CSCI 4210 — Operating Systems
CSCI 6140 — Computer Operating Systems
Project 1 (document version 1.2)
Process Simulation Framework

Overview

- This project is due by 11:59:59 PM on Monday, September 14, 2015. Projects are to be submitted electronically.
- This project will count as 4% 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 or v15.04.
- 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 (or performing) I/O.

Processes in state (a) reside in a simple queue. Much like waiting in line at the grocery store, processes are serviced using a first-come-first-served (FCFS) algorithm (though this will be expanded in future projects).

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 CPU burst.

After the CPU burst is complete, 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 end of the queue.

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.).

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 `t_cs` as the time, in milliseconds, that it takes to perform a context switch (use a default value of `13`).

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 filename is `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 reads this input file, adding processes to the queue in the order shown (i.e., not necessarily in numerical order). Therefore, in the above example, the processes will initially be on the queue in the order `1`, `2`, `4`, `3` (with process `1` at the front of the queue).

Note that depending on the contents of this input file, there may be times during your simulation in which the CPU is idle, because all processes are either performing I/O or have terminated. 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.

Required Output

Your simulator should keep track of elapsed time `t` (measured in milliseconds), which is initially set to `0`. As your simulation proceeds based on the input file, `t` advances to each "interesting" event that occurs, displaying a 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.

```
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 the previous page and default configuration parameters, your simulator output would be as follows:

```
time 0ms: Simulator started [Q 1 2 4 3]
time 13ms: P1 started using the CPU [Q 2 4 3]
time 181ms: P1 completed its CPU burst [Q 2 4 3]
time 181ms: P1 performing I/O [Q 2 4 3]
time 194ms: P2 started using the CPU [Q 4 3]
time 468ms: P1 completed I/O [Q 4 3 1]
time 579ms: P2 terminated [Q 4 3 1]
time 592ms: P4 started using the CPU [Q 3 1]
etc.
```

Please match the above example output **exactly**, as testing and grading will be automated to the extent possible. Further, when your simulator ends, display that event as follows (and skip the very last context switch):

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

Submission Instructions

To submit your project, please create a single compressed and zipped file (using `tar` and `gzip`). Please include only source and documentation files (i.e., do not include executables or binary files!).

To package up your submission, use `tar` and `gzip` to create a compressed `tar` file using your RCS userid, as in `goldsd.tar.gz`, that contains your source files (e.g., `main.c` and `file2.c`); include a `readme.txt` file only if necessary.

Here's an example showing how to create this file:

```
bash$ tar cvf goldsd.tar main.c file2.c readme.txt
main.c
file2.c
readme.txt
bash$ gzip -9 goldsd.tar
```

Submit the resulting `goldsd.tar.gz` file via the corresponding project submission link available in LMS (`http://lms9.rpi.edu`). The link is in the Assignments section.