THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 1, January 1983
Printed in U.S.A.

The 3B20D Processor & DMERT Operating System:

Overview, Architecture and Performance of DMERT

By J. R. KANE, R. E. ANDERSON, and P. S. McCABE

(Manuscript received March 22, 1982)

A process-oriented operating system, the Duplex Multiple Environment Real Time (DMERT) operating system, was designed for the 3B20D Processor and offers both real-time and time-shared operation, with emphasis on high availability. The design objectives and architecture of the DMERT operating system and an explanation of how the system performance is characterized are presented. A companion article describes in depth the DMERT operating system design.

I. INTRODUCTION

The direct predecessor of the 3B20D, the 3A Processor, included a real-time monitor known as the Extended Operating System (EOS).[1] Experience with EOS demonstrated that applications could develop their software with less effort and that synergy resulted between the hardware and software developers. Hence, the hardware is optimized to support the software and the software, in turn, uses the hardware in the most effective manner. The success of EOS led to the decision to support 3B20D applications with a more extensive operating system than EOS. The operating system that resulted is known as the *D*uplex *M*ultiple *E*nvironment *R*eal *T*ime (DMERT) operating system.

The basic architecture of DMERT is based on an earlier system named MERT,[2] which was derived from the *UNIX** operating system.[3] Both the *UNIX* and the MERT operating systems were origi-

* Trademark of Bell Laboratories.

nally developed to execute on commercial equipment; today *UNIX* operating systems are used widely for time-sharing on a variety of computers. The "D" in "DMERT" reflects one of the characteristics that distinguishes it from the previous two operating systems, namely that DMERT is designed to execute on a *duplex* 3B20 Processor. Thus, the DMERT architecture draws upon concepts from EOS, MERT, and *UNIX* operating systems.

The applications using the 3B20D Processor have been described in detail in a previous article.[4] Notice that while different, they have several common characteristics. First, a major component of the application is software. Second, the major mission of this software is real-time oriented with response times as short as several milliseconds. Third, each application has a need for continuous operation 24 hours a day, 7 days a week and hence stringent processor availability requirements. Fourth and finally, each application is to be operated over a long period of time, which requires extensive software for monitoring and reporting on system status as well as changing and upgrading the system while it is in operation.

This paper describes the development objectives of DMERT, which were chosen to satisfy the above application characteristics. The operating system architecture used to achieve these objectives is next described. Finally, this article describes the performance characterization of DMERT. The design details of DMERT are presented in depth in the next article[5] in this Journal.

II. DMERT DESIGN OBJECTIVES

2.1 Support multiple real-time applications

It is necessary for the DMERT operating system to support many applications, each with different real-time demands. Some applications include data bases that need many disk jobs serviced quickly. Others control telecommunication equipment requiring rapid response to an event such as an interrupt and dedicated processing capacity for an interval thereafter. To satisfy these diverse needs, a design objective was established to provide modularity in the operating system to allow a high degree of application tailoring.

2.2 Improve application development productivity

The real-time applications of the 3B20D Processor often have major software components that are not time critical. The "rule of thumb" stating that 80 percent of the time is spent on 20 percent of the software generally applies to these applications. Hence, a design objective of DMERT was to support a feature-rich operating system environment for the non-time-critical software while retaining real-time responsiveness for the rest.

To increase productivity of the developers, an objective of efficiently supporting a programming language at a level substantially higher than assembly language was established. (See the previous article[6] for further details.)

To allow technology upgrading of the 3B20D Processor without impacting the application software, an objective of shielding application programmers from hardware implementation details, such as memory protection[7] and I/O devices,[8,9] was established.

2.3 Error tolerant design

To meet the reliability objectives of the 3B20D Processor, it is necessary to support software packages for error checking and recovery. Some of these are described in subsequent articles.[10–12] To reduce the complexity of both the operational and recovery components of the system, a design objective was established to separate recovery software from the core of the system.

An objective of incorporating extensive internal consistency and integrity checks within all software components was established to ensure that critical software modules protected themselves from errors in other parts of the system.

III. DMERT ARCHITECTURE

3.1 Processes

One of the basic design goals for DMERT was to build modular and independent processes, each having localized data known only to itself. Hence, the notion of a process is fundamental to the DMERT architecture, which is essentially composed of a kernel and a collection of cooperating, concurrent processes. The following sections define what a process is, how processes communicate with each other, and how they are used by DMERT applications.

3.2 Definition of a process

A process is a collection of related, logical segments (programs and data) that can be brought into memory to form an executable entity. A segment is the basic memory entity in DMERT. A segment is composed of 1 to 64 pages, each 512 32-bit words in length. Segments can grow dynamically in increments of a page. A process consists of at least three segments: code or text, a stack used for temporary data, and a special type of data segment called a process control block (pcb). The pcb segment contains unique information that identifies the process to the operating system. This information includes the process number, type of process, priority, and address space qualifiers that define the virtual address space for a process. Each process has its own virtual address space of up to 128 segments. These virtual addresses

are mapped to physical addresses by 3B20D hardware under the control of the DMERT operating system.

A process can be dynamically created to perform a set of functions and then to gracefully terminate itself when its task is finished. Processes that continuously perform work remain "alive" at all times; however, they may sleep or be inactive until an interrupt fires. These features allow main memory to hold only the processes necessary, at a given point in time, to support the application.

3.3 Process types

DMERT has four basic types of processes: kernel, kernel process, supervisor, and user. DMERT may be viewed as a hierarchy of virtual machines, where successive levels put additional restrictions on access rights and further remove the programmer from the details of the physical machine. However, the higher levels may take advantage of services provided by the lower levels. In general, the higher the level, the more services available to the application programmer; the lower the level, the more real-time-efficient the program execution. This level structuring of virtual machines permits DMERT to manage real-time applications, while at the same time providing the flexibility of a time-sharing system for background tasks. This approach avoids contention for system resources with the high-priority tasks and simplifies the implementation effort for lower priority tasks.

3.3.1 Kernel

The DMERT kernel provides the most primitive virtual machine. The kernel handles hardware interrupts, timer interrupts, and operating system traps. In all cases, the kernel saves the state of the interrupted process, provides whatever service is requested, and then restores the state of the interrupted process. The kernel services are fairly primitive but they execute efficiently.

Also part of the DMERT kernel are special processes that provide scheduling, memory management, and other services. For example, the memory manager and the scheduler are two of the special processes in DMERT. The memory manager loads processes into main memory, selects segments to be swapped out to disk when additional main memory is required, and provides routines that may be called by the kernel. The scheduler controls the execution of time-shared processes, that is, supervisor and supervisor-user processes. Special processes behave as kernel processes, except that they do not have their own virtual address space, but rather reside in the kernel's address space. These special processes communicate with the kernel through function calls instead of operating system traps, and they have access to global system data.

3.3.2 Kernel processes

Kernel processes comprise the next virtual machine layer in DMERT. They are completely interrupt-driven and are designed to provide time-critical processing in a real-time environment. Kernel processes have their own virtual address space. However, they share the kernel's stack and the kernel's message buffer segment. Swapping is not necessary with kernel processes because their segments are locked in memory to ensure rapid response to events such as interrupts. The various peripheral device drivers and the file manager, which implements a hierarchical file system, are examples of kernel processes.

At process build time, kernel processes are set up to share the operating system's stack and message buffers. This design was chosen for quick access to arguments of operating system traps and fast message communications between processes. Since kernel processes use the kernel's stack, they must run until they complete their task and then return control to the kernel.

3.3.3 Supervisor and user processes

Supervisor processes form the third layer of virtual machine. These processes can use all the services provided by the kernel and kernel processes. Supervisor processes provide time-sharing services that can be considered background tasks. They share the real time of the processor with each other according to priorities administered by the scheduler, which is a special process. In general, supervisor segments are not locked in memory, but can be swapped out. Thus, supervisor processes take much longer to dispatch than either special or kernel processes.

Supervisor processes can be designed to run "stand-alone" or they may be used to implement a fourth virtual machine layer called user processes. The DMERT process manager is a supervisor process that does not support a user level. However, the *UNIX* supervisor provides a user environment identical to that seen by a *UNIX* program. This is done through code at the supervisor level that calls upon the services of lower virtual machine layers. DMERT can simultaneously support multiple supervisors, each supporting its own user processes. It should be noted that while DMERT treats a supervisor/user combination as a single process with a dual address space, both levels exist conceptually. Also, supervisor-level code executes more efficiently than user-level code because a supervisor has direct access to the lower level primitives, while the user interface to these primitives is coordinated by the user's supervisor.

3.3.4 Interprocess communication

DMERT provides a rich set of interprocess communication and

synchronization mechanisms including messages, events, interprocess traps, and shared memory. These interprocess communication primitives are fundamental to the DMERT structure. Most of the system services are requested by an exchange of events and messages between a requesting process and either a system process or the kernel.

3.3.4.1 Messages. Processes are in general separate and distinct entities. Two processes working together on a task must be able to exchange information. To satisfy this need messages may be sent from any level process to any other level process. These messages can be up to several hundred bytes long. The sender need only know the target process number and a pre-agreed format of the message. An optional acknowledgment capability is provided so the sender can synchronize actions with the receiver.

3.3.4.2 Events. Communications between processes may occur using an event mechanism. An event is a single bit that is set by DMERT or a process and can be interrogated by the receiving process. Presently, 32 bits are available of which the DMERT operating system reserves 16 bits for its use. Thus, two or more processes can communicate internal states using events.

3.3.4.3 Interprocess traps. A mechanism exists in DMERT to allow a lower-level process to support a higher-level process. A user-level process may trap to a supporting supervisor and a supervisor may trap to a kernel process. Trapping implies a transfer of control from one process to another with the passing of input parameters to the target process. The lower-level process returns status and control back to the trapping process after it has completed the required support work.

3.3.4.4 Shared memory. Processes are built with a view of their own virtual address space and in general cannot access any other process's process's address space. This affords protection; however, sharing large amounts of data is difficult. Cooperating processes that must exchange information at rates higher than those supported by messages or events can share segments. A shared segment is a part of the virtual address space of several processes simultaneously. The application must control access to the shared data.

3.4 Multiple environment support of applications

DMERT simultaneously supports both a real-time and a time-sharing philosophy. Kernel and kernel processes operate in a real-time environment and have first call on the available real time of the 3B20D Processor. The remaining time is shared among supervisor and user processes.

Most telecommunication applications build their own virtual machine or "operating system" as a kernel process that can respond to

the time-critical stimuli characteristic of telecommunications. This kernel-process approach also allows fine tuning of the telecommunications operating system, independent of the DMERT operating system. In at least one case, the application specific virtual machine is a kernel process that runs in emulation mode. Being a microcoded machine, the 3B20D Processor can efficiently execute another machine's instruction set. By using the emulation mode, existing debugged application code, including operating systems, can be carried forward to the 3B20D Processor and DMERT with little additional software development effort.

Some applications spread their functions over the existing DMERT virtual machines. For example, the time-critical functions related to disk and data link usage are implemented as kernel processes, and the administrative and postprocessing functions are implemented as supervisor and user processes. An application of this type is normally used as a data base management system and/or a data communications system. Generally, an application fine tunes its system by moving processes to different execution levels within the virtual machines.

IV. PERFORMANCE

The 3B20D/DMERT system is capable of providing computing services in a stand-alone mode; however, usually it is utilized by surrounding it with application hardware, firmware, and software. The application hardware may include additional units identical to those already a part of the processor complex (e.g., additional memory, disk, data links), or it may include hardware unique to the application system (e.g., bus controllers, or time- and space-division switches). The application software frequently includes drivers, schedulers, and fault-recovery facilities, as well as the more usual "application programs." As a result of this diversity of software interfaces to DMERT, the performance modeling and measuring of the application system requires an extensive performance characterization of DMERT, rather than the more traditional benchmark approach used in general-purpose, computer-performance evaluation. The following sections describe the approach taken and the type of performance data made available to DMERT applications.

4.1 Performance characterization

Since the application software may interface DMERT at all levels of the virtual machine (hardware, firmware, and the software levels of kernel, kernel process, supervisor, and user process), the performance characterization of the system must include data for all of these levels. In addition, the application can make use of a variety of DMERT system resources such as memory, peripheral devices, message buffers,

etc, and hence these resources must be accounted for in the performance characterization as well.

The 3B20D performance characterization addresses four areas: the operating system, input/output, DMERT services, and DMERT overhead. Each of these areas will be discussed in more detail, and the sum of these areas covers all significant aspects of DMERT performance, as well as providing some models for the application developers to use in the analysis of the application system performance.

4.1.1 Operating system characterization

The goal of the performance characterization of the operating system was to identify the cost in resources for every service available at every level. The predominant service interface in DMERT is the Operating System Trap (OST), and hence every significant OST in DMERT was characterized with respect to its central processing unit (CPU)-time at the various modes and execution levels available in DMERT.

The kernel, supervisor, and user OSTs cover a broad range of DMERT services:

(*i*) *Timing:* clock reading and setting, single and repetitive timeouts, and process sleeping requests.

(*ii*) *Memory management:* locking and unlocking of memory segments, growing and shrinking of memory segments, and swapping of memory segments.

(*iii*) *Scheduling and interrupting:* protecting against interrupts (critical region), priority changing, and fielding of interrupts.

(*iv*) *Interprocess communication:* interprocess message sending and receiving, and sending and fielding of interprocess events.

(*v*) *Process switching and other functions:* switching from one process to another, changing the execution level of a kernel process, creating another process, faulting another process, routing and rerouting of standard inputs and outputs.

4.1.2 Input/output system

The input/output (I/O) system of DMERT is characterized from an *internal* point of view; that is, each of the kinds of I/O services are characterized with respect to their primary resource consumption. The various I/O services are all measured with respect to the CPU-time consumed for each transaction and the maximum throughput rate based upon the elapsed time for each transaction.

A basic set of operations can be performed on most I/O devices: open, close, read, and write. Seeking is a disk-only service, and rewinding is a tape-only service. Except for data links, read and write operations can be invoked in several ways, depending on the physical organization of the data on the device. Most file operations can be

invoked directly or through the file manager. The devices supported by DMERT include disks, tapes, terminals, and data links. Finally, the disk I/O services can be invoked in several modes: normal (synchronous, buffered I/O), asynchronous (which allows the invoking process to continue running while the I/O request is being serviced), physical (no buffering constraints, or services) and synchronous writes (which guarantee an immediate write to the disk, rather than the potential delayed write possible in the normal mode).

Most of these I/O services are available to kernel, supervisor, and user processes via separate OSTs. Each OST is characterized with respect to CPU-time and maximum device throughput for each of the applicable cases.

4.1.3 DMERT applications services

Application software may utilize high-level services from DMERT as well as the more primitive OST-invoked services. The Craft Interface[12] is an example of this general category covered by the term DMERT services. The Craft Interface system provides an extensive and sophisticated set of terminal-oriented facilities that are used by both DMERT and the application software. Additional examples of DMERT services are the diagnostic and audit facilities that are a part of DMERT, but also may be invoked by application software.

Of these DMERT services, the Craft Interface has been characterized for performance owing to its importance to early DMERT applications. The performance characterization chosen was to measure the CPU-time usage of the three most important application services: the Program Documentation Standard (PDS) Shell, the Control and Display Administrator, and the Output Spooler. Each of these services was measured with a range of appropriate job sizes.

4.1.4 DMERT overhead

The final category in the performance characterization of DMERT is the system overhead. While system overhead is not invoked explicitly as a service, it provides the essential services of the operating system. There are several types of system overhead:

(*i*) *Functional:* provides for the capability of a multi-environment, real-time operating system by handling the timing-based facilities for interrupt servicing, scheduling, craft terminal polling, and data link polling.

(*ii*) *Sanity:* provides for sanity and overload monitoring by resetting hardware sanity timers, monitoring DMERT and application sanity timers, and checking for process lock-out conditions.

(*iii*) *Preventive maintenance:* provides for routine preventive maintenance exercising and running routine software audits.

(*iv*) *Fault maintenance:* provides for fault detection, location and recovery, including removing faulty units from service, and testing and restoring replacement units. This overhead is incurred only when a fault occurs.

(*v*) *Services:* provides for other non-electable services not covered by the previously described types, such as Craft Interface services and Plant Measurements services invoked in providing the processor system.

The DMERT operating system overhead is characterized in terms of CPU time and is expressed as a percentage. The first two types of overhead (functional and sanity) constitute the "continuous" overhead seen by the application, and cannot be controlled or throttled by the application. This component of the overhead is less than 5 percent for DMERT. The preventive maintenance overhead can be controlled in two ways: routine software audits can be throttled so that their peak resource usage is limited to a desired value, and the routine diagnostic exercising can be scheduled during times of light load. The fault maintenance overhead is measured as a single-fault, worst-case scenario for the fault detection, isolation, and recovery, as well as the testing and restoral, of the repaired unit. The total resource usage is averaged over the specified two-hour repair interval. The services overhead includes the normal administrative activities necessary to maintain and administer the processor complex. The total system overhead for functional, sanity, preventive, and fault maintenance and services is less than 15 percent for DMERT.

4.2 Performance measurement techniques

The key resource in the 3B20D/DMERT system, and in the applications systems built upon it, is CPU time. It is shared among many processes, both DMERT and application, in four execution modes, at 16 execution levels and 256 priority levels. The sharing is interrupt driven, with preemption from processes at higher priorities and at higher execution levels. While it is relatively simple to measure CPU time in an overall sense, it is a very complex job to measure the CPU time used by a particular process, function, or service. To solve this problem, the DMERT kernel was instrumented. Finally, hardware monitors were used to measure the performance of the processor complex, and also to verify the correctness and accuracy of the software performance measurement instrumentation.

4.2.1 Kernel instrumentation

The DMERT kernel was extended to provide detailed accounting of the CPU time usage (by execution mode and level) for the system as a whole, and also for each process. To keep the overhead low enough

to allow the instrumentation to be a permanent part of DMERT, a statistical sampling approach was used. Each 10 milliseconds the kernel records the process currently executing, together with the execution mode and level, assigning the previous 10 milliseconds to that process. Over a reasonable period of time (seconds or more) these statistical sampling results will converge arbitrarily close to the actual CPU time usage.

V. SUMMARY

An overview has been given for the objectives of DMERT based on its goal of providing a high-reliability, real-time processor system for telecommunications applications. This overview has indicated some of the development objectives of DMERT and has given the approach for the performance characterization of the whole system. Also included is a description of the various kinds of operating system overhead, including measured values for DMERT, and a description of the performance measurement instrumentation within the DMERT kernel.

VI. ACKNOWLEDGMENTS

Many members of Bell Laboratories' staff contributed to the successful introduction of DMERT. Key contributors to the work described in this article and not mentioned as authors or references were: S. F. Ho, P. C. Kao, T. K. Liu, T. M. Raleigh, E. P. Schan, D. S. Trushin, and S. M. Udupa.

REFERENCES

1. C. H. Elmendorf, "Meeting High Standards with the Extended Operation System," *Bell Lab Rec.,* (March 1980), pp. 97–103.
2. H. Lycklama and D. L. Bayer, "The MERT Operating System," B.S.T.J., *57,* No. 6., Part 2 (July 1978), pp. 2049–86.
3. D. Ritchie and K. Thompson, "The *UNIX* Time Sharing System," B.S.T.J., *57,* No. 6, Part 2 (July 1978), pp. 1905–29.
4. R. W. Mitze and W. C. Schwartz, "The 3B20D Processor & DMERT Operating System: 3B20D Processor and DMERT As a Base for ESS Applications," B.S.T.J., this issue.
5. M. E. Grzelakowksi, J. H. Campbell, and M. R. Dubman, "The 3B20D Processor & DMERT Operating System," B.S.T.J., this issue.
6. B. R. Rowland and R. J. Welsch, "The 3B20D Processor & DMERT Operating System: Software Development System," B.S.T.J., this issue.
7. M. W. Rolund, J. T. Beckett, and D. A. Harms, "The 3B20D Processor & DMERT Operating System: 3B20D Central Processing Unit," B.S.T.J., this issue.
8. R. E. Haglund and L. D. Peterson, "The 3B20D Processor & DMERT Operating System: 3B20D File Memory Systems," B.S.T.J., this issue.
9. A. H. Budlong and F. W. Wendland, "The 3B20D Processor & DMERT Operating System: 3B20D Input/Output System," B.S.T.J., this issue.
10. R. C. Hansen, R. W. Peterson, and N. O. Whittington, "The 3B20D Processor & DMERT Operating System: Fault Detection and Recovery," B.S.T.J., this issue.
11. J. L. Quinn and F. M. Goetz, "The 3B20D Processor & DMERT Operating System: Diagnostics," B.S.T.J., this issue.
12. M. E. Barton and D. A. Schmitt, "The 3B20D Processor & DMERT Operating System: Craft Interface," B.S.T.J., this issue.