Published: 05/26/67

Identification

Fault handling during initialization
A. Bensoussan

Purpose

This section describes how faults are handled during execution of the Multics initializer; it also provides a description of the interim fault interceptor module.

Introduction

During Multics initialization, before the Multics fault handling mechanism is in operation, only 5 types of faults are expected. They are:

a. Linkage fault

b. Missing page fault

c. Missing segment fault

d. Timer runout fault

e. Connect fault

The reason why linkage faults, missing page and segment faults are expected is explained in BL.5.00. The timer runout fault may occur at any time and must not be considered as an error; it will be ignored by the Multics initializer, except during the traffic controller initialization (see BL.11). The purpose of the connect fault, in Multics, is to clear the associative memory of the faulting processor; during Parts 3 and 4, the paging mechanism is in operation; each time a page is to be removed, page control sends a connect signal to all processors in the system.

When one of these 5 faults occurs the fault vector stores the control unit information in the concealed stack and transfers to the "interim fault interceptor" (IFI); the IFI saves the processor state, identifies the fault and calls the appropriate handler.

Fault handlers and Modes

The fault handler called by the IFI for a given fault is not necessarily the same during the whole initialization. In fact, most of the time, the handler to be called depends on what Part of the initialization the fault occurs. For instance, if a missing page fault occurs during Part 1, the IFI calls "interim1_page fault", while if Part 3 it would call the Multics page fault handler.

Therefore the IFI runs into three different modes:

Mode 1 is active during Part 1 and Part 2

Mode 2 is active during Part 3

Mode 3 is active during Part 4

The IFI can determine the current active mode by testing the value of its variable "mode"; it can switch from one mode to another by setting the variable "mode" to the appropriate value when it is told to do so by a call from the initializer control program. Figure 1 indicates the appropriate handler to be called by the IFI for each fault and each mode. The functions performed by these handlers and the calling sequences are described below.

1. Linkage fault

   In any mode the handler is the initialization linker. The calling sequence is:

   call linker (mach_cond,er_ret,er_code,option) /*see BL.7.01*/

2. Missing page fault

   a. Mode 1: the interim 1 page fault handler is called, which allocates the core required for the page and sets the page table word. The calling sequence is:

      call inter1_pagefault (scuptr) /*see BL.6.03*/

   b. Mode 2: The Multics page fault handler, now available, is called to handle a missing page fault. The calling sequence is:

      call pagefault (scuptr,dbrptr,errcode) /*see BG.4.00*/

   c. Mode 3: same as b.

3. Missing segment fault

   a. Mode 1: The interim 1 segment fault handler is called, which allocates the required core for the page table, manufactures the page table showing that all pages are missing and sets the descriptor segment word to point to the page table. The calling sequence is:

      call interim1_segfault (scuptr) /*see BL.6.03*/

   b. Mode 2: The fault handler used in mode 1 cannot be used in mode 2 because page control is in operation and expects the missing segment fault handler to perform more jobs: create Active Segment Table (AST) entry, set page table word to point to AST entry etc...(See BL.10.02). Since the Multics missing segment fault is not available yet, a second interim handler is called, which is compatible with page control. The calling sequence is:

      call interim2_segfault (scuptr,dbrptr,ringno, errcode) /*see BL.10.02*/

      The ring number is ZERO.

   c. Mode 3: The Multics handler, now available, is called. The calling sequence is:

      call segfault (scuptr,dbrptr,ringno,errcode) /*see BG.3.00*/

      The ring number is ZERO.

4. Timer runout fault

   a. Mode 1: The fault is ignored by calling no handler.

   b. Mode 2: The fault is ignored by calling no handler.

   c. Mode 3: The Multics timer runout procedure is implemented as an internal procedure in the Fault Interceptor and cannot be called by the IFI. The handler which is called performs the same function as an internal procedure in the IFI. The calling sequence is:

      call timer_runout /*see interim fault interceptor*/

This handler generates a timer runout interrupt for the processor on which it is executed. It should be noted that the timer runout interrupt handler ignores the timer runout interrupt if the "drain" switch in the process data block is ON, which disables the timer runout interrupt.

When the traffic controller initializer creates the File System Device Monitor process, in Part 4, it expects the timer runout interrupt to occur (BL.11). By setting the drain switch ON or OFF, the traffic controller initializer can disable or enable the timer runout interrupt as it wishes.

5. Connect Fault

   a. Mode 1: The fault is ignored by calling no handler.

   b. Mode 2: The connect fault is generated when a page is to be removed in secondary storage by page control. The action taken by the connect handler is to clear the associative memory of the faulting processes. The calling sequence is:

      call connect /*see interim fault interceptor*/

      The Multics connect handler is implemented as an internal procedure in the Fault interceptor and cannot be called by the IFI. The handler called by the IFI performs the same function and is implemented as an internal procedure in the IFI.

   c. Mode 3: same as b.

<u>Interim Fault Interceptor</u>

The interim fault interceptor (IFI) is an adaptation of the Multics fault interceptor for initialization purposes. It is transferred to by the fault vector when an expected fault occurs during initialization.

The following actions are taken when the IFI is entered after an expected fault:

a. Save the processor state in the concealed stack

b. Set lp-lb to point to the linkage section of the IFI

c. Allocate a new interrupt frame in the concealed stack

d. For linkage and missing segment faults only, switch from the concealed stack to the hardcore stack

e. Call the appropriate fault handler depending on the fault and also on the mode in which the IFI is currently running

f. For linkage and missing segment faults only, switch back to the concealed stack

g. Return the current interrupt frame

h. Restore the processor state

All these steps, except step e, are described in detail in the fault interceptor section (BK.3.03). Step e has been described in the previous paragraph.

<u>Internal procedures in the IFI</u>

As said above, the timer runout and the connect fault handlers called by the IFI are implemented as internal procedures in the IFI. Their functions are identical to the functions executed by the Multics handlers, that is:

1. Timer runout procedure

   a. Obtain the processor index number (0-7) from the processor data block.

   b. Use the processor index number to obtain the appropriate pattern for setting the time out interrupt cell for the processor on which the IFI is executing. This pattern is found by using the processor index number into the time out pattern array of the processor communication table (see BK.1.04).

   c. Set up and execute a "set memory controller interrupt cell" instruction whose address points to the memory controller through which the time out interrupt signal is sent to that processor.

2. Connect procedure

   a. Obtain the processor index number (0-7) from the processor data block.

   b. Execute a "clear associative memory" instruction.

   c. Clear the connect flag array entry for the executing processor in the processor communication table (BK.1.04).

<u>Entry points of the IFI</u>

The interim fault interceptor has 3 entry points:

1. <interim_fi>|[interim_fi]

2. <interim_fi>|[use_mode_2]

3. <interim_fi>|[use_mode_3]

The first entry is transfered to by the interrupt vector when any of the 5 expected faults occur.

The second entry is called by the initializer control program at the end of Part 2. The action taken by the IFI is simply to set the variable "mode" to the value 2.

The third entry is called by the initializer control program at the end of Part 3. The action taken by the IFI is simply to set the variable "mode" to the value of 3.

<u>Description of the Fault Vector</u>

When the Multics initializer is entered, the fault vector has been initialized by the bootstrap initializer in the following manner:

1. For the 5 expected faults mentioned above, the fault pairs contain:

scu = its(pds, scuptr,*),*	store control unit in concealed stack
tra = its(ifi, entry),*	transfer to interim fault interceptor

where "pds" represents the segment number of the process data segment, "scuptr" represents the offset of the SCU pointer in the process data segment, "ifi" represents the segment number of the interim fault interceptor and "entry" represents the offset of the entry point in the interim fault interceptor. This entry point is unique for all 5 faults.

2. For any other fault, the fault pairs contain:

scu = its(stop, control_unit),*	store control unit in segment stop
tra = its(stop, entry),*	transfer to segment stop

where "stop" represents the segment number of the segment stop, "control unit" represents the offset of the location where the control unit is to be stored in the segment stop and "entry" represents the offset of the entry point in the segment stop. A transfer to the segment stop means that a fatal error has occurred and that the Multics initializer has to stop.

The fault vector remains in this state during almost the whole execution of the Multics initializer. It will be changed in Part 4 by a call from the initializer control program to the fault initializer (call fault_init $ two).

Upon return from this call, the fault vector will be set to the final form required by the Multics system and will cause the Multics initializer to switch from the interim fault interceptor to the Multics fault interceptor.

	Mode 1 (Part 1 and 2)	Mode 2 (Part 3)	Mode 3 (Part 4)
linkage fault	linker	linker	linker
page fault	interim1_pagefault	pagefault	pagefault
segment fault	interim1_segfault	interim2_segfault	segfault
time out fault	none	none	timer runout
connect fault	none	connect	connect

This figure provides, for each fault, the handler called by the interim fault interceptor in each mode.

Figure 1