Published: 06/15/67

Identification

Interrupt handling during initialization
A. Bensoussan

Purpose

This section describes how interrupts are handled during initialization.

There are 2 distinct periods during initialization as far as the interrupt handling is concerned. During the first period, only interrupts coming from the bootload GIOC, following a read operation from the Multics System Tape, are expected. The second period starts in Part 1, as soon as the GIM and the Interrupt Interceptor are initialized and lasts the rest of the initialization; during this second period, the interrupt interceptor is in operation and can handle any interrupt.

First period

The only interrupt that is expected is the interrupt coming from the bootload GIOC when the tape reader reads the MST. The configuration is not known yet but the interrupt is expected from the bootload GIOC, status channel 0 or 1. We know that, by a Multics convention (BC.1.04), status channels 0 and 1 are assigned interrupt cell number between 0 and 11 in any GIOC. Therefore the bootstrap initializer has set the interrupt pairs 0 to 11 with SCU/TRA instructions in the 64 word interrupt vector block associated with the memory controller containing the base address of bootload GIOC. The base address of this 64 word block was passed by the bootload program to the bootstrap initializer through an index register (Register 1).

Interrupt pairs 0 to 11 are set as follows:

```
scu =its (tape_reader, control_unit),*

tra =its (tape_reader, interrupt),*
```

where "tape_reader" represents the segment number of the tape reader, "control_unit" represents the offset of the location where the control unit is to be stored in the tape reader segment and "interrupt" represents the offset of the entry, in the tape reader, that handles the tape interrupt.

Any other interrupt sent to the processor used by the bootload causes the Multics initializer to stop.

For reading the MST, the tape reader sets its word "event_cell" to the value 0; then it issues a connect GIOC instruction and loops, waiting for the value of event_cell to become non-zero. When the interrupt occurs, the interrupt vector transfers to the entry "interrupt" in the tape reader, which sets event_cell to the value 1 and restores the control unit.

Second period

After the system configuration table, the GIM and the interrupt interceptor have been initialized, a call is issued, in Part 1, to the interrupt initializer for setting the fault vector to its final form. The second period starts when a return from this call is experienced. From this point on the Multics initializer can execute I/O operations in any secondary storage device available to the file system.

During the second period, all interrupts sent by any device will be handled by the Multics mechanism, except that the module "wake-up" performs a special function during initialization. That is:

a. The interrupt vector transfers control to the appropriate entry in the interrupt interceptor.

b. The interrupt interceptor calls the appropriate interrupt handler.

c. The interrupt handler signals, in the Device Signal Table (DST), the fact that the device has sent an interrupt, and calls wake-up for the process associated with the device.

d. Wake-up performs a return.

Multiprocessor System Initialization

In a multiprocessor system initialization, the processor P(0) used by the bootload is not the only processor invoked by the Multics initializer. Any other processor P(i) may be awakened by an interrupt or a fault; the reasons are the following.

a. Interrupt. In Part 1, the configuration is made known, the GIM and the interrupt interceptor are initialized; from this point on, the file system initializer can read and write in any secondary storage device available to the file system in this configuration. When an I/O operation is completed, that has been requested by P(0), an interrupt is sent not necessarily to the processor P(0) but to the processor that happens to be the control processor for the memory controller containing the base address of the GIOC or drum involved in the I/O operation. Therefore, this processor must be prepared to receive an interrupt.

b. Connect fault. During Part 3 and Part 4, when a page is to be removed in secondary storage, associative memories of all processors have to be cleared. As explained in BK.3.08, when a Multics module must clear associative memories of all processors in the system, it issues a call to the master mode procedure "connect_generator". This routine sends a connect signal to each processor in the system.

c. A timer runout fault may occur at any time while processor P(i) is handling the connect fault or an interrupt.

When the bootload button is pushed, the processor P(0) is interrupted; it executes the bootload program and the bootstrap initializer in which it is given a descriptor segment and the associated DBR value; any other processor P(i) is sitting on a DIS.

In Multics, all processors have the same base address. Therefore any attempt to wake-up a processor P(i) by a fault or interrupt will cause P(i) to jump at the corresponding location in the fault-interrupt vector and to execute the pair instruction stored at this location. If the pair instruction is an SCU/TRA using the appending mode, this implies that the processor has been initialized, that is, it has been provided with a descriptor segment and the associated DBR value.

The most general solution is to have any processor P(i) execute the same fault or interrupt handler that would be executed by the processor P(0). This implies that any processor P(i) is provided with a descriptor segment describing all segments needed for the fault and interrupt

handling and that this descriptor segment describes a process data segment and a processor data segment that are private segments for processor P(i); furthermore the processor index number must be entered in the processor data block (this number is used by the timer runout fault handler and by the connect fault handler).

This will be done by a call to an entry of the traffic controller initializer, issued in Part 1 by the initializer control program, before fs_init_1 is called.

It should be noted that any processor other than P(0) is used only for handling an interrupt, a connect fault or a timer runout fault. After having handled the fault or interrupt handler, the processor returns to wait on a DIS.