Machine Project 2: Processor Scheduler

90 points + 10 bonus points
Due date: March 8, 1999

The simulated operating system currently uses a round robin scheduler. You are to modify it to be a version of multilevel feedback (MLF). In this version the priority of a process will determine which queue the process is to be placed upon when it requests time on the CPU. Each time the process waits (when a read or write request is issued)on the disk, increase its priority by two; each time it finishes its time slice, decrease its priority by one. Set the initial priority to 10 and never let it get above 20 or below 0.

Processes in the highest priority queue should get the shortest amount of time between time slice interrupts. Suppose the queues are numbered 0 to n where low priority processes are on queue 0. Then set the number of clock ticks you will give a dispatched process in queue k to $2^{(n-k)}$.

You are to also modify the scheduler to keep statistics on the processes. Keep track of how much CPU time each processes uses as well as how much time is spent waiting on the disk. Every 4000 time units, print out the time, whether or not the CPU is in use, the number of processes waiting on each queue (individually), and the accumulated statistics (including the priority) for each running process. Also print out a process' statistics upon termination along with the different averages; these will help in identifying bottlenecks in the system.

Be sure to protect against interference within the scheduler.

Print out the following statistics:

- CPU time per process

- Disk wait time per process

- Whether the CPU is in use or not

- Number of processes in each queue

- Priority of each process

The Assignment

Basic level.

You are to have four levels of queues in your MLF scheduler. To decide which priorities are assigned to which queue, you will need four cut-off priorities. Evaluate the effects on the performance of the system of using different cut-off points. Also determine where the bottlenecks are in the system. What would you change to improve performance?

(Hint: Read the cut-off points in at the startup of the program)

(Hint: Focus your attention to the files `schedule.H`, `scheduler.C`, `disk.H`, and `disk.C`. If you find yourself modifying code in many other files, you may be on the wrong track.)

Advanced level.

Same as the basic level, except for the following. Every 400 time units, artificially increase the processes' priority by moving them upwards one queue. This is called process aging.

How to proceed.

Implement the MLF scheduler as class MLFScheduler, which is a public subclass of class Scheduler. You will have to define or redefine the methods startup(), terminate(), dispatch (), and resume() to make the scheduler behave as a MLF scheduler (Take the class FIFOScheduler as an example to see how this can be done). The MLF scheduler is more complex than a FIFO scheduler, mainly because time-triggered preemptions can happen. This is not so difficult to handle. Whenever a process begins a CPU burst, initialize the maximum time it is allowed to execute (this is called the time quantum of the process). Before the CPU burst starts, set the CPU timer to expire at the end of the quantum (use the method set_Timer() in class CPU). If the burst ends before the timer expires, stop the timer, save the amount of quantum left for the process. If the timer expires during the CPU burst, the method handle_Timer_Interrupt() of the scheduler is called. Here, the MLF scheduler preempts the currently running process (by calling dispatch()), replenishes its quantum according to its priority, and puts the process back on the ready queue (by calling resume()).

What to Hand In

- Hand in hardcopies of all the source files that you have *modified*, and all of the source files that you have *created*.

- Clearly mark all new and/or changed code with highlighter. (If you change or add large sections of code, mark a line down the side of the page instead of line-by-line.)

- Also, hand in enough *output* from your simulation to convince us that your solutions are correct. DO NOT hand in the complete output – is is very lengthy. Just hand in enough to show that the system starts up and finishes correctly and that each aspect of your algorithm works correctly. Highlight the pertinent lines and jot down *brief* notes explaining why your output shows that your solution works. We will not be providing any sample output – we want you to be able to decide on your own if your output is correct.

- Finally, hand in an analysis of the effects on the performance of the system when using different cut-off points. Also determine where the bottlenecks are in the system. What would you change to improve performance? The complete analysis can be made in 500 words or less.

- Grading of these MPs is a very tedious chore. These handin instructions are meant to mitigate the difficulty of grading, and to ensure that the grader does not overlook any of your efforts.

- **Failure to follow the handing instructions will result in lost points.**