DMERT—An operating system for telecommunications systems

by S. F. HO, C. J. RICHARDSON, W. C. SCHWARTZ

AT&T Bell Telephone Laboratories
Naperville, Illinois

ABSTRACT

The duplex multi-environment real-time (DMERT) operating system is a process-oriented, fault-tolerant operating system designed to provide a versatile software base for telecommunication systems. DMERT provides general fault recovery capabilities, virtual machine layers to meet application needs, a UNIX environment, and I/O interfaces to peripheral devices. This paper gives a detailed description of the DMERT architecture and its capabilities.
INTRODUCTION

A major goal of the duplex multi-environment real-time (DMERT) operating system is to provide a versatile software base to fulfill the varied processing needs of telecommunication applications. While the needs of these applications are different, they have several common characteristics. First, a major component of these applications 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 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. To satisfy these needs, DMERT is designed to:

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 databases 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 that allows a high degree of application tailoring.

2. Improve application development productivity. Software for telecommunication applications is usually implemented in assembly language. To increase productivity of the developers, an objective of efficiently supporting the C programming language was established. Telecommunications systems often have major software components that are not time critical. Hence a design objective of DMERT was to support a UNIX interface as a familiar operating system environment for the non-time-critical software.

3. Be fault tolerant. To meet the reliability objectives of the applications, it is necessary to support software packages for error checking and recovery. In order 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.

In summary, DMERT is a process-oriented operating system designed to support both real-time and time-shared operations, with an emphasis on high reliability and availability. This paper outlines the DMERT capabilities and describes how these design objectives are achieved. The second section gives an architectural overview of DMERT. The process types, process communication primitives, and the time-sharing and real-time scheduling policies are described. The last section highlights DMERT features for achieving the high reliability and availability goals.

DMERT ARCHITECTURE

The architecture of DMERT is based on an earlier system, MERT, a real-time operating system derived from the UNIX operating system. The “D” in DMERT reflects one of the characteristics that distinguishes it from the previous two operating systems, namely DMERT is designed to execute on a fault-tolerant 3B20D duplex processor. Thus, the DMERT architecture is dependent on proven concepts from UNIX and MERT, which are extended to support highly reliable telecommunication applications.

One of the basic 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 and how processes communicate with each other.

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 bytes (512-bit) in length. Segments can grow dynamically in increments of a page. A process typically consists of four segments: code or text, a stack used for temporary data, a data segment containing global 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 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.

Besides the regular process entries for handling process
events and interrupts, any process may have a fault entry. A
process is entered at its fault entry when another process sends
a fault to this process, or a hardware/software fault is detected
by the system when the process is running. The purpose of
the fault entry is to give the faulted process an opportunity to
perform some recovery action based on where the process was
faulted and why. Every faulted process has a fault code that
indicates the nature of the fault and state information that
indicates the state of the process at the time the fault
occurred.

A process can be dynamically created to perform a set of
functions and then terminated when the task is completed.
Processes that continually perform work remain "alive" at all
times, however, they may sleep or be inactive until an event,
message, or interrupt occurs. An inactive process may be
swapped out to the disk, i.e., the process memory image is
copied to the disk and the memory occupied by the process is
released. This keeps main memory to be loaded with the
working set of processes at a given point in time.

**Process Types**

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

**Kernel**

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

Also part of the DMERT kernel are special processes that
provide scheduling, memory management, and other ser­
"vices. 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 pro­
ces communicate with the kernel through function calls
instead of operating system traps, and they have access to
global system data. For example, the memory manager and
the scheduler are two special processes in DMERT. The
memory manager loads processes into memory, selects seg­
ments to be swapped out to disk when additional main memo­
ry is required, and provides routines that may be called by the
kernel. The scheduler controls the execution of time-shared
processes, i.e., supervisor and UNIX processes.

**Kernel processes**

Kernel processes comprise the next virtual machine layer in
DMERT. They are completely interrupt driven and are de­
signed to provide time-critical processing in a real-time envi­
ronment. Kernel processes have their own virtual address
space. However, they share the kernel's stack and the kernel's
message buffer segment to provide quick access to arguments
of operating system traps and fast message communications
between processes. Kernel process segments are always
memory resident to ensure rapid response to events such as
interrupts. The various peripheral device drivers and the file
manager are examples of kernel processes.

**Supervisor and UNIX processes**

Supervisor and UNIX processes form the third layer of
virtual machine. These processes can use all the services pro­
"vided by the kernel and kernel processes. Supervisor and
UNIX 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. In general, these processes are not locked in
memory and can be swapped out. Thus, supervisor and UNIX
processes may take longer to dispatch than special and kernel
processes.

UNIX processes are actually supervisor processes, but a
shared library hides the supervisor interface and replaces it
with a UNIX environment. Conceptually, supervisor and
UNIX processes are different, but they are the same from the
operating system's point of view.

**Inter-process Communication**

DMERT provides a rich set of inter-process communication
and synchronization mechanisms including messages, events,
inter-process traps, and shared memory. These inter-process
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 pro­
cess and either a system process or the kernel.

**Messages and ports**

Processes are in general independent 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. The
sender needs only to know the target process number and a
pre-agreed format of the message. An optional acknowl­
edgement message is provided so the sender can synchronize
actions with the receiver.

Process ports permit processes to communicate with each
other without knowing each other’s process number. A process port is a globally known “device” to which a process may attach itself for receiving messages. Other processes may communicate with a process connected to a port by sending messages to that port. Thus process ports permit unrelated processes to communicate with each other.

Events

Communications between processes may occur using an event mechanism. An event is a one-bit message that can be sent from one process and be interrogated by the receiving process. Presently, 32 events are available, of which the DMERT operating system reserves 16 for its use. Application processes communicating using events can define the usages of the remaining 16 events. Thus, two or more processes can communicate internal states using events.

Inter-process traps

Trapping implies a transfer of control from one process to another with the passing of input parameters to the target process. The trapped process returns status and control back to the trapping process after it has completed the requested service. Any process may trap to another process, as long as the argument-passing protocol is mutually agreed upon.

Shared memory

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

Process Scheduling

The DMERT operating system simultaneously supports both a real-time and a time-sharing philosophy. Kernel processes operate in the real-time environment. The remaining processor time is shared among supervisor and UNIX processes.

Real-time

DMERT’s real-time allocation strategy is based on execution levels and preemptive scheduling. DMERT maintains a process hierarchy based on 16 execution levels. A kernel process can belong to levels 3 through 15 (levels 0 through 2 are reserved for the time-sharing environment). Kernel processes are used to implement tasks with stringent real-time requirements. DMERT dispatches processes at the highest execution level first. Generally, once a kernel process is dispatched, it is allowed to run to completion, i.e., until the kernel process relinquishes its control of the processor. However, if another kernel process at a higher execution level is awakened, DMERT preempts the executing process. Upon completion of the preemption process, if no other higher level processes were also awakened, DMERT resumes the suspended process.

DMERT applications are allowed to assign their own processes’ execution levels, thus allowing applications to control and distribute the real-time. This approach is flexible and supports a variety of applications.

Time sharing

The portion of real time not utilized by the kernel and kernel processes is time shared among supervisor and UNIX processes. Processes supporting the time-shared environment, such as the scheduler and memory manager, reside at execution level 2. These processes are at the bottom of the real-time hierarchy and gain control of the processor only after all other real-time work is completed.

Supervisor and UNIX processes execute at levels 0 and 1. The scheduling hierarchy of supervisor processes is based on software priority. The major difference between priority in the time-sharing environment and execution levels in the real-time environment is that DMERT adjusts software priorities dynamically depending on the I/O characteristics of the process and the system load, whereas execution levels are fixed.

RELIABILITY AND AVAILABILITY

The DMERT operating system must be able to support the stringent electronic switching system’s reliability requirements. To minimize the number of system failures and the associated down-time per failure, DMERT supports audits and overload control, progressive initialization, reconfiguration, preventive and corrective maintenance, field updates, and system updates. These features are described in the following sections.

Audit and Overload Control

The DMERT audit package verifies the validity of critical system data. Audit strategies are based on the inherent properties of the data structures and redundancies that are built into the structures. Audits are distributed throughout the system within processes that control the data to be audited. Audits can be issued by manual requests or the audit control structure. The DMERT system integrity monitor (SIM) is responsible for scheduling and dispatching all audits, and for handling all overload conditions. SIM receives overload conditions from DMERT operating system processes. The application and the craft are then notified that these conditions exist.

DMERT overload controls handle conditions in which critical system resources (e.g., message buffers, swap space, etc.) are in short supply or the system’s real-time performance falls
below a predetermined limit. These conditions occur when the system is overloaded with input requests, or sufficient resources are lost due to software errors over a long period of system operation. Most overload strategies involve changing the policy of assigning resources to processes and running audits to recover system resources. The combination of audits and overload control is a powerful mechanism to maintain system integrity.

**Progressive Initialization**

The DMERT recovery strategy attempts to minimize the service disruption caused by an initialization in response to hardware and software faults. Several levels of recovery actions are provided to match the level of initialization to the severity of the fault. Although DMERT attempts to recover at the lowest possible level, the recovery level is automatically escalated if the current level fails.

The initialization of application processes only is the least disruptive or the lowest level of initialization. Applications determine their own recovery strategies. This level of initialization can be requested by a craftperson or by an application process. DMERT administers the initialization counts and timers, but a DMERT operating system initialization is not taken.

The next level of recovery involves initializing DMERT processes as well as application processes. This level is the primary recovery mechanism in DMERT and uses a rollback strategy. The goal of this initialization level is to restore the system to a sane and operational state with minimal effect on service. Each process in the system is notified by its fault entry that a system initialization has been taken. Using state information that is maintained during normal operation, each process cleans up any transactions in progress and then returns. This strategy is effective because only a few processes are actually active at any given time.

If the rollback strategy fails, DMERT is rebooted from disk. Even when such a bootstrap occurs, several regions of memory are protected to maintain some continuity. The successively more severe levels of initialization involve reinitializing these protected regions. However, one protected memory region is preserved for applications, and is initialized only by manual request or a power up.

**Reconfiguration**

DMERT takes full advantage of the redundancy provided by the 3B20D processor. The equipment configuration database maintains information concerning the hardware configuration and hardware error rates. This provides a basis for automatic reconfiguration and allows the recovery strategy to be tuned to meet the needs of the individual applications.

In processing a hardware error interrupt, the unit causing the error is determined. The error count for that unit is then incremented and compared with its error threshold. If the threshold has not been exceeded, the unit remains in service. If the threshold has been exceeded, the configuration management routines decide on the corrective action. This decision is based on the availability and status of a replacement unit. Configuration options include removing the unit, switching in a replacement unit, or continuing operation on the faulty unit.

**Preventive and Corrective Maintenance**

DMERT provides a comprehensive set of diagnostics that can be invoked directly by the craftperson or under program control. Diagnostics ensure the operational capabilities of hardware units.

A routine exercise is performed daily to verify the operation of all units in the system. These units are diagnosed and a status report is generated indicating their conditions.

In addition to the routine exercise, if a unit is removed from service because of a fault condition, diagnostics are scheduled. If the unit fails diagnostics, a report is generated indicating the failure cause. If the unit passes diagnostics, it remains in service. However, to prevent a unit remaining in service that passes diagnostics, but fails repeatedly during actual operation, a count is kept of the number of times operational failure occurs. Any unit that exceeds a predetermined limit may be removed from service, pending some corrective action (e.g., more exhaustive diagnostics and unit replacement).

**Field Update**

Field update, which is typically called overwriting in traditional electronic switching systems, is the problem correction mechanism for DMERT. Field update may be used to modify data and programs on the 3B20D disk or in main memory. Field updates must be performed without disturbing system operations (e.g., call processing, critical system functions, etc.). The features of field update are the ability to change a file both instantaneously and in a temporary way, the ability to update a running function in a running process, the ability to coordinate changes to functions within a process, and the ability to change data contents or the structure of data in a running process. Changes made to the running process update the disk image of the process as well as the main memory image.

**System Update**

DMERT system update provides a safe, reliable mechanism to introduce new versions of DMERT and application software into the 3B20D/DMERT systems, while minimizing service disruption. System update differs from field update in the magnitude of the program and data changes being installed. Normally, a system update will replace all the software in the system, which is a complete reissue of DMERT, application software, and/or data. For this reason, system updates always include a memory reinitialization of all processes and data from disk. Only the protected memory areas are not reinitialized.
SUMMARY

The DMERT system has achieved its objective of providing a cost- and real-time-effective base for a wide variety of telecommunication systems. The concepts of multiple levels of functional support, reliability and availability features, and versatile I/O interfaces provide an adaptable base that can be tailored to many differing needs. More than one hundred DMERT systems have been installed in the field. These systems include electronic switching applications, database applications, as well as add-on extensions to existing switching machines to enhance processing power. The DMERT system is also the basis of a number of telecommunication system designs currently under way. This widespread use of 3B20D/DMERT marks it as a processor/operating system combination of significance in telecommunication systems.

REFERENCES
