CprE 308 Laboratory 5: Process Scheduling

Department of Electrical and Computer Engineering
Iowa State University

1 Submission

There is no report template for this lab. Instead, submit the following items:

- 10 pts - A PDF with a summary of what you learned in the lab session. This should be no more than two paragraphs. Include any particular details you found interesting or any problems you encountered
- 90 pts - The source code of your algorithm implementation (i.e. a completed scheduling.c)

2 Description

This project will allow you to explore three major scheduling algorithms: First-Come-First-Served, Shortest-Remaining-Time, Round Robin and Round Robin with Priority.

Download the scheduling.c from Canvas. This program contains a main function that will call four routines (one for each algorithm implementation you will write). You need to implement the scheduling algorithms described in Section 3. These functions are called at every timestep of the simulation to determine which simulated process runs next. The `process` structure is arranged as follows:

```
struct process {
    int arrivaltime;     /* Time process arrives and wishes to start */
    int starttime;       /* Time process actually starts*/
    int runtime;         /* Time process requires to complete job */
    int remainingtime;   /* Runtime remaining before process completes */
    int endtime;         /* Time process finishes */
    int priority;        /* Priority of the process */
    int running;         /* Convenience - indicates if the process is currently running */
    int finished;        /* Convenience - indicates if the process has finished */
};
```

The scheduling functions accept two parameters:

1. The array of `process` structures. You may use any of these values in your scheduling algorithm, but **you may not modify any of them**.
2. The current time of the simulation. You will need to use this to determine which processes are "ready", meaning they have arrived and they have not finished. (Processes do not block for I/O in this simulation.)

2.1 Output

The code in the main will print out the following information:

1. The start time of each process (the time instant when the process `first` runs, not each context switch)
2. The end time of each process (the time instant when the process's remaining time drops to zero)

After all simulated processes have finished for an algorithm, the average turn-around time (time from *arrivaltime* to *endtime*) is also printed. Sample output for a random seed `0xC0FFEE` is available on Canvas.

2.2 Grading

You will be graded based on the correct output for a set of random seeds. Focus on the correct output; you do not need to worry about efficiency of your implementation. The output for your Round Robin and Round Robin with Priority may be slightly different depending on your implementation. You will receive 10 points for code that compiles, runs, produces an output and terminates. You will receive 20 points for each successfully implemented algorithm.

3 Scheduling Algorithms

3.1 void first_come_first_served()

This algorithm simply chooses the process that arrives first and runs it to completion. If two or more processes have the same arrival time, the algorithm should choose the process that has the lowest index in the process array.

3.2 void least_remaining_time()

This algorithm will choose the program that has the least remaining execution time left, of those that are available to run. On a tie, choose the process with the lowest index in the process array.

3.3 void round_robin()

This algorithm cycles through the ready processes, running each for a 1-second quantum (i.e. a single simulation timestep) before switching to the next process.

3.4 void round_robin_priority()

This algorithm will be the same as the basic Round Robin algorithm except that it will also account for priority. Assume the largest value of the `priority` variable indicates the highest priority. For example, a process with priority level 1 will not run until all pending level 2 processes that have arrived finish, and level 0 processes will not run until all pending level 1 processes finish. Thus, the algorithm will run all higher priority processes to completion, each of them taking turns, before a lower priority process is scheduled. If a lower priority process has started, and a higher priority process arrives, the algorithm should immediately run the higher priority process.