MAINTENANCE AND FAULT RECOVERY SOFTWARE SUBSYSTEM DESCRIPTION AT&T 3B20D COMPUTER

CONTENTS PAGE

CONTENTS PAGE

1. GENERAL

1.01 This document provides general information relative to the maintenance and fault recovery procedures provided by the UNIX® RTR operating system.

1.02 This document is reissued to include information about the Four Bit Essential Field Feature and the Disk Limp Mode Enhancement Feature which applies to the UNIX RTR operating system (release 1) and to update areas of the document where changes and improvements have been made to the system. Revision arrows are used to emphasize

the significant changes. The specific reasons for reissue are listed below:

(a) Add paragraph 4.41 to incorporate information on the Four Bit Essential Field Feature.

(b) Add paragraph 7.07 to incorporate information on the Disk Limp Mode Enhancement Feature.

The Equipment Test List is not affected.

2. MAINTENANCE STRUCTURE

2.01 The maintenance structure as described herein is influenced heavily by three different sources:

(a) The self-checking maintenance philosophy of the computer

(b) The structure of the host operating system

(c) Design experience with other ***computer*** maintenance systems.

The UNIX RTR maintenance packages provide a reliable and flexible base for applications to build upon.

2.02 Figure 1 illustrates the major centers of maintenance activity in the system and how they relate to the 3-layer structure of UNIX RTR operating system.

2.03 Nondeferrable maintenance is associated with the kernel. The functions associated with nondeferrable maintenance are as follows:

(a) ***System Initialization:*** System initialization includes computer initialization and bootstrapping. Its function is to recover normal system operation after a fault has been detected.

(b) ***Equipment Configuration Data Base Manager:*** The equipment configuration data base (ECD) contains the current status of all computer and peripheral hardware known to the UNIX RTR operating system. The equipment configuration data base manager (ECDMAN) supplies access to this data base and other processes.

(c) ***System Control Process:*** The system control process provides basic maintenance related hardware access to both control units (CUs) in a system. It handles error interrupts and provides diagnostic access to the off-line CU.

(d) ***Configuration Management Library:*** The configuration management makes reconfiguration decisions based on the current status of the hardware found in the ECD and fault conditions reported by hardware drivers. One such decision might be to remove a faulty peripheral device from service.

2.04 Deferrable maintenance is executed by the UNIX RTR operating system. The processes that fall in this category are listed below:

(a) ***Maintenance Input Request Administrator (MIRA):*** The MIRA coordinates requests for maintenance actions (most typically diagnostics) that have been manually or automatically requested.

(b) ***Diagnostic Monitor (DIAMON):*** The DIAMON loads and executes a requested diagnostic.

(c) ***Trouble-Locating Procedures (TLPs):*** The TLPs translate the results of a diagnostic to a replacement pack list for the faulty unit.

(d) ***Coordinator Spooler Output Process (CSOP):*** The CSOP coordinates output directed to the maintenance teletypewriters (TTYs).

2.05 Maintenance functions appear at all levels of the operating system. Also, there is a close relationship between the structure of the UNIX RTR operating system and the structure of the maintenance functions themselves. That is, maintenance is not a stand-alone package. Some of the reasons for this structural integration follow:

(a) Maintenance functions tend to fall into different categories depending on the desired real-time response. For example, switching from one CU to another is to be done in a minimum amount of time and is therefore accomplished by a kernel process running at the highest hardware priority level. Diagnostics are a deferrable function that can be executed under the UNIX RTR operating system.

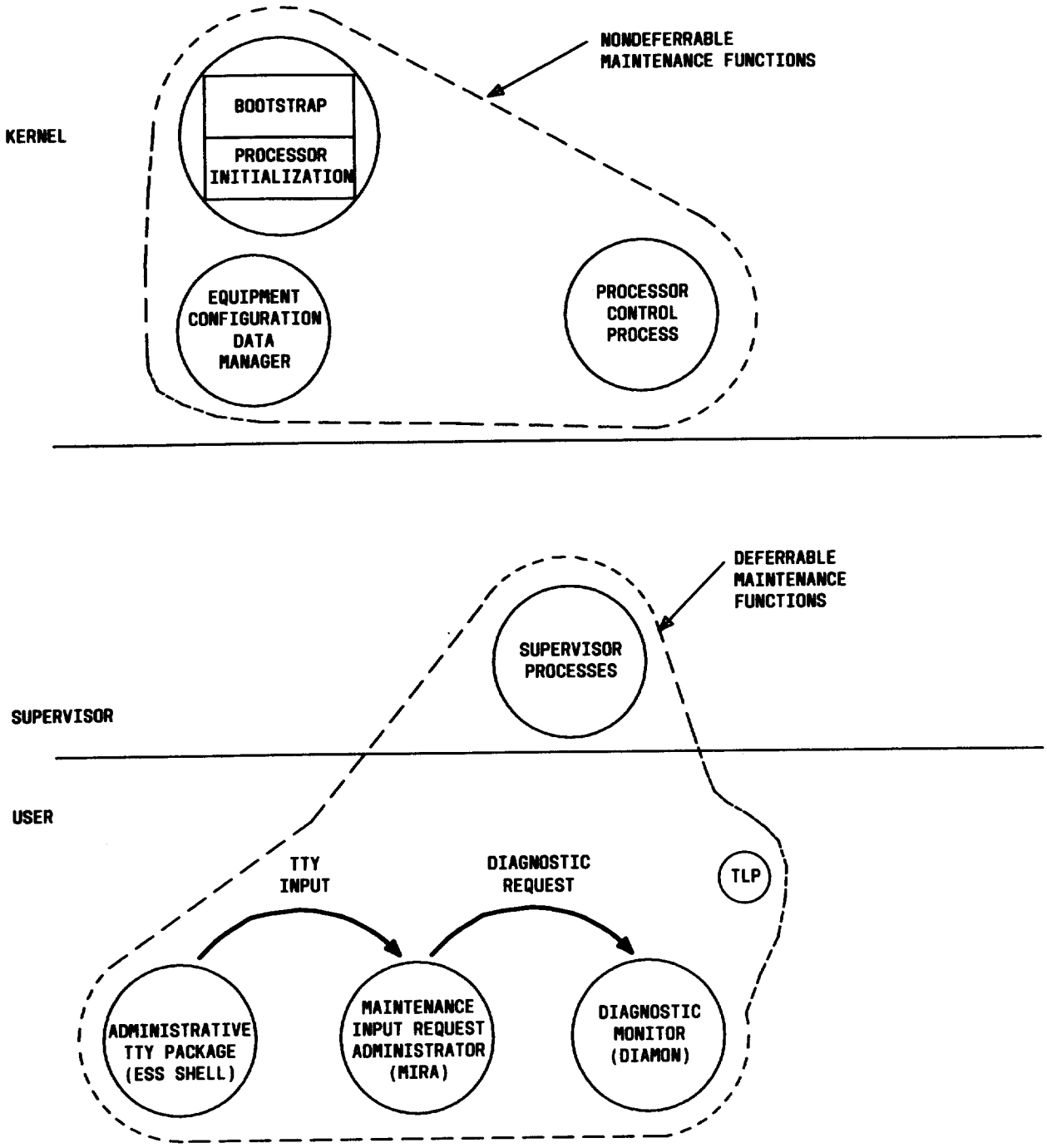

Fig. 1—UNIX RTR Operating System Maintenance Structure

(b) Some maintenance functions (such as ***computer initialization)*** do not assume that the operating system is running.

(c) Some maintenance functions are not self-sufficient but require the services provided by the operating system. In general, the level of service provided by the system improves in direct relation to the separation from the kernel. Therefore, it is reasonable to implement non-real-time critical functions under the UNIX RTR operating system.

2.06 ***Maintenance Objectives:*** Some of the maintenance objectives listed here apply to the 3B20D computer system as a whole — including hardware and software — while others apply specifically to the UNIX RTR operating system software maintenance package. The maintenance objectives are as follows:

(a) The expected amount of accumulated computer downtime should not exceed 2 hours per 40-year period for a fully redundant system configuration. This objective will be met with a balanced approach in the following areas:

(1) Duplication and self-checked computer hardware using high reliability components

(2) Adopting procedures which limit the amount of time that a duplex configuration must operate with one CU out of service

(3) Effectively handling memory faults through the use of hamming error detection and correction

(4) Extensive auditing of data

(5) Providing a reliable and rapid bootstrap from disk facility

(6) Providing limp mode capabilities to keep the system up when difficult error conditions are encountered

(7) Providing dead-start capabilities and emergency action capabilities for recovery from extraordinary problems.

(b) Although some of the features provided by UNIX RTR operating system are internally complex, application control interface for these features is very straightforward. System primitives are provided as necessary to control the maintenance feature package at high level.

(c) Whenever feasible, high-level decisions will be left in the hands of the application.

(d) In response to the trend towards remote maintenance operation, a common Switching Control Center System interface is provided with the capability to remote all control and display functions.

3. DEFERRABLE MAINTENANCE

3.01 Deferrable maintenance functions consist of the computer and peripheral diagnostic and diagnostic-related functions. Since diagnostics tend to be background-type tasks, the UNIX RTR operating system provides a suitable environment for the diagnostic control structure. The diagnostics are written in UNIX RTR operating system user processes. Diagnostics inherently need access to critical parts of the computer in order to thoroughly exercise the circuitry. Being at the user level in the operating system constrains the diagnostic access. Therefore, the diagnostics communicate with a kernel-level process and the computer control process maintenance driver to gain the necessary access. A diagnostic process under the UNIX RTR operating system consists of three elements. The DIAMON controls the execution of the diagnostics and serves as an interface between the diagnostic and the rest of the system. The 2-element diagnostic consists of a control program which interfaces with DIAMON and the diagnostic phase which contains the actual tests to be performed. Each phase is comprised of a collection of functionally related tests. The size of each phase is to some degree a function of the file system and thus was chosen so as to allow the diagnostic to be a sufficiently optimum structure under the file system.

3.02 Deferrable maintenance functions are executed under the UNIX RTR operating system. Most deferrable maintenance functions are related in one way or another to the human interface of the system. Some of the functions performed by the deferrable maintenance have to do with TTY message processing, fault analysis, and manipulation of diagnostic files. The processes that fall into this category are listed below.

(a) ***Electronic Switching System Shell:*** The Electronic Switching System shell handles input messages in the standard format common to all Electronic Switching Systems.

(b) ***Maintenance Input Request Administrator:*** The MIRA coordinates and schedules requests for maintenance actions (most typically diagnostic) that have been manually or automatically requested.

(c) ***Coordinator Spooler Output Process:*** The CSOP coordinates output directed to the maintenance TTY and the switching control center.

(d) ***Diagnostic Monitor:*** The DIAMON loads a requested diagnostic. The diagnostic is typically executed by the diagnostic controller cooperating with a kernel process (such as a device driver) using a diagnostic data table and the diagnostic control block (DCB).

(e) ***Trouble-Locating Procedures:*** The TLPs translate the results of a failing diagnostic to a replacement pack list.

A. Electronic Switching System Shell

3.03 The program documentation standard shell (pdshl) process is created upon system initialization or when the terminal comes on-line. The "pdshl" serves as a "cftshl" command interpreter that receives maintenance requests from the TTY. By incorporating this shell, standard Electronic Switching System TTY formats may be used to invoke the maintenance requests.

B. Maintenance Input Request Administrator

3.04 The MIRA process is created upon system initialization and is a single-image process (i.e., only one instance of MIRA exists at any time). The function of MIRA is to receive maintenance requests from a TTY as well as other processes. Typical requests are to remove or restore a unit as well as to diagnose the unit. Requests entered on a TTY may be interpreted by the system shell. The system shell invokes execution of an appropriate process which then sends a message to MIRA to arrange a particular request. Many processes throughout the system may need to communicate with the MIRA. These processes may or may not be UNIX RTR operating system processes. To allow the MIRA to be, in effect, globally known to all processes, it is attached to a system port. The DIAMON process was made a separate process in order to provide a structure in which a desirable amount of autonomy can exist. For effective scheduling of these tasks, there must be only one MIRA. Automatic requests can also be provided in this structure. An application-provided task could reside at the same level in the operating system and send messages via the port to MIRA at appropriate times to make diagnostic requests. From this point, the request can be the same as a request made from a TTY.

C. Coordinator Spooler Output Process

3.05 The CSOP is complimentary to MIRA in that it formats and displays all maintenance output messages directed to the maintenance TTY. As in MIRA, the CSOP is a single-image process which is globally accessible. Results of diagnostics are sent as a message or via a common file to CSOP whenever output is desired. This data will be properly formatted by CSOP for the TTY output. For restore and remove nondiagnostic-type output messages, DIAMON sends a message to CSOP causing it to output the proper response. Other maintenance output functions such as postmortem dumps are handled similarly. Data is sent as a message as or a common file to CSOP along with a format indicator to identify the format in which the data is sent to be output.

D. Diagnostic Monitor

3.06 Upon creation by MIRA, it is the function of DIAMON to carry out maintenance requests. The DIAMON constructs a DCB which contains all data relevant to the diagnostic; for example, which phase was requested, the unit type, the member number, etc. Also contained in the DCB is various independent data which is extracted from the ECD. The DCB is constructed as a shared segment between DIAMON and the diagnostic. Once the DCB is constructed, DIAMON creates the appropriate diagnostic control (DIAGC) process beginning the actual diagnostic. It is the function of the DIAGC process to open the appropriate diagnostic files and execute the diagnostic. It should be noted that since the diagnostics are executing under an operating system they are subject to being preempted by a higher priority task. Therefore, if they are executing code which cannot correctly perform a test if preempted during the test, they must block interrupts during that time and release the block at the end of the time. The results of

the diagnostic are handled by the control process. At the termination of the diagnostic, the test result is sent via a message to CSOP, the appropriate indicator is set in the DCB, and the process terminates. The DIAMON then sends a message to MIRA to inform that process that the request has been completed. The DIAMON process then terminates. The structure of the diagnostic system can be summarized as follows:

(a) The MIRA process is the receiver and scheduler of maintenance requests.

(b) The CSOP provides the handling and formatting of maintenance output to the TTY.

(c) The DIAMON processes can be created by MIRA to carry out the maintenance requests, including creation of the diagnostic process.

(d) The DIAGC process carries out the actual execution of the diagnostic. The device orders are handled by the device driver [disk or input/output processor] or maintenance driver for the computer.

E. Trouble-Locating Procedure

3.07 The TLP aids in locating a faulty circuit pack. Upon request, the TLP will analyze a diagnostic failure and output a list of suspected circuit packs ordered in such a way that the most probable pack is listed first, etc. Each time a pack is replaced, the appropriate diagnostic is rerun to determine if the replacement corrected the fault.

3.08 If the TLP option is specified on the diagnostic request, appropriate data from the diagnostic test results is saved by the DIAGC. After the diagnostic is executed, the trouble-location process (TLPR) is spawned. The TLPR uses test results to access a data base which associates failing tests with lists of circuit packs (and other components).

3.09 The interface of the TLPR to the diagnostic control structure is flexible enough to allow applications to use their own TLP or other TLPs if desired. A high degree of accuracy is achieved by combining techniques from the TLP of previous systems.

3.10 The major components of the TLP incorporate portions of the diagnostics, processes, and data. These are the fault signature, the tlpcall data table command, the TLPR, and the trouble-location data base (TLDB).

4. NONDEFERRABLE MAINTENANCE

4.01 Maintenance functions that fall into the general category of nondeferrable maintenance have the following common characteristics:

(a) They are closely related to the hardware.

(b) The real-time response is critical.

(c) The program modules must be locked in core because they cannot assume that the operating system is operating properly or even exists.

4.02 The following stimuli cause nondeferrable maintenance to take action:

(a) Recognition of a computer self-checking error of the stop and switch class means that the on-line CU has detected a fatal hardware error within itself that requires a switch to the other CU.

(b) An error interrupt in the on-line CU can either be hardware or software related or from the standby CU (memory errors). The action taken can be of varying degrees of severity.

(c) A peripheral device driver detects a change in the device that compromises its effectiveness and may require a reconfiguration.

(d) A manual request of a maintenance function is requested, such as to switch CUs.

(e) An inconsistency is discovered by an audit program.

4.03 The action taken by nondeferrable maintenance will be system initialization, system reconfiguration, and/or error report generation or status messages. All of the possible inputs to nondeferrable maintenance and resulting responses are summarized in Fig. 2.

A. System Initialization

4.04 A system initialization implies that both the computer hardware and the operating system are to be initialized. A UNIX RTR operating system

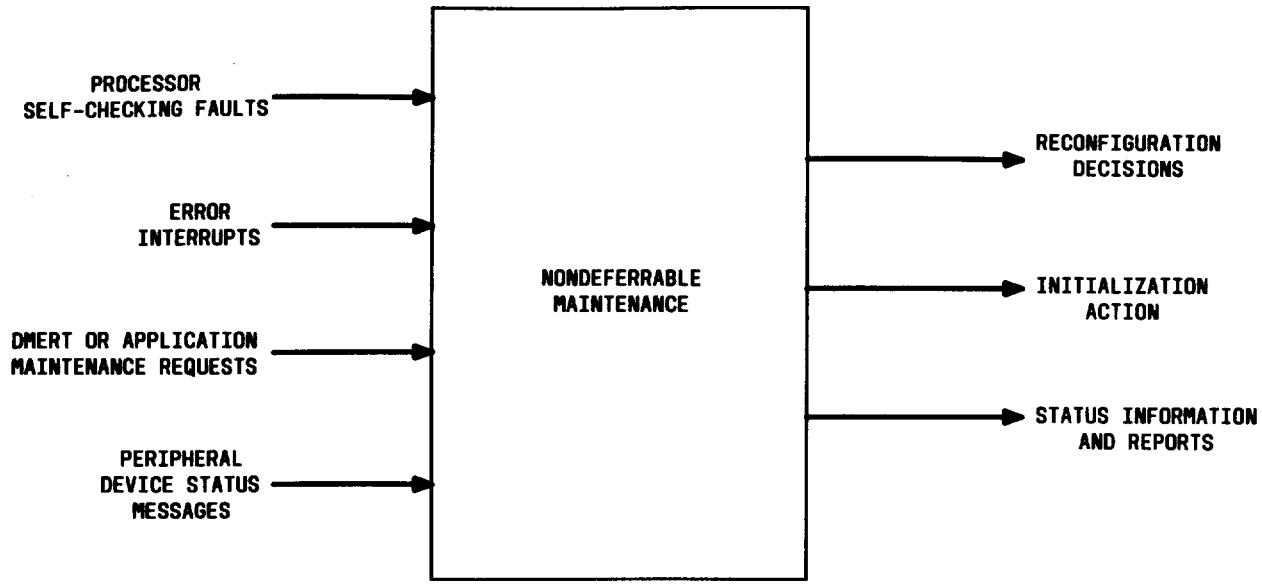

Fig. 2—Nondeferrable Maintenance

application can completely initialize its own processing without causing a system initialization. The UNIX RTR operating system cannot distinguish application initialization (or rollback action) from normal activities. However, when system initialization occurs, the application is impacted.

Initialization Levels

4.05 Four levels (1 through 4) of initialization are defined for UNIX RTR operating system (Table A). At each such level, specific action is taken to recover from a software or hardware fault. The initialization levels are related to a level-of-severity counter named initialization level (INITLVL). For each initialization level (except level 4), the application may define up to 15 levels of its own. The UNIX RTR operating system will not distinguish between application initialization levels other than to administer the level counter. On each initialization, the application level counter will be incremented.

4.06 The need for system initialization arises when a fault occurs of sufficient severity to prevent normal processing from continuing from the point of the fault occurrence. Initialization action can be as severe as a bootstrap or as simple as the initialization (or termination) of a single process. In real-time systems, it is important that the initialization action taken be carefully matched to the severity of the fault to prevent unnecessary loss of service. However, if the initialization action taken on a particular ini-

TABLE A

INITIALIZATION LEVELS

LEVEL	PROCESS ENTRY METHOD	ACTION
0	Application defined	Application defined
1	Fault	Minimal
2	Event fault	Bootstrap
3	Event fault	Bootstrap and reload ECD
4	Event fault (manual)	Bootstrap, reload ECD, and clear memory

tialization is not successful and another initialization occurs immediately, then the level of initialization action must be escalated. This type of progressive initialization philosophy has been implemented for UNIX RTR operating system.

4.07 Figure 3 represents a general view of initialization in an operating system environment. The operating system coordinates the activities and services requested from a set of processes that execute under the operating system. The operating system has an initialization module to which control is returned when a hardware or software fault is detected. After appropriate initialization action has been taken, control is returned to the operating system for normal processing. The term initialization, as it applies to process, means that the process is put into a known initialization or rollback state. Achieving the initialization state is generally a cooperative effort between the operating system and the process being initialized.

4.08 Assuming a progressive initialization philosophy in a real-time environment, several general points concerning system initialization can be made.

(a) Certain types of errors require more drastic recovery actions than others. However, if there is no differentiation between error types in the hardware, there is no choice other than to take the more drastic action.

(b) By its very nature, the progressive initialization philosophy requires that certain critical parameters concerning the nature of the initializations (such as how many initializations have occurred in the immediate past) be readily available. These processes can then select the appropriate initialization state.

(c) In a real-time environment, the set of processes running under the operating system can

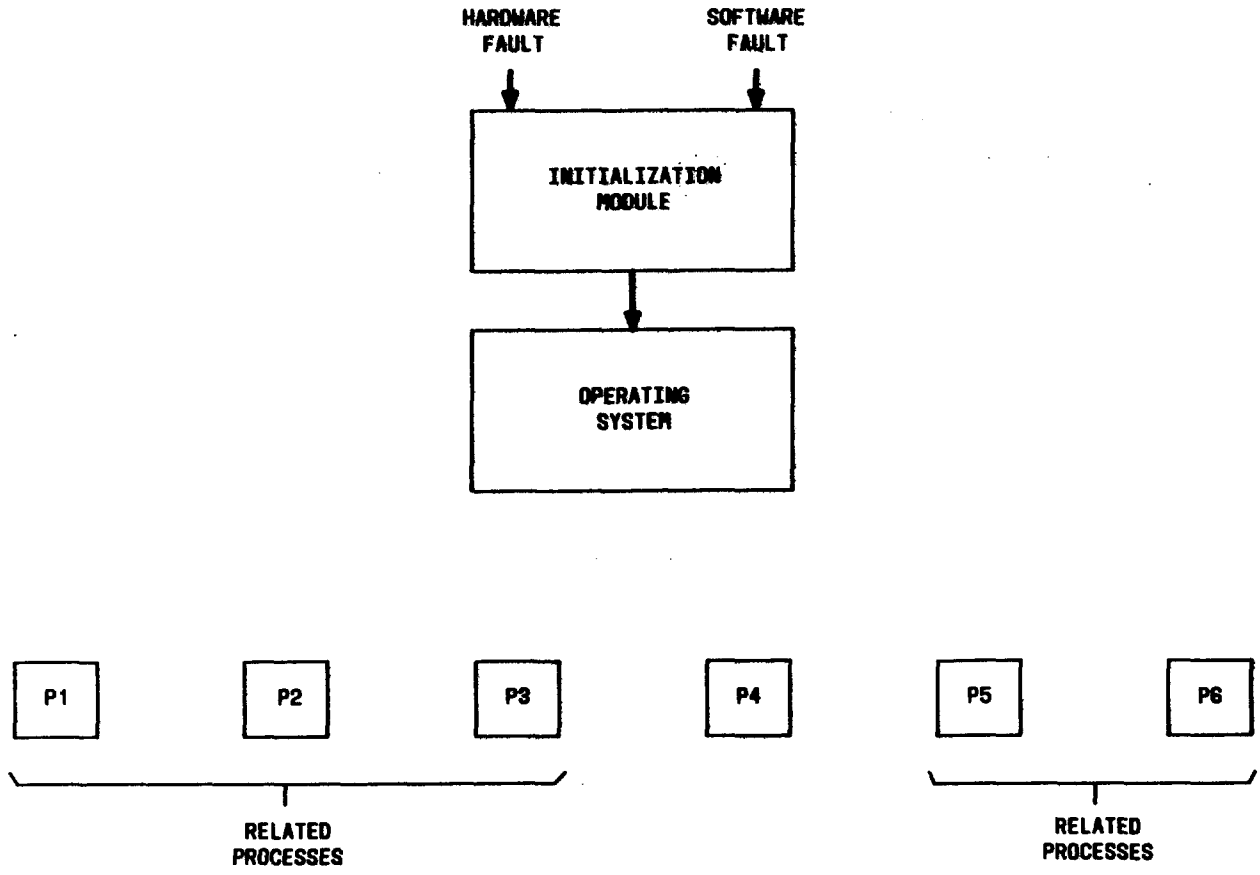

Fig. 3—Operating System Initialization

usually be grouped in closely coupled subsets. If one of the processes in a particular subset must be initialized, then the whole subset must be initialized.

4.09 The computer hardware and UNIX RTR operating system initialization software accounts for all of the above considerations. Figure 4 illustrates the flow of initialization as it is implemented for the UNIX RTR operating system. A hardware fault detected by a self-checking circuit in a computer normally leads to a stop and switch message being sent to the off-line CU that causes it to initialize. The internal signal generated in the off-line CU is referred to as a maintenance reset function (MRF). The initialization of the off-line CU proceeds as follows:

(a) A microcoded sequence initializes critical internal hardware registers. Depending on the state of the system, the outcome of the microcoded initialization normally results in a transfer to main store (MAS) code. (See A in Fig. 4.)

(b) The microcoded initialization is followed by additional hardware initialization (implemented in MAS code). This includes initialization of input/output channels, disabling of the off-line CU (in case it is still active), and reenabling of error check circuits.

(c) The operating system is initialized. The action taken will depend upon the state of the system upon entry into the initialization.

(d) Some or all of the application processes are initialized. Actual creation of the initialized processes may be required. The processes involved are informed of the initialization via a fault entry.

(e) Process initialization is followed by a post recovery phase during which various cleanup operations are accomplished, such as signaling to diagnose the off-line CU (faulty) circuit.

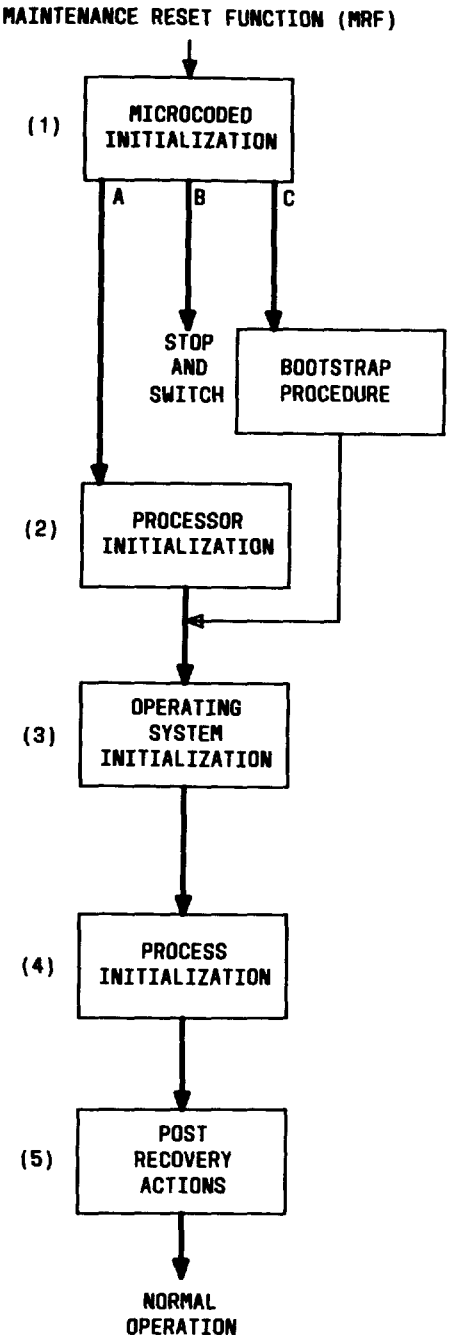

Fig. 4—UNIX RTR System Initialization

Bootstrap

4.10 The bootstrap function is the last resort autonomous capability of the computer initialization procedure. Earlier actions in the initialization sequence attempt to initialize the hardware in the on-line CU or switch control to the off-line CU. When these actions fail to recover system sanity, the bootstrap function is called to reinitialize the memory. The entire software system is reconstructed by reloading programs into MAS from an auxiliary memory device. The lowest level bootstrap preserves the ECD and the protected application segment. The next level reloads the ECD from the bootstrap device. The protected application segment is

only cleared on power-up boots or, if manually requested, through the emergency action interface (EAI).

4.11 ***Bootstrap Procedure:*** Reconstructing the software system memory is done in stages.

(a) First, a microcoded bootstrap function (micro boot) is called to load a small software bootstrap program and the volume table of contents (VTOC) from the moving head disk (MHD) into memory.

(b) Second, the small bootstrap program (little boot) utilizes the VTOC to find the main bootstrap program on the disk and load it into memory.

(c) Third, the computer initialization program saves status information about the computers for postmortem analysis and decides which CU should be active.

(d) Fourth, the main bootstrap program loads into memory the necessary program and data required to reconstruct the operating system.

(e) Fifth, the kernel initialization program [kernel boot (KBOOT)] restores the operating system and creates all required processes.

4.12 ***Boot Devices:*** More than one auxiliary device is designated as a boot device. Micro boot first attempts to load from the primary boot device if both boot devices are active. If only one is active, it will first attempt to load from the active boot device regardless of whether it is the primary or secondary boot device. If both boot devices are active and it is unable to load from the primary boot device, it will attempt to load from the secondary device. Channel addresses and device numbers are fixed in microcode. The EAI may be used to manually force a specific configuration by setting either the PRI-DISK or SEC-DISK EAI options along with forcing the appropriate CU active. The secondary boot device for one CU is the primary boot device for the other CU. Contention problems (both CUs trying to boot from the same device at the same time) are minimized by delay timing within the micro boot.

4.13 ***Bootstrap Progress Display:*** Visual indication to the craft person of progression through the bootstrap procedure is provided by the EAI. Each major step in the initialization will output at least one CU recovery message indicating either successful completion or the detected failure condition. Separate indications for each CU reveal its stage of recovery and failure encountered in the recovery attempt.

4.14 ***Micro Boot:*** Prior to micro boot being entered, hardware checks and interrupts will be blocked and the address translation buffer (ATB) and cache are disabled. Micro boot is entered, and it loads little boot from a partition on the system disk to MAS. Micro boot is written in microcode and performs the following functions:

(a) It resets the sanity timer. The first and second time-outs are set to occur 1.6 and 3.2 seconds after the timer is reset.

(b) It continues initialization of the computer for bootstrapping.

(c) It determines which device (primary or secondary) to use for the current boot attempt.

(d) It initializes the direct memory access (DMA).

(e) It sets all mask inhibit registers for DMA dual serial channel (DSCH) connection to the primary (or secondary) boot device.

(f) It clears, interrupts, and services requests pending in the DMA DSCH connected to the primary (or secondary) boot device.

(g) It enables the DMA channel that connects to the primary (or secondary) boot devices.

(h) It initializes the duplex dual serial bus selector (DDSBS) connected to boot devices being used.

(i) It initializes the bus interface controller (BIC).

(j) It initializes the peripheral interface controller (PIC).

(k) It initializes the MAS controller (MASC).

(l) It uses the disk boot command to load the VTOC on the disk into a temporary memory location in the MAS.

(m) It executes a delay loop waiting for the disk file controller (DFC) to make data available. If the timing interval times out, an access problem is indicated.

(n) It loads writable microstore using the disk boot command.

(o) It formats the disk boot command to load little boot and sends it to the DFC.

(p) If little boot is successfully loaded, program control is transfered to the starting address of little boot specified in the VTOC entry.

If a failure is detected, an output indication is provided via the EAI which indicates the reason for the failure.

4.15 ***Little Boot:*** Little boot is written in C-language which means it must establish a stand-alone environment since there is no operating system in the computer at this time. Little boot then performs the following functions:

(a) It initializes areas in core so that they will have good parity for use by both little boot and big boot.

(b) It determines which boot device is being used by reading a general-purpose register that was loaded with the boot device by microboot and then initializes boot parameters that both programs will use.

(c) It moves part of the VTOC that was loaded by micro boot into the initialization parameter area of MAS so that it will not be overwritten by big boot when it is loaded.

(d) It searches the VTOC for the big boot entry partition on the disk. Normally, the current version of big boot is loaded, but backup version can be loaded if manually requested (backup root EAI option).

(e) It initializes the page table and expanded device page table in the MAS.

(f) It initializes the direct memory access controller (DMAC) random access memory (RAM) for the boot device and then performs the DMAC setup.

(g) It formats the disk boot command to load the big boot and sends it to the DFC.

(h) It waits for the job to complete.

(i) It determines whether the job was successfully completed or not.

(j) If the job was successfully completed, it moves some of the saved initialization parameter data back to the initialization parameter area.

(k) If the job was not successfully completed, an output indication is provided via the EAI which indicates the reason for the failure.

(l) It passes control to the System Initialization Program.

System Initialization Program

4.16 The System Initialization resides in physical memory at a fixed address. The program is entered only from little boot or MRF microcode. Its function is to collect postmortem data, decide which CU should be active, initialize the core of the CU, and then pass control as quickly as possible to the kernel (or the bootstrap program if a bootstrap is in progress).

4.17 After receiving control, the system initialization performs the following operations:

(a) The registers of the CU in which the system initialization is executing are initialized with good parity data.

(b) The computer registers at the time of the initialization are saved in the postmortem data area in the UNIX RTR operating system maintenance data area. Postmortem data is collected for the running computer. [This may later be overwritten. See (d).]

(c) On entry to the system initialization, both CUs may be initializing simultaneously; however, only one CU will go on-line. The following sequence of decisions is used to select the on-line CU:

(1) Is this CU forced on-line? If so, this CU goes on-line. Being forced on-line is a manual override state.

(2) Is this CU forced off-line? If so, this CU goes off-line.

(3) Was this CU on-line before the initialization? If so, this CU goes on-line.

(4) Is the other CU stopped? If so, this CU goes on-line. If not, this CU checks a FIRSTMRF flag. If the flag is set, it goes on-line. If the flag is not set, the CU sets the flag and goes off-line.

(d) If a CU switch occurred, the postmortem data area is overwritten by data obtained from the CU that was previously running.

(e) The UNIX RTR operating system and application initialization levels are set to reflect the initialization level.

(f) An ATB base register is initialized to point at the kernel segment table. Control is passed to a specific point in the kernel with virtual addressing enabled for the nonbootstrap case. Control is passed to big boot for the bootstrap case.

4.18 ***Big Boot:*** Big boot is also written in C-language, so it too must establish a standalone environment in which to operate. Then big boot performs the following operations:

(a) It initializes a pointer to the DFC status word area.

(b) It initializes a pointer to the boot parameter save area initialized by little boot.

(c) It initializes a page table and an expanded device page table.

(d) It initializes the BIC and the PIC.

(e) It searches the VTOC for the root file to be loaded. Normally, the current generic is loaded, but a backup version can be loaded if it is manually requested.

(1) It uses the VTOC address entry obtained from the previous call that is used for block 0 of the file system.

(2) It loads the boot header.

(3) It waits until either the job is completed or a time-out occurs.

(4) It initializes the initialization parameter area with data from the boot file header.

(5) It adjusts the boot file segment table address entries. The segment table immediately follows the boot header, and the number of entries in it is specified in the boot header.

(6) It verifies the file size that is determined from the boot file i-node, checking with the file size in the boot header.

(7) It starts loading the boot file read, write, and execute (rwx) segments starting after the physical protected application segment (PAS). The boot file is loaded one segment at a time, and the segment table entry specifies the number of pages in that segment.

(8) After a segment is loaded, the page table entries are loaded for that segment with the physical addresses.

(9) For a level 4 bootstrap, which must be initiated manually, the PAS is cleared.

(10) The last rwx segment is the ECD that is part of the boot file. The EDC that is in the root file is a dummy. The VTOC is searched for ECD partition and the ECD is loaded. The boot time date data base (such as the plant measurement data base) is also loaded from the appropriate disk partition specified in the boot header.

(f) The boot file requires a certain number of pages that are not loaded with disk data. The pages that are read/write only must be initialized.

(g) It initializes the rest of MAS so that it has good parity and enables the cache if one exists and if a manual request to run without the cache has not been made.

(h) It loads the last word address for the MAS that is minimally equipped [whether it is in CU 0 or CU 1 for the system initialization].

(i) It provides an output System Recovery Message indication that either big boot detects failure or that it has successfully concluded.

(j) Control is transferred to kboot by a "change state."

If a failure is detected, an output System Recovery Message indication is provided via the EAI which indicates the reason for the failure.

4.19 ***Kernel Boot:*** Big boot transfers control to KBOOT which allocates a data area for the memory manager data and kernel stack and sets up a temporary memory map to reflect the boot image and data. The KBOOT enables interrupts and starts the highest priority process that is in the boot image followed by lower priority processes. Typically, processes are started in the following order:

(a) System control process Error Interrupt Handler

(b) System integrity monitor (SIM)

(c) Disk driver

(d) File manager

(e) Special processes such as the scheduler, the memory manager, or the capability manager

(f) Supervisory processes such as the UNIX RTR operating system boot or the process manager.

If there are process creation entries in the system generation specification file, KBOOT sends messages to the process manager to start these processes.

B. System Control Processes

4.20 The system control process modules of the operating system have responsibility for handling error interrupts, implementing CU reconfiguration, conducting system status audits, and providing maintenance access and control to the off-line CU. The system control process modules are implemented as operating system kernel processes.

(a) The system control process error interrupt handler is responsible for handling error interrupts and reconfiguring the CUs.

(b) The system control process maintenance driver is responsible for maintenance access and control to the off-line CU.

(c) The system control process audit is responsible for monitoring system status in both CUs.

System Error Interrupt Handler

4.21 Error interrupts can be caused by an error condition either in the on-line CU or off-line CU in a duplex configuration. Three types of error interrupts are on-line hardware, off-line hardware, and on-line software. All interrupts are handled by the on-line CU.

4.22 It should be noted that an error interrupt is distinct from the type of interruption caused by either a stop and switch or an MRF. The distinction is that an error interrupt may be graceful in the sense that execution can be resumed at the point of the interrupt after the error source has been cleared. When either a stop and switch or an MRF occurs, a hardware initialization follows with no possibility of returning to the point of the interrupt. The action required when an error interrupt occurs can be anywhere from benign to severe depending on the type of error and the system state at the time of the error.

4.23 When an error interrupt occurs, system control process error interrupt handler will take the following steps:

(a) It will classify the error by examining the error register in the CU or interrupting channel.

(b) It will increment an error counter by calling a configuration management error reporting function. This function will test to determine if the threshold has been exceeded. The error counts are contained in the unit control blocks for the appropriate unit controlled by system control process error interrupt handler [central control (CC), store, DMAC, or channel].

4.24 The actions taken on the error may be one of the following:

(a) Device may be removed if request is received from CONFIG (via a fault message). This would result in a CU switch if the remove request was for the on-line CU.

(b) Error information may be passed to other processes affected by the error.

(c) An initialization may be requested on certain error interrupts associated with the DMAC.

4.25 ***Hard Switch and Soft Switch Function:*** The system control process error interrupt handler will accept messages from CONFIG or any other process to accomplish the following:

(a) To cause an emergency or ***hard*** switch of CUs followed by a system initialization of the newly on-line CU.

(b) To cause a graceful or ***soft*** switch of the CUs. The function of the soft switch is to allow either CU in a duplex configuration to be used without the need for a system initialization.

The system initialization required for a hard switch is disruptive to activity in the system. A soft switch is transparent.

4.26 The following actions are taken during a soft switch:

(a) A soft switch message request is received by system control process error interrupt handler. If the off-line CU is not standby, send a failure response. If the off-line CU is standby, proceed.

(b) Block interrupts and freeze DMA activity.

(c) Enable input/output in the off-line CU over the maintenance channel to avoid missing any interrupts.

(d) Save the state of the CC registers, channel registers, DMAC RAM, and the interrupt stack in memory.

(e) Start the off-line CU over the maintenance channel.

(f) Complete the shutdown of the currently on-line CU. Terminate its activity with a HALT instruction.

(g) Continue execution in the newly on-line CU by restoring the state of the items saved in (d). Channel interrupt registers must be ORed with the value saved in memory to prevent the loss of interrupts that occur while the switch is in progress.

(h) Enable the DMAC and interrupts.

(i) Send success response message to the system integrity monitor.

4.27 ***Handling of Memory Faults:*** The computer has hamming error correction and detection. Multiple bit memory errors will result in a hardware error interrupt. Single bit errors will also result in an error interrupt being generated. The system control process error interrupt handler will then perform the following actions:

(a) It will record failing address and syndrome bits in memory.

(b) It will correct data in memory by reading and rewriting failing location.

(c) It will call configuration management routines to increment error counts.

(d) It will call the recovery message formatter to format the error data in memory and output it to the memory logfile.

If the error threshold in the ECD is exceeded, the configuration management routines will request a switch.

System Maintenance Driver

4.28 Processes in the system need to have special-purpose access to a particular CU for various reasons. Therefore, it is logical to group all of these special-purpose functions within the confines of one process, the system control process maintenance driver. The system control process maintenance driver provides an interface between the off-line CU and higher level programs. The system control process maintenance driver handles maintenance access to the off-line CU over the maintenance channel and the memory update unit. The system control process maintenance driver will deny access to the off-line CU when it is in the standby state.

4.29 The following functions are performed by the system control process maintenance driver:

(a) It executes a series of maintenance channel orders over the maintenance channel to the other CU. The on-line CU has access to the microcontrol unit of the off-line CU via the maintenance channel. A maintenance channel order can cause a single microinstruction to be executed in the off-line CU.

(b) It copies to or from the off-line CU memory. The on-line addressing mode is virtual, and the off-line addressing mode is physical. The most common use for this function is to copy diagnostics to the off-line CU and to copy results back.

(c) It updates a block of physical memory in the off-line CU of a duplex pair by performing a direct memory copy over the memory update unit. This function would be used when restoring the off-line CU to service.

(d) It initiates execution in the off-line CU at a specified physical address. This function is required to initiate a diagnostic that executes in the off-line CU.

(e) Same as (d) except it first blocks all peripheral interrupts and freezes DMAC in the on-line CU. Execution time is limited to 3 milliseconds. At the end of this period, it allows peripheral interrupts and releases DMAC.

(f) It initializes the off-line CU; that is, it puts it in a known state so that an off-line program can execute or so that the computer can be restored to service. This function is required between phases of the CU diagnostics.

4.30 An important point concerning the philosophy embodied in paragraph 4.29 (a) through (f) is that misuse of the maintenance functions can cause complete system insanity. However, it is not intended to include elaborate safeguards or checks in the system control process maintenance driver that could potentially overrule an application maintenance request in this process.

System Control Process Audit

4.31 The system control process audit provides a periodic verification of the CU hardware status. Its main function is to provide a useful CU hardware configuration.

4.32 The system control process audit monitors the duplex configuration. Inconsistent hardware status and unacceptable hardware configurations are corrected where possible. This test should run periodically approximately once every 20 seconds.

4.33 The system control process audit does not audit all hardware in the CU complex. The interface with the EAI hardware is audited by a separate EAI audit. The DMA channel inhibit/enable flags are audited by the drivers via periodic "pio" commands over all DMA channels under their jurisdiction.

4.34 The status information in the system status registers and the ECD are checked for consistency and validity. A maintenance channel failure and a read off-line store failure are considered to be caused by the powered down state of the other CU. The audit does not fail any test because the other CU is powered down if the other CU is not marked ACTIVE or STANDBY. The backup maintenance channel bits in the hardware status register of the on-line CU are checked. The sequence in which these tests are performed is important. Some tests depend upon conclusions of previous tests.

C. Configuration Management

4.35 The Configuration Management System (Fig. 5) is responsible for the nondeferrable portion of configuration management of the computer. Following are the main components of the Configuration Management System:

(a) ***Equipment Configuration Data Base :*** The ECD contains the attributes and current status of computer hardware units.

(b) ***Equipment Configuration Data Base Function:*** The ECD function (ECDFUNC) is a collection of library functions that provide ***kernel*** access to the ECD.

(c) ***Equipment Configuration Data Base Manager:*** The ECDMAN is a collection of library functions that provide ***nonkernel*** access to the ECD.

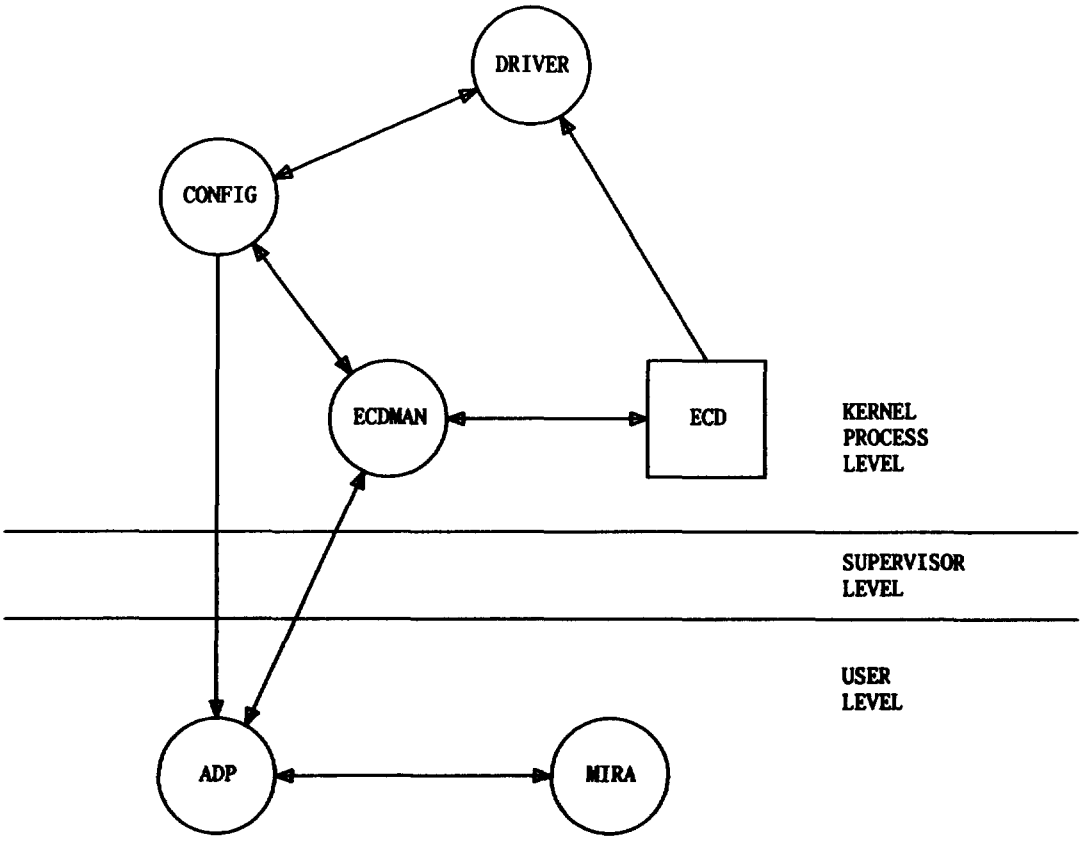

Fig. 5—Configuration Management System

(d) ***Configuration Control:*** The configuration control (CONFIG) is the central software entity responsible for configuration of the system.

The Configuration Management System interfaces to the automatic diagnostic process (ADP) and indirectly to the MIRA process in order to initiate diagnostics.

Configuration Control

4.36 The CONFIG is the central software entity that is responsible for system configuration. Generally, the functions of CONFIG are as follows:

(a) Deciding whether or not to honor a conditional ***remove unit*** request.

♦(b) Deciding whether or not to honor a manual ***remove unit*** request.♦

(c) Maintaining the error counters in the ECD.

(d) Changing the major state of a unit and informing other processes of the changed state. The actual state change is done by the driver responsible for the unit.

4.37 Although CONFIG is responsible for system configuration, it can be overridden by several sources. These sources are as follows:

(a) Hardware initialization and reconfiguration sequencers which cause MRFs and bootstrap when critical faults are detected

(b) Manual intervention from the EAI

(c) Unconditional remove, restore, and switch commands from the terminal or application programs.

4.38 The CONFIG uses information from driver processes and the ECD to determine reconfiguration actions. All conditional requests for removal

(from the terminal or application programs) are sent to the appropriate device driver. The driver then sends the request to CONFIG. The CONFIG accesses the ECD to determine current configuration and status of the unit and any associated backup, redundant, or spare units. If the reconfiguration is allowable, CONFIG sends the device driver a message affirming the action. Whenever CONFIG allows a conditional removal due to errors, a message is sent to ADP to initiate a diagnostic of the removed unit.

4.39 The CONFIG will not allow a conditional removal to occur if the unit is essential and does not have an available replacement. In this case, CONFIG will request a reinitialization of the hardware and continue to use the unit. Also, if several units connected to the same higher unit are failing, CONFIG will determine if the higher unit can be removed.

4.40 The ECD contains error counters for each hardware unit. When a driver discovers an error in one of its units, it reports the error to CONFIG. The CONFIG will record the error in the appropriate error counter in the ECD and will determine if the error rate is excessive. If the error rate is excessive, the unit is deemed faulty and CONFIG will determine if the unit can be removed.

4.41 ▶***Four Bit Essential Field Feature:*** This feature is effective with UNIX RTR operating system (release 1). With this feature, CONFIG will not allow a manual removal request to occur if the unit to be removed is the last unit in service. The automatic removal of units by the REX process and the existing fault recovery processes of the drivers are not affected by this feature.◀

D. System Integrity Monitor

4.42 The System Integrity Monitor (SIM) is responsible for the software integrity of the UNIX RTR operating system. It is also responsible for scheduling and dispatching all audits. Hardware integrity is the responsibility of fault recovery. The SIM performs the following functions:

(a) It ensures the integrity of system bootstrap initializations and generates boot progress system recovery messages.

(b) It administers the 3B20D computer hardware sanity timers and application sanity timers.

(c) It monitors integrity processes.

(d) It monitors overload conditions.

(e) It provides an interface between UNIX RTR operating system and the application integrity monitor (AIM).

System Integrity Monitor Initialization

4.43 The SIM is a kernel boot process, which is initiated by the kernel immediately after the system control processes error interrupt handler is activated. The SIM executes at priority level 13, which is the highest priority process excluding the system control processes error interrupt handler, test utility system (TUS), generic access package (GRASP), and the processes executing in a critical region. These latter processes execute at levels 14 and 15.

UNIX RTR Operating System Interfaces

4.44 The main function of SIM is to receive software integrity exception data (information regarding software faults) from operating system programs and report them to the application and to the craft.

4.45 The UNIX RTR operating system-owned audits, overload detection checks, some boot processes, and other processes detect software integrity faults and report them to SIM.

4.46 ***Interface to UNIX RTR Operating System-Owned Audits:*** All audit information (success or failure) is handled by the SIM. The SIM is responsible for the management of failure data and failure thresholds for all audits. When a failure threshold for a particular audit is exceeded, the SIM detects the failure and makes corrections. Audit failures which can be corrected are recorded in a system error file. The corrective action taken by SIM depends on the process that caused the failure.

4.47 ***Interface to UNIX RTR Operating System Boot Processes:*** The SIM is responsible for verifying the successful creation of the following operating system boot processes:

- File manager
- Disk driver

- Process manager
- Error interrupt handler
- Capability manager
- Scheduler
- Utility manager
- Memory manager.

4.48 When a system initialization starts, the SIM is notified via an event by the kernel. The above boot processes are then expected to send the SIM an initialization status message (i.e., success or failure) within 30 seconds. Also, these boot processes will inform the SIM of any integrity faults encountered during initialization.

4.49 If the SIM does not receive initialization status messages from all boot processes within 30 seconds, the SIM will request another system initialization. No prior notification of this initialization will be given to the applications. Thus, basic sanity can be defined as the reception of all expected successful initialization status messages from UNIX RTR operating system boot processes by the SIM.

4.50 In addition to the boot processes listed in paragraph 4.47, two other processes may send initialization status messages to SIM but are not required to do so. These processes are the IOP driver and the AIM.

4.51 Each initialization status message contains the identity of the reporting process and an EAI output message. The SIM computes the EAI boot step number based on the process identity. The SIM then forwards the EAI output message along with the boot step number to the EAI.

4.52 As each successful initialization status message is received by the SIM, the output message sent by SIM to the EAI is displayed on the maintenance terminal as a success system recovery message.

4.53 If a boot process cannot successfully initialize, a failure initialization status message is sent to notify the SIM of system error. The appropriate information is sent to the EAI to be displayed on the maintenance terminal as a failure system recovery message, and the SIM requests another system initialization.

4.54 The SIM does not determine the scheduling of the boot processes. The application should ensure that the boot processes are not suspended from running by their own processes.

4.55 ***Other Interfaces:*** The SIM also interfaces to portions of the craft interface subsystem for the exchange of integrity information with the applications and craft persons. Since these UNIX RTR operating system interfaces are involved in the application interfaces, they are described in paragraphs 4.57 through 4.61.

Application Interfaces

4.56 The SIM serves as the main interface for the exchange of integrity information between the operating system and all application integrity processes. The SIM will process applications, requests, and instructions. Most messages sent to applications will be handled by the SIM. The SIM interface to the application is via the AIM. The AIM is a kernel process running at the same priority as SIM. In addition, the SIM will handle messages from the maintenance terminal via the craft interface subsystem.

4.57 ***Interface to the Overload Monitor:*** When the SIM receives an overload fault or message from a UNIX RTR operating system process (paragraphs 4.77 through 4.98), the SIM will inform the AIM of the overload condition (paragraphs 4.89 and 4.90).

4.58 ***System Initialization Message (SYSIMSG) Status:*** Both UNIX RTR operating system and application processes may obtain information about the most recent system initialization that has occurred by sending SIM a system initialization status message. The SYSIMSG is a memory resident data structure that is built by various initialization processes during initialization. The SYSIMSG contains the following items:

(a) The initialization source (manual, hardware, or software request)

(b) Initialization request parameters, including the requested UNIX RTR operating system and application initialization levels

(c) The UNIX RTR operating system and application initialization levels that were executed

(d) The configuration options selected on the EAI terminal

(e) The inhibit options selected on the EAI terminal.

On a phase level 0, SIM reads various areas in low core in order to construct this information for use when an SYSIMSG request of this type is received. On phase levels 1 through 4, the SYSIMSG is obtained from the postmortem dump area where it was stored by the system initialization programs.

4.59 Whenever the audit terminates, SIM prints an error message and sends an event to the AIM.

Sanity Timer

4.60 ***Application Sanity Timer:*** The SIM provides a software sanity timer for use by AIM to ensure that application processes are sane and are being scheduled by the operating system. The timer is activated whenever AIM sends SIM a message. Once the application sanity timer has been activated, AIM must send SIM an event at least once in every time-out interval. If SIM fails to receive this event during any time-out interval, it will invoke the PHASE OST. The craft may inhibit the phase by setting INH_SFT_CHK on the EAI terminal when the system is bootstrapped.

4.61 It is the responsibility of the application process to activate the application sanity timer after a system is bootstrapped. The SIM will ensure that it does not time out during a phase 1. The AIM must ensure that it does not time out during a phase 0. The SIM will not deactivate the application sanity timer after either a phase 0 or 1.

4.62 ***Hardware Sanity Timer:*** The SIM is responsible for administering the hardware sanity timer in both the on-line and off-line CUs during normal system operation. During the system initialization sequence (before SIM is created), bootstrap programs and the kernel are responsible for administering the sanity timers.

4.63 The SIM resets the on-line CU sanity timer to time out in 1000 milliseconds and the off-line CU sanity timer to time out in 1600 milliseconds. If a sanity timer times out, a stop and switch signal will be generated by the hardware unless that signal has been disabled manually by selection of the "INH-TIMER" option on the emergency action page. This inhibit only disables the stop and switch; it does not stop the sanity timer from counting. If the stop and switch has not been inhibited, it will result in a system initialization.

4.64 The SIM requests a time-out event every 800 milliseconds to administer the sanity timers. This provides a 200-millisecond "cushion" before the on-line CU sanity timer will time out. This cushion partially defines system insanity; more than 200 milliseconds of continuous activity at or above the SIM execution level (level 13) is deemed to be system insanity and may be detected by the sanity timer timing out.

4.65 The SIM will always administer the on-line CU sanity timer. Administration of the off-line CU sanity timer is complicated by potential interference with diagnostic testing of the off-line CU, by potential maintenance channel failures, and by use of the backup maintenance channel. The following summarizes the administration of the off-line CU sanity timer:

(a) If the sanity timers are inhibited from generating MRFs, the off-line CU sanity timer will not be reset. This provides a way for extraordinary diagnostic testing to be performed with no interference from SIM.

(b) When the sanity timer inhibit is eventually turned off, the off-line CU sanity timer will generate an MRF in the off-line CU because the sanity timer will have timed out. To clean up after this MRF, SIM will send a success system recovery message with step number *C* to extinguish the "recovery" light on the EAI terminal.

(c) When using the maintenance channel to reset the off-line CU sanity timer, SIM will first initialize the maintenance channel hardware and then check for normal returns from all maintenance channel commands to determine if the maintenance channel is working properly. If a maintenance channel failure is indicated, the backup maintenance channel will be used to reset the off-line CU sanity timer.

Creation of Integrity Processes

4.66 The SIM is responsible for the creation and subsequent restart (in case of termination) of critical UNIX RTR operating system processes. These processes include the user level automatic restart process (ULARP). The user process monitor is used to monitor UNIX RTR operating system and application-user processes.

User Level Automatic Restart Process

4.67 The SIM is responsible for software system integrity and the creation of ULARP. The ULARP is a user level process whose functions are to create and monitor critical user processes and to restart automatically any monitored processes which terminate.

4.68 ***Initialization of the ULARP Processes:*** During a bootstrap procedure, SIM will create the UNIX RTR operating system supervision. In the "pcreate" message sent to the process manager, the field will be set to BOOTCHAN. Which will indicate to the UNIX RTR operating system that this is a bootstrap procedure. At this time, the UNIX RTR operating system "init" function will execute the ULARP process. When SIM receives the acknowledgment message on the successful creation of the UNIX RTR operating system, SIM will log the process identification and wait for the start-up event from ULARP. This event will cause SIM to send ULARP a start-up message. This message will contain SIMBOOT in the message text indicating a bootstrap procedure as well as other system information needed for a successful ULARP start-up.

4.69 Once ULARP has completed execution of the run command file, it will send a message to SIM. This message will inform SIM of the status (either success or fail) of the execution of the run command file. That status will be given to ULARP whenever it is necessary to restart ULARP and whenever ULARP is instructed to reread its process name file.

4.70 ***ULARP Termination:*** The SIM will receive a "death-of-child" message on the ULARP process if either of the following conditions exist:

(a) The UNIX RTR operating system "init" sequence fails to create ULARP.

(b) The ULARP dies.

4.71 Under most circumstances, when the "death-of-child" message is received, SIM will assume that ULARP was prematurely terminated and will attempt to restart it. There are, however, two conditions under which ULARP will not be restarted by SIM:

(a) The termination message is received before the start-up event from ULARP indicating a UNIX RTR operating system failure to successfully execute ULARP.

(b) An event is received by SIM prior to the termination message indicating a ULARP failure to initially access the process name file.

If ULARP is not restarted, SIM will display a system recovery message on the maintenance terminal to indicate the reason for the ULARP failure.

4.72 ***Restarting the ULARP Process:*** Following are three conditions under which SIM will attempt to restart the ULARP process:

(a) The ULARP is prematurely terminated, such as during a phase level 1. In this case, SIM will receive a "death-of-child" message and will immediately attempt to restart ULARP.

(b) The PDS shell command "INIT:ULARP!" is entered on the EAI terminal. When this command is entered, the event is sent to SIM. When this event is received, SIM will check the ULARP process identification. If it is running, a message will be sent to ULARP instructing ULARP to read its process name file and attempt to start all monitored processes which are not executing at that time. If ULARP is not running, SIM will attempt to restart it.

(c) The craft initialization selection is entered on the EAI terminal. This selection will cause the EAI handler to send SIM a fault. Upon receiving this fault, SIM will terminate all craft processes. If ULARP is running, SIM will send it a message. If ULARP is not running, SIM will restart ULARP.

For each ULARP restart, SIM will first attempt to create the ULARP process. When SIM receives the start-up event from the new ULARP process, it will

terminate all craft processes and then send ULARP a start-up message. If ULARP is not successfully created, the craft processes are not terminated.

Error Logging and Reporting

4.73 The SIM reports error conditions to the maintenance terminal via the EAI or the output spooler. The SIM error logging strategy uses the craft interface output spooler to record all integrity error conditions.

4.74 ***EAI:*** During system initialization, the SIM reports error conditions using system recovery messages via the EAI.

4.75 ***Output Spooler Interface:*** The SIM sends output error messages to the maintenance terminal, the read-only printer, and the switching control center data link via the output spooler. The SIM error message has the format

REPT:SIMCHK a b

where a is the SIM error code and b is supplementary data. The SIM error codes are listed in Table B. The supplementary data is binary information that may be examined to provide additional information. The optional binary data is described in Table C. The occurrence of this message (and any supplementary data) is recorded in the SIM log file.

4.76 ***Error Logging:*** All system error conditions are recorded by SIM in the SIM log file. The overload monitor process also records errors in the SIM log file. The entries in the SIM log file are as follows:

- ***SIMER:*** SIM error conditions
- ***OVLDER:*** Overload system error conditions.

E. Overload Monitor

4.77 ***Overload Monitor:*** The UNIX RTR operating system overload monitor is a subfunction of the SIM. Its primary function is to receive reports of overload conditions from other operating system processes and inform the application program and craft persons that these conditions exist.

4.78 ***Overload Detection:*** The detection of overload conditions, with only a few exceptions, is not the responsibility of the overload monitor. The overload monitor does not routinely query system resources seeking overload conditions. Frequent queries would simply add to overload situations, while infrequent queries would not detect overload in time to be of practical use. For this reason, overload detection is done through the use of code embedded in operating system processes which control resources being utilized. As resources are allocated to UNIX RTR operating system processes, checks are made for overload conditions. As overload is detected, notification is made to the overload monitor. When the overload condition has been cleared, the overload monitor is again notified. The overload monitor has no work unless the system is in overload or is recovering from overload.

4.79 ***Reporting Overload Condition to SIM and the Application:*** All overload conditions except file system overflow are reported to SIM through the use of fault codes in the range 81 to af, hexadecimal. A unique fault code is used to report each level of overload and each time an overload condition is cleared. File system overflow conditions are reported using messages because detection of file system overflow is done partially by a user-level program.

4.80 The SIM reports all overload conditions to AIM using the application port. Whenever an overload fault or message is received by SIM, it is forwarded to the process that is attached to the application port.

4.81 ***Reporting Overload Condition to the Craft:*** Whenever SIM receives an overload fault, an "REPT:SIMCHK" output message is printed with error code 602 followed by the overload fault code. Whenever SIM receives a file system overflow message, an "REPT FS" output message is printed giving the name of the affected file system.

4.82 The following two mechanisms report overload conditions to the craft:

(a) For every overload condition reported to the overload monitor, an REPT:SIMCHK output message is generated. The error code and data field of that message indicate the type of overload fault that was received.

TABLE B

SIM ERROR CODES

CODE	MEANING
*1	System initialization failure.
2	System initialization status message received with unknown process number.
*12	Unexpected fault received by SIM.
101	Audit control subsystem initialization failure.
102	Audit record initialization error.
103	The System Integrity Output Formatter (SIOF) program is not running. Some output messages were lost.
104	Routine audit scheduling was not allowed.
121	Failure to get an audit record from the Equipment Configuration Data Base (ECD) using its record ID.
122	Failure to update an audit record in the ECD.
123	Failure to open a keyed sequence of audit records in the ECD.
124	Failure to get a record in a sequence of audit records.
125	Failure to get an audit record using the audit name and member number.
126	Failure to read the SIM control record.
127	Failure to update the SIM control record.
128	Failure to open a sequence of audit instance records in the ECD.
129	Failure to get a record in a sequence of audit instance records.
130	Failure to get an audit instance record using its record id.
131	Failure to update an audit instance record in the ECD.
132	Failure to get an audit instance record using its instance name and the RID of the associated audit record.

*Indicates conditions that may result in a system initialization phase, unless software checks are inhibited.

TABLE B (Contd)

SIM ERROR CODES

CODE	MEANING
141	Audit marked active in its ECD record when it is not running.
142	Invalid message received from the process manager or the utility manager regarding an audit process.
143	Audit reply received from an inactive audit.
144	Timeout occurred for an inactive audit.
145	Invalid reply received from an active audit.
146	SIOF output buffer overflow; some audit output messages were lost.
147	Failure to find one and only one audit process for the specified utility id.
148	Audit error report received from an inactive audit.
149	Invalid request to inform an audit process when the audit's ECD record is modified.
150	Invalid request to run or stop an audit.
151	Audit blocking condition cleared because it was invalid.
152	Failure to create a transient audit process.
153	Unable to take recovery action for an audit failure.
154	Audit inhibited by SIM because it could not be run, timed out, died, or was faulted while correcting errors.
155	Routine audit scheduling malfunctioned and is being reinitialized. All routine audits are temporarily inhibited.
156	Failure to reinitialize routine audit scheduling after it had malfunctioned.
157	Routine audit scheduling has been temporarily inhibited because SIOF is not running.
158	Failure to reinitialize routine audit scheduling after SIOF stopped running.
181	Failure to access an audit record in the plant measurements data base.

TABLE B (Contd)

SIM ERROR CODES

CODE	MEANING
182	Failure to create an audit record in the plant measurements data base.
183	Failure to open the plan measurements data base.
*200	Failure of the application integrity monitor (AIM) process to reset the application sanity timer within the timeout interval.
201	Failure to activate or deactivate the application sanity timer.
202	AIM was not connected to port PT_AIM when a PHASE 0 was requested.
206	AIM not connected to port PT_AIM for reception of overload faults.
207	AIM not connected to port PT_AIM for soft-switch request.
208	AIM not connected to port PT_AIM for notification of CU switch completion.
300	Unrecognized acknowledgement message received by SIM.
301	Unrecognized USRACK message received by SIM.
306	Message with invalid type received by SIM.
307	Unrecognized MSFAULT message received by SIM.
310	Unrecognized OST entry received by SIM.
401	Failure to respond to a request from another process for information about the most recent system initialization.
602	Overload status change.
605	Kernel-level overload monitor process failed to initialize.
606	Supervisor-level overload monitor process failed to initialize.

(b) All output messages generated by the overload monitor are also logged in the system integrity log file from which they may be retrieved by the craft through use of the OP:LOG spooler utility command.

4.83 ***Detection of UNIX RTR Operating System Overload Conditions:*** Overload detection occurs within the operating system as operating system resources are allocated. These overload detections have been implemented for all of the following overload conditions:

- Message buffer
- Memory manager
- Insufficient swap space on disk
- Segment descriptor table

TABLE C

SIM SUPPLEMENTARY DATA

ERROR CODE	BINARY DATA
1	If a boot process indicator message is received from a kernel process, the appropriate byte is set to 1; otherwise the byte remains at 0. Boot progress indicators, from left to right: BYTE — PROCESS 0 — File Manager 1 — Disk Driver 2 — Process Manager 3 — Error Interrupt Handler 4 — Capability Manager 5 — Scheduler 6 — Utility Manager 7 — Memory Manager
2	Process number of process sending the message.
12	Fault code, followed by the virtual address within SIM at which the fault occurred.
102	ECD record id (RID) of the audit record that was not initialized.
121	RID used in attempt to get an audit record from the ECD.
122	RID used in attempt to update an audit record in the ECD.
126	RID of the SIM control record (as specified in the ECDORG record).
127	RID of the SIM control record (as specified in the ECDORG record).
128	If not zero, the RID of the audit record for which a sequence of instance records could not be opened.
130	RID used in attempt to get an audit instance record from the ECD.
131	RID used in attempt to update an audit instance record in the ECD.
132	RID of the audit record for which an associated instance record could not be gotten from the ECD.
141	RID of the audit record in error.
142	RID returned in the message.
143	RID of the audit record, as specified in the reply.
144	RID of the audit record, as specified in the timeout identification.

TABLE C (Contd)

SIM SUPPLEMENTARY DATA

ERROR CODE	BINARY DATA
145	RID of the audit record, as specified in the reply.
147	Utility ID (pcode) of the process, as specified in the audit record.
148	RID of the audit record, as specified in the error report.
149	RID of the audit record, as specified in the request, followed by the process number from which the request was received.
150	RID of the audit record, as specified in the request.
151	RID of the audit instance record process number of the blocking process, and the elapsed time in milliseconds since the audit instance was blocked.
152	RID of the audit record for which the audit process was not created.
153	RID of the recovery audit that could not be run.
154	RID of the audit that was inhibited.
300	Message identity
306	Message type
307	The fault code specified in the message.
310	The invalid OST number, followed by the process number of the calling process.
602	Overload fault code: AE — Message buffer 90% full. AD — Message buffer 70% full. AC — Message buffer overload cleared. A9 — Kernel process memory overflow. Swappable main memory is at minimum size. A8 — Kernel process memory has grown to within 1/4 megabyte of its maximum size. This has reduced the size of swappable memory to within 1/4 megabyte of its minimum size. A7 — Memory overload cleared. A6 — Disk swap space 80% full. A5 — Disk swap space overload cleared.

TABLE C (Contd)

SIM SUPPLEMENTARY DATA

ERROR CODE	BINARY DATA
602 (Contd)	A4 — Segment descriptor table overflow.
	A3 — SDE table nearly full.
	A2 — SDE table overload cleared.
	A1 — Dispactcher control table overflow.
	A0 — DCT table 70% full.
	9F — DCT table overload cleared.
	9A — Kernel level process lockout.
	99 — Kernel level lockout cleared.
	98 — Kernel process timed entries.
	97 — Timed entry overload cleared.
	96 — Supervisor/user level process lockout.
	95 — Supervisor/user level lockout cleared.
	92 — File manager overloaded.
	91 — File manager overload cleared.
	90 — A disk file controller is overloaded.
	8F — Overload cleared on one DFC.

- Dispatcher control table
- Kernel process timed events and system lockup
- Supervisor/user level lockout
- File manager
- Insufficient file system space
- Disk driver
- IOP driver
- Data link input/output
- Output spooler.

The thresholds for the detection of these overload conditions are fixed.

4.84 ***Message Buffer:*** The kernel monitors the number of message buffers currently allocated. When the number of messages allocated reaches 70, 90, or 100 percent, the kernel sends a fault to the overload monitor. A different fault is used for each level.

4.85 The operating system tries to prevent message buffer overload conditions from occurring by protecting itself against "message hogs." The processes that have the highest probability of becoming message hogs are low-priority receiving processes to which kernel processes are sending messages.

4.86 ***Memory Manager:*** The memory manager has a limit to the amount of main memory

that can be filled with nonswappable processes. When creation of a new kernel process is requested, the memory manager first checks that the allocation of memory to that process will not exceed the limit. The memory manager will monitor the amount of nonswappable main memory currently allocated. When the amount allocated reaches 80 percent of the maximum, the memory manager sends a fault to SIM.

4.87 ***Insufficient Swap Space on Disk:*** The amount of swap space on disk currently in use will be monitored to detect overload. When the amount used reaches 80 percent of the total space allocated, the memory manager sends a fault to SIM.

4.88 ***Segment Descriptor Table Entry:*** The creation of any new process requires a minimum of three entries in the segment descriptor table (SDT). The maximum number of SDT entries required by a process is 128; however, most processes use less than six. The number of entries in the SDT table will be monitored to detect overload. When the number of free entries in the SDT table falls below 50, the memory manager sends a fault to SIM.

4.89 ***Dispatcher Control Table Entry:*** The kernel monitors the number of entries in the dispatcher control table (DCT); when this number reaches 70 percent, the kernel sends a fault to the overload monitor. When the DCT entries are 100 percent allocated, the kernel sends another fault to the overload monitor.

4.90 ***Kernel Process Timed Events and System Lockup:*** To ensure that timed events are receiving adequate response from the kernel, a low-level nonkillable kernel process has been implemented to monitor timed events. This process will repeatedly call for a single 5-second time-out. If the time-out is not received within 5.2 seconds three consecutive times, a fault is sent to the overload monitor.

4.91 This same kernel process is used to check for lockup of the system by a kernel process below the level of SIM. If a kernel process between levels 3 and 12 gets into an infinite loop, SIM will continue to administer the sanity timer. In order to detect this, the low-level kernel monitor process is required to send an event to SIM every time it is entered. If that event is not received at least once every 10 seconds, a fault is sent to the AIM.

4.92 ***Supervisor/User Level Lockout:*** To ensure that supervisor and supervisor/user level programs do not remain locked out of execution by system overload, a nonkillable supervisor process has been implemented. The execution priority of this process has been set to a level that is below the priority of standard UNIX RTR operating system user processes. If this process gets executed periodically, it will be certain that all normal UNIX RTR operating system user processes are also being executed. If any AIM wishes to execute processes at a lower priority so that they run only when the system is idle, the overload monitor will not be triggered when those processes fail to run.

4.93 It is the responsibility of the nonkillable supervisor process to send an event to SIM every minute. If this event is not received at least once in 5 minutes, the operating system will be considered to have locked out supervisor/user level programs and a fault will be sent to the AIM for corrective action.

4.94 ***File Manager:*** The file manager has a limited number of blocks in its internal task table. It is designed to use the system message buffers as a holding area for incoming requests. Whenever it has resources available to handle a new request, it dequeues a new message.

4.95 The file manager monitors the number of message buffers on its input queue. When the length of its input message and queue equals 20 times the number of its internal task blocks, it sends a fault to the overload monitor.

4.96 ***Insufficient File System Space:*** The file system overload is a much trickier problem to monitor than file manager task queue overload. The number of free blocks needed in a particular file system depends entirely upon the use of that file system. A file system into which little or no data is written can be almost full all the time without any harm to the operating system. On the other hand, the file system into which the error logs are written will tend to fill up if it is not frequently purged of old data. When this happens, critical error data may be lost.

4.97 The following capabilities are required in order to solve file system overload conditions:

(a) A monitor program has been implemented that will periodically record the number of free blocks in every mounted file system. This pro-

gram attempts to predict file system overflow in advance based upon the rate at which free blocks are being consumed.

(b) Whenever the file manager detects that a file system has run out of either i-nodes or blocks of free space, it will send a message to SIM identifying the file system. ⧫The monitor program repeats the message to SIM every monitoring period in which a lack of free blocks is detected.⧫

(c) Craft interface commands are provided for the general maintenance of UNIX RTR operating system file systems.

4.98 ***Disk Driver:*** The disk driver administers a separate job queue for each DFC in the system. Therefore, each controller is subject to its own overload conditions. Overload on a DFC depends on the number of jobs waiting for the controller and also on the size of these jobs. Two criteria are used to determine when a DFC is overloaded:

(a) A DFC will be considered overloaded if its internal queue of 64 jobs becomes completely full.

(b) A DFC will also be considered overloaded if incoming jobs are required to wait too long because pending jobs are too big.

The disk driver will send a fault to the overload monitor whenever either of the above conditions occur on a DFC.

5. ENHANCED TERMINAL ACCESS

5.01 This feature decreases the likelihood of a terminal suspend (inability to communicate with the system) weakening the system.

5.02 The various activities included in this feature will add or enhance four categories of system capability: Prevention, Detection, Recovery and Data Reporting.

- ***Prevention*** -The prevention mechanisms insure that should a requested service be required to be completed (in order to prevent a suspend from occurring), the requested service either in fact completes or the resultant suspension is broken within a reasonable period of time.

- ***Detection*** -Detection techniques determine when a process is in a "permanent" suspended mode.

- ***Recovery*** -Once permanent suspensions are detected, recovery aims at getting processes out of their suspended states without taking a system reboot. Phase 1 and other limited process reinitialization techniques can be used.

- ***Data Reporting*** -This technique captures and reports conditions suspended to lead to terminal suspends.

5.03 This feature is present in all new releases and effort is being made to incorporate it into earlier generics. Also, it plays a very important role in the system because it eliminates some terminal suspends.

6. MAINTENANCE INTERRUPT

6.01 A level 1 that is hardware initiated and does not involve a memory copy will be called a maintenance interrupt. A level 1 that is software initiated and does not involve a memory copy will be called a phase or an interrupt. If the UNIX RTR operating system initiates it (a utility identification < 0X400), this will be called an interrupt. If the application initiates it and the application uses the PHASE WOP OST call, the application can specify whether it is to be termed a phase or a maintenance interrupt.

7. FAULT RECOVERY

7.01 The function of fault recovery, as the name implies, is to recover the system from a fault that has been recognized by the self-checking hardware circuits of the 3B20D computer. Fault recovery includes system initialization and bootstrapping. System initialization and bootstrapping occur automatically based on the progressive initialization philosophy implemented for UNIX RTR operating system. In progressive initialization, an unsuccessful initialization attempt followed by another initialization escalates the initialization level. When the initialization levels have been escalated to the full range, manual actions must be initiated via EAI. If manual actions are also unsuccessful, support computer diagnostics may have to be run to locate the trouble.

7.02 For nonbootstrap initializations, after it has initialized the computer, system initialization enables virtual addressing and passes control to the kernel. The system initialization is never reentered unless another system initialization occurs. The following actions are taken by UNIX RTR operating system:

(a) Every kernel process must be faulted since input/output in progress is interrupted by an initialization. Also, the suspension process that was running (when the initialization occurred) is faulted. The kernel interrupt stack in the cache is lost on every initialization.

(b) The System Control Process Error Interrupt Handler is dispatched since it is the highest priority process. The system control process error interrupt handled initializes the channel hardware and sets up the kernel interrupt attach table (via OST call).

(c) The SIM sends a detailed message concerning the initialization to a port dedicated to the application. The process attached to this port controls the application initialization procedure as appropriate.

(d) The UNIX RTR operating system and the application initialization level counters are cleared at the end of the initialization interval.

7.03 ***Operating System to Application Interface:*** Associated with each initialization are an operating system initialization level and an application initialization level. The initialization actions performed by the operating system are controlled solely by the operating system initialization level. The operating system will either fault or reinitialize the application processes depending on the operating system initialization level. When an initialization occurs, an application can request a message containing various details about the initialization including both the operating system and the application initialization levels. The process attached to this port adjusts the response of the application to the initialization depending on the contents of the message.

7.04 ***Application to Operating System Interface:*** The application specifies the number of application levels for each of the operating system initialization levels and defines what actions application code will perform for each operating system initialization level. Also provided is an operating system primitive called phase that allows an operating system process as well as an application to cause an initialization. The operating system initialization level 0 allows the application to perform an initialization without the operating system performing any initialization functions.

A. Limp Modes

7.05 In the face of certain multiple faults, a limp mode may recover a useful but degraded level of performance. The following limp modes may be initiated manually via the EAI:

- Hardware checks blocked
- Error interrupt blocked
- Cache disabled
- Software error checks inhibited.

7.06 Effective with UNIX RTR operating system (release 1), the disk limp mode feature allows the system to run (with limitations) without disk access, thus reducing system outages. This prevents a total system outage when an essential disk pair is lost. The disk driver determines when disk limp mode is necessary and initiates actions to enter disk limp mode. When essential disks are repaired, a manual bootstrap is required to configure the disks back into the system.

7.07 ***Disk Limp Mode Enhancement Feature:*** This feature provides two enhancements to the disk limp mode capability of the UNIX RTR operating system (release 1). The first enhancement eliminates the need for a manual bootstrap [load disk from tape (LDFT)] by allowing the system disks to be restored from tape(s) while in the disk limp mode. This decreases system downtime by eliminating the system downtime that accompanies the LDFT method to restore disk from tape(s). The second enhancement allows in-core database to be preserved when coming out of disk limp mode. A good, current, in-core database is provided by the limp mode ECD recovery when coming out of disk limp mode.

B. Software Faults

7.08 Some software faults within a process will be discovered by the kernel and will result in the

process being faulted by the kernel. Other software faults will be discovered by the process itself. A recommended response to the discovery of software faults is to notify SIM via a message specifying an appropriate action to be taken by SIM. If the software problem cannot be corrected by one means, SIM will resort to other more drastic actions to clear it.

8. DISK RECOVERY IMPROVEMENTS

8.01 This feature seeks to improve the reliability of the disk subsystem which is done partly by reducing the amount of time the system is vulnerable to an outage (due to a single fault in the active MHD or DFC) and by reducing that outage time if one occurs. This feature consists of the following improvements:

(a) It provides for reduction of simplex time for routine maintenance

(b) It reduces occurrences of extended vulnerability caused by read errors on the active MHD

(c) It makes faster repair of data on the disks

(d) It makes faster auditing and correction of disk data while the disk is active

(e) It reduces outage recovery time.

8.02 The following improvements make up the disk recovery feature:

- Short Term Improvements
- Medium Term Improvements

8.03 ***Inhibit Routine Exerciser (REX) DFCs and MHDs*** — This improvement will reduce the vulnerability time by eliminating unnecessary routine diagnosis and restorals. Whereas active MHDs were being diagnosed daily, manually initiated diagnosis of both (MHDs and DFCs) are performed monthly. Due to the hardware and firmware being used constantly, errors and failures in most of the hardware are found operationally when they occur.

8.04 ***One Block Retry Of MHD Restore Read Job Failure*** — This improvement provides valuable information about the location of the read failure. It is expected that with this improvement the number of occurrences of read errors causing MHD restorals to abort will drop significantly. Also, when the restore does abort, the information provided will reduce the repair time needed to obtain duplex MHD operation.

8.05 ***Invalidate VTOC To Preclude Booting Only On Active Partitions*** — The purpose of this improvement is to reduce the system outage time by providing a manually initiated capability to bootstrap on a disk that quite likely has good inactive partitions. One big advantage of this capability is that it can be invoked remotely via the switching control center (SCC) link for an unattended office, greatly reducing outage time in these cases. Another advantage is that it allows for recovering on inactive partitions rather than going to a "shelf copy" or tape, which typically is more out of date than the inactive partitions.

8.06 ***Alarm Disk Exerciser Error Messages*** — This improvement is to reduce the number of occurrences of extended vulnerability by alerting craft to repair disk data errors when they are first detected. A number of disk exerciser errors must occur before the disk will automatically be removed from service. If the data errors go unrepaired, they may cause an MHD restore to abort at some later time and put the system in an extended vulnerability situation.

8.07 ***Disk Compare Command*** — This command [See Input Manual (IM)] provides for detection of read problems under much more desirable circumstances than when the restore process finds the read problems (namely, both MHDs are active). When the read problem is found, the repair time will be much less, due to the fact that the MHD with the read problem can be removed from service, reinitiated, and restored while the other disk is active. There will be no period of extended vulnerability as there is when the restore aborts due to read errors. This procedure does not require any assistance from a support center because it is performed manually. It is done when read problems are found operationally by Unix RTR operating system.

8.08 The compare feature will also provide for detection and correction of data mismatches between duplexed disks. Mismatch problems arise occasionally in the field and are not easily detected. This command provides for periodic detection of mismatches and permits manually initiated correction

for situations where the mismatch is long lived. Support center assistance is required to determine which disk has the correct value. With the aid of the "mount" and the "OP:FNAME" commands, partition numbers and block numbers can be translated into filenames for mismatched data in partitions which contain file systems.

8.09 ***Modify PDT to Backup Application Disks on Tape*** — This modification, along with the existing 3B20D computer resident "makedisk" program, will provide a general tape backup capability for all disks.

9. RECOVERY ON PREVIOUSLY ACTIVE DISKS

9.01 This feature helps guarantee that in any recovery sequence the disks that are brought into service would be the disks that were in service before the recovery began. This minimizes the problem of recovering valuable data from out-of-service disks after initializations when initialization was not caused by faulty disks.

9.02 This feature interacts with an application process before starting a disk restore — This will allow the application to determine if valuable data is about to be overwritten (by the disk restore process) and abort the restore. The application can then take action to recover the data.

9.03 The application using this feature will be responsible for the processes to communicate with the Unix RTR operating system port. Also, it will determine if valuable data exists on an OOS disk, recover the data, and initiate the disk restore (if it was aborted for this reason). These processes can be at user level and should be monitored by ULARP.

10. GLOSSARY

10.01 The following terms are used in this section:

Automatic Diagnostic Process: A UNIX RTR operating system that generates automatic diagnostic requests. When a "unit removed" message is received from CONFIG, this process will send a "diagnose unit" message to MIRA to initiate a diagnostic. This process also schedules routine diagnostics for all units.

Block: 512 contiguous bytes.

Bootstrap: A technique used by a computer to bring itself into a desired state by means of its own action.

Cache: A limited capacity, very fast semiconductor memory which can be used in combination with lower cost but slower large capacity memory giving the effect of a larger and faster memory.

C-Language: A general-purpose programming language closely associated with the UNIX RTR operating system.

Deferrable Maintenance: Those functions executed by the UNIX RTR operating system.

File: An organized collection of information directed toward some purpose.

I-list: A list of i-nodes.

I-node: A 64-byte structure that defines and specifies a file.

Interrupt: A break in the normal flow of a system or program occurring in such a way that the flow can be resumed from that point at a later time.

Nondeferrable Maintenance: Those functions associated with the UNIX RTR operating system kernel that cannot be swapped to the disk.

Off-Line: A CU is off-line when it is not in the active state and is not controlling the system. The off-line CU is the unit which is not in active control of the system configuration and execution but which may be running (executing diagnostics for example) or performing off-line functions.

On-Line: A CU is on-line if it is in the active state in which it can execute code. More specifically, for a duplex computer, the on-line CU is in active control of the system configuration and execution; its mate CU, the off-line CU, may be running (executing diagnostics), but it is not in control.

Real-Time: The actual time during which a physical process operates.

Register: A hardware entity that contains 32 bits of data.

Stack: A portion of memory for temporary storage that often operates on a last-in, first-out principle and contains process-save states.

11. ABBREVIATIONS

11.01 The following is a list of abbreviations used in this section.

ABBREVIATION	TERM
ADP	Automatic Diagnostic Process
AIM	Application Integrity Monitor
ATB	Address Translation Buffer
BTC	Bus Interface Controller
CC	Central Control
CONFIG	Configuration Control
CSOP	Coordinator Spooler Output Process
CU	Control Unit
DCB	Diagnostic Control Block
DCT	Dispatcher Control Table
DDSBS	Duplex Dual Serial Bus Selector
DFC	Disk File Controller
DIAGC	Diagnostic Control
DIAMON	Diagnostic Monitor
DMA	Direct Memory Access
DMAC	Direct Memory Access Controller
DSCH	Dual Serial Channel
EAI	Emergency Action Interface
ECD	Equipment Configuration Data Base
ECDFUNC	ECD Function
ECDMAN	Equipment Configuration Data Base Manager
GRASP	Generic Access Package
INITLVL	Initialization Level
IM	Input Manual
IOP	Input/Output Processor
KBOOT	Kernel Boot
LDFT	Load Disk From Tape
MAS	Main Store
MASC	Main Store Controller
MHD	Moving Head Disk
MIRA	Maintenance Input Request Administrator
MRF	Maintenance Reset Function
OOS	Out-of-service
OVLDER	Overload System Error Conditions
PAS	Protected Application Segment
PDS	Program Documentation Standard
PIC	Peripheral Interface Controller
RAM	Random Access Memory
RID	Record Identification
rwx	Read, Write, and Execute
SCC	Switch Control Center
SDE	Segment Descriptor Table
SIM	System Integrity Monitor
SIMER	SIM Error Conditions

ABBREVIATION	TERM
SIOF	System Integrity Output Formatter
SYSIMSG	System Initialization Message
TLDB	Trouble-Location Data Base
TLP	Trouble-Locating Procedures
TLPR	Trouble-Location Process
TTY	Teletypewriter
TUS	Test Utility System
ULARP	User Level Automatic Restart Process
VTOC	Volume Table of Contents