Published: 10/01/68
(Supersedes: BJ.5.02, 03/24/67)

Identification

Pre-emption
Robert L. Rappaport, Michael J. Spier

Purpose

As mentioned in section BJ.5.00, a running process may be interrupted and made to give its processor to a ready process which has a high-priority level number. We name this kind of interrupt a `pre-emption' interrupt and say that the running process has been pre-empted in behalf of the higher-priority ready process.

This section describes the implementation of pre-emption in Multics.

Introduction

The generalized rule for pre-emption is as follows:

> When a process is put on top of the ready list, if its level number is of higher-priority than the level number of a currently-running process, and if that process has run for at least as long as the high-priority process intends to run then the running process is pre-empted.

The scheduler (see BJ.5.01) computes the time allotment as a function of the process' current priority; the lesser the priority, the larger the time-allotment. A running process is pre-empted only if it has already run for at least as long as the higher-priority process' time allotment.

This is in order to insure that a low-priority process will never get into a situation where it spends most of its process-time thrashing around between the ready-state and pre-emption.

However, it is rather expensive to determine, whenever some process is put on the ready-list, what the priority of all the currently-running processes is, and how much is left of their time-allotment (one might have to stop them all and look into their timer-registers in order to detemine that).

For reasons of efficiency, a less generalized technique has been adopted which permits pre-emption to be carried out at discrete points of time only, but which can be implemented without too much overhead. In Multics, the timer-runout is a software-generated interrupt; the flipping over of the timer register actually causes a hardware fault which is intercepted by the Fault Interceptor (FIM). The FIM sets the processor's timer runout interrupt cell to "on", thus converting a fault into an interrupt.

The Traffic Controller takes advantage of this mechanism. Instead of loading a process' integral time-allotment into the timer registers, it loads only a `time-quantum' into that register. A time-quantum is a system-constant, and corresponds to the amount of time during which a process is allowed to run without risking pre-emption (a typical time-quantum could be 2 seconds). A process' time-allotment is thus used up in a series of time-quanta, and the timer-runout interrupt normally occurs when the last time-quantum of this current time-allotment has been exhausted. The FIM keeps count of the number of time-quanta that a process has used, and it also knows the number of time-quanta that any other process intends to run. So whenever a running process gets a timer-runout fault because its time-quantum has been used up, it goes into the FIM and there looks to see whether or not it still has time-quanta left. If not, it sends itself a timer-runout interrupt. If it does have time left, it checks to see whether or not the process on top of the ready-list has a higher-priority level number. If it does, and if that process intends to run for a number of time-quanta which is equal to or less than the number of time-quanta this process has already used up, then it pre-empts itself by sending itself a pre-emption interrupt. Else it loads a fresh time-quantum into its timer register and resumes its interrupted execution.

This scheme works fine for most processes. However, certain system processes cannot tolerate to wait for one whole time-quantum to pass before pre-emption becomes effective. Such system processes as the Device Manager Process or the File System Device Manager or the Traffic Controller System process, which all have the system's highest-priority level number (level 1, which no user process can ever have), must be able to instantaneously pre-empt any other running process. In order to emphasize this point, let us assume that a Tape Drive Device Manager does not have special powers of pre-emption. On a heavily-loaded system it would then conceivably read a magnetic tape at the speed of one record per time-quantum, which is rather slow.

On the other hand, all these level-1 processes are known to perform very fast computations, and are guaranteed to keep the processor for very short periods of time only. Therefore, whenever a level-1 process is put on the ready-list, the lowest-priority non-level-1 process on the running-list is automatically pre-empted, without regard to the number of time-quanta that it has used up.

We name the level-1 pre-emption "system pre-emption" and the non level-1 "user pre-emption".

System Pre-emption

The system pre-emption mechanism is invoked whenever a process is put on the ready list. It goes through the following steps (assume that process A is being put on the ready list):

1. If process A's level number is unequal to 1, return.

2. Find the running process with the lowest-priority (highest value) level number (let's name it process B).

3. If process B's level number is equal to 1, return.

4. Call pre-empt (processor) where the argument is the number of the processor on which process B is currently executing.

For reasons of efficiency (processes are very frequently put on the ready list), step 1 is executed in-line in the Traffic Controller primitive which threads processes into the ready list. Only when the ready process is known to have level 1 is a call made to the pre-emption module.

User Pre-emption

The user pre-emption machanism in invoked whenever a process has exhausted a time-quantum and is executing in the Timer-runout Fault Handler. The fault handler goes through the following steps:

1. Increments a count of `time-quanta-used'. This count is maintained in the process' PDS and indicates the number of time-quanta that the process has used up out of its current time-allotment.

2. Sees whether or not the process still has time left out of its time-allotment. Whenever a time-quantum is loaded into the timer-register, it is deducted from the process' time allotment (the last value loaded into the timer-register might therefore be less than a full time-quantum).

3. If no time available, it generates a timer-runout interrupt for this process. Return.

Following is the pre-emption mechanism:

4. Compare this process' level number to the level number of the top-most process on the ready list (a variable, named "highest_loaded", is maintained by the Traffic Controller and contains the level-number of the top-most ready process).

5. If this level number is of higher-priority than the level number of this process, and if the time-allotment for that level (found in array "level_coefficient" in segment tc_data) is lower-than or equal-to this process' time-quanta-used count then call pre-empt (processor) where the arguments is the processor number of this process (self pre-emption).

6. Else, load a time-quantum into the timer register, return.