Programming Project 03

CPU Scheduling Simulator

CS 441/541 – Spring 2014

Due Dates & Grading

This project is worth 100 points total with a small number of bonus points for those that work in groups. Grades will be assessed as follows:

Project Available		March 24
Component	**Points**	**Due Date**
Notification of Group Participation		March 26
Project 3		April 7
Program Correctness	85	
Style & Documentation	10	
Testing	5	
Partner Evaluation	+3	
Project Total	100	

Late assignments will be subject to the late policy in the syllabus. Your assignment must be written in C (not C++). No credit will be given for assignments written in any programming language other than C. Your project must be able to be compiled using a Makefile by typing the command `make` without any additional arguments in the assignment directory.

For this project you have the option to work in groups of at least two (2) and at most three (3) people. You must notify the professor by the date indicated above if you intend to work in a group for this project. For those that choose to work in groups there is a total of **3 bonus points** available to you. If you wish to be randomly assigned a group let the professor know before the notification deadline.

Project Overview

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, as needed.

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 has no effect and is not required for the other scheduling algorithms.

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).

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 for each process (in order):

- Process identifier
- CPU burst length
- Priority
- Waiting time
- Turnaround time.

Then your program will calculate and display 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 test.txt
Scheduler   :  1 FCFS
Quantum     :  0
Sch. File   : test.txt
--------------------------------
Arrival Order:  4,  2,  3,  1
Process Information:
 4        3       7
 2        3      10
 3        5       7
 1        7       1
--------------------------------
Running...
--------------------------------
 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   : test.txt
--------------------------------
Arrival Order:  4,  2,  3,  1
Process Information:
 4        3       7
 2        3      10
 3        5       7
 1        7       1
--------------------------------
Running...
--------------------------------
 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
--------------------------------
```

Style & Documentation

Your assignment should be coded in a **consistent style** according to the **Style Requirements** handout.

Documentation is a critical piece of software development. Unfortunately, it is often left to the last minute and forgotten. In this assignment (and all of the assignments in this course) you will provide a single document containing at least the following pieces of information (each identified in the documentation as separate sections):

- Author(s), date
- Brief summary of the software
- How to build the software
- How to use the software
- Examples
- How you tested your software.
- How to use the included test suite – Make sure to identify the purpose of each test.
- Known bugs and problem areas

Documentation should be in the form of a plain text or PDF document in the base of the assignment directory submitted. If you choose to create a PDF, it does not matter with what software you choose as long as it is easily readable. If you choose to create a plain text document then it must be named `README`, and is cleanly formatted so that it is easily readable.

If you work in a group, then each group member must *individually* complete a group evaluation using the appropriate Quiz on D2L by the assignment due date to receive the extra credit points. The evaluation will contain the following information (all information provided is private and will only be reviewed by the professor):

- You name, your partner's name, date.
- Summarize your contribution to the group.
- Summarize your partner's contribution to the group.
- Any additional comments you have regarding the group.

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 shell each testing various aspects of the assignment. These are in addition to any test input files provided by the instructor. These files should be placed in a **tests** directory under the assignment directory. Each input file should be clearly labeled, and its usage should be described in the documentation.

Useful Documentation & Hints

General advise on completing this assignment Start early, and ask questions.

Defensive programming is an important concept in operating systems: an OS cannot simply fail when it encounters an error, therefore it must check all input parameters before it trusts them. In general, there should be no circumstances in which your C program will core dump, hang indefinitely, or prematurely terminate. Therefore, your program must respond to all input in a reasonable manner; by "reasonable", we mean print an understandable error message and either continue processing or exit, depending upon the situation. It is strongly recommended that you check the return code of all system and library function calls as this will often catch errors in how you are invoking them.

Test, test and test some more! The instructor will be extensively testing your assignment, so you should too.

You may find the following documentation useful in writing your program:

- From the **stdlib.h** library:
  - **strtol()**: Convert a string to an integer.
  - **strtod()**: Convert a string to a float.
- From the **string.h** library:
  - **strlen()**: Length of a string.
  - **strdup()**: Duplicate a string
  - **strncmp()**: Compare two strings (See *man strncmp*).
  - **strtok()**: String tokenization (See *man strtok*).

Packaging & handing in your assignment

You will turn in a single compressed archive (either bzip/gzip'ed tar file, or zip file) of your completed assignment directory to the D2L Dropbox for each portion of the assignment. The compressed archive file should reference both your name, the assignment, and part of the assignment (e.g., **project3-jjhursey.tar.bz2**). The assignment directory should contain the following items:

- Documentation (either plain text or PDF)
- A testing directory (i.e., `tests`) containing *at least* 5 unique input files.
- A `Makefile` to build the software
- All of the necessary source files to compile your software