PUBLISHED: 9/13/66

Identification.

The Fault Interceptor
Chester Jones

Purpose.

The Fault Interceptor Module is the interface between the hardware fault mechanism and the procedures for handling the various fault conditions. The Fault Interceptor Module is responsible for saving the processor state when that processor faults, for calling the appropriate procedure for handling the fault, and for restoring the processor state after control returns from the fault handling procedure. The supervisor protection mechanism (Section BD.9) requires that crossing a wall in either direction be detected by a fault. Since it contains parts of the wall crossing mechanism, the Fault Interceptor Module is accessible in every ring in the sense that it can be entered without first crossing a wall, and it is able to switch rings when necessary. The Fault Interceptor Module must execute privileged hardware instructions and inhibit interrupts during certain crucial operations. Therefore, the Fault Interceptor Module is a hand-coded, master mode procedure that can be entered only as a result of a hardware fault condition.

Summary of the Fault Interceptor Actions.

When a Multics processor generates a fault, control passes (automatically, through the processor fault vector) to the Fault Interceptor Module which executes as part of the process that is running at the instant the fault occurs. While executing within the ring in which the fault occurs, the Fault Interceptor Module temporarily saves the processor state in the Process Concealed Stack (Section BJ.1.05) that belongs to the running process and makes space available for safe-storing the processor state should another fault occur. Then, the actions of the Fault Interceptor Module vary, depending on the probable cause of the fault.

For user faults, the Fault Interceptor Module performs the following steps:

1. Copies the safe-stored processor state from the Process Concealed Stack into the paged stack for the protection ring in which the fault occurred.

2. Uses the paged stack for that protection ring to call the procedure for handling the fault in that ring.

3. Copies the safe-stored processor state from the paged stack into the Process Concealed Stack.

4. Checks the validity of the safe-stored processor state.

5. Restores the processor state to return control to the point at which the fault occurred.

For system faults (except for missing-page faults, connect faults, and timer runout faults), the Fault Interceptor Module performs the following steps:

1. Switches control to the hard core ring.

2. Switches from the Process Concealed Stack to the paged stack for the hard core ring.

3. Calls the procedure for handling that fault.

4. Switches back to the Process Concealed Stack from the hard core ring paged stack.

5. Switches control back to the ring in which the fault occurred.

6. Checks the validity of the safe-stored processor state.

7. Restores the processor state to return control to the point at which the fault occurred.

(For a description of the actions of the Fault Interceptor Module in response to missing-page faults, timer runout faults, and connect faults, see Sections BK.3.06. BK.3.07, and BK.3.08 respectively.)

Actions of the Fault Interceptor Module.

1. The processor state is stored in the current interrupt frame of the Process Concealed Stack. Pointers to the current interrupt frame are maintained at the base of the Process Concealed Stack. The processor state is stored in three steps:

   a. The processor control unit is stored (store control unit instruction) using the scu pointer. (This instruction is executed in the processor fault vector.)

   b. The arithmetic registers are stored (store registers instruction) using the sreg pointer.

   c. The address base registers are stored (store bases instruction) using the stb pointer.

2. The values for the linkage pointer and linkage base registers are determined and loaded into the lp-lb base register pair. (Since the Fault Interceptor Module is not called, it must establish its own linkage values.) Since the Fault Interceptor Module is a master mode

procedure, the segment number of its linkage section will always be one more than the segment number of the Fault Interceptor Module itself.

3. A new interrupt frame is allocated in the Process Concealed Stack in preparation for the next fault (or internal interrupt). Since a processor may fault (or accept an internal interrupt) in the course of handling an earlier fault (or internal interrupt), care is taken to prevent a possible overlapping of the new interrupt frame and the interim stack in use at the instant of the fault. A new interrupt frame is allocated as follows:

   a. The value of the stack base register, sb, (stored in Step 1) is examined. If the stack base register, sb, does not contain the segment number of the Process Data Segment, then the new interrupt frame is allocated immediately following the current interrupt frame. (It is important to emphasize that sb indicates whether the new interrupt frame should be allocated relative to the stb pointer or relative to sp|18.) In this case, the base location of the new interrupt frame is obtained by adding 24 (i.e. the length of an interrupt frame) to the stb pointer stored at the base of the Process Concealed Stack. If the stack base register contains the segment number of the Process Data Segment, then the sp-sb base register pair points to the base location of the stack frame in use at the instant of the fault and sp|18 points to the base location of the next (intended) stack frame (i.e. next sp). In this case, a constant, k, is added to the value of next sp to obtain the base location of the new interrupt frame. (Currently, k is equal to 32.)

   b. A back pointer to the previous interrupt frame is fabricated and stored into locations 22-23 of the new interrupt frame.

   c. The stb, sreg, and scu pointers stored at the base of the Process Concealed Stack are adjusted to point to the appropriate areas within the new interrupt frame.

4. For system faults only, control is switched to the hard core ring of the running process. The basic mechanism for passing control from one protection ring to another is a "load descriptor base register" instruction whose address points to the value for the descriptor base register in the new ring. Control is switched to the hard core ring as follows:

   a. The value for the descriptor base register in the hard core ring of the running process is obtained from the Process Data Block (Section BJ.1.04).

b. A "load descriptor base register" instruction is executed. It is important to note that the instruction following the "ldbr" instruction is executed in the hard core ring.

c. The paged stack is switched to the hard core ring paged stack.

5. The safe-stored processor state is copied from the Process Concealed Stack into the paged stack for the protection ring in which the processor is executing. (Note that the processor is executing in the hard core ring for system faults; for user faults, the processor is executing in the ring in which the fault occurred.)

   a. The values of the stack base register pair, sp-sb, are obtained. The stack pointer register, sp, points to the base location of the stack frame in use and sp|18 points to the base location of the next (intended) stack frame.

   b. The interrupt frame containing the processor state (Step 1) is copied from the Process Concealed Stack into the paged stack. (Since the Process Concealed Stack has been "pushed down," a missing-page fault during the attempt to copy the processor state is acceptable.) The base location of the interrupt frame into which the processor state is copied is obtained by adding 32 to the value of next sp.

   c. An interim stack is created immediately following the interrupt frame. The first stack frame of the new interim stack is initialized.

   d. The stack base register pair, sp-sb, is set to point to the base location of the newly created interim stack.

6. The space required for temporarily saving the processor state is returned to the Process Concealed Stack (i.e. the Process Concealed Stack is "popped-up" one level.) The values for the stb, sreg, and scu pointers are derived from the back pointer stored in locations 22-23 of the current interrupt frame.

7. A standard CALL is issued to the appropriate module to handle the fault. This module may CALL other modules. Until a RETURN is made to the Fault Interceptor Module, all steps for handling the fault are taken by other procedures. When control returns to the Fault Interceptor Module, the sequence of steps presented here continues.

8. For system faults only, control is returned to the ring in which the fault occurred.

9. The safe-stored processor state is copied from the paged stack into the current interrupt frame of the Process Concealed Stack.

10. The safe-stored processor state is checked to insure that the control unit information is valid.

11. The processor state is restored to return control to the point at which the fault occurred.

Hardware Interface.

Each of the 32 faults included in the GE-645 fault repertoire has a corresponding pair of instructions, the fault vector pair, which the processor executes automatically when it generates the fault. The 32 fault vector pairs constitute the processor fault vector, a 64-word block of core memory whose base address is determined by switch settings on the processor maintenance panel. Each of the 32 fault vector pairs has a 5-bit fault code (00000-11111) which the processor uses to locate the fault vector pair for that fault. The 32 fault vector pairs are ordered within the processor fault vector by increasing fault codes. When the processor control unit detects a fault condition, it takes a "snapshot" of its internal status and aborts the entire processor. At the end of the abort cycle, the processor executes the fault vector pair that corresponds to the condition causing the fault.

When a Multics system is initialized (Section BL), each fault vector pair is set to store the processor control unit in a safe place (usually in the Process Concealed Stack) and transfer to the procedure for handling the corresponding fault (usually in the Fault Interceptor Module.) In general, the fault vector pairs have the form shown below. In the following example, pds represents the segment number of the Process Data Segment (Section BJ.1.03), fim represents the segment number of the Fault Interceptor Module, and entry represents the offset in the Fault Interceptor Module that corresponds to the entry point for the fault condition.

```
Inhib   on
scu     =its(pds,4,*),*   control unit into Concealed Stack
tra     =its(fim,entry),* transfer to entry point
```

It is important to note that the addresses in the fault vector pairs are 18-bit absolute core addresses which point, indirectly, through 72-bit ITS-pointers, to segments that belong to the running process. Both the processor fault vector and the set of ITS-pointers are "wired" into core memory and cannot be paged onto secondary storage.