CprE 308 Laboratory 5: Process Scheduling

Department of Electrical and Computer Engineering
Iowa State University

1 Submission

Submit the following items via Canvas:

- A summary comparing the scheduling algorithms and anything you learned in the project. No more than 2 paragraphs.**(10 pts)**
- Source code of your algorithm implementation along with the Makefile **(90 pts)**

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.

Download file `scheduling.c` and read its main() function. Then fill in the code for the four following functions:

- `void first_come_first_served(process *proc);`
- `void shortest_remaining_time(process *proc);`
- `void round_robin(process *proc);`
- `void round_robin_priority(process *proc);`

The structure `process` is defined as follows:

```
typedef 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 (for algorithm 2.4)*/

    /* Values algorithm may use to track processes */
    int starttime;
    int endtime;
    int remainingtime;
    int flag;
} process;
```

The first three values (arrivaltime, runtime, and priority) are inputs to the scheduling algorithm (they are generated randomly), 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 functions as follows:

2.1 void first_come_first_served()

(20 pts) 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()

(20 pts) 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()

(20 pts) This algorithm tries to be fair in time allocation and will give each available process a time slot to run. In this implementation assume that the scheduler can switch between processes every second.

2.4 void round_robin_priority()

(30 pts) 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 HINT

In this assignment, your program is judged only by its validity of output, not by the time complexity. You don't need to implement complex data structures and sorting algorithms. To get you started, below is a pseudo-code template that can be used for all 4 of the algorithms.

NOTE: You are not required to use this code template. Feel free to code it your way.

```
int time = 1;  // time stamp
int finished = 0;  // number of finished processes
while (finished < NUM_PROCESSES) // while there are still unfinished processes
{
    /* TODO: loop through *proc and find out which process should run */

    // Assuming the process to run is at index "i" of proc array
    if (proc[i].starttime == 0)  // this is the first time this process gets to run
    {
       proc[i].starttime = time;
       proc[i].remainingtime = proc[i].runtime;
       printf("Process %d started at time %d\n", i, proc[i].starttime);
    }

    /*TODO: determine how many time slots to give this process
      HINT: in FCFS, the process should be given as much as it needed;
      in other algorithms, you can assign only one time slot and re-evaluate
      in the next iteration
    */

    // Assuming we are giving this process "t" times
    // Assign time to this process and update time
    proc[i].remainingtime -= t;
    time += t;

    // If this process finished, announce it
    if (proc[i].remainingtime==0)
    {
        proc[i].endtime = time; // mark the end time
        proc[i].flag = 1; // mark process as done
        finished++;
        printf("Process %d finished at time %d\n", i, proc[i].endtime);
    }
}

/* TODO: At the end, loop through *proc and calculate the average wait time.
   For each process, wait time = endtime - arrivaltime. */
```

2.6 Example 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. 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

--- Shortest remaining time
...
...

--- Round Robin
...
...

--- Round Robin with priority
...
...
```