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

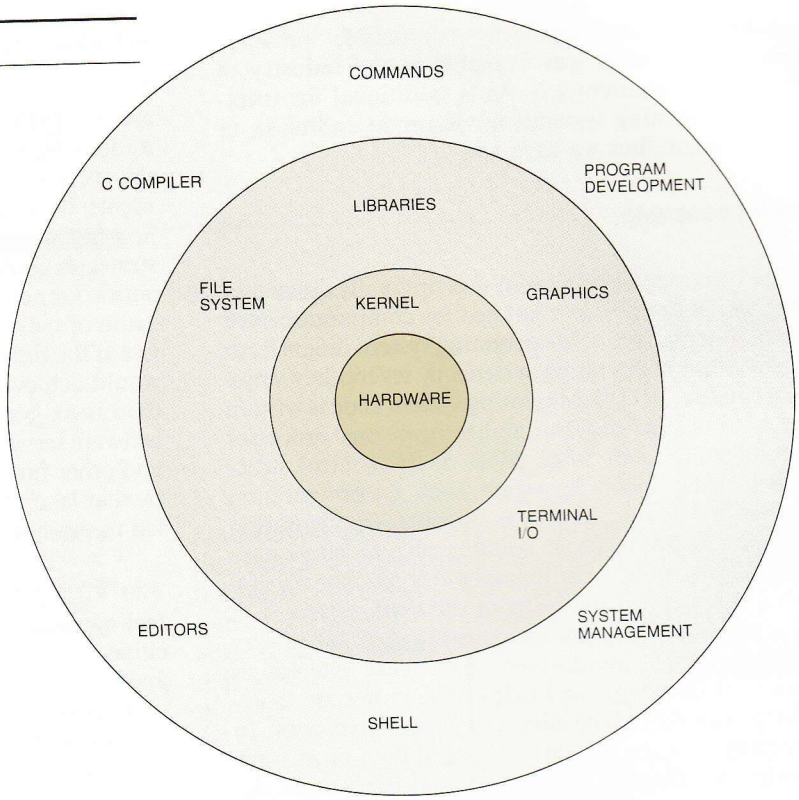


Figure 2 AIX system structure

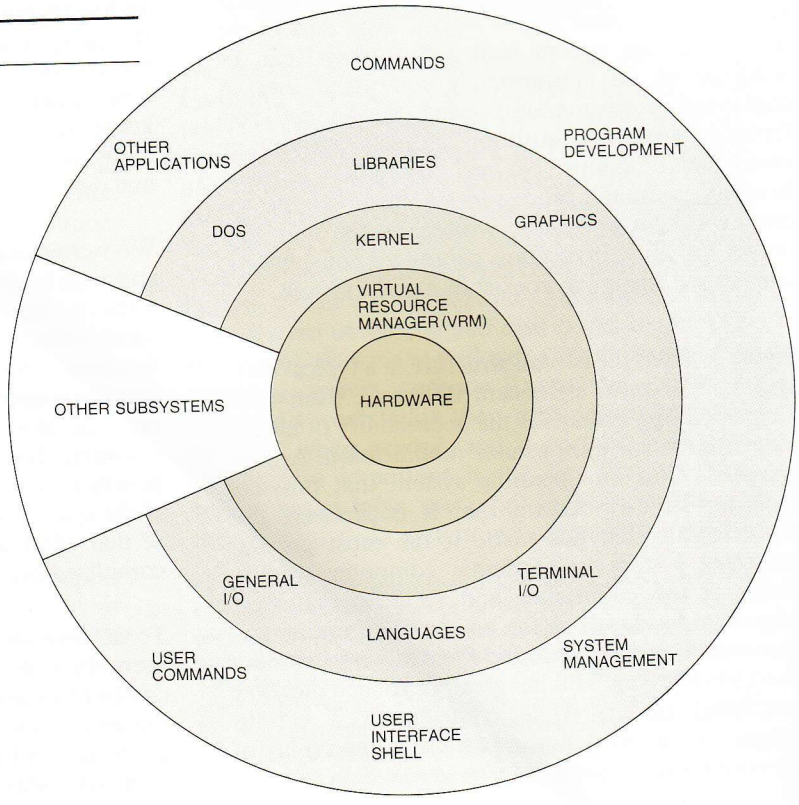
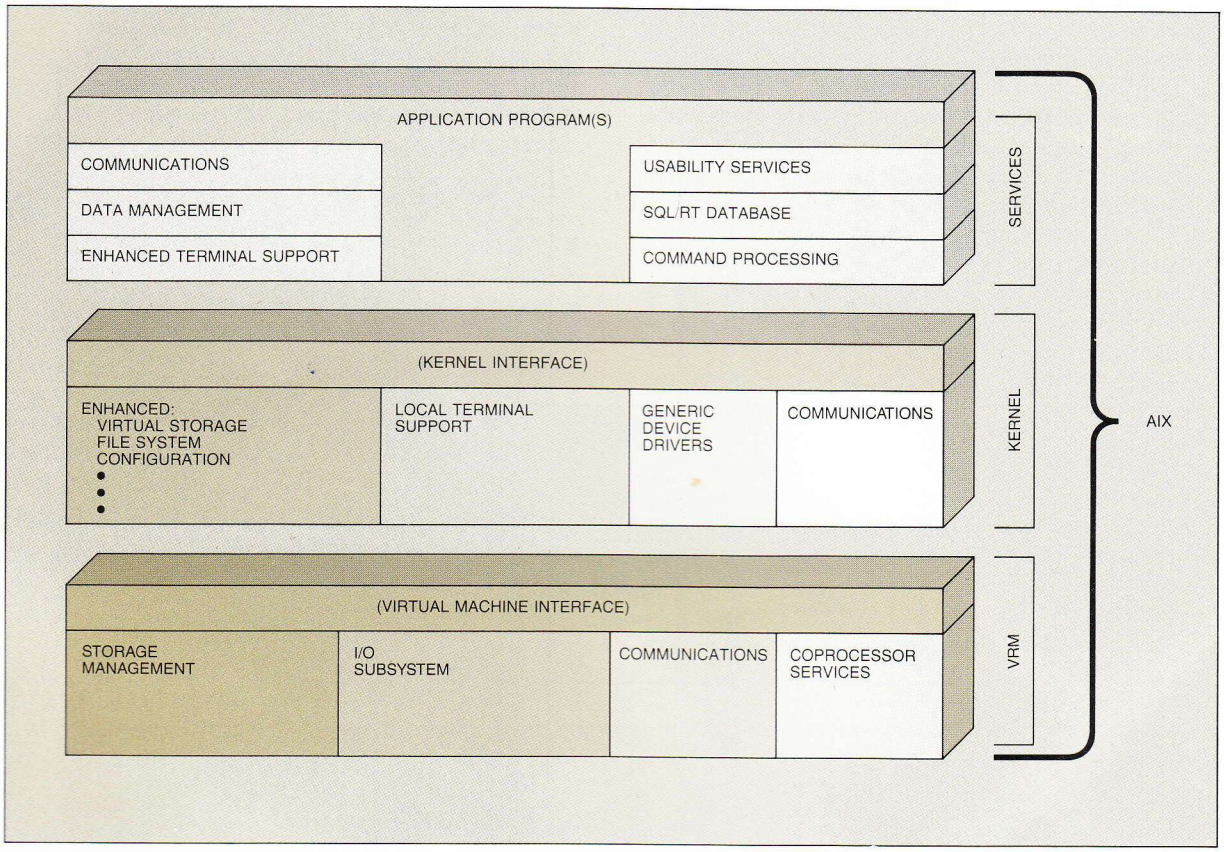


Figure 3 **Interface levels of AIX**



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.

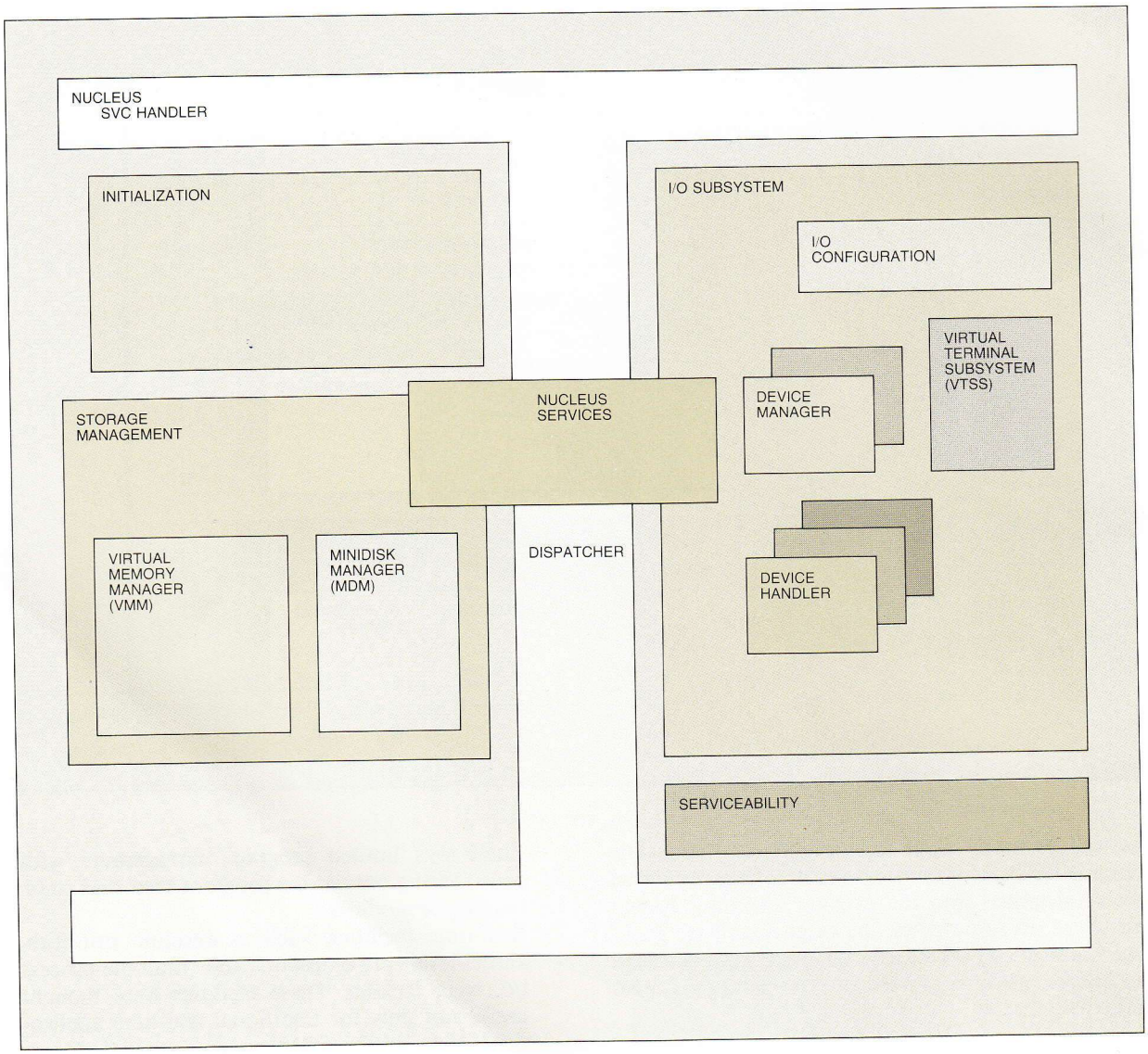
Creating a virtual environment for the AIX kernel

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

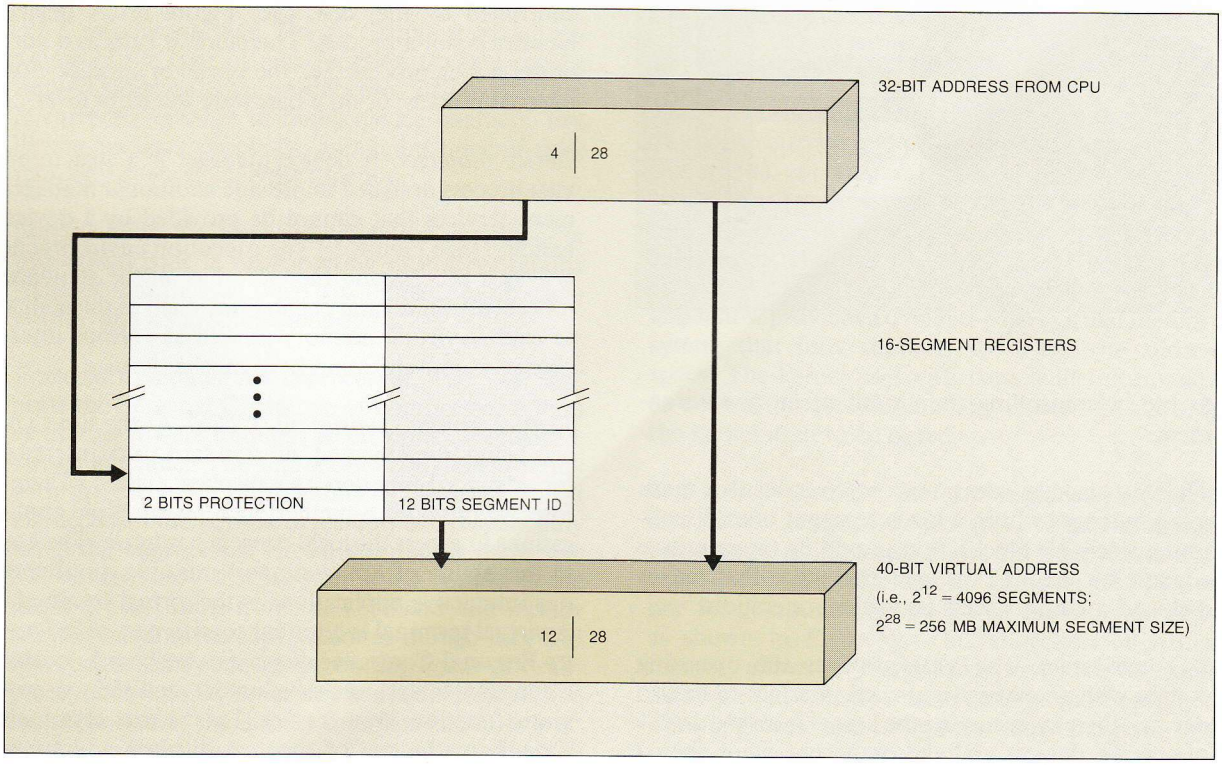
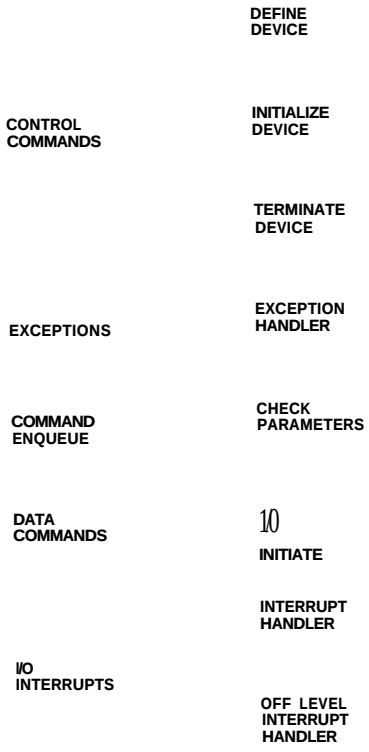


Figure 6 Device handler structure



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 (Le., 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 múltiple applications to access the local consolé hardware. The key to this ability of AIX to support múltiple 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 terminais time-share the use of the real displays and input