Copyright © 1983 American Telephone and Telegraph Company
THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 1, January 1983
Printed in U.S.A.

The 3B20D Processor & DMERT Operating Systems:

Fault Detection and Recovery

By R. C. HANSEN, R. W. PETERSON, and N. O. WHITTINGTON

(Manuscript received March 18, 1982)

The 3B20D Processor is designed to be a high-availability system for utilization in electronic switching systems. This high availability translates into the development of numerous features and capabilities for the 3B20D that distinguish it from other processors. The reliability objectives for the processor are described and related to the subsystems that have been developed to meet each objective. This article discusses processor and peripheral fault recovery, system integrity, and other software subsystems that provide the high availability and maintainability for the processor.

I. INTRODUCTION

The 3B20D Processor has extensive maintenance subsystems associated with it and is designed to meet the high-availability standards of Bell System electronic switching systems. This implies that the processor must perform within an objective of not more than two minutes downtime per service year when used in an electronic switching application. The many subsystems that have been developed to provide the high-availability capability are described in this article. In particular, software and hardware fault recovery are discussed along with the microcode assists for the recovery.

Much evolution has taken place in recovery architectures for electronic switching systems.[1,2] Earlier processor systems used extensive hardware-matching algorithms that resulted in intricate software recovery.[3,4] More recent hardware technologies have enabled the cost-effective design of processor systems with unique fault-detection capabilities.[1,5,6] These capabilities have led to much simpler recovery software. This article describes the detection mechanisms for the 3B20D and the software maintenance architecture.

II. SYSTEM RELIABILITY REQUIREMENTS

The reliability objective for the 3B20D Processor system, as with other similar systems, is to keep the overall system unavailability—i.e., the time that the system cannot be utilized by operational (call processing) functions—below 2.0 minutes per year.[7] In keeping with the ESS processor tradition, the total system downtime is allocated to four general categories: hardware, software, recovery, and procedural.

The processor has 0.4 minute per year allocated to malfunctions in the system hardware. Like other highly reliable systems, the 3B20D is equipped with redundant hardware units. Thus, failures must occur in both redundant units before the system is unable to establish a working configuration. In the case of simultaneous failures in both units, until one is repaired and system integrity is reestablished, the system is considered unavailable. This portion of the overall system downtime is a function of the failure rates of the various components (FIT rate), the system architecture, and the mean time to repair (MTTR). The hardware reliability model for the 3B20D Processor within a given application is dependent on the hardware configuration used and the maintenance technique used (this determines the repair time).

The processor has 0.3 minute per year allocated to malfunctions in the processor operational software. This is a classification of software faults that can render the system features inoperative. This allocation includes cases such as software faults that require a bootstrap to recover the system. As in the case of other high-availability systems, the 3B20D/DMERT system has a design objective of having no software failures the system cannot recover from. To help recover the system against software failures, DMERT has three levels of defenses that attempt to recover the system from such faults: hardware protection, system integrity monitor, and audits. The 3B20D Processor has several levels of hardware protection that detect the sanity of the system software. The system integrity monitor in the DMERT system has an elaborate scheme of software and hardware sanity timers as well as overload detectors that protect the system against software "resource hogs." DMERT audits include all of the explicit audits in the system as well as the defensive checks built into the common processor software. The intent of the audits is to help defend important processes against data mutilation.

The processor has 0.7 minute per year allocated to limitations in fault-recovery programs. These failures are classified by the inability of fault-recovery software to achieve a working configuration of the system due to some hardware failure condition even if a working state of the hardware is possible. These cases are characterized by the necessity for manual intervention to reestablish system integrity or by an automatic initialization to regain system integrity.

The 3B20D has a comprehensive fault-recovery scheme that attempts to recover the system from all foreseeable single hardware fault conditions. In several cases, recovery mechanisms are generated for multiple fault situations (e.g., memory failures) when that is considered to be a probable situation.

Finally, the processor has 0.6 minute per year allocated to procedural errors. This category covers cases where a craft person uses an improper maintenance procedure or follows a poorly designed procedure that results in a machine outage. The 3B20D is designed with a defensive craft interface using the PDS (Program Documentation Standards) and MML (Man Machine Language) languages.[8] The craft interface also includes emergency action and display-page capabilities that attempt to simplify the complexities of maintaining the 3B20D.

The system reliability requirements also include the various aspects of maintaining the 3B20D. These maintainability aspects include diagnostics, transient error analysis, emergency recovery procedures, routine maintenance procedures, growth and retrofit capabilities, system and process update capabilities, and field utilities. Diagnostics are provided to detect and assist the repair of classical hardware failures in the system. The diagnostic requirements include sufficient run-time performance so that a rapid repair can be carried out. Diagnostics provide greater than 90 percent fault detection.

The ability to repair circuitry exhibiting transient failures is provided through fault-recovery error reports. For example, data about transient memory faults is printed out to the craft and includes address and pack location where the error was detected. If that circuit pack continues to have a history of transient errors, the craft has sufficient information to effect a repair. Error analysis capabilities are provided on the 3B20D through the use of fault-recovery messages and error logs.

Emergency recovery procedures are provided to reconfigure the system when automatic recovery does not succeed. These capabilities allow the craft to repair the 3B20D in case of catastrophic failures. These procedures include use of the emergency action page, processor recovery message analysis, and dead-start diagnostics. Routine maintenance procedures are provided to keep the 3B20D in peak operating condition. Growth and retrofit procedures allow hardware additions and removals without affecting the system service. Finally, various utilities are provided with the DMERT system to locate system problems in field installations.

III. GENERAL RELIABILITY AND MAINTENANCE ARCHITECTURE

In this section, we provide an overview of the 3B20D fault-recovery architecture that is described in further detail in later sections. Figure

1 illustrates the hardware architecture of the 3B20D. As is indicated in the figure, the processor system has very loose coupling between any of the mate subsystems. The memory to memory update coupling is provided to keep both active and standby memories identical. This allows the switching of processors without losing the integrity of the software running on the system.

The other coupling between the processors is through the maintenance channel. The maintenance channel provides two capabilities important to the integrity of the processor. First, it provides a control and communication bus for the purpose of diagnosing the off-line processor from the on-line processor. Second, it provides low-level maintenance control for fault-recovery programs so that a switch in processor activity can be carried out with no operational interference. In addition, other maintenance controls can be exerted over the channel to start an initialization sequence on the other processor or to stop execution on the other processor. One other coupling, the Dual Duplex Serial Bus Selector (DDSBS), allows either processor to talk to any peripheral controller. Thus, no matching techniques are utilized between major subsystems or peripherals in the 3B20D for the purposes of fault detection in the hardware. This means that unique fault-detection techniques are essential in each subsystem of the 3B20D.

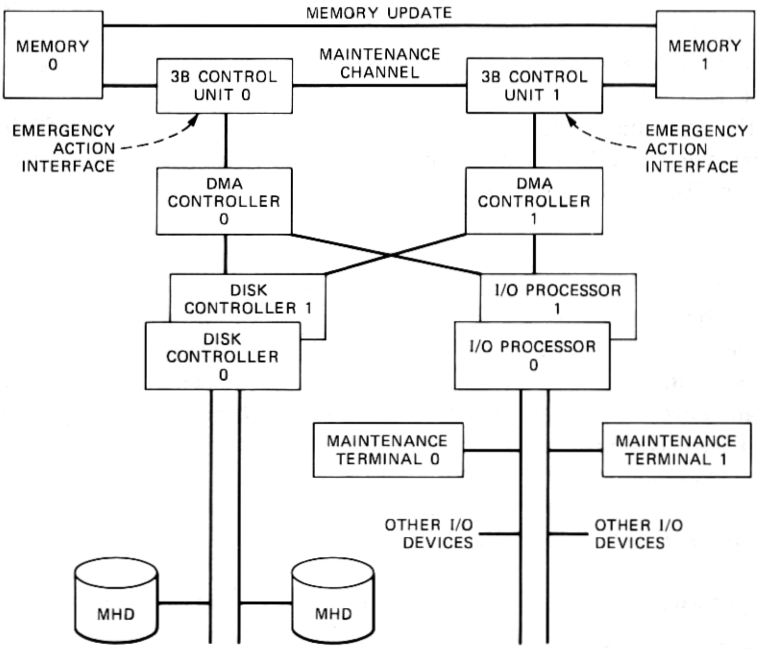


Fig. 1—The 3B20D system architecture.

To provide these detection algorithms, extensive use of local matching circuits, parity techniques on all buses, Hamming detection with single-bit error correction on the main store, cyclic redundancy codes on the disks, and numerous sanity timers throughout the control unit and peripherals are used as the primary fault-detection techniques. In addition, routine diagnostics are used to detect failures in the fault-detection hardware itself. Other routine sanity checks are used to ensure that peripheral subsystems are healthy. Finally, system-integrity checks catch certain subtle problems that are not caught by unique detectors.

3.1 Fault-recovery architecture

When any of the unique detectors determine an error condition, an error interrupt (or error report in the case of certain peripherals) is registered in the processor. The most severe of these will result in automatic hardware sequences that switch the activity of the processors (hard switch). Less severe errors result in microinterrupts that enter microcode and software charged with recovery of the system.

The microcode and recovery software provides a layered approach to the recovery architecture. Figure 2 illustrates this architecture with microcode providing low-level access to the hardware while the recovery software provides the high-level control mechanisms and decision making. This technique has resulted in several hardware design modifications requiring minimal change to the recovery software.

Figure 3 illustrates the principal architecture and features provided by the recovery software. The bootstrap and initialization routines provide a fundamental set of microcode and software algorithms to control the processor initialization and recovery. These actions are stimulated by a Maintenance Restart Function (MRF), which repre-

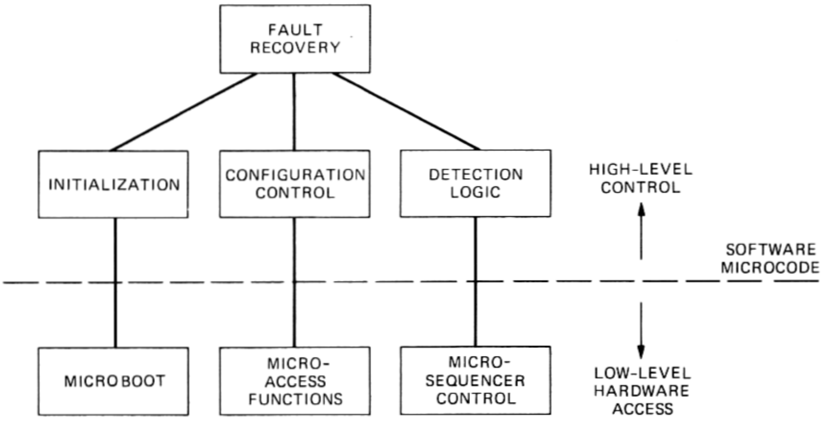


Fig. 2—Maintenance software structure.

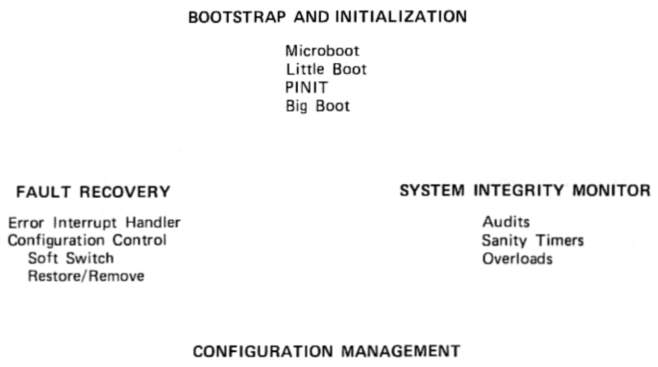


Fig. 3—Maintenance architecture.

sents the highest priority microinterrupt in the system. An MRF sequence can be stimulated from either hardware- or software-recovery sources.

The fault-recovery and system-integrity packages control fault detection and recovery for hardware and software, respectively. The Error Interrupt Handler (EIH) is the principal hardware fault-recovery controller. It receives all hardware interrupts and controls the recovery sequences that follow. The configuration-management program (CONFIG) determines if this particular error is exceeding predetermined frequency thresholds. If a threshold is exceeded, CONFIG requests a change in the configuration of the processor to a healthy state. Thus, CONFIG serves as an error-rate analysis package[10] in the 3B20D maintenance architecture for both hardware and software errors.

3.2 Software integrity architecture

Software fault recovery is very similar in architecture to hardware fault recovery. Each major unit of software is expected to have associated with it error-detection mechanisms (defensive checks and audits), error thresholds (provided by the system-integrity monitor and CONFIG), and error-recovery mechanisms (failure returns, data correcting, audits, and initialization techniques). In addition, both SIM (System Integrity Monitor) and EIH oversee the proper execution of the process. SIM ensures that a process does not put itself into an infinite execution loop or excessively consume some system resource (e.g., message buffers). EIH, through the use of hardware and microcode detectors, ensures that processes do not try to access memory outside of defined limits or execute instructions that are not permitted to the process. Each process has initialization and recovery controls (analogous to hardware) so that a recovery can be effected. Figure 4 illustrates the software-recovery architecture.

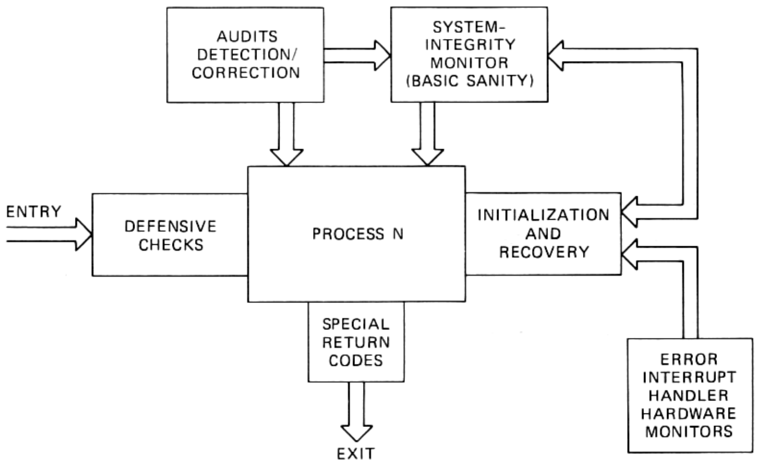

AUDITS DETECTION/ CORRECTION
SYSTEM-INTEGRITY MONITOR (BASIC SANITY)
ENTRY
DEFENSIVE CHECKS
PROCESS N
INITIALIZATION AND RECOVERY
SPECIAL RETURN CODES
EXIT
ERROR INTERRUPT HANDLER HARDWARE MONITORS

Fig. 4—Software fault-recovery architecture.

If recovery actions result in the removal of hardware units from the system, diagnostics[9] are dispatched automatically to analyze the specific problem. Audits represent the software counterpart for diagnostics with the exception that the routine interval is much shorter and correction is possible in the case of audits.

IV. FAULT RECOVERY

In this section, we describe the fault-recovery strategies associated with the 3B20D Processor. In particular, we describe the fault recovery and initialization strategy along with the microcode assists required to carry out these functions. We also describe the manual control capabilities provided by the processor and software. These control mechanisms are termed emergency mode. Finally, we describe some of the software audit and integrity techniques in the DMERT operating system.

4.1 Fault recovery and system initialization

Fault-recovery strategies are based on the fault-tolerant architecture of 3B20D. Major hardware units are fully duplicated. This duplication provides a high probability that a combination of operational units can be retained in the face of faults. The mate processors are only loosely coupled; interprocessor connections are limited to the maintenance channel and memory-update circuitry. This architecture forms the foundation of the hardware-recovery strategy employed in the 3B20D, namely to isolate an entire faulty processor as opposed to attempting fault resolution at the subunit level.

DMERT is a modular operating system that provides a wide range of protection from various types of classical errors. Examples include write-protected memory areas, memory ranges that can be used only for text execution, and protected virtual address spaces. Thus much of the recovery from these types of errors is built into DMERT directly. Those overt recovery actions that are required are greatly simplified by the underlying architecture. Hard faults and other conditions requiring recovery actions are treated according to their severity. Fault categories that will be described individually are hard faults, thresholded faults, configuration faults, sanity time-outs, and software-requested recovery actions.

The 3B20D has built in self-checking circuitry designed to detect hard faults as soon as they occur. This circuitry simplifies recovery since early fault detection limits the possible damage done by the fault. Faults in this category indicate that the processor is no longer capable of proper operation and results in an immediate stop of the currently running processor and a switch to the standby processor. Since the standby processor does not match the active processor instruction by instruction, an initialization sequence is required to start execution properly.

Some types of faults and errors are not severe enough to justify an immediate stop and switch recovery action. Examples of errors of this kind are hardware faults detected in the standby processor memory and software errors such as write-protection violations. Another type of error in this category is hardware faults that are handled by self-correcting circuitry. Although most faults are detected by self checking, some units, such as main memories, have fault rates that justify self-correcting capabilities. Disks also are self correcting through the use of cyclic redundancy codes. All errors in this class are reported to the recovery system as error interrupts.

Recovery software classifies the interrupt by type, gathers and saves all available information about the interrupt, and reports the error to the system configuration-management package. If a particular software process is suspect as the cause of the interrupt, such as in a software-triggered event, the process that was running at the time of the interrupt is faulted and entered at its fault entry after a stable system configuration is guaranteed. The fault entry of a process contains recovery and initialization sequences that are special to the process involved.

All error interrupts are reported to configuration management. Errors are logged against the failing unit and error rates are compared to allowed error thresholds. If the affected threshold is exceeded, further action is required and is based on several factors. If the faulty unit is essential to the system and a mate unit is available, the faulty unit will be removed from service and scheduled for diagnostic testing.

If there is no available mate unit, the faulty unit will be initialized and returned to service since in the case of essential units it is better to have a faulty unit than no unit. Nonessential units are removed and scheduled for diagnostic testing whenever their error thresholds are exceeded.

Each processor has a sanity timer that will result in an initialization if it times out. The active processor maintains both its own and the standby sanity timer so that if the active processor is completely dead, an initialization of the standby processor will be triggered by a sanity timer time-out.

The system provides an Operating System Trap (OST) for use by software to request an initialization. This capability is used by critical system processes when they encounter errors that preclude performance of a critical system function. Initializations occur when an error or fault has been detected that cannot be recovered from without a change in hardware and/or software status. A stop and switch to the other processor may or may not be associated with any given initialization. All initializations include actions of varying severity depending on what is required to deal with various faults and errors.

The first event in the initialization sequence is a hard-wired transfer to a fixed location in the CU microstore where microcode makes a decision as to whether to bring this processor on-line or to switch to the other processor. If the current initialization is of level two or higher, the appropriate processes and data bases are loaded from disk. All available data about the initialization trigger is saved and a decision is made to bring this processor on-line or stop for the off-line initialization.

The DMERT kernel initialization or bootstrap routine is then called to restart system processes or to fault active processes as appropriate. The initialization is now complete and the system has returned to normal operation. If an initialization does not recover the system to an operational state, another and more severe initialization will be triggered automatically. Whether to escalate or not is controlled by the initialization interval. Any initialization that occurs during a window of time following the previous initialization will escalate to the next higher level. The length of the initialization interval is a system generation parameter that is established by the application. In addition to the DMERT initialization levels, provision is made for an application to specify between one and sixteen levels for each DMERT level. For example, if the application specifies two levels for DMERT level one, the normal execution of initialization levels would be (1,1), (1,2), (2,1), · · ·, where the first number indicates the DMERT level and the second number is the associated application level.

Data about various recovery actions taken by the system are supplied to provide all possible information about what went wrong and to provide data that can be used by maintenance personnel to assist them in isolating difficult faults. Recovery data are provided in several forms. Each error interrupt is accompanied by a printout containing available information about the state of the processor when the interrupt occurred. A more difficult problem is presented by initializations. Since they are more severe than interrupts and in fact represent a discontinuity in processing, gathering and preserving error data is more difficult. Initializations, as well as interrupts, can occur at a rate much too fast for data to be printed. The solution is to save all pertinent data in a protected area of memory for printing after the system has recovered.

Various kinds of error data are not generally printed as a part of the standard system output but instead are saved in error files on the system disks. Examples of this kind of data are device-driver errors and failing-memory data. One of the more useful pieces of data output by the system are Processor Recovery Messages (PRM). These are low-level one-line messages that are printed in real time. The PRMs thus represent progress marks through the recovery sequences and are extremely useful in those cases where stability cannot be achieved or postmortem data cannot be gathered.

4.2 Special microcode for recovery

A large fraction of the total amount of CU microcode is provided to aid in recovery. The bulk of this recovery microcode is in PROM because most functions are required in spite of the power history of the CU or its boot devices. Functions that are required even if the CU is not ready to execute its instruction set include microinterrupt processing, maintenance channel assists so that one processor can access the other processor and microcode to initialize hardware subsystems. Additional recovery microcode that resides in writable microstore (WMS), extends the processor's instruction set to provide convenient diagnostic and recovery software access instructions. When diagnostic performance requirements do not justify a special instruction, a microstore scratch area is available that can be loaded with arbitrary microsequences that can then be executed for special tests or functions. Before software can run, the WMS must have been loaded from disk. This happens initially as part of the processing of the MRF microinterrupt.

4.2.1 MRF and microboot

When a maintenance restart interrupt occurs, a long sequence of microsteps begins to establish system sanity. Both processors may be in their MRF sequence at the same time and each one may try to become the active processor. The MRF code first makes decisions on whether to do an off-line initialization or an on-line initialization. If a processor determines that it has just powered up, it clears main store and does an off-line initialization unless forced on-line.

A number of tests are made on data in the system status register, SSR, to select one of four possible actions: microboot, tapeboot, processor initialization, or stop and switch. The simplest actions are to begin execution of a processor initialization program called PINIT or to stop and switch to the other processor. This is accomplished by sending a switch command over the maintenance channel to the other processor.

Tapeboot is a complex sequence of microcode that is only done when requested manually via the craft interface. Its function is to create a new system disk from tape. Using the tape device and disk device selected by the craft interface it initializes those I/O units and initializes the WMS from tape. A boot program, called load disk from tape, is read from tape into main store, and memory-management tables are created to allow it to run the hardware complex without the operating system. This program then reads the tape to make a DMERT disk image.

Microboot uses information on the DMERT disk to initialize the writable microstore and read in the first software boot program called little boot. To do this, it must first select the disk drive to use as a boot device. If the craft interface has forced either the primary or secondary boot device, it uses that device. Otherwise, microboot selects a disk drive based on the state of hardware control bits. Alternate boots will use alternate devices. Microcode is read from the disk and then copied to WMS. Finally, little boot is read from the boot partition and given control.

4.2.2 Microaccess assists

Although the MRF sequence is the most complex microcode recovery assist, both diagnostics and recovery software have special microcode. There are six maintenance channel assists in PROM. They are:

Write Main Store
Read Main Store
Write Writable Microstore
Read Microstore
Write Utility Circuit
Read Utility Circuit

In addition there is microcode in WMS to support a set of instructions provided for the diagnostic and recovery software. Diagnostics have instructions to manipulate the maintenance channel and aid in I/O diagnoses. They also share instructions with recovery. These instructions include groups of instructions for:

On-Line Main Store Controller
Off-Line Main Store Controller
Maintenance Store Operations

Finally, both diagnostic and recovery software use privileged instructions (shared with the operating system) to read or write special registers. They also can activate unit initialization sequences that are used in the various parts of the MRF microcode.

4.3 Emergency modes

Emergency mode on the 3B20D refers to the facilities and procedures provided to prevent the system from experiencing a total outage. For example, emergency facilities are applied when the system is unable to recover automatically. The most characteristic of these are: duplex failure of the control unit, duplex failure of the system disks, duplex failure of the essential I/O devices, failure of fault recovery to find a working configuration of hardware, software faults that will not allow the system to operate properly, errors that destroy the integrity of the disks, and software overwrites that introduce catastrophic errors into the software.

Emergency mode capabilities are built into the system to address these mechanisms that can fail the 3B20D as a system. The emergency action interface (EAI) on the 3B20D provides for manual initialization capabilities that can recover the system from several of the conditions mentioned above. This interface allows the craft to select a processor and disk configuration in a case where certain configurations may be leading to the problem. The interface also allows the craft to reconfigure the system to handle maintenance hardware failures. For example, the craft can inhibit error sources and sanity timers through EAI commands, thus allowing recovery from certain maintenance failures even though both processors are affected. The EAI also provides capabilities for craft initializations to deal with loss of subsystem capabilities.

The 3B20D also provides the craft with other manual capabilities through the port switch select, the disk power inverter select, and the unit power switches. These can be used to reconfigure the system to handle certain problems in the system. In rolling bootstrap conditions, the 3B20D outputs diagnostic information through processor recovery messages. This information provides a gross diagnostic capability in the event of a complete system outage.

Tape load boot is an emergency procedure provided for the situation where a system has destroyed its only valid copies on disk of the generic software. Here the site would have a tape copy of the generic and data base, and read the tapes into the disk via the EAI tape load boot facility.

The final backup repair procedure is the dead start diagnostics. Primarily used as an installation tool, the dead start diagnostics allow for the repair of a completely sick processor by using a remote host processor.

The craft interface provides the mechanism through which the status of the system can be determined, the configuration of the system's hardware or software can be changed, and special emergency actions can be taken during catastrophic failures of system component.[8] The emergency action interface (Fig. 5) allows the craft in the field to access the system during times when a major portion of the system is nonfunctional to the point where the normal capabilities of the craft interface cannot be used. The limited capabilities of the emergency action interface include forcing failing hardware off-line or on-line, notification of the status of critical system resources, and forcing a reinitialization of the system.

4.4 Software integrity

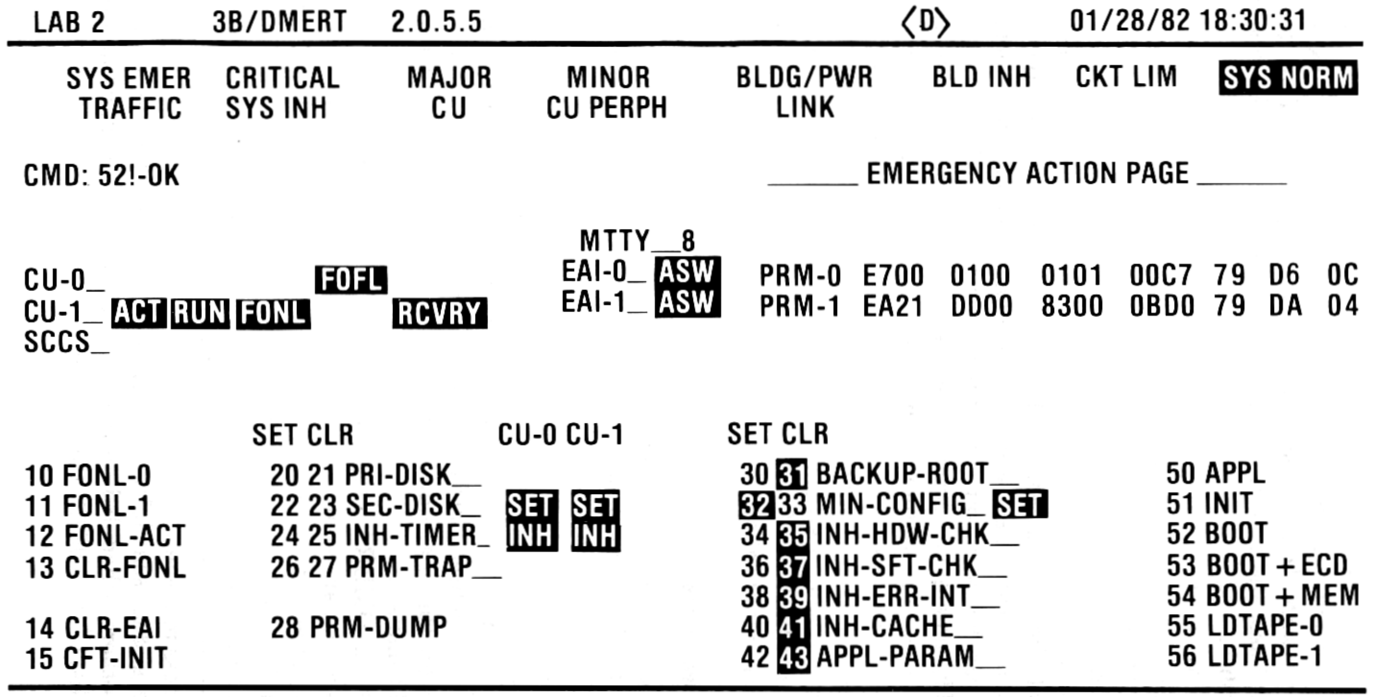
Section III described the architecture of the software integrity system. In this section, we describe some of the specific audit and overload measures that have been included in the DMERT system.

The DMERT audit package verifies the validity of critical data structures. Most audits exist throughout the system within the processes that control the data to be audited. In some cases, several audits are invoked consecutively to form a sequenced mode audit. Most requests for running audits come from an audit control structure, i.e., the audit manager.

Audits in DMERT verify data, not functions. The basic types of auditable data are system resources and stable data. Though most of the auditable data in the operating system resides in the kernel, additional data resides in other critical processes, such as the file manager and device drivers. Smaller amounts of auditable data reside in supervisor processes, such as in the *UNIX** operating system and the process manager.

Some audits, scheduled on a regular basis, are known as routine audits; others, scheduled on request, are known as demand audits. A partial list of the DMERT audits follows:

* Trademark of Bell Laboratories.



```
LAB 2        3B/DMERT    2.0.5.5                               ⟨D⟩        01/28/82 18:30:31

     SYS EMER   CRITICAL    MAJOR     MINOR       BLDG/PWR    BLD INH    CKT LIM    SYS NORM
     TRAFFIC    SYS INH      CU      CU PERPH       LINK

CMD: 52!-OK                                           ______ EMERGENCY ACTION PAGE ______

                                            MTTY__8
CU-0_                       FOFL           EAI-0_ ASW   PRM-0  E700  0100  0101  00C7  79  D6  0C
CU-1_ ACT RUN FONL               RCVRY     EAI-1_ ASW   PRM-1  EA21  DD00  8300  0BD0  79  DA  04
SCCS_

                 SET CLR             CU-0 CU-1       SET CLR
10 FONL-0         20 21 PRI-DISK__                    30 31 BACKUP-ROOT__               50 APPL
11 FONL-1         22 23 SEC-DISK_  SET  SET           32 33 MIN-CONFIG_ SET             51 INIT
12 FONL-ACT       24 25 INH-TIMER_ INH  INH           34 35 INH-HDW-CHK__               52 BOOT
13 CLR-FONL       26 27 PRM-TRAP__                    36 37 INH-SFT-CHK__               53 BOOT + ECD
                                                      38 39 INH-ERR-INT__               54 BOOT + MEM
14 CLR-EAI        28 PRM-DUMP                         40 41 INH-CACHE__                 55 LDTAPE-0
15 CFT-INIT                                           42 43 APPL-PARAM__                56 LDTAPE-1
```

Fig. 5—Emergency action interface page.

(*i*) Message buffers—This audit finds and frees lost message buffers, i.e., messages that have been on a process's queue for extended periods of time.

(*ii*) Scheduler—This audit checks for linkage errors in the scheduler's ready and not-ready lists.

(*iii*) Memory manager—This audit recovers lost swap space and corrects any overlap of swap space.

(*iv*) File manager—This audit checks all internal file manager structures: task blocks, buffers, mount table, etc. The audit corrects the information and has the ability to back out an aborted task and free its resources.

(*v*) File system—This audit is demanded by the file manager whenever a file system is mounted read/write. It checks and corrects the file system's super block free list, and free-block bit map. This audit verifies the integrity of the mounted file systems concurrent with their use.

V. MAINTAINABILITY

The maintainability of the 3B20D system is the second vital component that guarantees the overall high reliability required of the system. There are conditions where automatic recovery is unable to restore the system to a fully functioning state. This is where maintainability is critical to satisfying DMERT's high-reliability requirements. The basic premise of maintainability is to provide basic data-gathering and data-analysis mechanisms as well as the ability to act on the results of that analysis. These mechanisms must be able to collect and analyze diagnostic and debugging information from various hardware and software components within the system in order to isolate the error. These mechanisms must then allow the craft to control and modify the configuration of the system based on the diagnostic and debugging information collected. Furthermore, these mechanisms must yield their information as quickly as possible while disturbing the rest of the system as little as possible.

Maintainability comprises such areas as diagnostics, transient-error analysis, routine maintenance procedures, field utilities, and plant measurements. Once the error has been isolated and analyzed, the problem must then be corrected as quickly and benignly as possible. This procedure is termed updatability, and it includes such aspects as growth and retrofit for hardware, emergency fixes, function update, and system update for software. Maintainability is quite naturally partitioned into diagnostics (hardware) and the various field utilities (software).[11] However, central to the ability of the craft to maintain and control the 3B20D hardware and software is the ability to interface to the various maintenance facilities provided within the system. This is one of the very important capabilities of the craft-interface system. The craft interface provides the craft and others with the means to request diagnostics, receive error-analysis reports, initiate emergency-recovery procedures, gather plant-measurements data, and exercise routine maintenance programs. In addition, the craft-interface system allows configuration control by providing access to growth and retrofit procedures, system- and function-update capabilities, emergency-fix facilities, and the various field utilities. This section discusses the capabilities of the subsystems, which provide basic maintainability of the DMERT system. Diagnostics are discussed in Ref. 9.

One component of the maintainability required of DMERT-based systems is the ability of these systems to accept hardware and software changes in a way that does not interfere with their primary tasks. In other words, a DMERT-based system must be able to accept changes without disturbing call processing, networking, or other critical functions. DMERT supports this through several aspects of updatability. The first is growth; the ability to add or remove hardware and related software components to the running system. Growth extends from physically connecting new equipment—such as memory boards—through informing the system of its existence, exercising it, logically connecting it into the system's configuration, and committing its use in the system. Other subsystems—such as a hardware and software fault recovery and diagnostics subsystem—then take over to ensure that the new system component continues to be sane and usable.

[illegible]

The second aspect of updatability is retrofit: the ability to replace hardware components in the system with similar components of a different vintage or with different capabilities or interface characteristics. Retrofit procedures may "de-grow" or remove old units and then grow or add new ones. They also may add the new units first and then perform a transition from the old units to the new. Thus, retrofit of units may involve extensive periods of time where old and new units coexist in the system. Retrofit may also involve substantial software changes to interact with new units and to recognize the existence of both old and new units.

The third component of updatability, field update,[12] deals exclusively with software and data file changes in DMERT. Such changes are done logically, on a file-by-file or functional level. Just as with growth and retrofit, field update can install or replace system programs or files, inform the system about them, logically connect them into the system, exercise them in that state, and then commit to or back out of them. Field update is intended primarily for installing fixes or small features that do not perturb the system's architecture.

The fourth updatability component, system update, allows program and data changes of much greater magnitude, up to complete software system replacement. A bootstrap is required to install the changes for any system update. By using disk redundancy or backup copies of sections of DMERT's disks, system update can prepare a new, partial, or total version of the system on disk and then switch to it (and back, if necessary). Where field update performs a logical change of files, system update does a physical change of a set of partitions (file systems and/or file partitions).

VI. SUMMARY

This article has described the basic architecture of the fault-recovery and system-integrity subsystem for the 3B20D Processor. These subsystems are tied into the maintainability aspects of the processor. All of the features provided are responses to the reliability objective of no more than two minutes downtime in each year of service. The features and architecture continue in the tradition of former high-availability processors.

VII. ACKNOWLEDGMENTS

The authors thank D. G. Gilbert, G. T. Surratt, B. G. Niedfeldt, and D. J. Fitch for their assistance with various sections of this article.

REFERENCES

1. R. C. Hansen, "System Reliability Strategies," Proc. Nat. Elec. Conf., *35* (1981), pp. 40–51.
2. P. D. Carestia and F. S. Hudson, "No. 4 ESS: Evolution of the Software Structure," B.S.T.J., *6,* No. 6 (July 1981), pp. 1167–1201.
3. R. W. Downing, J. S. Nowak, and L. S. Tuomenoksa, "No. 1 ESS Maintenance Plan," B.S.T.J., *43,* No. 5 (September 1964), pp. 1961–2019.
4. P. W. Bowman et al., "1A Processor: Maintenance Software," B.S.T.J., *56,* No. 2 (February 1977), pp. 255–87.
5. T. F. Storey, "Design of a Microprogram Control for a Processor in an Electronic Switching System," B.S.T.J., *55,* No. 2, (February 1976), pp. 183–232.
6. 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.
7. Jonas Butvila, "Reliability and Its Impact on System Design," Proc. Nat. Elec. Conf., *35* (1981), pp. 43–7.
8. M. E. Barton and D. A. Schmitt, "The 3B20D Processor & DMERT Operating System: Craft Interface," B.S.T.J., this issue.
9. J. L. Quinn and F. M. Goetz, "The 3B20D Processor & DMERT Operating System: Diagnostic Tests and Control Software," B.S.T.J., this issue.
10. M. M. Meyers, W. A. Routt, and K. W. Yoder, "No. 4 ESS Maintenance Software," B.S.T.J., *56,* No. 7 (September 1977), pp. 1139–67.
11. G. P. Eldredge, and J. G. Chevalier, "The 3B20D Processor & DMERT Operating System: Field Utilities," B.S.T.J., this issue.
12. R. H. Yacobellis, J. H. Miller, and B. G. Niedfeldt, "The 3B20D Processor & DMERT Operating System: Field Administration Subsystems," B.S.T.J., this issue.