OS Project 1 B07902082 黃哲維

Design

After scanning in the inputs, the main starts scheduling the given processes. The policies given for the processes are determined and ordered in the scheduler. Non-preemptive has its own determination while the others work similarly.

The processes are given “x” amount of time quantums based on the scheduling policy and then it would call the next process. Each process would only finish when they reach their execution time limit.

FIFO is the quickest to implement and usually does not take as long as the other policies. The principle is:

- Sort all processes according to ready time from small to large.
- For each process, if its ready time is the current time, put the process in a queue.
- If no process is currently being executed, and there is a process in the queue, the process is taken out of the queue and executed until the execution of the process ends.
- The process in execution will be given a higher priotity.

For round robin, we have a timer for the given

- Use the timer to fix the cycle, let the executing process enter the queue after executing one cycle, and select the next executing process from the queue.
- Sort all processes according to ready time from small to large.
- For each process, if its ready time is the current time, put the process in a queue.
- If there is no process currently being executed, and there is a process in the queue, then take the process from the queue and execute it, and calculate whether the process will end in this execution cycle. If it is not over, reduce the remaining execution time by one cycle after executing one cycle, and put the process in the queue.
- The process in execution will be given a higher priotity like FIFO.

Shortest job first selects the process that has the shortest execution time. The principle is given as:

- Select the process with the shortest execution time to execute. The difficulty in practical application is how to estimate the execution time required by the process. The SJF applied here is non-preemptive, without interrupting the tasks that are still being executed, and there is also PSJF.
- Sort all processes according to the ready time from small to large. If the ready time is the same, then sort according to the execution time from small to large.
- For each process, if its ready time is the current time, the process is put into a heap, which is sorted according to execution time from small to large.
- If no process is currently being executed and there is a process in the heap, the process with the smallest execution time is taken out and executed until the execution of the process ends.
- The process in execution will be given a higher priotity.

Preemptive shortest job first operates as the process that has the shortest remaining execution time would be selected next. The principle goes as follows:

- Sort all processes according to the ready time from small to large. If the ready time is the same, then sort according to the execution time from small to large.
- For each process, if its ready time is the current time, the process is put into a heap, which is sorted according to the remaining execution time from small to large.
- If the remaining execution time of the current heap top process is smaller than the currently executing process, replace the currently executed process with the process taken from the heap, and put the originally executed process into the heap.

- If no process is currently executed and there is a process in the heap, the process with the smallest execution time is taken out for execution.
- The process in execution will be given a higher priotity.

Core Version

Linux zavien-virtual-machine 4.14.25 #6 SMP Tue Apr 28 19:52:59 CST 2020 x86_64 x86_64 x86_64 GNU/Linux

Actual VS. Theoretical Results

The actual results versus my own theoretical results differed in some areas which is seen in “dmesg” command within the terminal. Obviously the stdout result should be the same if my math calculations is on par. With the time given in dmesg from our system call using printk, the time was surprsingly a bit longer than I expected for a few such as for round robin. It is most likely due to do the number of context switches and just the sheer amount of constant number of times it needs to re-determine conditions. This grows exponentially as more processes and longer execution times are given. Preemptive shortest job first had similarities except it did not grow as exaggerately with greater number of processes and longer execution times.