TO: MSPM Distribution
FROM: J. H. Saltzer
SUBJ: BK.3.01-03 and BK.3.06
DATE: 10/02/67

Sections BK.3.01, 02, 03, and 06 are revised here. The main change is that the Fault Interceptor Module handles process faults by issuing a signal, rather than by calling a user fault handler. The fault_stack is also added, a mechanism found necessary to implement wall-crossing faults.

Published: 10/02/67
(Supersedes: BK.3.01, 09/13/66)

## Identification

Overview of Fault Handling
Chester Jones

## Discussion

A fault, by Multics definition, is a condition (such as accumulator overflow) detected within the processor hardware which causes that processor to depart from the procedure it was executing. Conditions which cause faults are not necessarily error conditions. For example, a program reference to an area not currently in core memory causes a "missing-page" fault.

Faults occuring in the Multics system are divided into two mutually exclusive categories, named system and process. System faults include those faults which indicate hardware or software failures and other faults which require system-wide fault handling strategies. System faults may occur at any time, regardless of which user process has control of the processor. [They may not, in general, be considered to be "programmed" by the running process.] The memory parity fault, the illegal descriptor fault, and the directed faults are examples of system faults. Process faults, on the other hand, always occur as a consequence of some action in the running process. Overflow, derail, and fault tag 2 (linkage fault) are examples of process faults. MSPM Section BK.3.02 contains a complete breakdown of the GE-645 faults into system and process fault categories.

The basic philosophies for handling system and process faults are quite different. The underlying philosophy of handling system faults is that they are the responsibility of the operating system; they are handled on a system-wide basis. Their handling must be invisible to the user process. Certain of the system faults have reserved meanings in Multics. For example, directed fault 0 is reserved to indicate a missing page or segment; the connect fault is reserved to mean "clear your associative memory." Such reserved system faults are viewed as calls originating in the processor hardware to "hidden" modules of the operating system. The role of the fault handling routines is to transform system faults into "automatic" calls to the appropriate Multics modules to handle the faults.

The underlying philosophy of handling process faults is that they are the responsibility of the user process; they are handled on a per-process basis. Multics contains a standard procedure for handling each fault. However, there is at least one point in each process fault handling procedure where control may pass to some other procedure in the process if that process is administratively entitled to provide alternative handling for that fault. Process faults are viewed as signals similar in nature to condition signals but originating in the processor hardware (see MSPM Section BD.9.04). The fault handling mechanism for process faults has been subsumed under the condition handling mechanism in Multics. Thus, the role of the fault handling routines is to transform process faults into appropriate condition signals.

## The Fault Interceptor Module

All faults are passed directly to the Fault Interceptor Module by a transfer instruction in the processor fault vector. This transfer instruction is set up at system initialization or reconfiguration time. The Fault Interceptor Module is a standard Multics procedure segment, except for its unorthodox method of entry. It operates in master mode, but not in absolute mode.

Conceptually, the Fault Interceptor Module is quite simple. It saves the processor state in a safe place and calls the appropriate handler to service the fault. The handler, a standard Multics procedure, performs the appropriate action, modifies the machine conditions (if necessary), and returns control to the FIM, which restores the processor state and returns control to the point at which the fault occurred.

Actually, the Fault Interceptor Module is slightly more complex than as described above in order to satisfy the following requirements.

1. Fault identification. Not all faults recognized by the GE-645 processor hardware have unique meanings in Multics. For example, directed fault 0 may indicate a missing page, a missing segment, or an out of bounds reference depending on the situation in which it occurs. The illegal procedure fault may indicate an illegal instruction, improper use of a privileged instruction, an access violation (which <u>may</u> be an outward wall crossing attempt in some instances), or an out-of-bounds reference (which <u>may</u> be one of the software simulated faults -- see MSPM Section BB.5.03). Therefore, before it can call the handler to service a fault generated by the hardware, the FIM must first identify the fault in order to determine which handler is "appropriate".

2. Ring switching. While system faults may occur at any time, in any protection ring of a process, procedures for handling system faults may not be called from outside the hardcore ring. Therefore, in order to maintain the illusion that system faults are invisible to the user process, the FIM must switch to the hardcore ring before it calls system fault handling procedures and switch back to the original ring before returning control to the faulting procedure. In addition, conditions associated with certain of the process faults must be handled in specific rings regardless of where they occur. For example, fault tag 2 (linkage fault) must be handled in the administrative ring whenever it occurs. So, in addition to its responsibility for calling the appropriate fault handling procedures, the FIM is responsible for first switching to the appropriate protection ring.

3. Stack management. Associated with each process in Multics is a (large) number of call stacks (see MSPM Section BD.9.xx) having a wide variety of protection modes. Since faults may occur at any time, in any protection ring of a process, there is no a priori way of knowing which call stack will be in use when a fault occurs. Several points should be noted. First, the FIM must save the machine conditions on a push-down list since "cascaded" faults may cause the FIM to be entered recursively. Second, many of the fault handling procedures must modify the machine conditions before returning control to the FIM. Therefore, the machine conditions must be stored in a stack that is (write) accessible to the fault handling procedure. Finally, not all fault handling procedures will return control to the FIM; the machine conditions cannot be stored in a "wired-down" stack. The FIM's responsibility, then, is to insure that the stack used to call a fault handling procedure is accessible to that procedure and that stack is in a consistent state when the fault handling procedure is called.

4. Supervisor protection. Since the machine conditions associated with a fault are write-accessible to the fault handling procedure, that procedure may freely modify the machine conditions before returning control to the FIM. In many cases, the machine conditions must be modified by the fault handling procedure and the FIM must use the modified copy to restore the processor state. The machine conditions may be modified so that, when the FIM restores the processor state, the system protection mechanism (MSPM Section BD.9) is violated. To prevent this the FIM must make a new copy of the now-changed machine conditions in a safe place and carefully examine that copy to check the validity of the modifications made by the fault handling procedure.

In order to tentatively identify a particular hardware fault, that is, in order to satisfy the first requirement, the FIM contains a list which relates hardware fault conditions to faults expected by Multics. The list contains an entry for each fault condition (or subcondition) that can occur. An entry in the list indicates whether the fault is a system fault or a process fault. For process faults, the entry includes the condition name associated with the fault and the ring number in which the condition is to be signaled. For system faults, the entry indicates which Multics module is to receive control upon occurrence of the corresponding system fault. Following a hardware fault, the FIM uses information captured by the hardware to determine which entry in the list to use. Once tentative identification is made, the FIM performs the necessary stack management and ring switching for that fault and calls the appropriate handler. In ambiguous cases, the FIM assumes the fault is a system fault, performs the usual system fault housekeeping, and calls the system fault handler. If the handler returns an error code meaning the fault was "encountered in a non-standard situation," then the FIM "un-does" the system fault housekeeping and treats the fault now or a process fault by signalling corresponding condition.

In order to perform its ring switching duties, that is, in order to satisfy the second requirement, the FIM must be accessible in every ring of every process and it must be able to switch rings when necessary. Since the ring mechanism for supervisor protection is implemented by using a different descriptor segment for each ring, the basic mechanism for switching rings is the (privileged) "load descriptor base register" instruction. This instruction and associated housekeeping are contained in a privileged Basic File System procedure, ring$load (MSPM Section BG.3.05). In order to abandon one ring in favor of another, the FIM issues a standard call to ring$load. Control returns to the FIM in the new ring. Following a wall-crossing fault, normal ring switching is performed in two steps. First, the Gatekeeper (MSPM Section BD.9.04), executing within the hardcore ring, performs the housekeeping (including stack switching) necessary for a "formal" wall crossing. Second, the FIM calls ring$load to switch to the descriptor segment for the target ring. It is important to note that the wall-crossing fault is a system fault; the Gatekeeper can only be called from inside the hardcore ring. (Clearly, a call to the Gatekeeper must not produce a wall-crossing fault.) Before it calls the Gatekeeper, the FIM must first call ring$load to switch to the hardcore ring.

The FIM's stack management duties following a fault depend on whether the fault is a system fault or a process fault. Following a process fault, the FIM calls signal (MSPM Section BD.9.04) to indicate the occurrence of the corresponding condition. However, before it can call signal, the FIM must first set up a new stack frame in the call stack for the ring in which the condition is to be signalled. This new stack frame, called a "placeholder" frame (see Figure 1), contains a copy of the machine conditions at the instant a fault occurred. (This step in stack management is also taken by the Gatekeeper when it switches stacks. (See also MSPM Section BD.9.01.) Following a system fault, the FIM switches to a special paged stack (named fault_stack) before calling the system fault handling procedure. In order to understand the use of fault_stack, consider the steps taken following a legitimate attempt to enter the hardcore ring. First, a process, executing in some ring other than the hardcore ring, calls a valid entry point to the hardcore ring, causing a wall-crossing fault. Since the wall-crossing fault is a system fault, the FIM switches to the hardcore ring (using ring$load) and tries to call the Gatekeeper to handle the fault. The FIM cannot use the standard paged stack for the hardcore ring in order to call the Gatekeeper since the Gatekeeper will insert a "placeholder" frame in that stack in preparation for switching to it. The FIM cannot use the wired-down Process Concealed Stack (MSPM Section BJ.1.04?) to call the Gatekeeper because the Gatekeeper may get missing-page faults. Therefore, the FIM uses a third stack the fault_stack, to make the call.

Following a process fault, the FIM makes a copy of the machine conditions in the "placeholder" frame of the call stack to be used as required by the fault handling procedure. A slightly more complete overview of the FIM may now be given.

When a Multics processor generates a fault, control passes automatically to the FIM which executes as part of the process that is running at the time of the fault. While executing within the ring in which the fault occurs, the FIM safe-stores 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 or interrupt occur. Then, the actions taken by the FIM vary, depending on the probable cause of the fault. For missing-page faults, the fault interceptor switches to the hardcore ring and uses the Process Concealed Stack to call the Basic File System (Section BG) to supply the missing page. For other system faults, the fault interceptor switches to the hard core ring copies the safe-stored processor state from the Process Concealed Stack into the fault_stack that belongs to the running process, and calls the appropriate procedure for handling the fault. For process faults, the FIM consults its list of faults to determine the number of the ring in which the associated condition is to be signalled and calls the Gatekeeper to switch rings if necessary. Then, the FIM builds a "placeholder" stack frame in the call stack and deposits copies of the machine conditions in the "placeholder" frame. Finally, the FIM calls signal to report the occurrence of that condition.* If, and when, control returns from the fault handling procedure, the fault interceptor checks the validity of the processor state, restores the processor state, and returns control to the point at which the fault occurred.

Catastrophe Module

The catastrophe module (Section BK.3.04) performs the initial handling of system faults that indicate either an appending hardware malfunction or some possibly fatal error in the operating system. For example, following an illegal descriptor fault, the catastrophe module checks the validity of the descriptor for the fault interceptor before transferring control to the fault interceptor.

The catastrophe module is an absolute mode module that can be entered only as a result of a hardware error condition.

*To avoid the need to prelink a path through the signal procedure to the linker, the FIM makes a special core of fault-tag 2 by calling the linker directly instead of signalling. No loss of generality occurs, for a process could not replace its linker by such a simple Technique as setting a new on-condition anyway -- since dynamically set on-conditions depend on having a linker available to operate properly when signalling occurs.

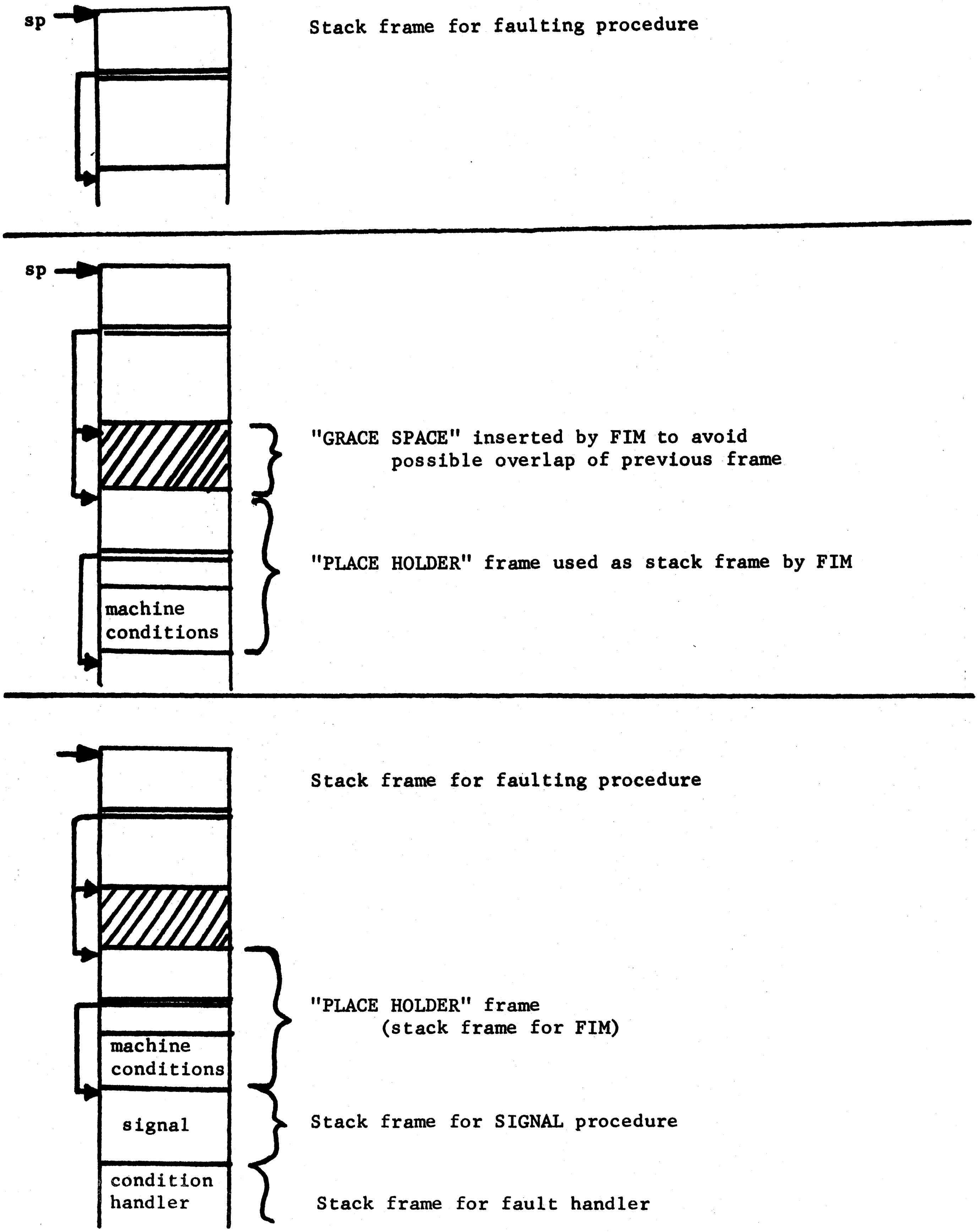
sp
Stack frame for faulting procedure
sp
"GRACE SPACE" inserted by FIM to avoid possible overlap of previous frame
"PLACE HOLDER" frame used as stack frame by FIM
machine conditions
Stack frame for faulting procedure
"PLACE HOLDER" frame (stack frame for FIM)
machine conditions
signal
Stack frame for SIGNAL procedure
condition handler
Stack frame for fault handler

Figure 1