536

Evolution of storage facilities in AIX Version 3 for RISC System/6000 processors

by A. Chang
M. F. Mergen
R. K. Rader
J. A. Roberts
S. L. Porter

The AIX* Version 3 storage facilities include features not found in other implementations of the UNIX† operating system. Maximum virtual memory is more than 1000 terabytes and is used pervasively to access all files and the meta-data of the file systems. Each separate file system (subtree) of the file name hierarchy occupies a logical disk volume, composed of space from possibly several disks. Database memory (a variant of virtual memory) and other database techniques are used to manage file system meta-data. These features provide the capacity to address large applications and many users, simplified program access to file data, efficient file buffering in memory, flexible management of disk space, and reliable file systems with short restart times.

Introduction

AIX* Version 3 for the IBM RISC System/6000 processor continues the evolution of storage facilities within the framework of the UNIX† operating system that began with AIX Version 1 [1]. There are various motivations for this evolution, such as the following:

* AIX is a trademark of International Business Machines Corporation.

- Support for larger applications and more users.
- Direct access to file data by ordinary program statements.
- More efficient file buffering than that provided by a fixed-size buffer pool.
- More flexible units of disk space management than provided by fixed-size disks.
- A file subsystem that can recover from crashes and that has a shorter restart time.

The important concepts used in this evolution include very large virtual memory, integration of the file subsystem with logical volumes and virtual memory, and the innovation of database memory. The first two concepts come from earlier systems such as Multics [2] and the IBM System/38 [3]. Database memory was developed in the 801 minicomputer project at the IBM Thomas J. Watson Research Center, where all three concepts were combined in a prototype called CPR running on the IBM RT System [4].

This paper first explains how very large virtual memory is achieved while preserving the AIX program interface, based on architecture extensions described elsewhere [4, 5]. It then discusses the structure of UNIX-based file systems and how they benefit from the flexibility of logical volumes and from simplified buffering in virtual memory. Finally, the use of database techniques to implement a reliable file subsystem is described.

Very large virtual memory

The IBM POWER (Performance Optimization With Enhanced RISC) architecture with the AIX Version 3 operating system offers a maximum virtual memory

† UNIX is a registered trademark of AT&T.

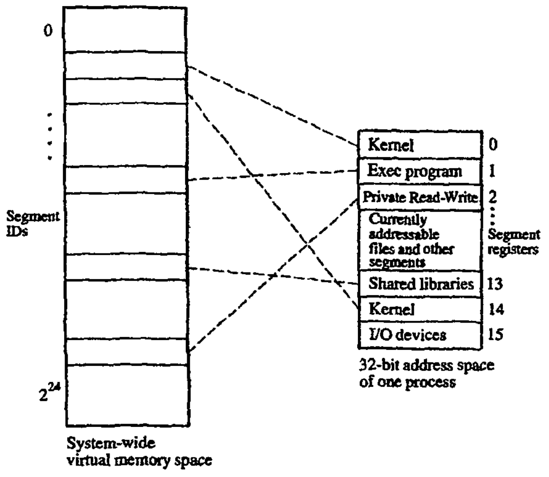
0
Segment IDs
2^24
System-wide virtual memory space
Kernel
Exec program
Private Read-Write
Currently addressable files and other segments
Shared libraries
Kernel
I/O devices
0
1
2
13
14
15
Segment registers
32-bit address space of one process

Figure 1

Process view of system-wide virtual memory.

space of more than 1000 terabytes. The addressing capacity is greatly increased over that of previous versions of AIX, without affecting programs outside the AIX kernel. This results from the inherently extensible virtual-memory architecture of the IBM RISC System/6000* (RS/6000) processor [4, 5], which is summarized as follows.

Memory-access instructions generate an address of 32 bits. Four bits select a segment register and 28 bits give an offset within the segment, providing access to 16 segments of up to 256 megabytes each. Each segment register contains a segment ID that becomes a prefix to the 28-bit offset, forming the virtual-memory address. The segment ID is 24 bits in the RS/6000, for a total of up to 16 million segments. The 52-bit virtual address refers to a single, large, system-wide virtual memory space, as shown in **Figure 1**.

The AIX process space is a 32-bit address space; in other words, programs use 32-bit pointers. Each process or interrupt handler is able to address only a portion of the system-wide virtual memory space—those segments whose segment IDs are in the segment registers. If desired, segment registers may be changed rapidly, allowing a process to access many more than 16 segments. However, only the AIX kernel can load a segment register, which enables the kernel to enforce access permissions for objects in virtual memory.

* RISC System/6000 is a trademark of International Business Machines Corporation.

The system call to load a segment register (*shmat*) does not specify the segment ID to be loaded, but rather a software capability for some object that the process is expected to address. Access permissions are checked prior to this, during system calls that open a file or get a virtual memory segment. If access is allowed, the kernel assigns a segment ID (of interest only to the kernel) and gives the process a capability (file descriptor or shared memory identifier) for access to the segment, used later in the *shmat* system call.

This 32-bit addressing and indirection through access capabilities give each process an interface that does not depend on the actual size of the system-wide virtual memory space. The maximum segment ID (16 million in the RS/6000 processor) must be large enough to accommodate the sum of all segments of all processes, and this maximum affects only programs inside the kernel.

Some segments are shared by all processes, some are shared by a subset of processes, and some are accessible to only one process. Sharing is achieved by simply allowing two or more processes to load the same segment ID.

The AIX Version 3 kernel loads a few segment registers in a conventional way in all processes, implicitly providing the memory addressability needed by most processes, as shown in Figure 1. There are two kernel segments, a shared-library segment and an I/O device segment, that are shared by all processes and whose contents are hidden (or read-only) to non-kernel programs. There is a segment for the current main program (*exec* system call) of the process, shared on a read-only basis with other processes executing the same program. There is a read-write segment that is private to the process. The remaining segment registers can be loaded (using the *shmat* system call) to provide more memory or memory access to files (described later), shared or private, read-only or read-write, according to access permissions.

The approach to implementing large virtual memory in AIX systems has been described elsewhere [1, 4]. An inverted page table, with one entry for each real page, limits the real memory required for virtual-address translation to a size related to real rather than virtual memory size. The external page tables, which contain the disk location of each virtual page, are themselves kept in pageable virtual memory, since they may be quite large. Disk-allocation maps, which record the allocation state of disk blocks, are treated similarly. The kernel page-fault handler uses techniques called *careful update* and *backtracking* [1, 4] to deal with page faults in its own data structures.

The AIX Version 3 virtual-memory implementation retains the preceding features and generalizes them, with two motivations: to efficiently represent sparsely used memory and to accommodate the very large addressing capacity of the architecture. External page tables and other structures that contain disk locations of pages (file system "meta-data," described later) are organized as a tree for each segment or file, with nodes present only where needed for data actually stored. These and other virtual-memory structures are allocated in several segments, separate from the kernel segments mentioned previously. The page-fault handler must find the segment that contains the tree for the segment or file in which a page fault has occurred, and then load a segment register to address it. These tree-containing segments also contain an external page table for themselves, with a small portion pinned in real memory to terminate recursion during page-fault handling.

Another useful consequence of very large virtual memory is that it somewhat simplifies memory allocation within the kernel itself. Tables whose required size depends on system load or on the number of users, processes, or segments, can be allocated in virtual memory at the size of the maximum design point. Unused portions of these tables and the corresponding external page-table tree nodes consume no real memory or disk space. These resources are assigned only after a page fault in a previously unused area.

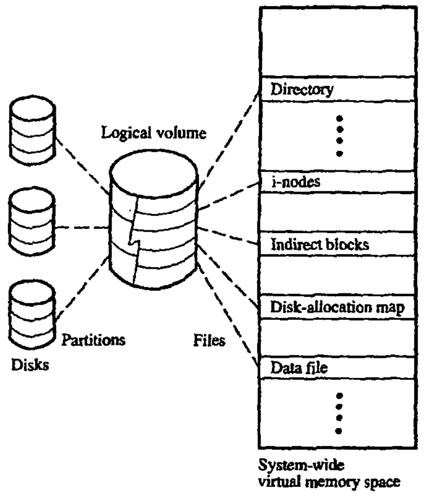

Figure 2

File system in a logical volume and virtual memory.

Files in logical volumes and virtual memory

A file in the UNIX-based operating system is an unstructured byte string stored in a set of not necessarily adjacent disk blocks. File system *meta-data* (i-nodes, indirect blocks, and directories) are used to find and access the files. An *i-node* is a small block that contains information such as file owner, access permissions, file size, and the locations of a few data blocks for a small file. Larger files have a tree of *indirect blocks*, rooted in the i-node, which contain the data-block locations. A *directory* is a file that contains pairs of the form (file name, i-node location) organized for searching by file name. Directories can contain names of other directories, thereby forming a hierarchy or tree. A file is named by a variable-length sequence of names that gives a directory search path, starting from the root directory, to the directory that contains the location of the file i-node.

As in most UNIX-based implementations, AIX Versions 1 and 2 separate total disk space into partitions, such as one disk or a contiguous portion of one disk, sometimes called a *minidisk*. Each partition contains a file system with its own directory tree, i-nodes, indirect blocks, and files. These separate file systems are formed into a single naming tree by making the root directory of each file system, except one, become a directory in the tree of another file system (using the *mount* system call).

AIX Version 3 retains this model but generalizes the concept of disk space. Each file system occupies a *logical volume* of disk space composed of one or more disk partitions rather than one disk partition. A partition is contiguous on one physical disk, but a logical volume may have partitions on more than one disk in a group of related disks, as shown in **Figure 2**. All partitions in the disk group are of one size, but logical volumes may differ in the number of partitions they contain. Each disk contains a description of the group it is part of, e.g., what logical volumes are in the group and what partitions belong to each volume.

This generalization of disk space has two important results. First, the space available to a file system can be expanded by adding a partition to the logical volume, without stopping the system or moving any other partition. If necessary, a disk can be added to the group to supply more free partitions. The size of a file system is not limited by the size of a physical disk. Second, a logical volume can be *mirrored* to enhance data availability. Each partition of such a volume has one or two other partitions allocated on different disks to hold identical copies of the data.

The file systems and virtual-memory pager do not use the physical disks directly, but instead consider each logical volume as a device and call a logical-volume device driver. This component translates a logical-volume disk-block location into a physical-block location or mirrored locations. If mirroring, the driver reads the fastest copy or writes all copies identically. If a permanent disk error occurs, the driver can assign a new location for the block and, if mirrored, read a good copy and write the data into the new location.

Previous versions of AIX have two possible places to buffer the disk blocks of a file in memory. The default is the kernel buffer cache. In addition, the *shmat* system call can be used to map an entire file into a virtual-memory segment in the caller's space, where it is addressable by ordinary memory-access instructions. The virtual-memory pager is responsible for the disk I/O for that file, when triggered by events such as page fault or request for a free real page. The *read* and *write* system calls can access the data in a file in either place.

File buffering is simplified in AIX Version 3 by not using the kernel buffer cache. Files are always mapped into virtual-memory segments when first opened and there is a separate virtual-memory segment for the different meta-data of each file system, that is, one for i-nodes, one for indirect blocks, one for the disk-block allocation map, and one for each directory, as shown in Figure 2. The kernel addresses these segments by loading their segment IDs as needed. For example, during *read* and *write* system calls, the kernel loads the file segment ID to move data between the file and the caller's area. The *shmat* system call simply makes file segments addressable by non-kernel programs, as shown in Figure 1. Ordinary program statements may then be used to access any byte in such a file, without further system calls, which may simplify some programs considerably.

The mapping of files into virtual-memory segments is also simplified. Rather than copying disk-block locations from the file system into external page tables for virtual memory, AIX Version 3 simply makes the file system meta-data (i-nodes and indirect blocks) available to the virtual-memory pager. The page-fault handler accesses the file meta-data directly, to find a disk-block location needed for page-in or to save a newly allocated disk-block location for eventual page-out.

The virtual-memory pager uses all real memory as a common buffer pool for the most recently referenced pages of all virtual segments, including those used to address all opened files. The availability of a large portion of real memory as a buffer may significantly improve file-access performance for some applications, compared to the usual fixed-size buffer cache.

Database memory

Other UNIX operating systems invoke a utility program, *fsck*, to detect possible file system damage after a crash. The *fsck* program reads all the meta-data (i-nodes, indirect blocks, and directories) and recommends and performs repairs (such as putting orphan files in the *lost and found* directory and deleting very damaged files) if needed to restore a consistent state. A *consistent* state has properties such as the following: The number (0 or more) of directory entries that point to an i-node exactly equals a link count in the i-node; each disk block belongs to, at most, one file (one pointer in an i-node or indirect block). Repair by the *fsck* program may sometimes be impossible.

AIX Version 3 uses database memory to achieve a more reliable file subsystem. Specifically, the kernel implements database memory for the segments that contain directories, i-nodes, and indirect blocks. Also, changes to disk-block allocation maps are recorded in the same log file used to implement database memory. The result is that these file system meta-data are consistent or can be made consistent after a crash simply by application of recent records from the log file. Recovery time with this database approach is much faster than with the *fsck* program, since it is related only to the amount of log data produced by recent file system activity, rather than to total file system size as with *fsck*.

Database memory, as described in [4], is files in virtual memory with the additional implicit properties of access serializability and atomic update, similar to those of database transaction systems. Database memory can be shared and accessed concurrently by many processes executing different transactions. The processes use ordinary memory-access instructions to read and write the contents of database memory, and they indicate the end of each transaction by a call to the kernel. They need do nothing else to coordinate with each other. The system is designed to ensure that the permanent results in database memory are as if the transactions execute one after another in some order rather than concurrently and, in the case of failures, as if each transaction executes completely or not at all.

To implement database memory, the kernel must detect and control the memory accesses of each process. The RT and RS/6000 systems have a data-locking hardware assist [4] to provide this function in a virtual-memory segment if required. Each real page-table entry contains a transaction ID field and lock-bits to control access to each memory line (128 bytes in the RS/6000) of the 4-Kbyte page. A register contains the transaction ID of the process currently executing. Hardware prevents access and interrupts the process if the locks in the page-table entry belong to another transaction or if they disallow the attempted load or store operation. In the RS/6000 architecture, the hardware also grants and sets locks in the page-table entry without interrupt, when only one transaction is accessing a page or when all accesses in a page are read accesses [5].

Lock interrupts call a lock manager in the kernel. It searches a lock table for locks of other transactions in the same page. If conflicting locks exist, the current process must wait until they are released. Otherwise, the lock is granted in the lock table and in the page table, and the current process can continue.

Selected standard database techniques are used to make updates atomic. Updated pages are not permitted to page out to permanent-file disk space unless the updating transaction has ended. When a transaction ends, its updates are recorded in a log. The write locks held by the transaction determine which memory lines are copied to the log file. An end-transaction record is logged, and then the transaction's locks are released. If failures occur, pages updated by uncompleted transactions are discarded from real memory or temporary disk space. Recent log records (those from pages possibly not paged out to permanent file space) of completed transactions (those with end-transaction records in the log) are used to update permanent-file disk blocks and thereby complete the transactions.

The AIX Version 3 file system meta-data reside in database memory segments. Each transformation of the meta-data by the kernel from one consistent state to another is treated as a transaction: for example, create a file, close a file after writing into new data blocks, or remove a file. In each of these examples, changes to several areas of meta-data are made into an atomic unit by the logging and recovery operations previously described.

As in any system where lock requests may occur in any order, the use of database memory may cause deadlock—that is, two or more processes waiting forever for the other(s) to release locks. Deadlock detection and recovery are well understood, but the required undoing and redoing of lost work would be complicated inside the file subsystem of the kernel. Instead, deadlock involving file system meta-data is avoided by a combination of techniques which differ somewhat from those used with pure database memory. Reading is allowed freely and does not conflict with write locks. Conventional software locking for some operations provides the proper serialization. Hardware locking is applied to indirect blocks only during end-transaction processing, when all modified locations are again updated in a standard order. Finally, if a transaction modifies several i-nodes, it does so in a standard order.

Conclusions

The architecture extensions of the IBM RISC System/6000 processor and the more general AIX Version 3 implementation provide a total virtual space of more than 1000 terabytes and a storage interface consistent with previous versions of AIX. File data may be addressed directly in virtual memory with ordinary program statements. Logical volumes and file buffering in virtual memory allow file systems to grow to very large configurations with very little space management and tuning effort. File-space mirroring and database memory are designed to improve file system availability, reliability, and recovery time.

Although most of the ideas described in this paper are not new, their combination in an industry-compatible workstation product may be new. Database memory is a new approach to storage concurrency and recovery, previously studied only in a research setting and employed in AIX Version 3 only for the file system meta-data. Other uses for it may be discovered within the kernel, based on experience with AIX Version 3. Also, database memory might be useful for non-kernel programs, as an option at the system call interface.

References

1. J. C. O'Quin, J. T. O'Quin, M. D. Rogers, and T. A. Smith, "Design of the IBM RT PC Virtual Memory Manager," *IBM RT Personal Computer Technology*, Order No. SA23-1057, pp. 126–130, 1986; available through IBM branch offices.
2. A. Bensoussan, C. T. Clingen, and R. C. Daley, "The Multics Virtual Memory: Concepts and Design," *Commun. ACM* **15**, 308–318 (May 1972).
3. *IBM System/38 Technical Developments*, Order No. G580-0237, 1978; available through IBM branch offices.
4. A. Chang and M. F. Mergen, "801 Storage: Architecture and Programming," *ACM Trans. Computer Syst.* **6**, 28–50 (February 1988).
5. R. R. Oehler and R. D. Groves "IBM RISC System/6000 Processor Architecture," *IBM J. Res. Develop.* **34**, 23–36 (1990, this issue).

Received July 21, 1989; accepted for publication February 19, 1990

Albert Chang *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Chang is a Research Staff Member in the Computer Sciences Department of the IBM Research Division in Yorktown Heights. He recently completed a temporary assignment at the Austin laboratory as a programmer working primarily on the virtual memory and file system components of the AIX Version 3 kernel. He holds the B.S. and Ph.D. degrees in electrical engineering from the the University of California at Berkeley.

Mark F. Mergen *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Mergen is Manager of Systems Architecture in the Advanced RISC Systems Department at the Thomas J. Watson Research Center. After working in FORTRAN compiler development at RCA in Cherry Hill, New Jersey, he joined IBM in 1967 as a systems engineer working with university customers. In 1970, he joined the IBM Data Processing Division in Gaithersburg, Maryland, in a group which produced job management subsystems (HASP and JES2) for the OS/360 and MVS operating systems. Later work included a prototype virtual memory version of OS/360 and a prototype high-performance transaction system for System/370, with databases in virtual memory. Since joining the Research Division in 1980, Dr. Mergen has worked on the integration of database transaction concepts with virtual memory; he shares a patent for a hardware-locking mechanism invented in this work. His continuing interests are in computer architecture and operating systems. He has B.S. (mathematics) and M.D. degrees from the University of Wisconsin.

Robert K. Rader *IBM Advanced Workstations Division, 11400 Burnet Road, Austin, Texas 78758 and IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Rader received a B.S. in physics from Stanford University in 1961 and a Ph.D. in physics from the University of California at Berkeley in 1970. From 1970 to 1972 he was a research physicist in elementary particle physics at the Centre des Ȇtudes Nucléaires, Saclay, France. He left the field of physics to work on a large interactive system for computer-based education at the University of Illinois in 1972. Dr. Rader joined IBM in 1982 as a Research Staff Member at the Thomas J. Watson Research Center, as a member of the Systems Organization group. He has worked on advanced I/O architecture, fault-tolerant computers, and system measurement and performance. Since 1986 he has been on assignment to the Advanced Workstations Division in Austin, Texas, where he joined the work on AIX Version 3 virtual memory management as a member of the AIX Base Kernel Architecture Department.

Jeffrey A. Roberts *IBM Advanced Workstations Divisio. 11400 Burnet Road, Austin, Texas 78758.*

Scott L. Porter *IBM Advanced Workstations Division, 11400 Burnet Road, Austin, Texas 78758.* Mr. Porter joined IBM in 1984 after receiving a B.S. degree in computer and information science from the University of Florida. He is a Senior Associate Programmer in AIX kernel development and is involved with the implementation of the AIX Version 3 virtual memory manager.