22. Important Linker/Loader Options

Command Line Option	Description
-bmaxdata:*bytes*	Specifies the maximum amount of space to reserve for the program data segment, for programs where the default size of this region is a constraint. If your program allocates large arrays, statically or dynamically, specify `-bmaxdata` when linking the program. The resulting executable uses the large data model and can have a data region larger than a single segment, up to two GB (in 32-bit mode).
-bmaxstack:*bytes*	Specifies the maximum of amount of space to reserve for the program stack segment, for the program where the default size of this region is a constraint. If the program has large amounts of automatic data, or otherwise exceeds the soft limit on stack size for a program, specify `-bmaxstack` to define the soft limit up to 256 MB when linking the program.
-bdynamic -bshared -bstatic	`-bdynamic` and `-bshared` are synonymous. When `-bstatic` is in effect, shared objects are statically linked into the output file. When `-bdynamic` is in effect, shared objects are linked dynamically.

The `-bmaxdata` and `-bmaxstack` parameters are involved in the large program support capability and large file support capability of your application.

These options are passed directly to the `ld` command and are not processed by compilers.

Appendix E. Migration Documentation

This appendix contains information from two existing documents that discuss 32-bit to 64-bit application migration.

E.1 C for AIX: 32-bit to 64-bit Migration Considerations

This section outlines various portability considerations in moving programs from 32-bit to 64-bit mode.

- Constants
- Assignment of Long Types to Integer and Pointers
- Structure Sizes, Alignment, and Bitfields
- Miscellaneous
- Interlanguage Calls with FORTRAN

Source of this Material

The following text originates from the HTML documentation for the C for AIX compiler and may be found at:

`file:/usr/vac/html/en_US/ref/rucl64mg.htm`

or on the IBM Intranet:

`http://w3.torolab.ibm.com/~batthis/C_Manual/ref/rucl64mg.htm`

E.1.1 Constants

The limits of constants changed. This table shows changed items in the limits.h header file, their hexadecimal value, and decimal equivalent. The equation gives an idea of how to construct these values.

Table 23. Changed Limits in limits.h

Type	Hexadecimal	Equation	Decimal
Signed long min (LONG_MIN)	0x8000000000000000L	$-(2^{63})$	-9,223,372,036,854,775,808
Signed long max (LONG_MAX)	0x7FFFFFFFFFFFFFFFL	$2^{63}-1$ (-LONG_MIN-1)	+9,223,372,036,854,775,807
Unsigned long max (ULONG_MAX)	0xFFFFFFFFFFFFFFFFUL	$2^{64}-1$	+18,446,744,073,709,551,616

In C, type identification of constants follows explicit rules. However, programs that use constants exceeding the limit (relying on a 2's complement representation) will experience unexpected results with the LP64 model. This

is especially true of hexadecimal constants and unsuffixed constants, which are more likely to be extended into the 64-bit long type. In each case, constants typed as long integers will have different values depending on whether the program was compiled in 32- or 64-bit mode.

Problematic behaviors will generally occur at boundary areas such as:

- constant >= `UINT_MAX`
- constant < `INT_MIN`
- constant > `INT_MAX`

Some examples of undesirable boundary side effects are:

Table 24. Undesirable Boundary Side Effects

C Constant Assigned to Long	32-bit Mode	64-bit Mode
-2,147,483,649 (INT_MIN-1)	+2,147,483,647	-2,147,483,649
+2,147,483,648 (INT_MAX+1)	-2,147,483,648	+2,147,483,648
+4,294,496,726 (UINT_MAX+1)	0	+4,294,967,296
0xFFFFFFFF (UINT_MAX)	-1	+4,294,496,295
0x100000000 (UINT_MAX+1)	0	+4,294,967,296
0xFFFFFFFFFFFFFFFF (ULONG_MAX)	-1	-1

Currently, the compiler gives warning messages for out of range when attempting to assign a value larger than the designated range into a long type. The warning message is:

```
1506-207 (W) Integer constant 0x100000000 out of range.
```

This warning message may not appear for every case.

Unsuffixed constants are more likely to become 64-bit length if they are in hexadecimal. To avoid this possibility, explicitly suffix all constants that have the potential of impacting constant assignment.

When you bit left-shift a 32-bit constant and assign it into a long type, signed values are sign-extended and unsigned values are zero-extended. The examples in the table below show the effects of performing a bit-shift on both 32- and 64-bit constants, using the following C code segment:

```
long l=constantL<<1;
```

The following is a table that shows values before and after the bit-shift:

Table 25. Constant Values after Bit-Shift

Initial Constant Value	Constant Value after Bit-Shift	
	32-bit	64-bit
0x7FFFFFFFL (INT_MAX)	0xFFFFFFFE	0xFFFFFFFE
0x80000000L (INT_MIN)	0	0x100000000
0xFFFFFFFFL (UINT_MAX)	0xFFFFFFFE	0x1FFFFFFFE

E.1.2 Assignment of Long Variables to Integers and Pointers

In 32-bit mode, int, long and pointer types have the same size and can be freely assigned into each other. In extended mode, integer and long types can be assigned into pointer types, and vice versa, with only a warning. In ANSI mode, assignment between integral and pointer types will generate a severe level message. The following warning message is generated in extended mode:

```
1506-068 (W) Operation between types "int" and "int*" is not allowed.
```

Using int and long types in expressions and assignments can lead to implicit conversion through promotions and demotions, or explicit conversions through assignments and argument passing. The following should be avoided:

- Using integer and long types interchangeably, leading to truncation of significant digits or unexpected results.
- Passing long arguments to functions expecting type int.
- Exchanging pointers and int types, causing segmentation faults.
- Passing pointers to a function expecting an int type, resulting in truncation.
- Assignment of long types to float, causing possible loss of accuracy.

Assigning a long constant to an integer will cause truncation without warning. For example:

```
int i;
long l=2147483648; /* INT_MAX+1*/
i=l;
```

What will be the value of i? We know that `INT_MAX+1` is 2147483647+1 (0x80000000), which becomes `INT_MIN` when assigned into a signed type. Truncation occurs because the highest bit is treated as a sign bit. The rule here is that there will be a loss of significant digits.

Similar problems occur when passing constants directly to functions, and in functions that return long types. Making explicit use of the L and UL suffix will avoid most, but not all, problems. Alternately, you can avoid accidental conversions by using explicit prototyping. Another good practice is to avoid implicit type conversion by using explicit type casting to change types.

E.1.3 Structure Sizes, Alignments, and Bitfields

Structures may face potential porting problems.

The LP64 specification changes the size, member and structure alignment and bit fields of all structures that are recompiled in 64-bit mode. Structures with long integers and pointers will at least double in size under 64-bit mode, depending on the alignment. Sharing data structures between 32- and 64-bit processes is no longer possible, unless the structure is devoid of pointer and long types. Unions that attempt to share long and int types, or overlay pointers onto int types will now be aligned differently or be corrupted. In general, all but the simplest structures must be checked for alignment and size dependencies.

The alignment for `-qalign=full`, power or natural changes for 64-bit mode. Structure members are aligned on their natural boundaries. Long types and pointer types are word-aligned in 32-bit mode, and doubleword aligned in 64-bit mode. Additional spaces could be used for padding members. Structures and unions will double in size if they contain long and pointer types.

The alignment for `-qalign=twobyte, packed, or mac68k` do not change for 64-bit mode. Long and pointer types in `twobyte` and `mac68k` alignment are still aligned by the halfword. They remain aligned by bytes in packed mode.

Structures are aligned according to the strictest aligned member. This remains unchanged from 32-bit mode. Because of the padding introduced by the member alignment, structure alignment will not be exactly the same as in the 32-bit mode. This is especially important when you have arrays of structures that contain pointer or long types. The member alignment will change, most likely leading to the structure alignment to change to doubleword alignment (if there are no long long types, double types and long double types).

Structure bit-fields in 32-bit mode are limited to 32 bits, and can be of type signed int, unsigned int or plain int. Bit fields are packed into the current word. Adjacent bit fields that cross a word boundary will start at storage unit. This storage unit is a word in power and full alignment, halfword in the `mac68k` and

twobyte alignment, and byte in the packed alignment. In 64-bit mode, bitfields will be limited to at most 64-bit in size, and can be type signed int, unsigned int, plain int, long, and unsigned long. If it is typed long, or unsigned long, the maximum size is 64 bits, and the storage unit in power or full alignment is double word. The alignment will remain the same in all other alignment modes. This means that adjacent declarations of bit fields of type long can now be contained into one storage unit. Since long bitfields were not permitted before in 32-bit mode, this is usually not a portability problem.

E.1.4 Miscellaneous Issues

The sizeof operator will now return size_t, which is an unsigned long.

The length of the integer required to hold the difference between two pointers is ptrdiff_t, and is a signed long type.

Masks will generally lead to different results when compiled in 64-bit mode from their 32-bit mode behavior.

Many include files have pointers and structures in them, and their inclusion in 64-bit mode will change the size of your data section even if your program does not use structures and pointers explicitly.

E.1.5 Interlanguage Calls with FORTRAN

A significant number of applications use C and FORTRAN together, by calling each other or sharing files. Such applications are among the early candidates for porting to 64-bit platforms for its abilities to solve larger mathematical models. Experience shows that it is easier to modify data sizes/types on the C side than the FORTRAN side of such applications. The following table lists the equivalent FORTRAN type in the different modes.

Table 26. FORTRAN Types

C type	FORTRAN 32-bit	FORTRAN 64-bit
int	LOGICAL	LOGICAL
unsigned int	INTEGER	INTEGER
signed long	INTEGER	INTEGER*8
unsigned long	LOGICAL	LOGICAL*8
pointer	INTEGER	INTEGER*8

A user must not mix XCOFF object formats from different modes. A 32-bit FORTRAN XCOFF can not mix with a 64-bit C XCOFF object and vice versa.

Since FORTRAN 77 usually does not have an explicit pointer type, it is common practice to use INTEGER variables to hold C pointers in 32-bit mode. In 64-bit mode, the user should use INTEGER*8 in FORTRAN. Fortran 90 does have a pointer, but it is unsuitable for conversion to the basic C types.

In 64-bit mode, FORTRAN will have a POINTER*8 that is 8 bytes in length as compared to their POINTER which is 4-bytes in length.

E.2 README: C for AIX

Following is the README.C file that is shipped with the current C for AIX compiler.

Source of this Material

The following text originates from the REACME.C file shipped with the C for AIX compiler and may be found at:

`file:/usr/vac/html/en_US/index.htm`

```
* COMPONENT_NAME: (vac) C for AIX compiler * * FUNCTIONS: README * * (C)
COPYRIGHT International Business Machines Corp. 1994,1996,1997 * * All
Rights Reserved. * Licensed Materials - Property of IBM * * US Government
Users Restricted Rights - Use, duplication or * disclosure restricted by
GSA ADP Schedule Contract with IBM Corp. *

================================================================================
These filesets install files to the /usr/vac directory. There are no
executables present in /usr/bin, due to conflicts with the existing C for
AIX and C Set ++ for AIX 3.1.4 products. This product will overwrite the C
for AIX 4.1 compiler in /usr/vac. If you have the C for AIX 4.1 compiler
installed, you should uninstall it before installing C for AIX 4.3. To
uninstall C for AIX 4.1, enter: installp -u vac installp -umemdbg

Add /usr/vac/bin to your path or invoke the compiler by full path name,
such as /usr/vac/bin/c89 helloworld.c.

The online hypertext documentation for this product can be found in
/usr/vac/html. Use a standard browser to view the online documentation by
entering the line below for the web page location.

file:/usr/vac/html/en_US/index.htm

This README has the following contents:
```

A. 64-bit migration issues.

i. Language-specific issues

ii. Operating System-specific issues

iii. Installation issues

B. Changes from CSet 3.1.4 to C for AIX 4.3

A. 64-bit migration issues -------------------------

i. Language-specific issues that can affect 64-bit compilation.

1. Unsuffixed numbers A number like 4294967295,(UINT_MAX) when parsed by the compiler will be typed signed long in 32-bit mode. This is OK since signed long is 4 bytes and mixing long with int is usually OK in 32-bit mode.

In 64-bit mode, this same number will choose signed long, which is 8-byte, leading to some operations like comparing the sizeof(4294967295) to return 8.

In general, suffix all constants. The fix for the above case is to write the number as 4294967295U. This will allow the compiler to pick unsigned int.

2. Undeclared functions return int. Any function that returns a pointer should be declared in 64-bit mode. Otherwise, the compiler will assume they return an int and truncate the resulting pointer even if you were to assign it into a valid pointer. Code such as:

int a=calloc(25);

which used to work fine in 32-bit mode, will now get a truncated pointer silently.

Even -qwarn64 can not trap this since we have an int to int assignment.

The fix is to include the correct header which is actually stdlib.h and not malloc.h as you might believe.

3. Structure with pointers and long types will change size and alignment in 64-bit mode. Any structure or unions with pointer or long types will change size and alignment in 64-bit mode. Some structures may not change size because they happen to fall on the exact 8 byte boundary even in 32-bit

mode.

4. __int64 is a long type in 64-bit mode. __int64 types will look like long long in 32-bit mode, but long in 64-bit mode. When they look like long long types, they will not be promoted into. When they look like long types, they will participate in promotion rules and arithmetic conversions.

5. Long bitfields in 64-bit mode may change in future In 32-bit mode, only in extended mode was non-integer bitfields tolerated (but not respected). But if you have long type bitfields in 64-bit mode, their exact alignment may change in future even if the bitfield size is under 32-bit.

6. Va_arg function parameters involving structures of 8 byte multiples passed by value If passing a structure of 8-byte multiple by value to a va_arg argument in 64-bit mode, the member values may not be accessed properly. This is a known limitation of the Operating System.

7. Beware of zero extension from unsigned int to unsigned long in 64-bit extended mode. When you zero extend an unsigned int that is negative, such as 0xFFFF FFFF, you will get a preservation of the bit pattern resulting in 0x0000 0000 FFFF FFFF. This can cause 64-bit results to become a large positive number.

ii. Operating System-specific issues that may affect your code:

1. time_t has changed size from 4.2 to 4.3 Library functions which take an argument of time_t or return type time_t may find type mismatches with your existing code in 32-bit mode because time_t has changed from long type in AIX 4.2 to int type in AIX 4.3.

2. MB_CUR_MAX has changed from int to size_t in AIX 4.3 MB_CUR_MAX is a macro defined in stdlib.h that calls _getmbcurmax(). This function now returns size_t which has always been unsigned long. It used to be prototype to return int in AIX 4.2.

3. setlocale in 64-bit mode If you have user locales defined, you need to recompile them in 64-bit mode using localedef. This generates a 32-bit and 64-bit versions of your locale file. Otherwise, calling setlocale in 64-bit mode will not find the user defined locale file. However, localedef in 4.3 supports only the charmap that came with the 4.3 distribution. If you need to use the charmaps that came from an older AIX distribution, you must explicitly copy them into your directory before using localedef with your custom locale definition file. Also, localedef by default is set up to use /bin/cc and /usr/bin/cc. The C for AIX 4.3 compiler does not setup links in /usr/bin or /bin in respect of CSet distributions. Since localedef requires the use of a 64-bit compiler, you need to run /usr/vac/bin/replaceCSet to replace the links with the C for AIX 4.3 links, run localedef, then run

restoreCSet to restore the links as they were before.

4. Make doesn't discriminate object formats. Make only discriminates on timestamp of files. The one case where this can cause problem is when you try to add the same named 32 and 64-bit object into the archive. Running make in 32-bit mode, then 64-bit mode will not update the 2nd object. Make only checks the timestamp of the first object it finds with the correct name.

5. int64 is typedefined in inttypes.h If you use int64 as a variable name, this is now a typedef in inttypes.h

6. Header file predefined types that are based on long There are many header file predefined types such as size_t and ptrdiff_t which remains the same type regardless of 32 to 64-bit mode. This presents subtle opportunity for differences when compiling the same code in different mode of the compiler. Although size_t remains the same type (unsigned long), the length of size_t will change between different modes of AIX. This can subtly cause library functions that return or take size_t to change behaviour in 32-bit to 64-bit mode. Specifically, sizeof will return an 8-byte value in 64-bit and a 4-byte value in 32-bit mode. The same applies to prtdiff_t which is signed long in both modes.

iii. Installation issues

1. Operating System level-specific files During installation, packaging will determine your AIX Operating System level. This will decide the correct OS-specific version of the following files to install: vac.cfg libhmd.a libhmu_r.a libhm.a libhmd_r.a libhu.a libhm_r.a libhmu.a libhu_r.a profiled/libhmd.a profiled/libhmu_r.a profiled/libhm.a profiled/libhmd_r.a profiled/libhu.a profiled/libhm_r.a profiled/libhmu.a profiled/libhu_r.a

B. Changes from CSet 3.1.4 to C for AIX 4.3
---

1. The main() function must be visible to IPA (i.e. in a .o compiled by IPA), thus if main() is in a .a, .S, or .so or any other file type, IPA will return an error.

2. Since IPA is responsible for calling the linker when it is active, the -# option (which shows the compilation steps without executing them) will not display anything after the IPA 2nd pass because IPA is not actually called. This is not a bug and is known expected behaviour.

3. #pragma options arch=suboption in source files is not supported.

4. C++ compile is not supported thus the C++ stanzas (xlC, xlC_r, etc.) cannot be used. The C stanzas (xlc, cc, etc.) will not compile C++ (.C, .cpp, .cxx) suffixes. The -+ option is also not supported.

5. Some utilities which were shipped with CSet 3.1.4 are not shipped with C for AIX 4.3. These are tcov, the Source Code Browser, and the HeapView Debugger. Thus the options -a, -ae, -qbrowse, and -qhd are not supported. However, some of the functionalities of the old HeapView Debugger can be found in the new memory debug routines accessible via the -qheapdebug option.

6. The default links in /usr/bin (xlc, cc ,c89, ... etc) will not be set up to point to /usr/vac/bin of C for AIX 4.3. This is done at the discretion of the product installer using replaceCSET. After using this script, /usr/bin/xlc (cc, c89, etc.) will point into /usr/vac/bin/xlc (cc, c89, etc.). If the links are not setup, they may point to the old CSet release (3.1.4 and older), and you may not be using the new C for AIX 4.3 release. If you want individual scripts in /usr/bin to point to the new C for AIX 4.3 release and also maintain simultaneous usage of /usr/bin/xlc (cc, c89, etc.) to point to the old CSET compiler, then create these scripts in /usr/bin:

vaxlc vacc vac89 vacc128 vaxlc128 vacc_r vaxlc_r

All these scripts could be links to the /usr/bin/vaxlc script which simply contains:

```
#! /usr/bin/ksh exec /usr/vac/bin/${0##*va} "$@" #end of script vaxlc
```

Note that use of this script is not supported as it may have unknown side effects through aliases and macros in the .kshrc file. This is provided only as a suggestion.

7. When using -qipa, some symbols which are clearly referenced or set in the source code may be optimized away by IPA, resulting in loss of these symbols in debug, nm or dump outputs. This is normal. Using -g with IPA will generally result in non-steppable source.

8. Files compiled with older versions of IPA cannot be mixed with files compiled with C for AIX 4.3 IPA.

9. Invoking xlc by itself will generate a return code of 40 after displaying the help file. Invoking xlc with an option but without a file will generate a return code of 249.

10. Using IPA with #pragma disjoint may produce incorrect code.

11. When generating a listing with -qipa using -qlist, IPA will generate an a.lst file which may overwrite any existing a.lst file. Also if your source file is named a.c, the IPA listing will overwrite the normal compiler listing a.lst. This is a known limitation. You can cause IPA to generate an alternate listing via the option -qipa=list=<filename>.

12. Any pdf files generated with CSet 3.1.4 cannot be mixed with new pdf files as optimizations are performed differently. Pdf files should always be created using the same release/ptf level. Both -qpdf1 and -qpdf2 should use the same xlCcode.

13. The profile directed feedback option (-qpdf1) generates a -lpdf to look for the libpdf.a archive. This archive is in /usr/vac/lib. The user should add the option -L/usr/vac/lib to allow -qpdf1 to find the correct library. This is a known issue.

14. In the config file, the stanza attribute "inline" defines the location of the Intermediate code inliner. This is no longer used and the file xlCinline is also no longer shipped. This does not affect the -Q or -qinline option. The -Q and -qinline options are still functional.

15. This release of C for AIX uses License Use Management (LUM) instead of iFOR/LS. For details, please read the hardcopy README instructions that came with the product and /usr/vac/README.password.

Appendix F. Special Notices

This publication is intended to give Technical Sales Representatives, Technical Consultants, System Administrators and System Engineers an in-depth understanding of 64-bit AIX performance and tuning. Developers involved in writing 64-bit applications and migrating existing 32-bit applications to the 64-bit environment will find the chapter on migration useful. See the PUBLICATIONS section of the IBM Programming Announcement for AIX Version 4.3 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM ("vendor") products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate

them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

The following document contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples contain the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

You can reproduce a page in this document as a transparency, if that page has the copyright notice on it. The copyright notice must appear on each page being reproduced.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX®
AT ®
C + +/MVS
IBM ®
Micro Channel ®
PowerPC ®
PowerPC 601 ®
PowerPC 604 ®
RS/6000
SP
AS/400 ®
C Set + + ®
DB2 ®
InfoExplorer
OS/400 ®
PowerPC Architecture ®
PowerPC 603e ®
RISC System/6000 ®
RT ®

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

Java and HotJava are trademarks of Sun Microsystems, Incorporated.

Microsoft, Windows, Windows NT, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

Pentium, MMX, ProShare, LANDesk, and ActionMedia are trademarks or registered trademarks of Intel Corporation in the U.S. and other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names may be trademarks or service marks of others.

Appendix G. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

G.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How To Get ITSO Redbooks” on page 187.

- *AIX Version 4.3 Differences Guide*, SG24-2014
- *AIX Version 4.2 Differences Guide*, SG24-4807
- *RS/6000 Performance Tools in Focus*, SG24-4989
- *Understanding IBM RS/6000 Performance and Sizing*, SG24-4810
- *Customizing Performance Toolbox and Performance Toolbox Parallel Extensions for AIX*, SG24-2011
- *Benchmarking in Focus*, SG24-5052

G.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** and receive updates 2-4 times a year at significant savings.

CD-ROM Title	Subscription Number	Collection Kit Number
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RS/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RS/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
Application Development Redbooks Collection	SBOF-7290	SK2T-8037
IBM Lotus Redbook Collection	SBOF-6899	SK2T-8039
Tivoli Redbooks Collection	SBOF-6898	SK2T-8044
IBM RISC/6000 Redbooks Collection PDF	SBOF-8700	SK2T-8043

G.3 Other Publications

These publications are also relevant as further information sources:

- *AIX Versions 3.2 and 4 Performance Tuning Guide*, SC23-2365
- *Performance Toolbox for AIX: Guide and Reference*, SC23-2625
- *Performance Toolbox Parallel Extensions for AIX: Guide and Reference*, SC23-3997
- *Optimization and Tuning Guide for Fortran, C, and C++,* SC09-1705
- *Writing a Device Driver for AIX V4.1*, SC23-2593
- *PE Operation and Use Vol. 2 - Tools Reference,* SC28-1980
- *AIX Version 4 System Management Guide: Operating System and Devices*, SC23-2525
- *AIX Version 4 System Management Guide: Communications and Networks*, SC23-2526
- *AIX Version 4 System User's Guide: Operating System and Devices*, SC23-2544
- *AIX Version 4 System User's Guide: Communications and Networks*, SC23-2545
- *AIX Version 4 Problem Solving Guide and Reference*, SC23-2606
- *AIX Version 4 Messages Guide and Reference*, SC23-2641
- *AIX Version 4 Files Reference*, SC23-2512
- *AIX Version 4 Commands Reference*, SBOF-1851 (Contains the following publications that may also be ordered separately.)
  - *AIX Version 4 Commands Reference, Volume 1*, SC23-2537
  - *AIX Version 4 Commands Reference, Volume 2*, SC23-2538
  - *AIX Version 4 Commands Reference, Volume 3*, SC23-2539
  - *AIX Version 4 Commands Reference, Volume 4*, SC23-2540
  - *AIX Version 4 Commands Reference, Volume 5*, SC23-2639
  - *AIX Version 4 Commands Reference, Volume 6*, SC23-2640
  - *Go Solo 2*, SR28-5705
- *X/Open Single Unix Specification Go Solo: How to Implement and Go Solo with the Single Unix Specification*, SR28-5705
- *AIX Version 4 General Programming Concepts: Writing and Debugging Programs*, SC23-2533
- *AIX Version 4 Communications Programming Concepts*, SC23-2610

- *AIX Version 4 Technical Reference*, SBOF-1852 (Contains the following publications that may also be ordered separately.)
  - *AIX Version 4 Technical Reference, Volume 1: Base Operating System and Extensions*, SC23-2614
  - *AIX Version 4 Technical Reference, Volume 2: Base Operating System and Extensions*, SC23-2615
  - *AIX Version 4 Technical Reference, Volume 3: Communications*, SC23-2616
  - *AIX Version 4 Technical Reference, Volume 4: Communications*, SC23-2617
  - *AIX Version 4 Technical Reference, Volume 5: Kernel and Subsystems*, SC23-2618
  - *AIX Version 4 Technical Reference, Volume 6: Kernel and Subsystems*, SC23-2619
  - *AIX Version 4 Technical Reference, Volume 7: AIXwindows*, SC23-2620
  - *AIX Version 4 Technical Reference, Volume 8: Enhanced Xwindows*, SC23-2621
  - *AIX Version 4 Technical Reference, Volume 9: Enhanced Xwindows*, SC23-2622
  - *AIX Version 4 Technical Reference, Volume 10: Enhanced Xwindows*, SC23-2623
  - *AIX Version 4 Technical Reference, Volume 11: Master Index*, SC23-2624
- *AIX Version 4 Assembler Language Reference*, SC23-2642
- *AIX Version 4 Quick Reference*, SC23-2529
- *AIX Version 4 iFOR/LS Tips and Techniques*, SC23-2666
- *AIX Version 4 AIXwindows Programming Guide*, SC23-2632
- *AIX Version 4 Enhanced Xwindows Programming Guide*, SC23-2636

How To Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at `http://www.redbooks.ibm.com`.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **PUBORDER** – to order hardcopies in United States
- **GOPHER link to the Internet** – type `GOPHER WTSCPOK.ITSO.IBM.COM`
- **Tools disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get BookManager BOOKs of redbooks, type the following command:

  ```
  TOOLCAT REDBOOKS
  ```

  To get lists of redbooks, type the following command:

  ```
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  ```

  To register for information on workshops, residencies, and redbooks, type the following command:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1998
  ```

  For a list of product area specialists in the ITSO, type the following command:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
  ```

- **Redbooks Web Site on the World Wide Web**

  `http://w3.itso.ibm.com/redbooks/`

- **IBM Direct Publications Catalog on the World Wide Web**

  `http://www.elink.ibmlink.ibm.com/pbl/pbl`

  IBM employees may obtain LIST3820s of redbooks from this page.

- **REDBOOKS category on INEWS**
- **Online** – send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**

  With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an e-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** (Do not send credit card information over the Internet) – send orders to:

	IBMMAIL	**Internet**
In United States	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America	dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** – send orders to:

IBM Publications	IBM Publications	IBM Direct Services
Publications Customer Support	144-4th Avenue, S.W.	Sortemosevej 21
P.O. Box 29570	Calgary, Alberta T2P 3N5	DK-3450 Allerød
Raleigh, NC 27626-0570	Canada	Denmark
USA		

- **Fax** – send orders to:

United States (toll free)	1-800-445-9269
Canada	1-800-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1) 408 256 5422 (Outside USA)** – ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **Direct Services** – send note to `softwareshop@vnet.ibm.com`

- **On the World Wide Web**

Redbooks Web Site	http://www.redbooks.ibm.com
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Internet Listserver**

  With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an e-mail note to `announce@webster.ibmlink.ibm.com` with the keyword `subscribe` in the body of the note (leave the subject line blank).

IBM Redbook Order Form

Please send me the following:

Title	Order Number	Quantity

First name	Last name	
Company		
Address		
City	Postal code	Country
Telephone number	Telefax number	VAT number

☐ Invoice to customer number _____

☐ Credit card number _____

Credit card expiration date	Card issued to	Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

List of Abbreviations

ABI	application binary interface
API	*Application Programming Interface*
AIO	asynchronous Input/Output
AIX	advanced interactive executive
ANSI	American National Standards Institute
APAR	authorized program analysis report
API	application program interface
AS/400	Application System/400
ASCII	American National Standard Code for Information Interchange
ATM	asynchronous transfer mode
bf	bigfoot
bfrpt	bigfoot report
BOS	base operating system
BSD	Berkeley software distribution
BSS	Basic Software Standard
CAD	computer aided design
CAM	computer aided manufacturing
CDE	Common Desktop Environment
CD-ROM	compact disk-read only memory
CDLI	common data link interface
CHRP	Common Hardware Reference Platform
COFF	common object file format
CPU	central processing unit
DAC	dual address cycle
DB2	DATABASE 2
DMA	direct memory access
EB	exabyte
ESID	effective segment identifier
fdpr	feedback directed program restructuring
FIFO	first in/first out
FORTRAN	formula translation
FP	floating-point
FPR	floating-point register
G	giga
GB	gigabyte
GPR	general purpose register
GUI	graphical user interface
HIPPI	high performance parallel interface
HTML	Hypertext Markup Language
I/O	input/output
IBM	International Business Machines Corporation
IEEE	Institute of Electrical and Electronics Engineers
ILP	integer, long, pointer
ILS	international language support

IP internet protocol

IPC inter-process communication

IPV6 Internet Protocol Version 6

ISV independent software vendor

IT information technology

ITSO International Technical Support Organization

JFS journaled file system

K kilo

KB kilobyte

L long

LPP licensed program product

LVCB *Logical Volume Control Block*

LVM logical volume manager

M million

MB megabyte

MCA micro-channel architecture

MHz megahertz

MTU maximum transmission unit

NCS NetWork Computing System

NDD network device driver

NDP neighbor discovery protocol

NFS network file system

OpenGL open 3D graphics library interface

OS/2 operating system/2

OS/400 operating system for AS/400

P2SC *Power2 Single Chip*

PB petabyte

PC personal computer

PCI peripheral component interconnect

perfPMR performance program modification request

PL1X programming language one extended

POSIX portable operating system interface for computer environments

POWER performance optimization with enhanced RISC

PPC PowerPC

PSSP Parallel System Support Programs

PTF program temporary fix

PTX Performance Toolbox

PV *Program Visualizer*

R register

RAID redundant array of independent disks

RAM random access memory

RFC request for comments

RISC reduced instruction set computer

rmss reduced memory system simulator

RR round robin

RS RISC system

RSS resident set size

RT RISC technology

SDP solution developer program

SMIT	System Management Interface Tool
SMP	symmetric multiprocessors
SP	Scalable POWERparallel
SQL	structured query language
stem	scanning tunneling encapsulating microscope
TB	terabyte
TCP	transmission control protocol
TOC	table of contents
UL	unsigned long
V	version
VM	virtual memory
VMM	virtual memory manager
VSM	Visual Systems Management
X	X Window System
X11R6	X Window System Release 6
XCOFF	extended common object file format
XL	extended language
XPG	X/OPEN portability guide

Index

Numerics

A

B

C

D

F

G

H

I

J

L

T

V

W

X

ITSO Redbook Evaluation

AIX 64-bit Performance in Focus
SG24-5103-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at http://www.redbooks.ibm.com
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@vnet.ibm.com

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction _____

Please answer the following questions:

Was this redbook published in time for your needs? Yes___ No___

If no, please explain:

_____

_____

_____

_____

_____

What other redbooks would you like to see published?

_____

_____

_____

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

_____

_____

_____

_____

_____

SG24-5103-00
Printed in the U.S.A.