Saltzer

DRAFT: 9/12/67 Comments
from Anonymous Evans.

Pre-emption

There are two issues concerning pre-emption and the scheduler: The first has to do with the policy issues of deciding under what circumstances pre-emption should take place. The second has to do with the mechanism by which the pre-emption actually transpires. We consider first the first.

In general, the scheduler performs a pre-emption when the process for which it is working seems enough greater in priority than a process which is currently running that the currently running process should be stopped in favor of the process for which the scheduler is working. In CTSS the issue is a good deal easier to resolve. There the criterion is as follows: The process being scheduled will pre-empt the process currently running if the process currently running has already executed for more time than the process being scheduled is asking for. For example, suppose we are about to schedule a process to run for 8 seconds. If the process currently running has only run for 7 seconds, no pre-emption will take place. However since the scheduler is invoked and polls on every quantum, one second from now the scheduler will observe that the current process has run for 8 seconds and that a process in a queue wishes 8 seconds. At that time, there will be a pre-emption.

This concept does not carry over very well into Multics. For one thing there is no polling, so the decision to pre-empt or not to pre-empt has to be made at the time the scheduler is first invoked. Secondly, the issue is more complicated because there will several running processes at any given time (as many as there are processors in the system). For example it is rather difficult for the scheduler to determine how long a particular process has been executing on a processor since it was last scheduled.

(Then could be) of the scheduler would it just set a short time limit and put him back in front) Second time?

this follows from second. you can find out how long the current process has run.

Since "time", as far as the scheduler is concerned, is measured in memory addresses, the only way it could get the desired information would be to know what value the processor's timer register had when initially loaded and what value it currently has. The former datum is available in the APT, but the latter datum could only be read from a register in another processor, an impossibility. An approximation to desired value could be achieved if the time at which the process began running were available in the APT. We could approximate by calculating microseconds rather than memory accesses. This possibility does exist since there is an APT entry called time_last_run. We could establish the convention that this entry would contain the time running began for all processes which are in running status. There seem to be serious race conditions in accessing this sort of data, however.

A convenient possibility for Multics, considering the data bases available, would be for a pre-emption to take place if the process being scheduled was to go into a queue higher than the queue number of a running process. Thus the scheduler working for a process which it was about to put into queue 4 would perform a pre-emption if there was a process running in queue 6. Question: Should it pre-empt a process in queue 5? The answer is probably no, although that is not clear.

A difficulty with this approach is that a process may be pre-empted after it has executed for only 10 or 15 microseconds. Doing so involves a rather high overhead.

We consider now the actual mechanism by which the pre-emption takes place, assuming that we know under what circumstances we wish to try to pre-empt. Actually we assume that pre-emption is done by a black box: The scheduler calls a suitable module which does the pre-emption. What we are concerned with here is what happens next. It should be understood that a process which has set interlocks on critical data bases (i.e., a process whose block_lock_count is non-zero) is to be scheduled with high priority. Suppose we are scheduling a process in queue 5, and observe that there is now a process running in queue 10. Clearly we pre-empt him. Unfortunately, when the target process of the pre-emption receives the pre-emption, it will call the scheduler from restart. Assume that the scheduler observes at this time that block_lock_count is non-zero. It will then schedule into queue 2 (say). Thus the pre-emption has been transmitted, but it did no good (for the processor into) queue 5. This sort of thing seems even more serious if there is another process in queue 7 that could have been pre-empted.

There are several possibilities. One would be for the scheduler to look at block_lock_count of the receiving process before sending the pre-emption. If this datum were non-zero, the scheduler would select another process for pre-emption. The difficulty here is the possibility of race conditions. The scheduler in one memory cycle could find block_lock_count to be zero, while it would be set non-zero in another cycle or two. Thus by the time the pre-emption was received, the target would have priority. It seems possible to obviate this problem with a suitable interlock scheme, although doing so seems rather complex. Another possibility is to ignore this problem entirely. Thus the target process in the example just given would indeed reschedule itself in queue 2, but it would do so for only a small burst. As soon as this burst were done, the process originally doing the pre-emption would presumably be at the top of the ready list and would run. Thusly,

the pre-emption would take longer than the sender had in mind, but the delay does not seem excessive. At the moment this approach seems to be the most plausible one to use.

And also no longer than the next possible situation, if there were no other potential pre-empters.

It seems to me that you have this area down solidly enough to go aboard and propose a complete scheduling algorithm.