CS441 Project 2
CPU Scheduling Simulator
CS 441/541 – Fall 2018

Project Available		Sept. 13
Component	**Points**	**Due Date (at 11:59 pm)**
Scheduler program	75	
Documentation	10	
Testing	10	
Style	5	
Total	100	Sept. 27
Submission	individual	Autolab

Objectives

The goal of this assignment is to simulate different scheduling algorithms. It also serves as simpler C program to write than future assignments to make sure you are prepared. Be sure to take the time to write quality software with good design, data structures, and functional decomposition, as well as tests and documentation. Tests are included in the assignment to check your work, as well as on Autolab to see how well your code performs.

Packaging & handing in your projects

You will turn in a single compressed archive of your completed project directory to the appropriate Autolab project. The compressed archive should be downloaded from the BitBucket website, which is a `.zip` archive. If, for some reason, you cannot download the source from the BitBucket website then turn in a manually compressed archive of the project as a zip file.

Deliverables:

Your project submission is required be written in a **consistent style** according to the **C Style Guide**. You will turn in a single compressed archive of your completed project directory to Autolab. Be sure to carefully review the entire specification in detail for the list of requirements.

○ Program must compile with the Makefile provided without warnings or errors.
○ Command line arguments fully supported, and proper error checking in place (e.g., if the user forgets to supply an integer value to the `-s` parameter).
○ Correct implementation of FCFS
○ Correct implementation of SJF
○ Correct implementation of Priority
○ Correct implementation of RR
○ Complete documentation & code styled according to the **C Style Guide**.
○ Set of 5 unique tests

Project Description

In this project, you will implement a CPU scheduling simulator. You will be reading from a file the following information: number of processes, process arrival order, CPU burst per process, and priority per process. Command line parameters will select which scheduling algorithm to use and the quantum value (if required).

The following command line arguments can occur in **any order** on the command line. All other command line options should be considered file inputs. Your program needs to only handle processing the first file encountered, but be aware that other command line parameters (e.g., `-s`) may occur after the file name.

- Required: `-s #`
  Scheduling algorithm to use identified by number. The table below describes the association of parameters to scheduling algorithms.

`-s 1`	First-Come, First-Served (FCFS)
`-s 2`	Shortest-Job First (SJF)
`-s 3`	Priority
`-s 4`	Round-Robin (RR)

- Required for RR only: `-q #`
  The quantum to use in the Round-Robin scheduling algorithm. This option is not required for the other scheduling algorithms, and, if provided for those algorithms, is ignored.

File Format

The file format provides all of the information for the processes that you will need to schedule. The first line of the file contains a single number identifying the number of processes in the file that you will be scheduling (`4` in the example below). The number of processes will range from 1 to an undefined upper bound (your program will need to handle a large number of processes). You may assume that the file is formatted correctly.

The subsequent lines in the file each describe one process indicating the process identifier, the CPU burst length, and the priority of the process. For example, the `4 3 7`, in the example below, indicates that process `4` has a CPU burst of `3` and priority `7`. A lower number means higher priority. You may assume that the process identifiers are integers, but do not assume that the process identifiers are contiguous starting at `1`. For example, you may be processing a file with the following set of five processes: `12, 24, 23, 103, 99`.

The order of the processes in the file is their arrival order. In the example below, the processes arrive in the following order: `4, 2, 3, 1`.

```
4
4 3 7
2 3 10
3 5 7
1 7 1
```

Example Output

After your program parses the command line arguments, your program will display these parameters. This should identify the scheduling algorithm by name and index.

After reading the test file, you will need to display the arrival order and process information. The process information displayed before running the scheduling algorithm will be the process identifier, the CPU burst length, and the priority of the process.

Your program will then run the appropriate scheduling algorithm keeping track of the waiting time, and turnaround time for each process. When the scheduling algorithm starts running your program should display the message `Running...`

Once the scheduling algorithm has finished with all of the processes, it will then display the following information (in the order below) for each process (in arrival order) with each line prefixed with the # symbol:

1. Process identifier
2. CPU burst length
3. Priority
4. Waiting time
5. Turnaround time.

Then your program will calculate and display (prefixed with the # symbol) the average waiting time, and average turnaround time. This information will be displayed after displaying information for each process.

Below are two examples:

```
shell$ ./scheduler -s 1 given-tests/level3.txt
Scheduler   :  1 FCFS
Quantum     :  0
Sch. File   : given-tests/level3.txt
-------------------------------
Arrival Order:  4,  2,  3,  1
Process Information:
 4        3       7
 2        3      10
 3        5       7
 1        7       1
-------------------------------
Running...
##################################################
#  #    CPU     Pri       W       T
#  4      3       7       0       3
#  2      3      10       3       6
#  3      5       7       6      11
#  1      7       1      11      18
##################################################
# Avg. Waiting Time   :   5.00
# Avg. Turnaround Time:   9.50
##################################################
shell$ ./scheduler -q 3 test.txt -s 4
Scheduler   :  4 RR
Quantum     :  3
Sch. File   : given-tests/level3.txt
-------------------------------
```

```
Arrival Order:  4,  2,  3,  1
Process Information:
 4        3       7
 2        3      10
 3        5       7
 1        7       1
-------------------------------
Running...
##################################################
#  #    CPU     Pri       W       T
#  4      3       7       0       3
#  2      3      10       3       6
#  3      5       7       9      14
#  1      7       1      11      18
##################################################
# Avg. Waiting Time   :   5.75
# Avg. Turnaround Time:  10.25
##################################################
```

Testing

Testing is an important part of the software development life cycle. You are expected to create a set of **at least five (5)** input files for your program each testing various aspects of the project. These are in addition to, and distinct from any test input files provided by the instructor. These files must be placed in a **tests** directory under the project directory. Each input file should be clearly labeled, and its usage should be described in the documentation.

These tests must be distinct from the tests provided in the `given-tests` directory. **Do not add to or modify** the contents of the `given-tests` directory as you changes **will not** be preserved in grading.

You can run the given tests at any time by following the directions displayed when you run the `make check` command.

```
shell$ make check
By default check will run all levels.
You can run an individual level by using one of the following:
   check-level-1
   check-level-2
   check-level-3
   check-level-4
   check-level-5
   check-level-6
   check-level-7


mmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm
mmmmmmmmm Level 1 Tests mmmmmmmmmmmm
mmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm
----------------------------------------------------------------------
Running the command:
        shell$ ./scheduler -s 1 given-tests/level1.txt
```

...
