CSCI 4210 — Operating Systems
CSCI 6140 — Computer Operating Systems
Project 1 (document version 1.0)
Process Simulation Framework

Overview

- This project is due by 11:59:59 PM on Thursday, February 18, 2016. Projects are to be submitted electronically.
- This project will count as 8% of your final course grade.
- This project is to be completed **individually**. Do not share your code with anyone else.
- You **must** use one of the following programming languages: C, C++, Java, or Python.
- Your program **must** successfully compile and run on Ubuntu v14.04.3 LTS.
- Keep in mind that future projects will build on this initial project. Therefore, be sure your code is easily maintainable and extensible.

Project Specifications

In this first project, you will implement a rudimentary simulation of an operating system. The initial focus will be on processes, assumed to be resident in memory, waiting to use the CPU. Memory and the I/O subsystem will be covered in future projects.

Conceptual Design

A **process** is defined as a program in execution. For this assignment, processes are in one of the following three states: (a) ready to use the CPU; (b) actively using the CPU; and (c) blocked on I/O.

Processes in state (a) reside in a simple queue, i.e. the ready queue. This queue is ordered based on a configurable CPU scheduling algorithm. In this first assignment, there are two algorithms to implement, i.e., first-come-first-serve (FCFS) and shortest job first (SJF). And note that both algorithms will be applied to the same set of simulated processes.

Once a process reaches the front of the queue (and the CPU is free to accept a process), the given process enters state (b) and executes its current CPU burst.

After the CPU burst is completed, the process enters state (c), performing some sort of I/O operation (e.g., writing results to the terminal or a file, interacting with the user, etc.). Once the I/O operation completes, the process returns to state (a) and is added to the ready queue (its position within the queue is based on the CPU scheduling algorithm).

Simulation Configuration

The key to designing a useful simulation is to provide a number of configurable parameters. This allows you to simulate and tune a variety of scenarios (e.g., a large number of CPU-bound processes, multiple CPUs, etc.).

Therefore, define the following simulation parameters as tunable constants within your code:

- Define `n` as the number of processes to simulate. Note that this is determined via the input file described below.
- Define `m` as the number of processors (i.e., cores) available within the CPU (use a default value of `1`). Note that we will use this in a future project.
- Define `t_cs` as the time, in milliseconds, that it takes to perform a context switch (use a default value of `9`). Remember that a context switch occurs each time a process leaves the CPU and is replaced by another process.

Input File

The input file to your simulator specifies the processes to simulate. This input file is a simple text file that adheres to the following specifications:

- The file must be named `processes.txt`.
- Any line beginning with a `#` character is ignored (these lines are comments).
- All blank lines are also ignored.
- Each non-comment line specifies a single process by defining the process number, the CPU burst time, the number of bursts, and the I/O wait time; all fields are delimited by `|` (pipe) characters. Note that times are specified in milliseconds (ms) and that the I/O wait time is defined as the amount of time from the end of the CPU burst (i.e., before the context switch) to the end of the I/O operation.

An example input file is shown below.

```
# example simulator input file
#
# <proc-num>|<burst-time>|<num-burst>|<io-time>
1|168|5|287
2|385|1|0
4|97|5|2499
3|1770|2|822
```

In the above example, process `1` has a CPU burst time of `168ms`. Its burst will be executed `5` times, then the process will terminate. After each CPU burst is executed, the process is blocked on I/O for `287ms`. Process `2` has a CPU burst time of `385ms`, after which it will terminate (and therefore has no I/O to perform).

Your simulator must read this input file, adding processes to the queue based on the scheduling algorithm. For FCFS, processes are always initially added in process number order. Therefore, in the above example, processes will initially be on the queue in the order `1`, `2`, `3`, `4` (with process `1` at the front of the queue). For SJF, processes will initially be on the queue in the order `4`, `1`, `2`, `3`.

After you simulate both the FCFS and SJF algorithms, you must reset the simulation back to the initial set of processes and set your elapsed time back to zero. In short, we wish to compare these algorithms with one another given the same initial conditions.

Depending on the contents of the given input file, there may be times during your simulation in which the CPU is idle, because all processes are busy performing I/O. When all processes terminate, your simulation ends.

All "ties" are to be broken using process number order. As an example, if processes 2 and 5 happen to both finish with their I/O at the same time, process 2 wins this "tie" and is added to the ready queue before process 5.

Also, do not implement any additional logic for the I/O subsystem. In other words, there are no I/O queues to implement here in this first project.

CPU Burst, Turnaround, and Wait Times

CPU Burst Time: CPU burst times are to be measured for each process that you simulate. CPU burst time is defined as the amount of time a process is **actually** using the CPU. Therefore, this measure does not include context switch times.

Note that this measure can simply be calculated from the given input data.

Turnaround Time: Turnaround times are to be measured for each process that you simulate. Turnaround time is defined as the end-to-end time a process spends trying to complete a single CPU burst. More specifically, this is the arrival time in the ready queue through to when the CPU burst is completed. Therefore, this measure includes context switch times.

Wait Time: Wait times are to be measured for each process that you simulate. Wait time is defined as the amount of time a process spends waiting to use the CPU, which equates to the amount of time the given process is in the ready queue. Therefore, this measure does not include context switch times that the given process experiences (i.e., only measure the time the given process is actually in the ready queue).

Required Output

Your simulator should keep track of elapsed time `t` (measured in milliseconds), which is initially set to zero. As your simulation proceeds based on the input file, `t` advances to each "interesting" event that occurs, displaying a specific line of output describing each event.

Note that your simulator output should be entirely deterministic. To achieve this, your simulator must output each "interesting" event that occurs using the format shown below (note that the contents of the ready queue are shown for each event except for the end-of-simulation event).

```
time <t>ms: <event-details> [Q <queue-contents>]
```

The "interesting" events are:

- Start of simulation
- Process starts using the CPU
- Process finishes using the CPU (i.e., completes its CPU burst)
- Process starts performing I/O
- Process finishes performing I/O
- Process terminates (by finishing its last CPU burst)
- End of simulation

Given the example input file from page 2 and default configuration parameters, your simulator output would be as follows:

```
time 0ms: Simulator started for FCFS [Q 1 2 3 4]
time 9ms: P1 started using the CPU [Q 2 3 4]
time 177ms: P1 completed its CPU burst [Q 2 3 4]
time 177ms: P1 blocked on I/O [Q 2 3 4]
time 186ms: P2 started using the CPU [Q 3 4]
time 464ms: P1 completed I/O [Q 3 4 1]
time 571ms: P2 terminated [Q 3 4 1]
time 580ms: P3 started using the CPU [Q 4 1]
etc.
time <t>ms: Simulator ended for FCFS

time 0ms: Simulator started for SJF [Q 4 1 2 3]
time 9ms: P4 started using the CPU [Q 1 2 3]
time 106ms: P4 completed its CPU burst [Q 1 2 3]
time 106ms: P4 blocked on I/O [Q 1 2 3]
time 115ms: P1 started using the CPU [Q 2 3]
time 283ms: P1 completed its CPU burst [Q 2 3]
time 283ms: P1 blocked on I/O [Q 2 3]
time 292ms: P2 started using the CPU [Q 3]
etc.
time <t>ms: Simulator ended for SJF
```

Match the above example output **exactly** as shown above. Note that when your simulation ends, you must display that event as follows (and skip the very last context switch):

```
time <t>ms: Simulator ended for <algorithm>
```

Finally, in addition to the above output (which should simply be sent to `stdout`), generate a `simout.txt` file that contains statistics for each simulated algorithm. The file format is shown below (with `#` as a placeholder for actual numerical data). Round to exactly two digits after the decimal point for your averages. And note that averages are averaged over all executed CPU bursts.

```
Algorithm FCFS
-- average CPU burst time: ###.## ms
-- average wait time: ###.## ms
-- average turnaround time: ###.## ms
-- total number of context switches: ##
Algorithm SJF
-- average CPU burst time: ###.## ms
-- average wait time: ###.## ms
-- average turnaround time: ###.## ms
-- total number of context switches: ##
```

Submission Instructions

We will use the Homework Submission Server (HSS) for this assignment.