Operating Systems (CS:3620)

Assignment 3 (Implementing Scheduling Algorithms) Total points: 100

Due: 16 days (by 11:59 pm of October 23 (Sunday), 2016)

This assignment will contribute 10% to your final grades.

You can participate in a group of maximum two members.

You can maintain the same group from the previous assignment or form a new group.

Objective: The objective of this assignment is to implement three process scheduling algorithms that we have seen in class, namely, `First Come First Serve (FCFS)`, `Shortest Job First` (SJF), and `Shortest Time-to-Completion First` (STCF).

Your implementation of an algorithm should take as input the name of a job description file (described below) as a command line argument, run the respective scheduling algorithm on the jobs given by the job description file, and calculate the average response time and average turnaround time.

Input—Job Description File: The job description file has the following format. *You can assume the input files will be well formed.* The first line of the job description file will be an unsigned integer N where the value of N respects $1 \leq N \leq 100$. N here refers to the total number of jobs your program should consider. Then description of N jobs will follow. The description of each job starts with an unsigned integer arrivalTime referring to the job's arrival time where arrivalTime respects $0 \leq \text{arrivalTime} \leq 100$. The next line of the job description contains an unsigned integer executionTime referring to the total time the job requires to complete. executionTime respects the following constraint $1 \leq \text{executionTime} \leq 100$. Following is an example job description file. Comments such as "——> Number of jobs" are added for clarity. The job description file you have to work with will not have such comments.

```
2   ————> Number of jobs

0   ————> Arrival time of job 1
100 ————> Execution time of job 1

10  ————> Arrival time of job 2
10  ————> Execution time of job 2
```

Output generated by your program: Each time one of your program is executed, it will produce two lines of output in the standard output (**STDOUT**). The first line containing "f_1" where f_1 is the average turnaround time as a fractional number followed by a newline. The second line containing "f_2" where f_2 is the average response time as a fractional number followed by a newline. In both cases, you will print 5 decimal digits for the fractional part. The following is a sample output. "\n" is added to the sample output for clarity. In the output, this will be replaced by a real new line. Comments such as "——> Average turnaround time" are added for clarity. Your program's output should not have such comments.

```
6.66667\n ————> Average turnaround time
0.00000\n ————> Average response time
```

Submission: Please submit your source code through ICON. Your submissions should be compliant with the following directory structure. If you are doing this assignment individually, please follow the individual submission guideline. In case, you are doing this assignment as a group, please follow the group submission guideline.

Individual Submission: The top level directory of your submission should be named ⟨last-name⟩-⟨first-name⟩-Assignment-3. If a student's name is Bob Marley and he were to submit the assignment, his top level folder name will have the name `Marley-Bob-Assignment-3`. The top level directory should contain three directories, namely, `FCFS`, `SJF`, and `STCF`. Each of these directories should contain the necessary source code for the respective scheduling algorithms. The top level directory should also contain a file named `PL` which is described in the programming language section below. You will then zip your top-level directory (e.g., `Marley-Bob-Assignment-3.zip`) and submit them.

Submission for groups of 2 students: The top level directory of your submission should be named ⟨last-name$_1$⟩-⟨last-name$_2$⟩-Assignment-3. If a group have two students, namely, `Bob Marley` and `Omar Chowdhury`, their top level folder name will have the name `Marley-Chowdhury-Assignment-3` or `Chowdhury-Marley-Assignment-3`. The top level directory should contain three directories, namely, `FCFS`, `SJF`, and `STCF`. Each of these directories should contain the necessary source code for the respective scheduling algorithms. The top level directory should also contain

a file named `Members` which will contain the full names of the group members, and their responsibilities for solving this assignment. The top level directory should also contain a file named `PL` which is described in the programming language section below. You will then zip your top-level directory (e.g., `Marley-Chowdhury-Assignment-3.zip`) and submit them.

Programming Language: You are free to use any programming language for this assignment (e.g., C, C++, Java, Assembly, Python). The list of programming languages provided here are just for demonstration, you are free to use any other language **not** in this list. **We *recommend* students to not use .NET or C#.**

1. Please include a file named `PL` in the top level folder in your submission where you mention which programming language and compiler version (e.g., `gcc (GCC) 5.3.1 20160406`) you or your group used to write this assignment.
2. Please also explicitly mention, in the same `PL` file, the **command** to compile your/your group's source code (e.g., `gcc FCFS.c -o fcs -O3 -Wall`). *Please do not say, e.g., to compile press F5 in Microsoft Visual Studio.* We may not have access to the specific IDE. Give the command to run it in a shell. For example, you can say `javac FCFS.java && java FCFS job.description` or `python FCFS.py job.description`.

Cheating and Collaboration: *This is a group exercise, you can discuss with other groups but cannot copy their source code. Please do not copy source code from Internet.* The scheduling algorithm implementations that can be found in the Internet have a lot of subtle bugs which can be used to fingerprint those implementations.

PLEASE READ—Assumptions, Hints, and Pitfalls:

- **Assumption—Different Start Times:** We assume for the FCFS, SJF, and STCF algorithms, jobs may have different start times.
- **Assumption—No I/O:** We assume for the FCFS, SJF, and STCF algorithms, jobs do not perform any I/O. Jobs are only CPU-bound.
- **Assumption—Different Execution Times:** We assume for the FCFS, SJF, and STCF algorithms, each job may have different execution time.
- **Assumption—Known Execution Times:** We assume for the FCFS, SJF, and STCF algorithms, we know each job's execution time.
- **Assumption—Preemptive & Non-preemptive:** FCFS and SJF work as a non-preemptive scheduler, i.e., once a job is scheduled, it is run to completion. STCF scheduling algorithm, however, works a preemptive scheduler, i.e., it may interrupt a running job to schedule another job.
- **Instantaneous job start:** Let us consider we have a job that arrives at time 0 and runs for 5 time units. Then there is another job that arrives at time 5 and runs for 10 time units. We can run the first job from time 0 to time 5. The next job arriving at time 5 can be instantaneously started at time 5. We do not have to start the second job at time 6.
- **Stable Sorting Jobs:** It is a good idea to first **stable**[1] sort the jobs in the ascending order of their arrival times. This is particularly helpful for satisfying the following breaking tie requirements of `FCFS`, `SJF`, and `STCF` algorithms. Note that, built-in sorting functions may not be stable. Please consult the documentation of the function you are using to check whether the sorting function is indeed stable.
- **Breaking Ties (Removing non-determinism):**
  1. For the `FCFS` algorithm, if you have two (or, more) jobs with the same arrival time, pick the job that appeared earlier in the input job description file. If you do not, then the results may vary. Let us assume you have two jobs A and B with the same arrival time 0. A runs for 5 time units and B runs for 10 time units. If you first execute A and then execute B, the average response time will be $\frac{0+5}{2} = 2.5$ whereas if you run B first then A, then the average response time will be $\frac{0+10}{2} = 5$. To make it easy for us to grade, we break this tie by making sure that if A appears before B in the input file then you will run A first then B in such a case.
  2. For the `SJF` algorithm, if you have two (or, more) jobs with the same arrival time and the same minimum execution time, you can pick to execute the jobs in any order. However, if you have two (or, more) jobs with the same minimum execution time but different arrival times, then pick the job with the minimum arrival time to execute first.

[1] `https://en.wikipedia.org/wiki/Category:Stable_sorts`

3. For the `STCF` algorithm, if you have two (or, more) jobs that have the same arrival times and the minimum remaining times, then you can choose to execute them in any order. However, if you have two (or, more) jobs with the same minimum remaining execution times but different arrival times, then pick the job with the minimum arrival time to execute first.

- For the `STCF` algorithm, it may be a good idea to maintain a clock with 1 time unit resolution. You run a loop that executes until all jobs are finished and it increments the current time by 1 time unit. In every time unit (or, every iteration of the loop), you try to figure out which job to run. However, this is not necessary if you can figure out a smarter way.
- Please start testing your implementations with small test cases which you can verify by hand and then go on to test your implementations with complicated cases. Book has a fair number of examples with answers. You can use them to test your implementation.

Grading Rubric: As you may have realized, everything in this assignment is carefully designed so that if you follow the instructions, it should lead to only deterministic choices. Each of your implementations will be checked with 10 random test cases. You will get points of a particular test case if your program's output matches the instructor's program's output. You will be given the assembly files (`*.s` files) for the instructor's implementation. You can compile and execute the instructor's implementation in the following way.

```
g++ -c FCFS.s -o fcfs.o
g++ fcfs.o -o fcfs
./fcfs job.info
```

Questions

1. (30 points) **Implementing First Come First Serve (FCFS) Scheduling Algorithm:** For this problem, you will implement the FCFS non-preemptive scheduling algorithm. Your program should take a job description file name (described above) as input, then apply the FCFS algorithm on the jobs and will generate two lines of output in the standard output. The first line of output will contain the calculated average turnaround time and the second line will contain the average response time, for the input jobs according to FCFS algorithm. Note that, the input and output format have been explained above.
2. (30 points) **Implementing Shortest Job First (SJF) Scheduling Algorithm:** For this problem, you will implement the SJF non-preemptive algorithm. Your program should take a job description file name (described above) as input, then apply the SJF algorithm on the jobs and will generate two lines of output in the standard output. The first line of output will contain the calculated average turnaround time and the second line will contain the average response time, for the input jobs according to SJF algorithm. Note that, the input and output format have been explained above.
3. (40 points) **Implementing Shortest Time-to-Completion First (STCF) Scheduling Algorithm:** For this problem, you will implement the STCF preemptive scheduling algorithm. Your program should take a job description file name (described above) as input, then apply the STCF algorithm on the jobs and will generate two lines of output in the standard output. The first line of output will contain the calculated average turnaround time and the second line will contain the average response time, for the input jobs according to STCF algorithm. Note that, the input and output format have been explained above.