Published: 06/30/67

Identification

Process Exchange during initialization
A. Bensoussan

Purpose

During initialization page control procedures are executed as part of the MULTICS initializer; when waiting for a page, the procedure "page_fault" issues a call to "Block" and when the page has arrived, the procedure "io_done" calls "wakeup". The "device signal table manager" is also executed as part of the MULTICS initializer and issues a call to "wakeup". But the MULTICS initializer has not yet set the proper environment required by Block and wake-up to perform their standard jobs. Therefore, when called during initialization, Block and wake-up execute special functions.

This write-up describes what these special functions are, how they can be executed instead of the standard ones, and why they have been designed the way they are.

Introduction

As explained in BL.5.00, a certain number of MULTICS segments are executed as part of the MULTICS initializer. It may happen that one of them is called but cannot perform its standard function during initialization.

The first idea for solving this problem is to take advantage of the segmentation facility by substituting to the called segment, a temporary segment, with the same name, which would perform an interim function F' instead of the standard function F. After initialization, the temporary segment would be replaced by the standard one. The segment structure would make this substitution easy but the fact that the hardcore supervisor is prelinked and the linkage sections combined makes the problem more complex. Therefore, the following solution has been chosen: Both functions F and F' are contained in the same segment; when the procedure is called it is able to switch to function F' during initialization and to function F after initialization. This implies that the code which executes F' will remain in the procedure after initialization even if it is not used any longer; but, as we shall see, the function F' is a simple function and requires a very small amount of code.

Block and Wake-up during initialization

During initialization Block and Wake-up perform the following interim functions:

Block calls the File System Device Monitor procedure and returns to the caller. Wake-up simply returns to the caller.

Block and Wake-up are written in such a way that they can execute in 2 modes. The mode is determined by the block_enable switch and the wake_up_enable switch in the Traffic Control Data Block. These switches can enable (standard function) or disable (interim function) the corresponding module.

The next paragraphs justify the choice for these interim functions executed by Block and wake-up during initialization.

Who calls Block and wake-up in MULTICS?

1. Block. During execution of a MULTICS process, block may be called by:

    1.1 Page control (page_fault), when the process is waiting for a page
    1.2 Segment control, when the process is waiting for a system data base to be unlocked
    1.3 The wait-coordinator, when the process is waiting for an event to occur
    1.4 The Quit interrupt handler.

2. Wake-up. This entry may be called by:

    2.1 Page control (io_done) when a page has arrived in core
    2.2 Segment control when a process unlocks a system data base that another process is waiting for
    2.3 The event channel manager (set_event) when an event has occurred
    2.4 The device signal table manager when a device sends an interrupt.

Who calls Block and wake-up during initialization?

Cases in which Block and wake-up may be called during initialization are 1.1, 2.1, and 2.4 (see previous paragraph).

Page faults are handled by the MULTICS mechanism; therefore Block will be called by page_fault(1.1) and wake-up will be called by io_done (2.1).

1.2 and 2.2 never happen during initialization since there is only one process using the system data bases.

1.3 and 2.3 never happen for the following reason: the only events (other than "page in core") that the MULTICS initializer has to wait for are completion of the following I/O operations:

Read tape	in the tape reader
Read_write drum	in fs_init_1
Read_write discs	in fs_init_1

The tape reader and fs_init_1 are executed in a one process environment; therefore there is no need for them to call the wait coordinator; they issue the connect operation and they loop until the status word shows that the operation is completed.

1.4 never happens since no Quit interrupt is generated.
2.4 happens each time an interrupt occurs since interrupts are handled by the standard interrupt handlers.

Why can't standard Block and wake-up be used?

One might think that, provided that the MULTICS initializer has a process id and an entry in the Active Process Table (APT), the system can be regarded, during initialization and as far as the process exchange is concerned, as a MULTICS system in which the number of active processes is 1.

But this is not correct; a one process system cannot constitute a regular system because MULTICS makes use of system processes.

Two system processes would be invoked if we used the standard Block and wake-up.

1. When the MULTICS initializer calls Block, get_work tries to find the next process to run in the ready list; but the ready list is empty. Therefore a system process, known as "idle" process, associated with the running processor is given to this processor.

2. When a device assigned to the basic file system sends an interrupt, the interrupt handler wakes-up a system process, known as the "File System Device Monitor" process.

These two processes do not exist during initialization; that is why standard versions of Block and wake-up cannot be used.

System processes simulation

Since during initialization there is only one process we have to find a way of simulating the two system processes mentioned above.

1. Idle process. The only purpose of this process is to keep the processor busy when there is nothing useful to do in the system. When Block is called in the MULTICS initializer, it is always because the process is waiting for a page. The simplest way of simulating the idle process is to keep looping in Block, testing the page table word that caused the missing page fault to occur, and to return from Block when the fault bits have been removed from the page table word. But this test already exists in page_fault: The page table word contains a software switch, the page out of service switch, which is set ON by get_page and reset by io_done. When Block returns to page_fault, this switch is checked: if ON (the page is not in core yet) page_fault calls Block again; if OFF (the page is now in core), page_fault returns to the fault interceptor.

   Therefore, the only function that Block has to perform in order to simulate the idle process, is to execute a RETURN.

   Since the MULTICS initializer is never Blocked, any call to wake-up for the MULTICS initializer can be a RETURN.

2. File System Device Monitor Process. In order to understand how this process is simulated during initialization it might be helpful to give a general description of the missing page fault handling.

   When a page fault occurs, page control is called; if this page has already been requested, page control calls Block; otherwise it determines on what device the page is stored, places the request in a job queue, and calls the Device Interface Module (DIM) associated with this device.

The DIM scans the job queue and places as many DCWs as possible in the hardware queue, for pages requested in the job queue. Then the DIM checks if any pages previously requested by other processes are now in core, even before the device sends an interrupt. For each of these pages, the DIM calls back page control, at the entry "io_done"; io_done builds the page table word, sends a wake-up to every process that is waiting for the page and returns to the DIM which deletes the request from the queue. Then the DIM returns to page control which calls Block.

Our process is now blocked. It can be awakened by two different ways:

a. If a running process takes a missing page fault which causes the DIM to be entered, the DIM, after having placed DCWs in the hardware queue, checks if some pages, previously requested, are now in core; if it finds our page, it calls io_done which sends a wake-up to our process.

b. If none of the running process causes the DIM to be activated before the device sends its interrupt, when the interrupt occurs the interrupt handler signals the event in the Device Signal Table (DST). Since this device is one device used by the file system, the interrupt handler sends a wake-up to the File System Device Monitor (FSDM) process. When awakened, the FSDM process executes the FSDM procedure. This procedure looks in the Device Signal Table, and, for each device available to the file system that has sent an interrupt, it calls the associated DIM which, as explained above, will cause our process to be awakened.

   Then it returns from block, page control returns to the fault interceptor which restores the processor state.

A simple way of simulating the FSDM process in the MULTICS initializer process would be the following: when wake-up is called with the process id of the FSDM process, since this process does not exist, wake-up performs the job that this process would do. But, the description given below for a page fault shows that, if we do so, because the FSDM

procedure is executed as part of the interrupt handler, interlock conflicts can arise in the DIM: when the DIM is executed as part of the FSDM process, it may find its queue locked because the interrupt occurred while the DIM was executing.

The only other place where the FSDM procedure can be executed is in Block. Wake-up, when called for the FSDM process, could set a flag signalling the fact that Block could test for executing the FSDM procedure. But this flag already exists in the Device Signal Table, and the FSDM procedure already contains this test in it.

Therefore, when wake-up is called for the FSDM process, it executes a RETURN. When Block is called, it executes the FSDM procedure and returns.

Conclusion

We can sum up this discussion as follows:

During initialization, Block is called only when waiting for a page; it does not block the process in the MULTICS sense, but loops on the page fault until the page table word is set with the address of the page. The function executed by Block during initialization is: call the FSDM procedure and return to the caller.

Wake-up is called by io_done and by interrupt handlers. When called by io_done, wake-up is only a return since the process is not blocked when waiting for a page. When called after interrupt associated with the file system, wake-up is also a return since the FSDM process is simulated in Block. When called after other interrupts, wake-up is also a return because the process is not Block but looping on a test of the status word. Thus: the function executed in any case by wake-up during initialization is merely a RETURN.