Advanced Interactive Executive (AIX) operating system overview

by L. K. Loucks
C. H. Sauer

The Advanced Interactive Executive (AIX) is the operating system used in the RT Personal Computer. It is a portable operating system architecture that is suitable for a wide range of computer architectures and customer requirements. Discussed in this paper are the structure and services of AIX.

This paper discusses the operating system for the RT Personal Computer™ that is known as the Advanced Interactive Executive (AIX)™. The RT Personal Computer system is a family of workstations based on the IBM 32-bit (RISC) microprocessor—named ROMP—and its corresponding high-function memory management unit.[1] (RT Personal Computer, Advanced Interactive Executive, and AIX are trademarks of the International Business Machines Corporation.) With this level of performance and functionality, IBM workstations reached the point at which it was practical and imperative to provide workstation users with an operating system that was as sophisticated as those used in mainframe computers. There were many considerations that compelled us to build an operating system for the RT Personal Computer that incorporated many of the currently most advanced system concepts.

The RT Personal Computer system includes sophisticated hardware features, such as high-function virtual storage, advanced all-points-addressable (APA) displays, real-time capability, and others, which can be fully exploited only by equally sophisticated software. Because most workstations operate in an increasingly interconnected environment, the operating system must be able to deal with communication functions—especially those that are taking place at the request of other users—without intervention by the workstation's user. In many cases, the distribution of resources is not uniform. Users need to be able to use programs, data, and peripheral devices that are not local to their own workstations. Perhaps most important is the fact that workstations require an operating system that provides an application execution environment that combines application program portability from IBM and industry environments with efficient use of the hardware.

We decided to base the core of the RT Personal Computer operating system on the AT&T UNIX® System V. (UNIX is developed and licensed by AT&T, and is a registered trademark of AT&T in the U.S.A. and other countries.) In addition to System V, we included many enhancements generally available in the industry, most notably some features of System V.2, and many from BSD (Berkeley Software Distribution) 4.2 and 4.3. (BSD 4.2 and 4.3 are variants of the UNIX system developed and distributed by the University of California at Berkeley.) We chose the UNIX operating system because it provides significant power to a workstation user, provides multiuser capabilities when needed, and is portable and open-ended. Also important is the fact that the UNIX system has a large user and application base. In choosing the UNIX system, we accepted the need to

make significant upwardly compatible enhancements over what was available in the industry to meet our requirements. As is traditional for UNIX-based operating systems, an acronym ending in IX was chosen: thus we have AIX.

UNIX concepts

The UNIX operating system was originally created in the 1970s to provide a test bed for computer science experimentation.[2] This operating system differs from conventional operating systems in several key ways. Essentially, all of the operating system code is written in C to ensure easy portability from one processor architecture to another. Most of the control structures of the operating system, such as configuration tables, are bound as late as possible. Configuration information is kept in editable files to allow easy modification for experimental purposes. The file system, often called the heart of the UNIX system, is a tree-structured hierarchy consisting of *directories* and *files*. Files are represented as linear byte spaces rather than records and fields. Directories are structured files describing files and other directories. In keeping with the objective of portability, most I/O is performed through generic devices. The generic devices are mapped to real I/O devices by user-replaceable routines called *device drivers*. Any part of the nucleus of the system (called the *kernel*) can be modified by an appropriately authorized user. A command-processing component (called a *shell*) performs parameter substitution and calls appropriate command programs. No real distinction is made between command processors supplied with the operating system and those written by the user that accept the same invocation parameter conventions, and several shells can coexist in a given system.

Figure 1 shows the overall structure of a typical UNIX system. The most significant difference from ordinary operating systems is the accessibility of all elements of the software to user modification. A UNIX system is thus an operating system that provides tools for its own redefinition. It is precisely this characteristic that has made it the most popular operating system in academic computer science. Many of the commands and facilities that were originally developed in the course of computer science experiments have found their way into production UNIX systems. This has greatly enriched the UNIX functional power, while contributing a certain amount of inconsistency, especially in the syntax of the command language.

AIX structure

The generality and portability of the UNIX system are achieved at some cost in optimum use of the underlying hardware. We had decided to start with the UNIX system as a base. In view of our requirements, however, we were faced with the question of how best to provide the required enhancements. Two strategies could be followed. One was to rewrite the entire kernel. Although theoretically possible, because of the amount of UNIX knowledge available in IBM at the time, it was unlikely that such an approach would achieve upward compatibility with the standard UNIX system. The other was to provide a set of software services for the kernel and modify the kernel and other functions to exploit the facilities provided by that layer. We chose the second approach, which led to the system structure shown in Figure 2.

The Virtual Resource Manager (VRM) controls the real hardware and provides a stable, high-level machine interface to the advanced hardware features and devices. (See Figure 3.) The kernel received corresponding enhancements to use the services of the VRM and to provide essential additional facilities. Although the VRM and the AIX kernel proper have been tuned to each other, we have not precluded the ability to build other operating systems to exploit the VRM services. Similarly, the techniques we used to virtualize existing types of devices would work for new device types as well. Both the VRM and the kernel have been deliberately made open-ended to allow the straightforward addition of new functions and device support.

We were dealing with a new hardware architecture and with large quantities of new and modified software in the system. Because of that, we felt that special efforts were required to ensure excellent performance. We adopted a policy of continuous performance assessment of the operating system, starting with the earliest availability of hardware and software. The performance group had to develop new tools and procedures to assess the performance of the system, while it was still immature. The results of that effort are visible in the performance of the completed product.

To achieve one of our primary goals of providing users the widest possible choice of applications and computing environments, we provided ways of moving applications and data to the RT Personal Computer from other systems, such as the IBM Personal Computer, other UNIX systems, and IBM mainframes,

Figure 1 Typical UNIX system structure

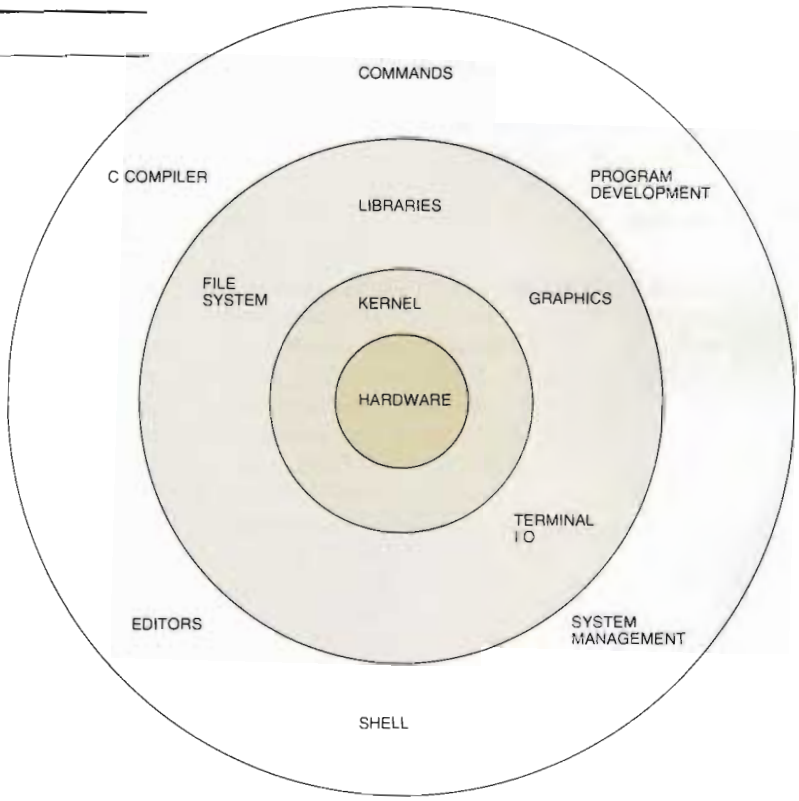


Figure 2 AIX system structure

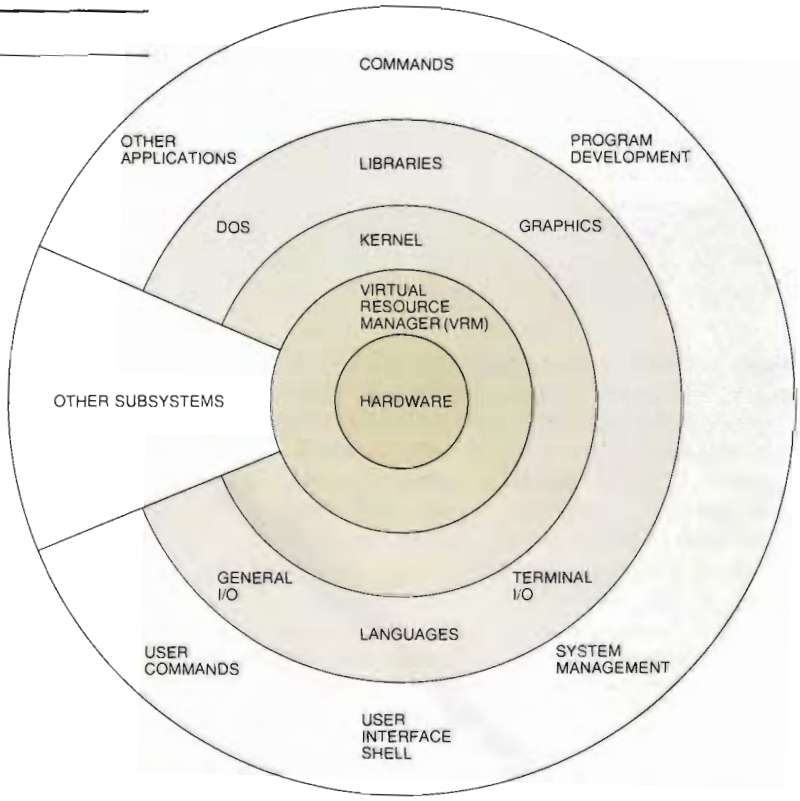
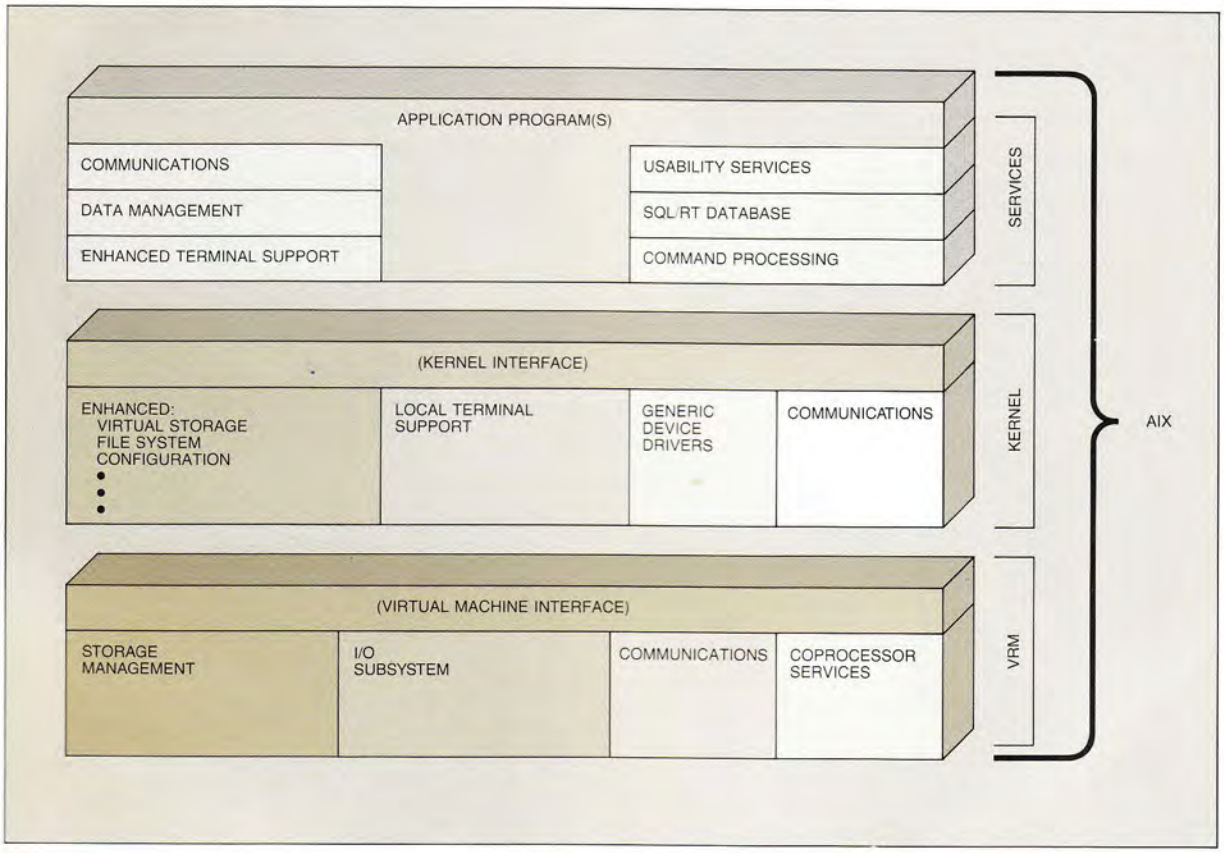


Figure 3 **Interface levels of AIX**



as well as many ways to interconnect them. The application development extensions above the kernel were integrated into the existing operating system structure. In some cases, the extensions were packaged and priced separately, but they were designed to operate as integral parts of the operating system after installation.

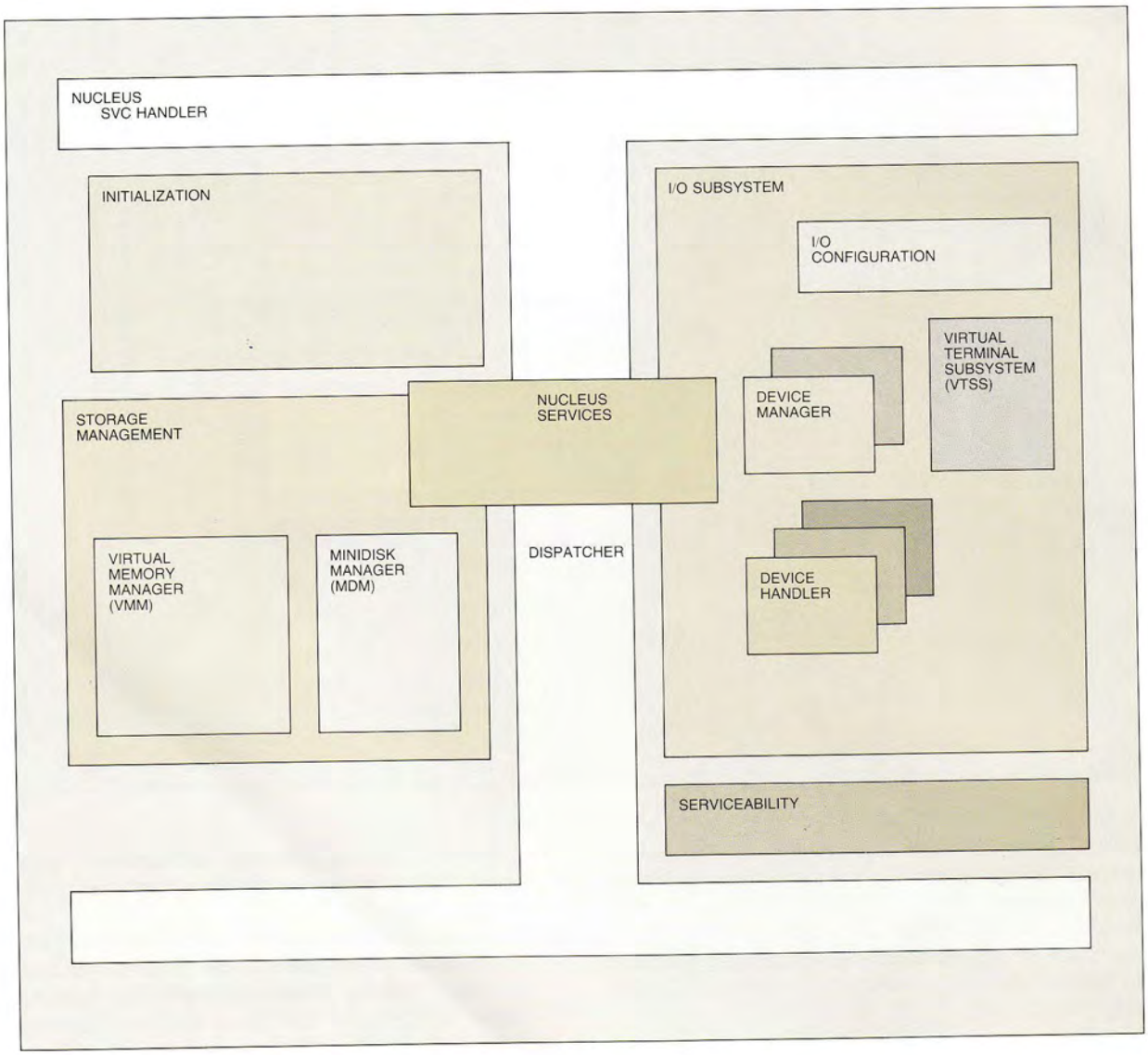
Creating a virtual environment for the AIX kernel

The existing structure and functions of the UNIX kernel were not sufficient for exploiting the advanced features of the RT Personal Computer hardware. The major deficiencies fell into the following areas:

- Lack of virtual-memory support had been a perceived deficiency in earlier systems. However, there were UNIX-based systems, such as BSD 4.2, that had provided virtual memory, but none of these had the capabilities that were possible with the RT Personal Computer.
- There was limited program management with code sharing only at the program level and static binding of modules.
- Real-time facilities, such as absolute priorities, kernel-level pre-emption, and multiple-process I/O, were lacking. These facilities were thought useful not only for traditional real-time applications, but also for complex communications services.
- There were limited facilities for dealing with dynamic I/O configurations. Most I/O changes required the rebinding of the kernel.

Instead of making major changes to the architecture of the kernel, we used the VRM to provide facilities to overcome these shortcomings. The VRM provides the services to implement a multitasking operating system while insulating the kernel from most of the details of the hardware implementation. The kernel has to be aware of only the problem-state instructions. All the other services, such as I/O device sup-

Figure 4 Virtual Resource Manager (VRM) structure



port, storage management of disk and memory, and hardware initialization, are provided. The VRM services are implemented in a comprehensive real-time execution environment.[3] Figure 4 shows the overall structure of the VRM.

VRM nucleus. The nucleus contains the basic services for the control of the ROMP processor, Memory Management Unit (MMU), and I/O Channel Controller (IOCC). These services include multiple pre-emptable processes, process creation and priority control, dynamic run-time binding of code, direct control of virtual memory, millisecond-level timer control, multiple pre-emptable interrupt levels, and an efficient interprocess communication mechanism for main and interrupt-level processes.

There is a virtual machine interface (VMI) that operates as follows. The kernel accesses the facilities of the VRM via a set of supervisor calls (SVC), virtual interrupts, and shared memory control blocks. Services may be executed synchronously as a call/return

or asynchronously via a queue to a device driver or process. These services provide the kernel with the capability of enabling and disabling virtual interrupts, returning from a virtual interrupt, processing machine communications interrupts such as a storage exception, and dispatching an operating system process. These are the basic facilities provided to implement a multitasking operating system kernel.

Storage management. A minidisk manager (MDM) provides the services to partition disk storage into logical areas that are independently managed. A minidisk is a contiguous area of disk storage that can be accessed by a logical block number, the size of which is specified by the kernel. This service also provides error recovery and bad block relocation. The VRM resides on a minidisk of its own in a standard AIX file system. Installation and space management on that minidisk are performed with standard AIX utilities.

Virtual memory manager (VMM). The ROMP/MMU virtual memory architecture, in combination with the VRM, gives the RT Personal Computer a demand-paged virtual memory of 1 terabyte, consisting of 4096 256-megabyte segments. Segments have a maximum of 256 megabytes, but typically they are much smaller. The ROMP contains 16-segment registers, permitting the addressing of 14 segments [plus I/O and Direct Memory Access (DMA) operations] at any time. (See Figure 5.) The VRM performs page-fault handling and manages the allocation of real memory, paging space, and virtual storage segments.[4] The VRM also provides the AIX kernel with interfaces to control these functions and to respond to a page fault by dispatching another process. These services provide a view of virtual memory as a collection of segments and pages that can be managed via SVCs. The segment services include create, destroy, change length, and protection, with a load-and-clear segment register(s) to provide addressability. A copy service that delays copying pages until they are referenced provides the necessary support for the UNIX fork primitive. The page services that pin, unpin, change protection, and purge provide other basic mechanisms for the kernel

Figure 5 Virtual memory addressing

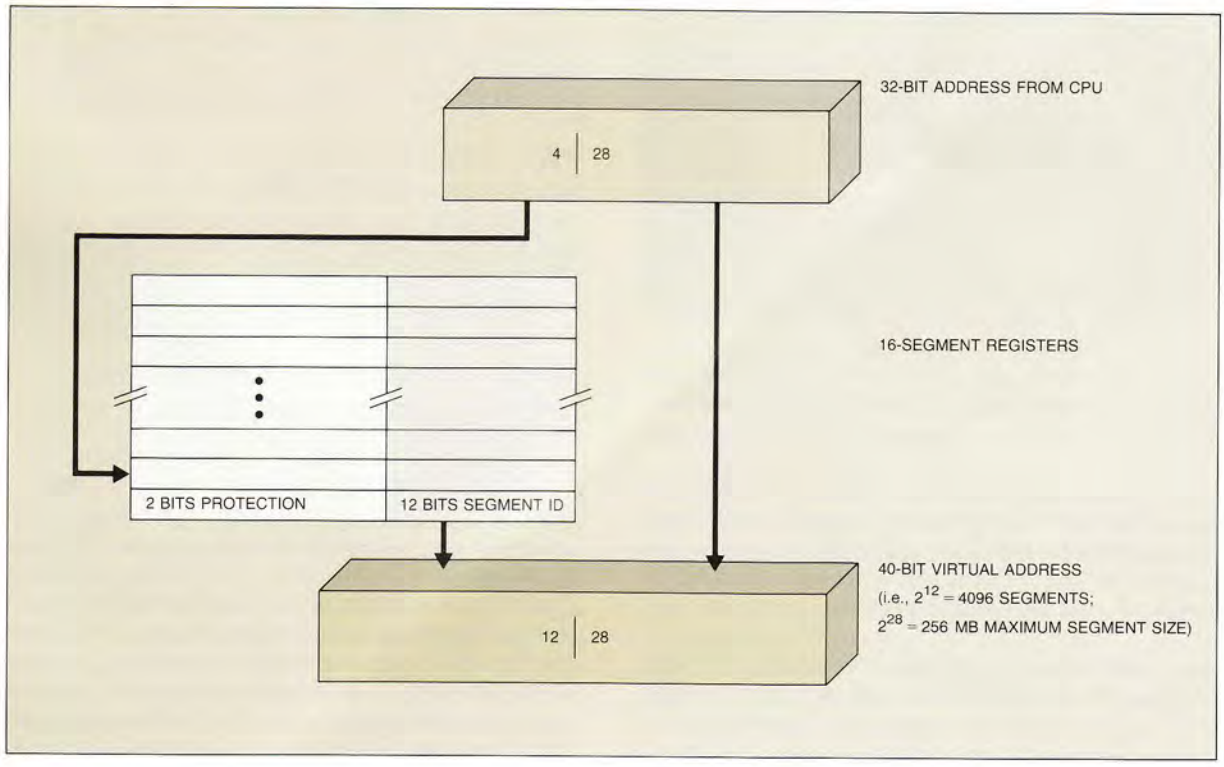
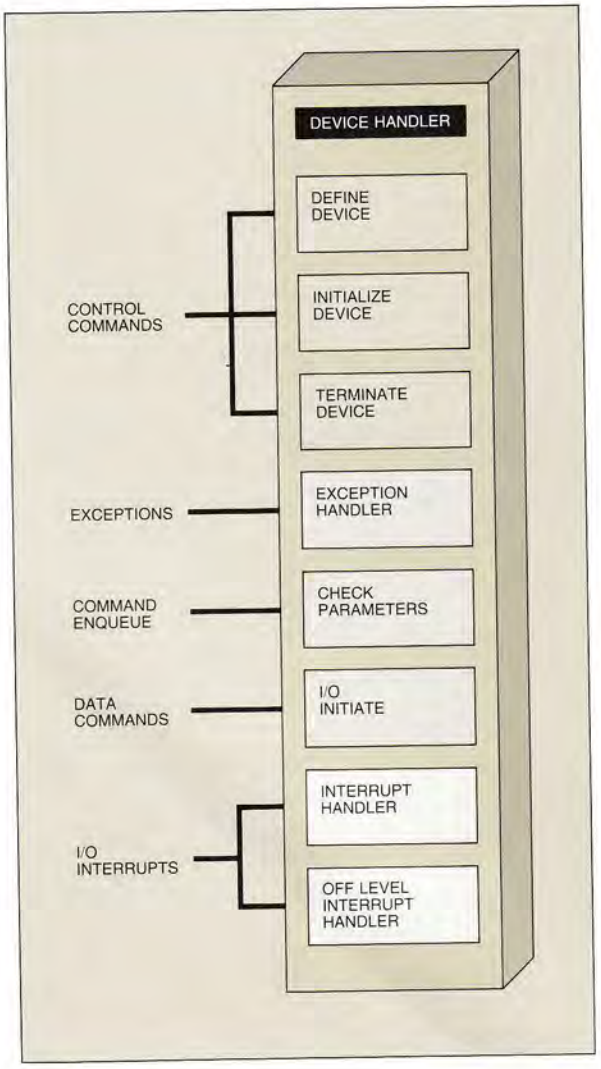


Figure 6 Device handler structure



to use in controlling the virtual memory environment.

The VRM also provides a map page service that maps memory pages within a given segment onto discontinuous disk file blocks, thus providing the primitive support for creating a single-level store that makes disk and memory access equivalent operations.

I/O subsystem. The VRM provides the operating system with an extensive queued interface to the I/O devices, thereby insulating the kernel from the details of specific devices and the management of shared devices. The devices that the kernel typically sees are those that are generic, such as generalized fixed-disk drives (i.e., minidisks) or serial ports. In those cases in which the generic devices are not appropriate or in which the real-time capabilities of the VRM environment are needed by the application, the user or a third-party programmer can write C or assembler-language code to implement the necessary function and dynamically add that code to the VRM.

Configuration. The configuration services provide facilities to add device support to the VRM. The Define Code SVC binds an executable module, called a *device handler*, into the VRM and Define Device provides the device-specific parameters to the handler. The correct device handler is typically selected on the basis of the currently installed hardware or via operating system configuration files and is dynamically bound into the VRM at start-up. However, these SVCs may be issued to the VRM at any time. A Query SVC provides the ability to determine the current configuration.

Device handler. A device handler is a very structured module designed to provide a queued interface to a device. There is a well-defined set of entry points that implement the functions of the driver; the execution environment for those entry points is strictly controlled by the VRM. (See Figure 6.) A device handler is not a process. Therefore, it runs as a part of the calling process, i.e., the kernel or another VRM internal process, or on a hardware interrupt level.

Device manager. A device manager is a structured VRM process designed to provide additional management services that cannot be provided by a device handler. (See Figure 7.) The execution environment is a VRM process and therefore has all the standard process attributes and capabilities, such as the ability to exploit virtual memory as well as various interprocess communication (IPC) mechanisms.

Virtual terminal subsystem (VTSS). At the time AIX was being implemented, no standard UNIX interface for advanced-function APA displays existed. Therefore, we provided a method to allow multiple applications to access the local console hardware. The key to this ability of AIX to support multiple simultaneous interactive applications is the virtual terminal.[5] A virtual terminal is a virtual counterpart of the real RT Personal Computer display(s), keyboard, locator (mouse or tablet), dials (valuator), and lighted program function keys (LPFK). The virtual terminals time-share the use of the real displays and input

devices. A virtual terminal can function either as a simulated ASCII terminal or as a high-function terminal (HFT) equivalent in power to the real hardware.

ASCII terminal simulation. The simulated ASCII terminal resembles a typical "*glass teletype*" (TTY), enhanced with functions to control sound, multiple fonts, and color. The functions are made available at the VMI through a set of standard I/O SVCs and through escapes in the data stream, as allowed in ANSI 3.64.

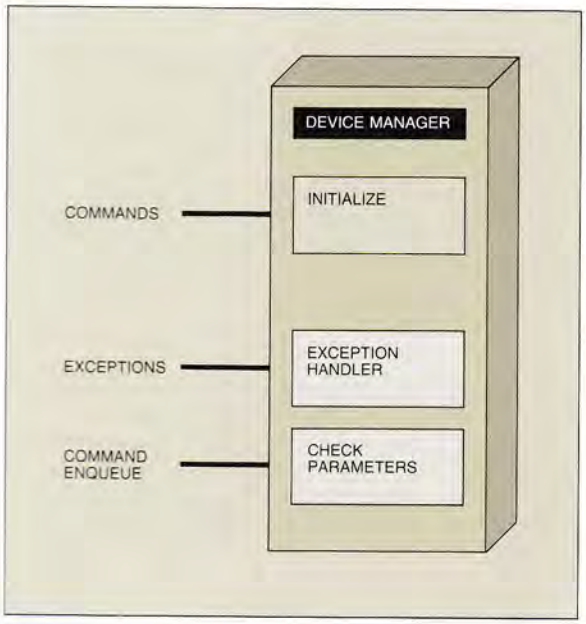
Monitored mode. ASCII terminal support is obviously not sufficient to support graphics and image on the local console displays. Therefore, an additional mode of an HFT (high-function terminal) virtual terminal was provided. This facility, called *monitored mode,* provides the support to allow an application in problem state to obtain controlled access to all hardware functions of the display. Also in this mode, data from the input devices are placed directly in the process address space by the HFT support in the VRM. The necessary services to control this access are also provided. These functions can be accessed directly by advanced applications through HFT facilities provided by the kernel, or more appropriately via the advanced graphics and windowing services provided by AIX.

HFT implementation. The HFT support is one example of the type of high-function I/O that can be implemented using the services provided by the VRM. It currently consists of many device handlers, two device-manager processes, and more lines of code than any other single VRM function.

Serviceability. Problem determination in system- or user-added code is supported by VRM serviceability facilities that include trace capabilities, dump capabilities, and an absolute debugger.

Personal Computer AT coprocessor. The VRM supports the Personal Computer AT coprocessor option as though it were another, albeit rather specialized, virtual machine.[6] The coprocessor runs concurrently with the execution of programs in the ROMP, but it has access to the keyboard, locator, and display only when the coprocessor virtual terminal is the active virtual terminal—that is, when it has control of the display. The inputs from the keyboard and locator are presented to the coprocessor as though they had been produced by the corresponding Personal Computer AT devices. If no display has been dedicated to the coprocessor, the display interface emulates a PC display on the system display. The VRM manages the shared system resources to ensure that the ROMP and coprocessor operate cooperatively.

Figure 7 Device manager structure



Building an enhanced kernel

The structure of the UNIX kernel was modified to allow it to operate in a VRM execution environment.[7] The kernel, and all of its processes, operate within a single virtual machine, as shown in Figure 8, and it uses the execution control facilities of the VRM to multitask within that machine. The kernel has been enhanced to use the VRM virtual memory services, and it now provides a demand-paged virtual memory system that fully supports the 1-terabyte address space. The kernel uses VRM page fault information to control process dispatching, as well as allowing the kernel itself to be paged.

The kernel occupies one (256-megabyte) segment. The code, computational data, and stacks are all contained within that segment. Each process is allocated three segments: one for program text (code), one for computational data, and one for the stack, as shown in Figure 9. This allocation of virtual memory allows very large programs with a very large data space to execute on the RT Personal Computer. This approach also simplifies many program and storage management functions. Functions such as

Figure 8 Operating system structure

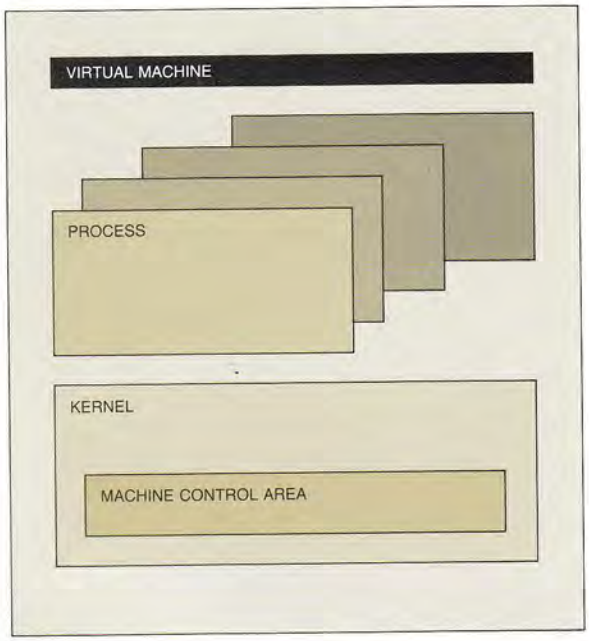
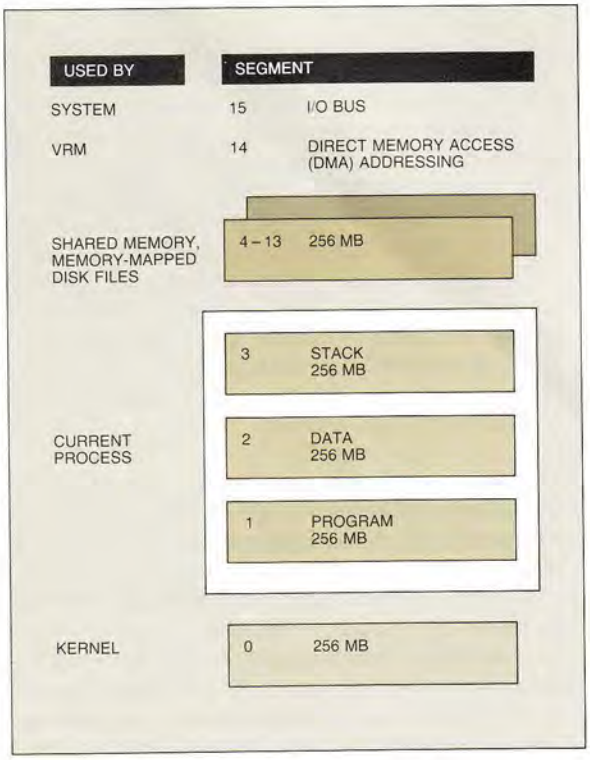


Figure 9 AIX virtual memory segment allocation

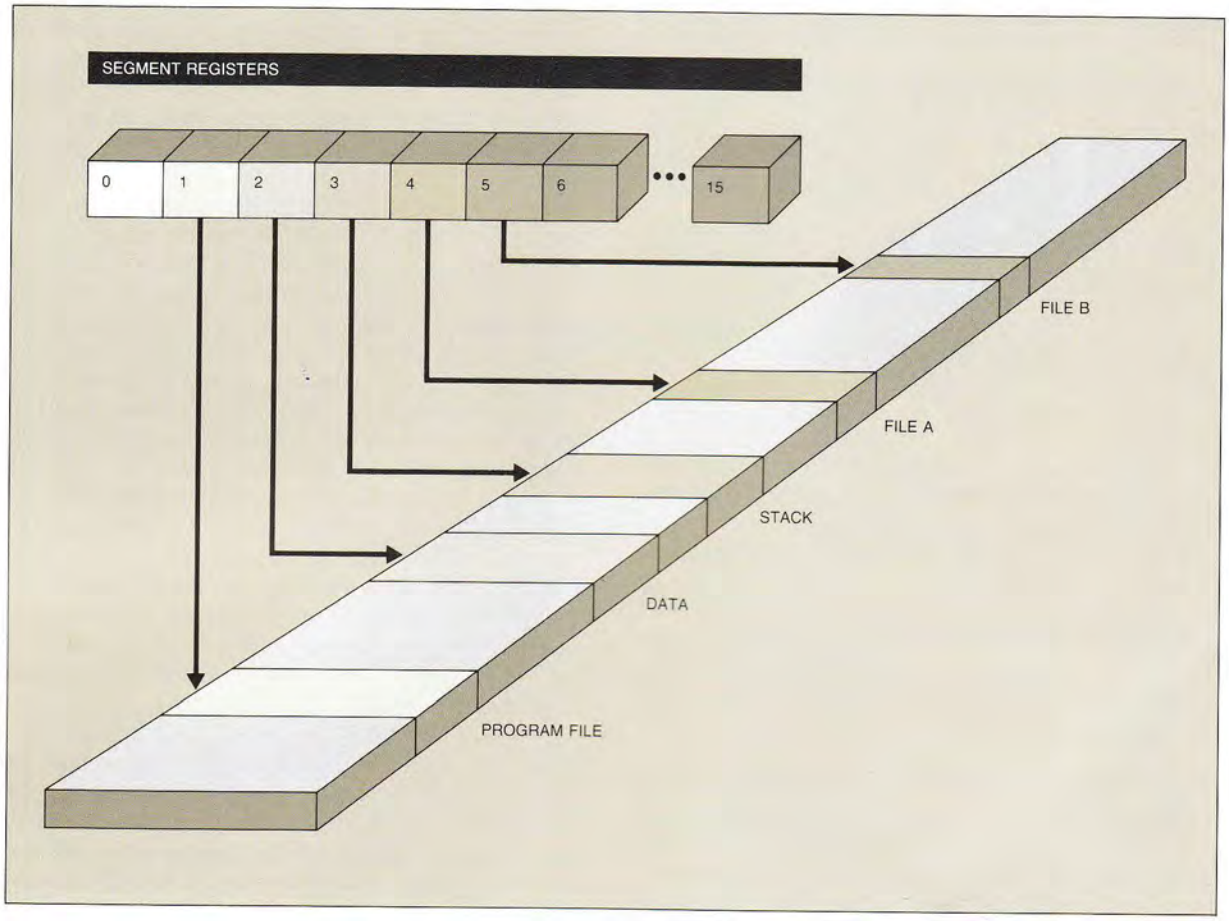


program sharing, computational storage allocation, and automatic stack growth are easier because all of the program sections are consistent among processes and are obviously large enough to allow simple techniques to be used. Additional segments can be obtained for use with private or shared data, shared code, or for mapped files.

Mapped files. A major extension of the file system was the exploitation of the VRM map page service to create a *single-level store* environment for program text (code) and data. This facility is called *mapped files*. A mapped file is one that is accessed through the virtual memory mechanism simply by loading data from the appropriate address. A segment can contain only one file. Figure 10 shows how files are mapped into the program's address space. Executable files (programs) and static initialized data are automatically mapped by the kernel at program invocation. A user data file can be mapped after it is opened via a simple extension to an existing system call. After a data file is mapped into a segment it may be accessed using any of the traditional kernel file I/O facilities, such as read, write, . . . , or it may be treated as memory and accessed directly.

Single-level store. Single-level store (SLS) technology provides a number of significant performance and space improvements over traditional methods. For programs, the load-and-execute method of execution requires that the operating system load the entire program into its address space before execution may begin. In addition, if the real memory is required for other purposes, the program must be paged out to backing storage. Contrast that procedure with the SLS approach. First, the program is simply mapped into the address space and given control at its entry point. Only the portions of the program that are needed for this invocation are ever actually read into real memory. Furthermore, if the real memory is required for other purposes, the program does not need to be paged out; it is simply paged in when required again for execution. This procedure has the benefits of quicker program start-up, reduced disk space because only a single copy of the program exists on disk, and elimination of paging out of program code. For data files, the advantages come from allowing the virtual memory manager to control all of the data of a process, both file data and computational storage. Therefore, it can allocate real memory in a more efficient manner. For example, consider a database application that is accessing a set of tables 10 megabytes in size, with that application executing on a machine that has 16 megabytes

Figure 10 Virtual memory of an application using mapped files

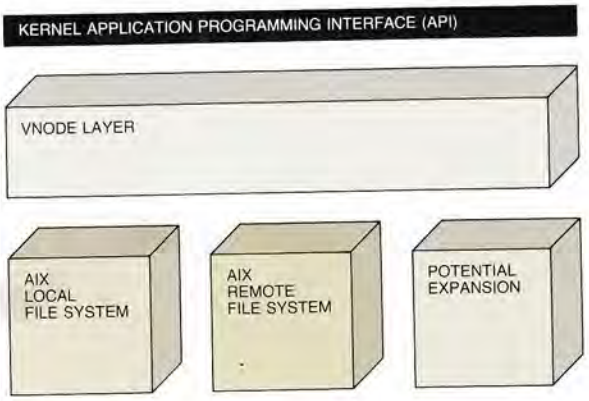


of real memory. After a few accesses, the essential portions of the database tables are in real memory, and the accesses that in traditional architectures were disk accesses are now memory accesses. The characteristic of program execution time changes from being I/O-limited to processor-limited, and, since processor speeds are increasing at a more rapid rate than disk access times, this change in the characteristic of the program is very beneficial. This is similar to the benefits of memory disks on personal computers, except that the allocation of resources is done dynamically rather than statically and the process is totally transparent to the user as well as to the program.

Database enhancements. Historically, UNIX-based database programs have used only the low-level disk I/O services of the kernel because the standard UNIX file system lacked several key features necessary to support them. This resulted in database programs that were not integrated with the system, unique sets of utility commands to be learned, and a general increase in the complexity of the system. We wanted to provide an integrated environment. Therefore, the kernel file system services were extended to provide the necessary facilities to allow us to add data management and relational database support that is built on top of the file system.[8] The enhancements included the ability to perform space management within a file, buffer cache synchronization on a file basis, and file- and record-level locking.

Performance and structure. We have added many performance improvements to the file system. The most notable are directory caching to speed up pathname lookup and the use of 2048-byte blocks. We

Figure 11 File system structure



have restructured the file system using the Sun Microsystems™ vnode definition to support multiple file system types in the kernel.[9] Figure 11 illustrates this approach. (Sun Microsystems is a trademark of Sun Microsystems, Inc.)

Interprocess communications (IPC). To assist in the writing of multiprocess applications, several enhancements were added to the standard system V IPC packages.

Signal enhancements. The traditional signal (asynchronous event notification) package has been augmented by a new package, compatible with the BSD 4.2 package, that provides more signal management services and cures a number of race conditions that were inherent in the original services. The standard signal package remains available for compatibility with existing application programs.

Message queues/semaphores/shared memory. Message queues were enhanced to provide an extended message structure that contains information useful for implementing security controls in servers. An additional option of semaphores reduces the process dispatches required in typical multiple-process applications, and new system calls were added that provide additional control over shared memory allocation and reclamation.

I/O management. The I/O management area of the kernel was restructured to make effective use of the I/O facilities of the VRM. Instead of a specialized device driver for each distinct device, we created a family of generic device drivers that are capable of supporting a number of unique devices of a given class. Unique device characteristics are supported by the VRM device handlers, which can be added or replaced dynamically. To implement this, the device-driver interfaces have been extended to allow dynamic binding of a kernel driver to a VRM device handler.

Configuration. In configuring a UNIX system, the administrator has historically needed an understanding of the internal structure and logic of the UNIX system, to be able to edit the configuration files correctly. We believed that it was unrealistic to impose such a requirement on our prospective users. Therefore, we set out to simplify the installation and configuration processes.[10] For those devices that can be identified internally, such as displays, the system performs an automatic configuration process. For devices that require explicit description, such as printers, we built a set of menu-driven utilities that obtain the necessary information from the user and make the required coordinated changes to all of the affected VRM and kernel system files. The interfaces to these utilities have been documented so that users or third-party programmers can add devices to be selected and described via the menus. These menus use the facilities shown in Figure 12, which were provided to allow users to add device and real-time application support.

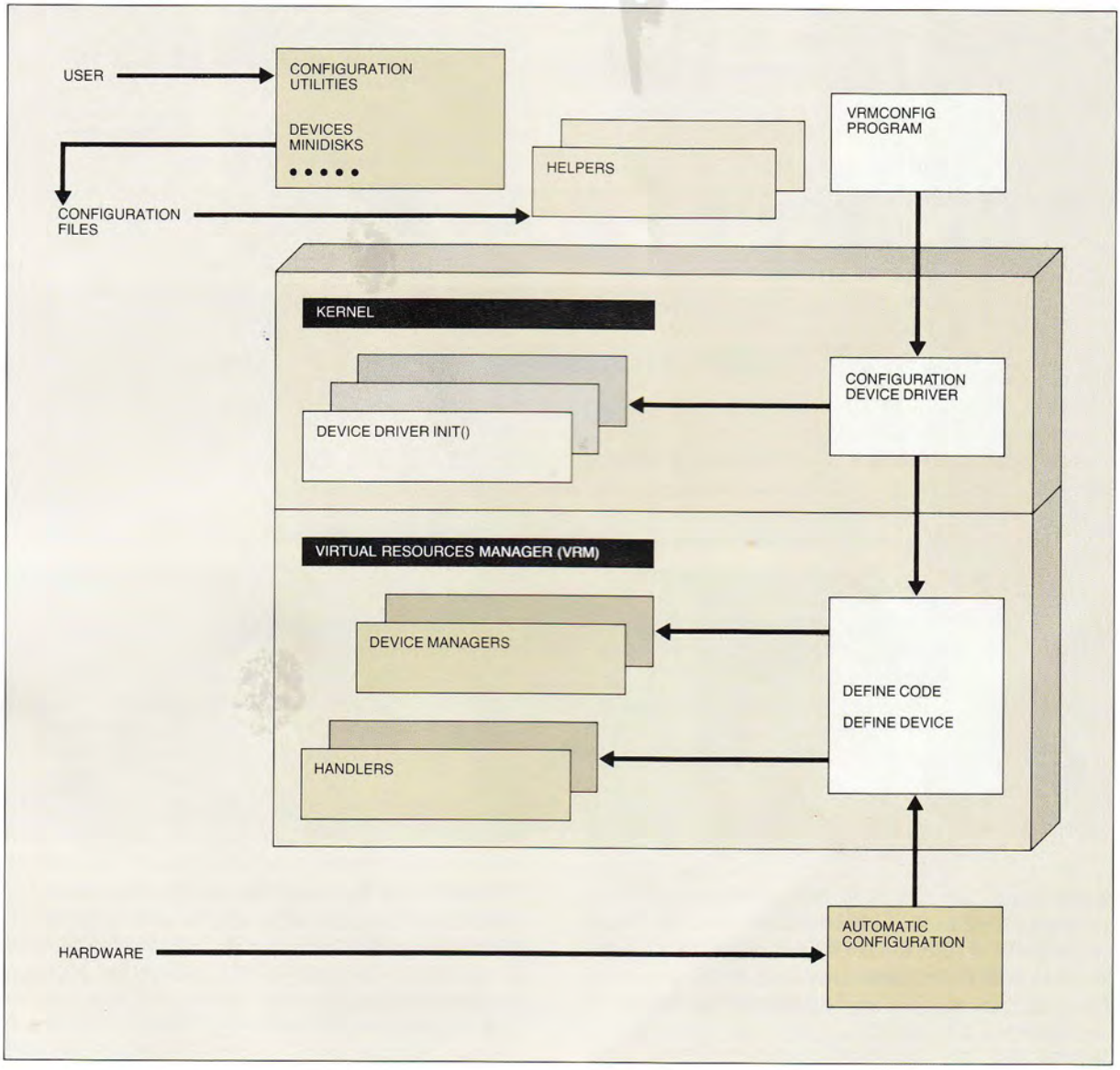
The I/O is typically configured at system start-up. The *vrmconfig* program, along with the helpers for each unique device type, reads the configuration files and adds the current device support to the running system. Additional support may be added any time by simply running *vrmconfig*.

Terminal support. The standard terminal support facilities of the UNIX operating system were extended to exploit the capabilities of the VRM local console support. In addition, several enhancements were made to the general character support that is applicable to all ASCII-class or teletype terminals (commonly known as TTYs). Figure 13 describes the overall AIX terminal structure.

Activity manager. We developed an activity manager to provide the support to manage virtual terminals. It has facilities for programs and users, such as to create or to terminate a virtual terminal or to start a program.

Character support. The TTY generic support was extended to provide support for screen paging. This facility is useful in controlling the output of stream-oriented applications, as well as providing a mecha-

Figure 12 I/O configuration

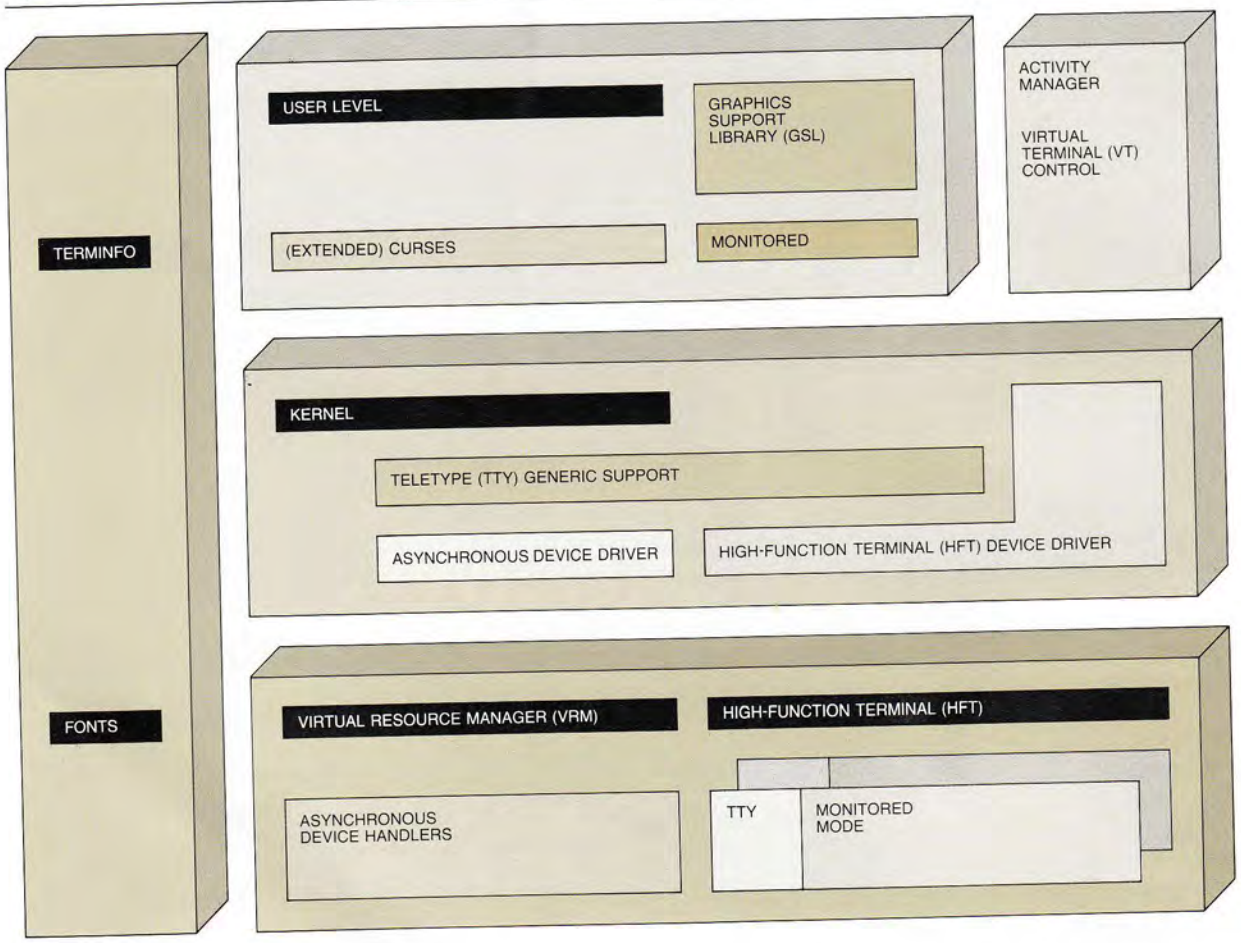


nism to prevent unseen output from going to inactive virtual terminals. In addition, an input-editing model patterned after the one provided in PC DOS was provided.

To allow existing applications to run unchanged and new character-oriented applications to use the RT Personal Computer facilities fully, we extended the ASCII character-oriented terminal model via private escape codes in the data stream and a new set of I/O controls to access features such as fonts, character sets, color, sound, and mouse input. These facilities are accessed through the kernel high-function terminal (HFT) device driver.

A package known as CURSES, which is a character-oriented window management package designed for TTY ASCII terminals, has received performance en-

Figure 13 AIX terminal support structure



hancements and has been compatibly extended to provide access to the extended font and other functions of the RT Personal Computer native displays. We also added functions such as screen division and layering logic to give applications a high-level, device-independent interface.

APA support. The monitored mode support provided by the VRM is managed by the kernel HFT device driver via a set of I/O controls and signals. These facilities ensure proper behavior by applications using this feature. If an application refuses to relinquish control of a virtual terminal, the HFT driver, after waiting for a specified time period, terminates the application. The application selects the mode in which to use the virtual terminal.

The Graphics Support Library (GSL) provides a set of high-performance graphic, text, and raster output primitives and a set of input functions for the local console. These functions are designed to provide an application programming interface to applications desiring this level of interface, as well as the APA device-driver function to higher-level graphics and window services.

Usability extensions

Single user. Because we expected RT Personal Computers to be used both as single-user workstations and as traditional UNIX time-shared systems, we believed that some changes were required to support the workstation user. We have made some alterations to reduce the number of situations in which a user has to exercise "superuser" authority. We added the ability to define more than one group to which a user belongs at any given time. This feature, derived from BSD 4.2, allowed us to define users as

members of the *system group*. System group members can perform a number of operations that previously could be performed only by a superuser; only the most hazardous commands are still restricted to superuser authority. This technique gives the user of a private workstation a simpler environment to work in, while preserving the existing AIX authority structure for multiuser environments. For users who wish to operate their systems in a manner similar to PC systems, a configuration option was added to allow automatic log-on at system start-up time.

Menu shell. The UNIX system has a dual-purpose command language. The commands have been designed from the beginning to be primitives of a command procedure programming language, sometimes at the expense of ease of use when individual commands are submitted from the terminal. This makes the management of files and the performance of common operations unnecessarily complex. Many UNIX installations solve this problem by building sets of procedures that effectively constitute a command meta-language. We chose to combine the solution to this problem with the construction of a full-screen interface to AIX.[11] The usability package provides Files, which is a full-screen file management utility similar to FILELIST on VM/CMS, and Tools, which is the ability to request the most common AIX commands via a menu interface. The dialog manager that is used to implement these utilities is general enough to serve application programs as well as AIX commands.[12]

The Files and Tools applications of the usability package can be extended to cover new types of files; new actions that can be performed against those files can be defined; and new tools—including complete full-screen applications—can be added. The dialog manager in the usability package can also be used to provide new full-screen applications with an interface that is consistent with the interface presented by Files and Tools.

PC DOS compatibility. AIX also includes a new shell that processes PC DOS commands, conversion programs that transform data from PC to RT Personal Computer format, and subroutines that allow applications to read DOS-formatted diskettes and minidisks.[13]

National-language support. The UNIX system has historically been an English-only operating system. We have added significant national-language support in the following form:

- An extensive character set including mathematics symbols and multinational characters has been added. For example, characters such as ä, è, π can be included in an RT Personal Computer data stream.
- The formatting of data such as currency, date, and time in accordance with the requirements of a particular country has been included. For example, in some countries dates are traditionally written with the year first; in other countries, dates are written with the month first.
- Data processing has been included that is consistent with the characteristics of a particular national language. For example, the operating system can sort in alphabetic order, according to the particular conventions of each supported country.

The RT Personal Computer international-character support benefits more than just non-U.S. English users. For example, an extensive character set of over 500 characters allows users to create text that includes non-alphanumeric symbols (such as many mathematical symbols).

Diagnosis and debug. To simplify the diagnosis of problems in AIX, we added several debugging tools that include a trace mechanism, a mechanism for logging of errors and system messages, and a memory-dump capability. The standard facilities were extended where necessary to deal with the unique features of AIX and the RT Personal Computer hardware.

Expanding the application development environment

To be able to support the full range of modern applications, AIX incorporated several functional extensions. The most significant enhancements have been the following:

- A broad spectrum of technical and commercial programming languages
- An SQL database manager and an indexed access method
- Industry standard graphics subsystems
- Connectivity enhancements to allow RT Personal Computers to communicate with both IBM and non-IBM systems, in Local-Area Networks (LANs) and over Wide-Area Network (WAN) telecommunications links
- Window support services
- Distributed services for interconnected RT Personal Computers to be used cooperatively and to

allow multiple applications to work effectively on the RT Personal Computer

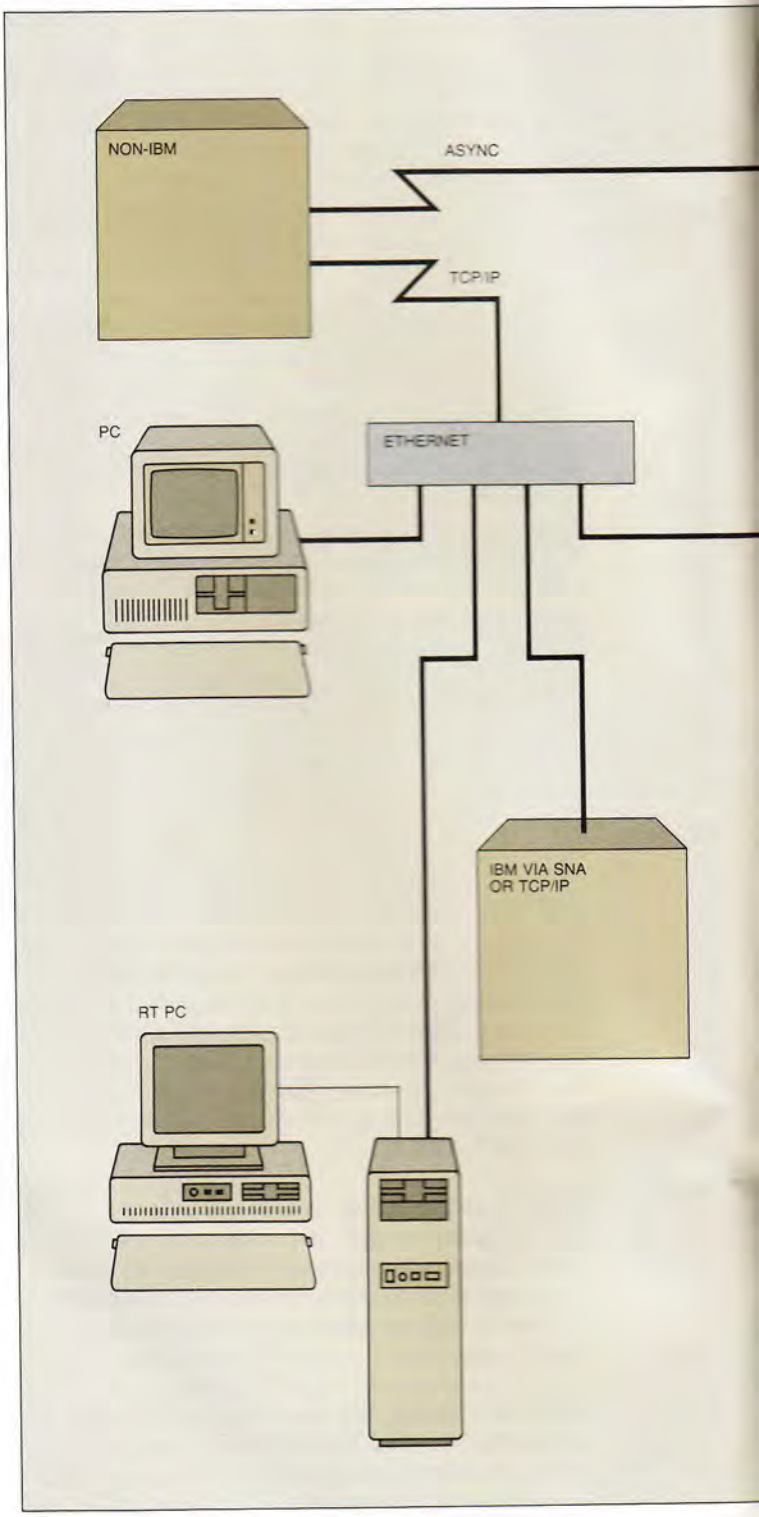
Languages. The higher-level language compilers for the RT Personal Computer were chosen on the basis of the number and types of programs that have been written in those languages. We selected dialects that would facilitate propagation of programs from the IBM Personal Computer, other IBM mainframes, and other UNIX systems, with language extensions where necessary to support the AIX environment. In some cases, the compilers have two modes—one for programs from the PC, and one for programs from minicomputer or mainframe environments. We developed a new subroutine linkage convention that supports multimodule programs written in several languages.[14]

Data management. One of the most critical requirements was for a database program supporting IBM SQL to provide both users and application programmers with relational database facilities. We also added a b-tree-based data management program that permits either record-level or field-level access. Although these subsystems are packaged separately from the operating system, they become an integral part of the file system when they are installed.

Graphics interfaces. AIX provides a family of different interfaces to the console display. Versions of a standard Virtual Device Interface (VDI) and a Graphics Kernel System (GKS) are available from third-party vendors. We provide graPHIGS™, an implementation of the emerging Programmer's Hierarchical Interactive Graphics Standard (PHIGS). (graPHIGS is a trademark of the International Business Machines Corporation.) This implementation provides application portability from IBM mainframes and supports all of the graphics device attachments of the RT Personal Computer, including the IBM 5085.

Communications. Because the RT Personal Computer is intended to be able to communicate with both IBM and non-IBM systems, AIX must include support for a wide variety of communications protocols. AIX supports Asynchronous, Bisynchronous (BSC), SDLC, CUT, DFT, Ethernet™, and Token-Ring connections. We have included both the industry-standard TCP/IP and the IBM SNA protocols (LU 1, 2, 3, and 6.2). The RT Personal Computer can be a bridge among multiple different LANs, able to support up to two Ethernet and two Token-Ring LANs from a single RT Personal Computer. Figure 14 summarizes the range of connectivity available. (Ethernet is a trademark of Xerox, Inc.)

Figure 14 Summary of RT Personal Computer connectivity



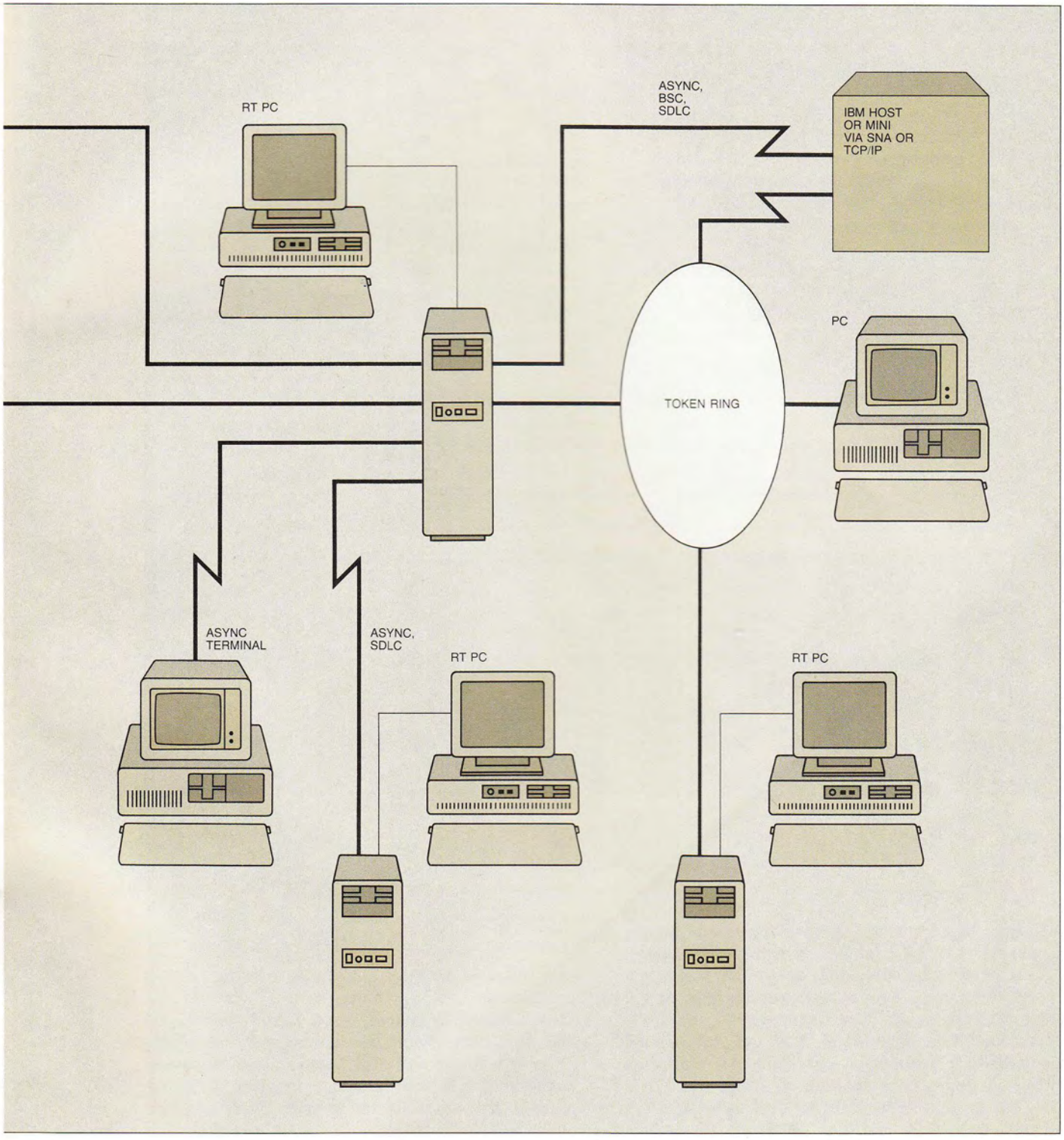
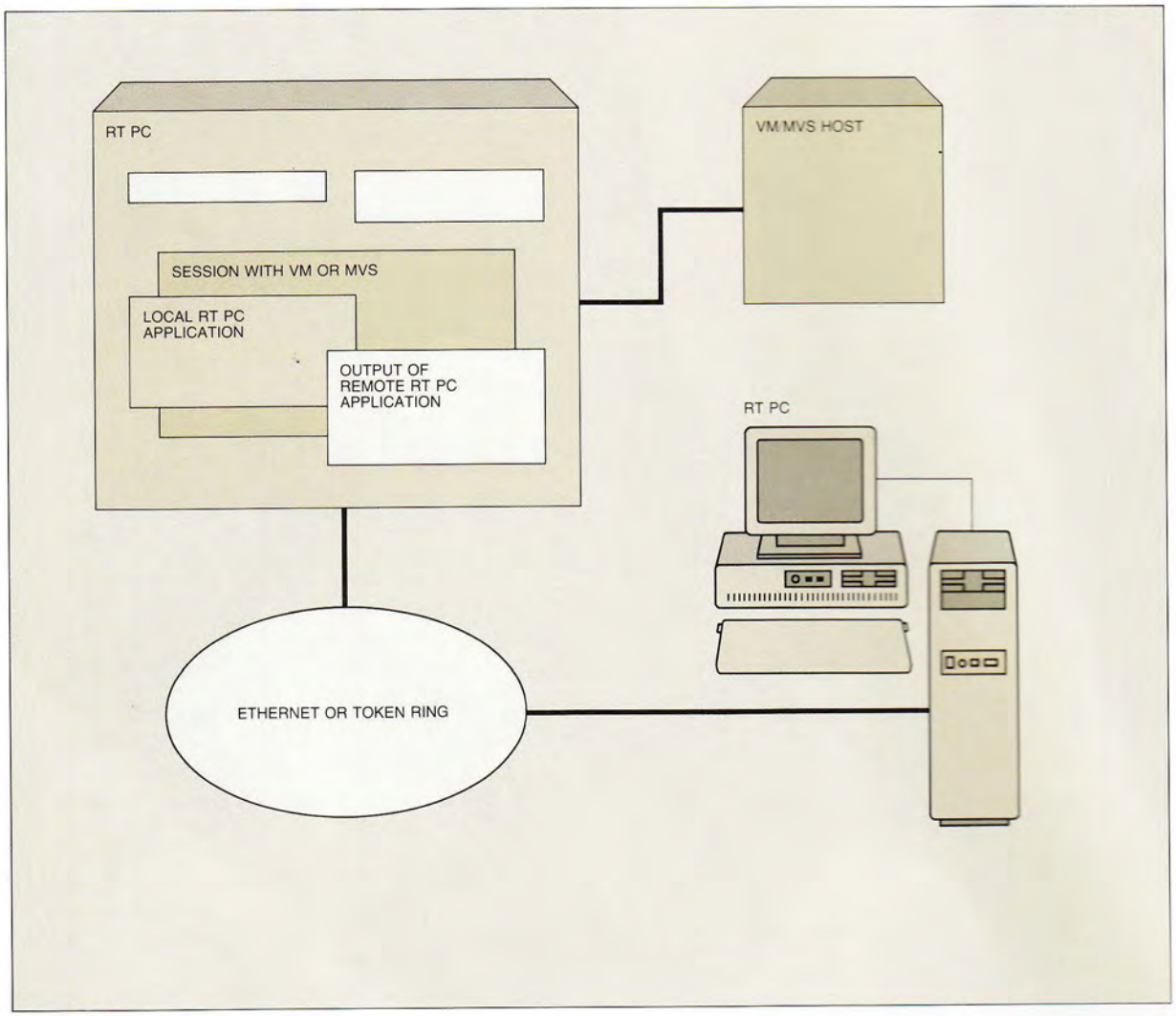

RT PC
ASYNC,
BSC,
SDLC
IBM HOST
OR MINI
VIA SNA OR
TCP/IP
PC
TOKEN RING
ASYNC
TERMINAL
ASYNC,
SDLC
RT PC
RT PC

Figure 15 X-Windows example



Window management. Another subsystem that is available is the X-Window System. X is a space-shared, multiprocess window service that was developed under the auspices of Project Athena at the Massachusetts Institute of Technology.[17] One of the key capabilities included in X is the TCP/IP-based network support that allows windows to contain data from X applications running on other computers that are connected via Ethernet or Token Ring. See Figure 15.

Distributed Services

RT *Personal Computer Distributed Services* (RT/DS) provides distributed operating system capabilities for the AIX operating system. These include distributed files with local/remote transparency, a form of single-system image and distributed interprocess communication. The distributed file design supports traditional AIX and UNIX file system semantics. This allows applications, including data management/

database applications, such as SQL/RT, to be used in the distributed environment without modification to existing object code. The design incorporates IBM architectures such as SNA and some of the architectures of Sun Microsystems NFS.[15,16]

Design goals. The primary goals in our design of distributed services were the following:

- *Local/remote transparency in the services distributed.* From both the user's perspective and the application programmer's perspective, local and remote access appear the same.
- *Adherence to AIX semantics and UNIX operating system semantics.* This is corollary to local/remote transparency. The distribution of services cannot change the semantics of the services. Existing object code should run without modification, including database management and other code that is sensitive to file-system semantics.
- *Remote performance and local performance.* This is also corollary to transparency. If remote access is noticeably more expensive, transparency is lost. Note that caching effects can make some distributed operations faster than a comparable single-machine operation.
- *Network media transparency.* The system should be able to run on different local- and wide-area networks.
- *Support of mixed administrative environments.* We believe networks are evolving toward heterogeneous collections of private machines, i.e., single-system-image clusters of machines, and machines that act as servers for multiple machine/system images. This necessarily implies multiple administrators with complex interrelationships.
- *Security and authorization.* Comparable to these are those for a single multiuser machine.

File system. Distributed Services uses remote mounts to achieve local-remote transparency. A remote mount is much like a conventional mount in the UNIX operating system, but the mounted file system is on a different machine than that mounted on directory. Once the remote mount is established, local and remote files appear in the same directory

Figure 16 Example shared file system

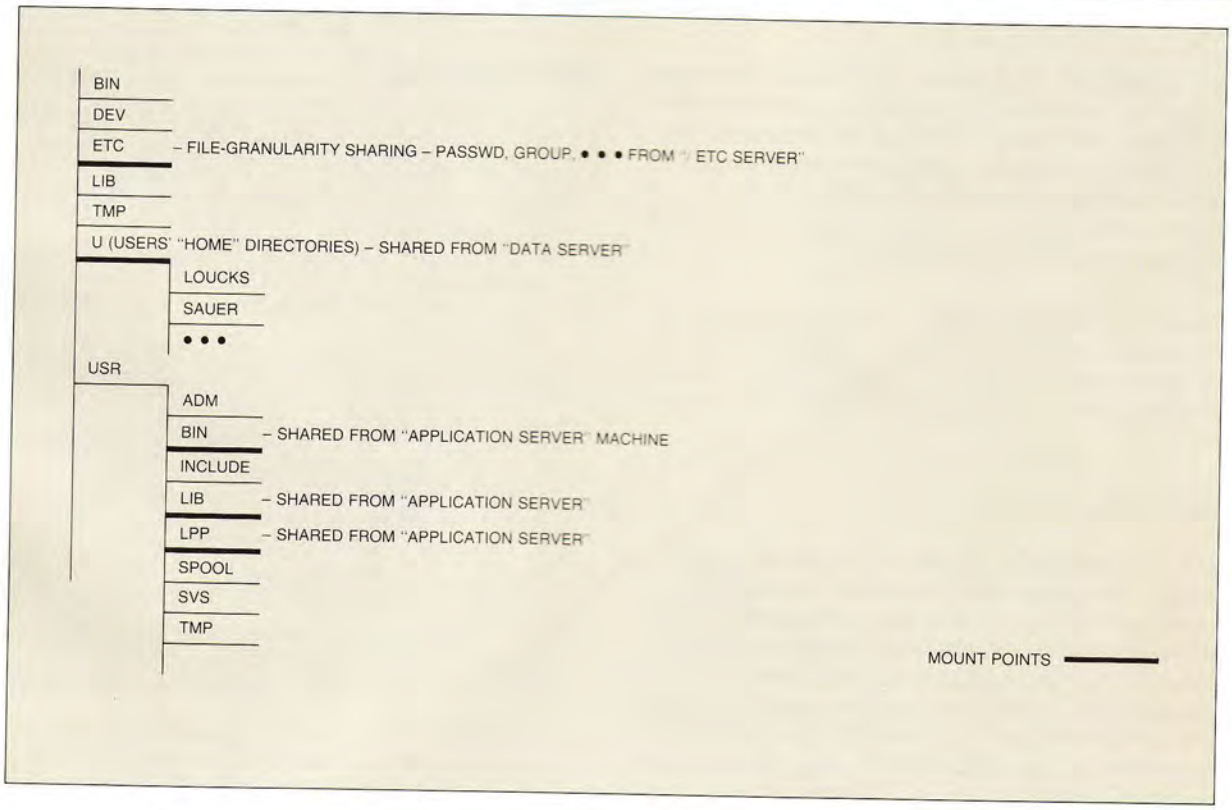
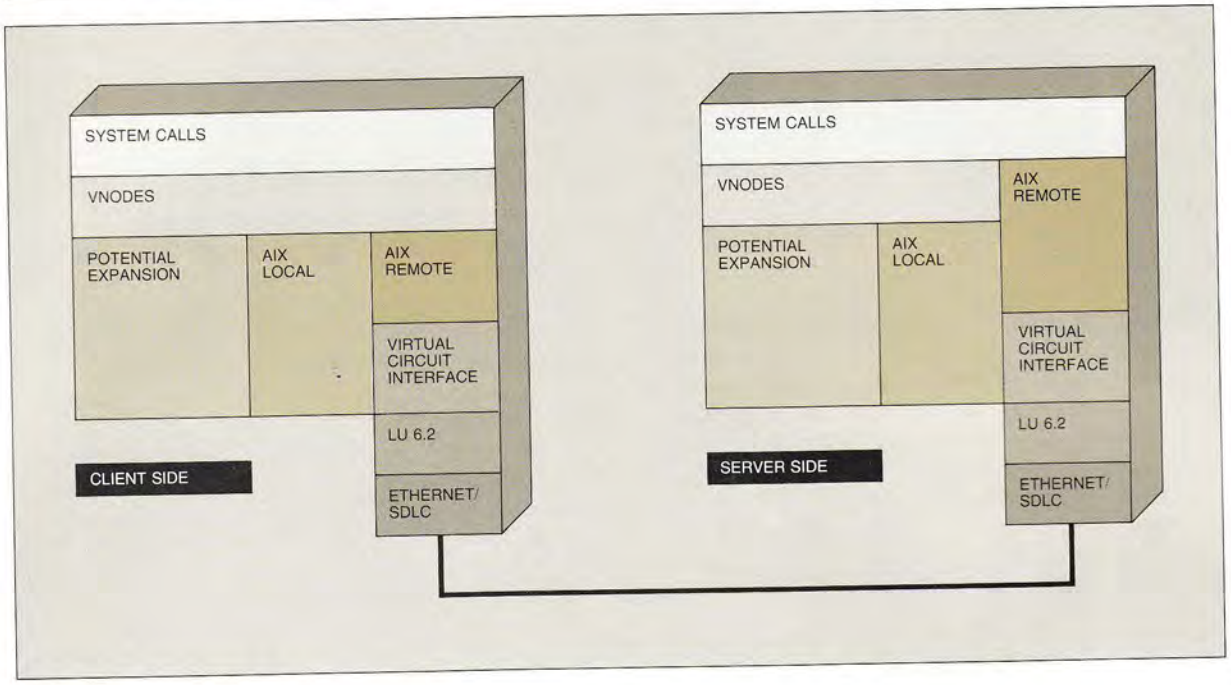


Figure 17 Architectural structure of Distributed Services file system



hierarchy, and—with minor exceptions—file system calls have the same effect regardless of whether files (directories) are local or remote. Mounts, both conventional and remote, are typically made as part of system start-up, and thus are established before users log on. Additional remote mounts can be established during normal system operation.

Figure 16 gives an example view of a shared file system seen by one machine in a single system image cluster. Figure 17 illustrates the structure of the RT Personal Computer Distributed Services files system (RT/DS).

Concluding remarks

AIX is a portable operating system. Written mostly in the C language, it is intended to be migratable to new hardware platforms as they emerge. We believe that in AIX we have combined a portable and versatile industry-standard application programming environment with contemporary hardware and operating system architecture. The resulting operating system is suitable for use with a wide range of computer architectures and customer requirements.

Cited references

1. R. O. Simpson and P. D. Hester, "The IBM RT PC ROMP processor and memory management unit architecture," *IBM Systems Journal* **26,** No. 4, 346–360 (1987, this issue).
2. D. M. Ritchie and K. Thompson, "The UNIX time-sharing system," *Communications of the ACM* **17,** No. 7, 365–375 (July 1974).
3. T. G. Lang, M. S. Greenberg, and C. H. Sauer, "The virtual resource manager," *RT Personal Computer Technology* (pp. 119–125), SA23-1057, IBM Corporation; available through IBM branch offices.
4. J. C. O'Quin, J. T. O'Quin, M. D. Rogers, and T. A. Smith, "Design of the IBM RT PC Virtual Memory Manager," *RT Personal Computer Technology* (pp. 126–133), SA23-1057, IBM Corporation; available through IBM branch offices.
5. D. C. Baker, G. A. Flurry, and K. D. Nguyen, "Implementation of a virtual terminal subsystem," *RT Personal Computer Technology* (pp. 134–136), SA23-1057, IBM Corporation; available through IBM branch offices.
6. R. Krishnamurty and T. Mothersole, "Coprocessor software support," *RT Personal Computer Technology* (pp. 142–148), SA23-1057, IBM Corporation; available through IBM branch offices.
7. L. K. Loucks, "IBM RT PC AIX kernel—modifications and extensions," *RT Personal Computer Technology* (pp. 96–109), SA23-1057, IBM Corporation; available through IBM branch offices.
8. J. M. Bissell, "Extended file management for AIX," *RT Personal Computer Technology* (pp. 114–125), SA23-1057, IBM Corporation; available through IBM branch offices.

9. S. R. Kleinman, "Vnodes: An architecture for multiple file system types in Sun UNIX," *USENIX Conference Proceedings*, Atlanta, June 1986, pp. 238–247.
10. S. Lerom, L. Terrell, and H. Advani, "Configuration methods for a personal computer system," *RT Personal Computer Technology* (pp. 91–95), SA23-1057, IBM Corporation; available through IBM branch offices.
11. P. J. Kilpatrick and C. Greene, "Restructuring the AIX user interface," *RT Personal Computer Technology* (pp. 88–90), SA23-1057, IBM Corporation; available through IBM branch offices.
12. T. Murphy and D. Verburg, "Extendable high-level AIX user interface," *RT Personal Computer Technology* (pp. 110–113), SA23-1057, IBM Corporation; available through IBM branch offices.
13. L. F. Brissette, R. A. Clauson, and J. E. Olson, "PC DOS emulation in a UNIX environment," *RT Personal Computer Technology* (pp. 147–148), SA23-1057, IBM Corporation; available through IBM branch offices.
14. J. C. O'Quin, "The IBM RT PC subroutine linkage convention," *RT Personal Computer Technology* (pp. 131–133), SA23-1057, IBM Corporation; available through IBM branch offices.
15. C. H. Sauer, D. W. Johnson, L. Loucks, A. A. Shaheen-Gouda, and T. A. Smith, "RT PC Distributed Services: File system," *ACM Operating Systems Review* (July 1987).
16. C. H. Sauer, D. W. Johnson, L. Loucks, A. A. Shaheen-Gouda, and T. A. Smith, "RT PC Distributed Services: Overview," *ACM Operating Systems Review*, 18–29 (July 1987).
17. R. W. Scheifler and J. Gettys, "The X window system," *ACM Transactions on Graphics* **5**, No. 2, 79–109 (April 1986).

Larry Loucks *IBM Advanced Engineering Systems IBU, 11400 Burnet Road, Austin, Texas 78759.* Mr. Loucks, the lead architect of the RT system, is a member of the IBM Senior Technical Staff. He received a B.A. in mathematics in 1967 from Minot State University, Minot, North Dakota. Mr. Loucks joined IBM in 1967 in the Fargo, North Dakota, branch office. In 1970, he transferred to Raleigh, North Carolina, where he worked on SNA and other communications products. In 1977, he transferred to Austin, Texas, where he worked on the IBM 5520 Administrative System and the RT system. He has received three IBM Invention Achievement Awards and has been honored with IBM Outstanding Innovation Awards for his work on SNA, on the 5520, and on the RT system.

Charles H. Sauer *IBM Advanced Engineering Systems IBU, 11400 Burnet Road, Austin, Texas 78759.* Dr. Sauer received his B.A. in mathematics and Ph.D. in computer sciences from the University of Texas at Austin in 1970 and 1975, respectively. He joined IBM at the Thomas J. Watson Research Center in 1975. From 1977 to 1979, Dr. Sauer was an Assistant Professor of computer sciences at the University of Texas at Austin. In 1979, he returned to IBM Research, and in 1982 transferred to the IBM Communication Products Division Laboratory in Austin, Texas. Currently, Dr. Sauer is a Senior Technical Staff Member and lead architect for AIX. Dr. Sauer has published three textbooks, *Computer System Performance Modeling*, co-authored with K. M. Chandy, *Simulation of Computer Communication Systems*, co-authored with E. A. MacNair, and *Elements of Practical Performance Modeling*, also co-authored with E. A. MacNair. He has received IBM Outstanding Innovation Awards for the creation and basic design of the Research Queueing Package (RESQ) and for the RT Personal Computer Virtual Resource Manager.

Reprint Order No. G321-5300.