Shortest-Job-First Scheduling

Summary: In this homework, you will be implement a program that simulates the Shortest-Job-First Scheduling algorithms.

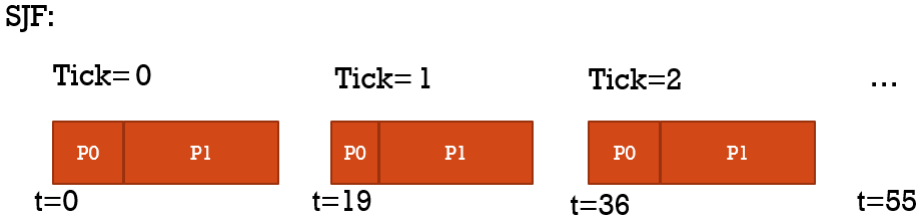
1 Background

In this assignment you will write a program that simulates using the shortest-job-first (SJF) algorithm(s) to pick which processes should be scheduled in what order. In the material, SJF is introduced as the basic idea that if we know how long each of the set of active processes that wants to run will take, then we can order them by their size, and then execute them in that order to minimize waiting and completion time. Later, this algorithm was extended with a live algorithm that predicted how long a process would take given it's previous history of execution.

This document is separated into four sections: Background, Requirements, Data File, and Submission. You have almost finished reading the Background section already. In Requirements, we will discuss what is expected of you in this homework. In Data File, we discuss the format of the simulation data. Lastly, Submission discusses how your source code should be submitted on BlackBoard.

2 Requirements [35 points]

In this assignment you will write a program that simulates scheduling processes. Your program needs to implement the SJF and SJF live algorithms. Your simulation should be divided into 'ticks' where each process is scheduled (based on t_n or τ_n values), and allowed to execute. The values of t_n represent the CPU burst times required by a process over it's lifetime. Note that a tick is different than the current time. A tick is simulating running a batch of processes, whereas time globally increases as each process in a batch runs. The first few ticks of the SJF on the processes given in Data File section are illustrated below:



For the live algorithm, you should rely on τ each iteration not $t_n$which is the actual time the process takes in that tick and a value unknown to the algorithm (it is trying to guess it!). At the end of each simulation, display the turnaround and waiting times. For the live algorithm, also compute the "estimation error" which is the sum of the absolute differences between each τ_n and t_n. When you calculate turnaround and waiting times, do so with respect to the beginning of the latest tick. That is, don't include waiting on the previous ticks since the process was executing at those times and is only waiting within the current tick. Sample output:

```
==Shortest-Job-First==
Simulating 0th tick of processes @ time 0:
  Process 0 took 6.
  Process 1 took 13.
Simulating 1th tick of processes @ time 19:
```

```
  Process 0 took 4.
  Process 1 took 13.
Simulating 2th tick of processes @ time 36:
  Process 0 took 6.
  Process 1 took 13.
Simulating 3th tick of processes @ time 55:
  Process 0 took 4.
  Process 1 took 13.
Simulating 4th tick of processes @ time 72:
  Process 1 took 6.
  Process 0 took 13.
Simulating 5th tick of processes @ time 91:
  Process 1 took 4.
  Process 0 took 13.
Simulating 6th tick of processes @ time 108:
  Process 1 took 6.
  Process 0 took 13.
Simulating 7th tick of processes @ time 127:
  Process 1 took 4.
  Process 0 took 13.
Turnaround time: 184
Waiting time: 40

==Shortest-Job-First Live==
Simulating 0th tick of processes @ time 0:
  Process 0 was estimated for 10 and took 6.
  Process 1 was estimated for 10 and took 13.
Simulating 1th tick of processes @ time 19:
  Process 0 was estimated for 8 and took 4.
  Process 1 was estimated for 11 and took 13.
Simulating 2th tick of processes @ time 36:
  Process 0 was estimated for 6 and took 6.
  Process 1 was estimated for 12 and took 13.
Simulating 3th tick of processes @ time 55:
  Process 0 was estimated for 6 and took 4.
  Process 1 was estimated for 12 and took 13.
Simulating 4th tick of processes @ time 72:
  Process 0 was estimated for 5 and took 13.
  Process 1 was estimated for 12 and took 6.
Simulating 5th tick of processes @ time 91:
  Process 0 was estimated for 9 and took 13.
  Process 1 was estimated for 9 and took 4.
Simulating 6th tick of processes @ time 108:
  Process 1 was estimated for 6 and took 6.
  Process 0 was estimated for 11 and took 13.
Simulating 7th tick of processes @ time 127:
  Process 1 was estimated for 6 and took 4.
  Process 0 was estimated for 12 and took 13.
Turnaround time: 200
Waiting time: 56
Estimation Error: 45
```

As an worked out example computation for SJF, turnaround time is computed as: (6+19) + (4+17) + (6+19) + (4+17) + (6+19) + (4+17) + (6+19) + (4+17)=184 and waiting time is: (0+6) + (0+4) + (0+6) + (0+4) + (0+6) + (0+4) + (0+6) + (0+4) = 40.

- Specific Requirements:
  - Read data on processes to simulate from local data file that is passed in as the first command line argument. [5 points]
  - Read data with dynamical memory allocation to support any number of processes and time steps. [8 points]
  - Simulate the shortest-job-first algorithm:
    * Simulates and displays each tick. [3 points]
    * Executes SJF algorithm on the set of processes. [4 points]
    * Displays turnaround and waiting times. [3 points]
  - Simulate the shortest-job-first live algorithm:
    * Simulates and displays each tick. [3 points]
    * Executes SJF live algorithm on the set of processes. [6 points]
    * Displays turnaround, waiting times, estimation error. [4 points]

3 Data File

Your program is required to read in a text file which contains a description of a set of processes which will be simulated. The first line lists the number of simulation "ticks", the second the number of processes. After that, processes are listed in sequence from 0 to the number indicated. Each process number is followed by a tau value, a alpha value, and data for actual run times equal to the number of simulation ticks. All numbers but alpha are integers.

In the case of the normal algorithm, you should ignore the τ and α information, and only use the $t_n$data that represents for a specific simulation tick, how much time that process wants. For the live algorithm, you should rely on τeach iteration, not $t_n$which is the actual time the process takes in that tick and a value unknown to the algorithm (it is trying to guess it). A sample data file is attached to this assignment (not the one we will test with) and an annotated version is shown below. **Be aware that the input file we use to grade will most likely use a different number of ticks and processes. You will need to use malloc to allocate the correct memory based on the parameters given in the file.**

```
8;ticks. see blackboard for the version of this file you need to support
2;process count
0;start of process 0
10;process 0 tau
.5;process 0 alpha
6;t_0 for process 0
4;t_1 for process 0
6;...
4
13
13
13
13;t_7 for process 0
1;start of process 1
10;process 1 tau
.5;process 1 alpha
13;t_0 for process 1
13;t_1 for process 1
13;...
13
6
```

```
4
6
4;t_7 for process 1
```

4 Submission

The submission for this assignment has one part: a source code submission. The file should be attached to the homework submission link on BlackBoard.

Writeup: For this assignment, no write up is required.

Source Code: Please name your main class as "LastNameSJFL.c" (e.g. "LastNameSJFL.c").

• If your program fails to compile out-of-the-box, there will be a mandatory deduction 20% from the assignment points.

• If you do not follow the file submission standards (e.g., the submission contains project files, lacks a proper header), there will be a mandatory deduction of 10% from the assignment points.

• If compiling or running your program differs from stated on the assignment and it is not specified as a requirement, please include a readme file with steps to execute the program. If you do not, there will be a mandatory 10% deduction from the assignment points.

Example readme

gcc SJFL.c -o SJFL
./SJFL data.txt