

# CprE 308 Laboratory 5a: Process Scheduling

Department of Electrical and Computer Engineering  
Iowa State University

Fall 2014

## 1 Submission

Include the following in your lab report:

- Your full name and lab section in the upper right corner of the first page in large print.
- A summary comparing the scheduling algorithms and anything you learned in the project.
- Source code of your algorithm implementation along with the Makefile
- Output of one run of the program where `srand()` is seeded with the value `0xc0ffee`.

## 2 Description

This project will allow you to explore three major scheduling algorithms, First-Come-First-Served, Shortest-Remaining-Time, and Round Robin. You will also implement one modification to the Round Robin algorithm to include three levels of priority.

The file `scheduling.c` contains a main function that will call four routines (one for each algorithm implementation you will write) with the same array of process structures. The structure is arranged as follows:

```
struct process {
    /* Values initialized for each process */
    int arrivaltime; /* Time process arrives and wishes to start */
    int runtime; /* Time process requires to complete job */
    int priority; /* Priority of the process */
    /* Values algorithm may use to track processes */
    int starttime;
    int endtime;
    int remainingtime;
    int flag;
};
```

The first three values (`arrivaltime`, `runtime`, and `priority`) are inputs to the scheduling algorithm (in our simulation, these are set using random numbers), and cannot be modified by your implementation of the scheduling algorithm. The last four values (`starttime`, `endtime`, `flag`, and `remainingtime`) are available to the algorithm for storing information, and you can use this as “workspace” for your algorithm. All time values are given in seconds. You are to implement the four routines as follows:

## 2.1 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.

## 2.2 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. Again, on a tie, choose the process with the lowest index in the process array.

## 2.3 void round\_robin()

This algorithm tries to be fair in time allocation and will give each process that is available to run, an equal amount of time. To keep your implementation simple, do not create a queue of running processes that loops, just loop through the process array. When new process comes, it just becomes available. So that the process running sequence of this algorithm will be unique according to the process id. In this implementation assume that the scheduler can switch between processes every second.

## 2.4 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. In this context, 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 higher priority process will finish before the lower priority process will run again.

**Note** For each of the above algorithms assume the scheduler can switch every second.

## 2.5 Output

For each algorithm print the time each process starts, the time the process finishes and the average time between arrival and completion for all the processes. Don't print the process switching information every second. You'll get a bunch of output if you do so. A sample output for the First-Come-First-Served algorithm is given below.

```
Process arrival runtime priority
0 10 25 0
1 69 36 2
2 87 20 0
3 1 16 2
4 46 28 0
5 92 14 1
6 74 12 1
7 61 28 0
8 89 27 0
9 28 31 1
10 34 33 2
11 82 13 1
12 93 32 0
```

```
13 85 33 0
14 87 11 1
15 57 35 1
16 2 10 0
17 27 31 0
18 34 10 0
19 78 18 1
First come first served
Process 3 started at time 1
Process 3 finished at time 17
Process 16 started at time 17
Process 16 finished at time 27
Process 0 started at time 27
Process 0 finished at time 52
Process 17 started at time 52
Process 17 finished at time 83
Process 9 started at time 83
Process 9 finished at time 114
Process 10 started at time 114
Process 10 finished at time 147
Process 18 started at time 147
Process 18 finished at time 157
Process 4 started at time 157
Process 4 finished at time 185
Process 15 started at time 185
Process 15 finished at time 220
Process 7 started at time 220
Process 7 finished at time 248
Process 1 started at time 248
Process 1 finished at time 284
Process 6 started at time 284
Process 6 finished at time 296
Process 19 started at time 296
Process 19 finished at time 314
Process 11 started at time 314
Process 11 finished at time 327
Process 13 started at time 327
Process 13 finished at time 360
Process 2 started at time 360
Process 2 finished at time 380
Process 14 started at time 380
Process 14 finished at time 391
Process 8 started at time 391
Process 8 finished at time 418
Process 5 started at time 418
Process 5 finished at time 432
Process 12 started at time 432
Process 12 finished at time 464
Average time from arrival to finish is 189 seconds
```

## 2.6 Grading

Each algorithm weighs for 25 points (10 for the output, 15 for the code),