CSCI 4210 — Operating Systems
CSCI 6140 — Computer Operating Systems
Project 2 (document version 1.1)
CPU Scheduling Algorithms

Overview

- This project is due by 11:59:59 PM on Monday, October 5, 2015. Projects are to be submitted electronically.
- This project will count as 5% 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 (consider using `http://c9.io`).
- Keep in mind that future projects will continue to build on this project. Therefore, be sure your code is easily maintainable and extensible.

Project Specifications

In this second project, you will extend the first project to study CPU scheduling algorithms. In particular, you will simulate different scheduling algorithms with the same given set of processes, comparing and analyzing your simulation results.

As with the first project, processes are assumed to be resident in memory, either waiting to use the CPU or blocked on I/O. Memory and the I/O subsystem will be covered in more detail in future projects.

Conceptual Design

A **process** is defined as a program in execution. 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 queue. The scheduler selects processes from this queue to hand off to the dispatcher, which then performs the necessary context switching to get each process running with the CPU.

For this project, the scheduling algorithms are: first-come-first-served (FCFS); shortest remaining time (SRT); and priority with aging (PWA). FCFS was implemented in Project 1.

When you run your simulation, it must simulate all three of these scheduling algorithms.

Shortest Remaining Time (SRT)

The SRT algorithm selects the process with the lowest CPU burst time from among all processes in the queue. The queue is therefore often implemented as a priority queue ordered by CPU burst time. Note that this is essentially a priority scheduling algorithm in which the priority measure is based on CPU burst time.

The SRT algorithm is a preemptive algorithm, i.e., when process `A` arrives and is to be added to the queue, its CPU burst time is compared with the remaining time of currently running process `B`. If the burst time of `A` is less than that of `B`, then a preemption occurs—i.e., process `B` is forced to relinquish its time with the CPU (and is returned to the ready queue) such that process `A` uses the CPU immediately (after a context switch). In such a case, process `A` is not added to the queue.

Priority With Aging (PWA)

The PWA algorithm selects the process with the highest priority from among all processes in the queue. If more than one process exists at this highest priority level, then FCFS is used. For this project, use an integer in the range `[0,5]` to represent priority, with priority `0` as highest priority.

For this project, implement PWA as a preemptive scheduling algorithm. This means that when process `A` arrives and is to be added to the ready queue, if it has a higher priority than that of currently running process `B`, then a preemption occurs—i.e., process `A` preempts process `B`.

To avoid starvation (of lower-priority processes), processes that are in the ready queue for extended periods of time "age" by increasing their priority over time. For this project, if a process is waiting in the ready queue for more than three times its CPU burst time, increase its priority by `1`.

Simulation Configuration

As noted in Project 1, the key to designing a useful simulation is to make it easily tunable via a number of configuration parameters. This allows you to simulate and tune a variety of scenarios (e.g., a large number of CPU-bound processes, multiple CPUs, etc.).

As with Project 1, 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, the I/O wait time, and the priority; 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>|<priority>
#
1|168|5|287|2
2|385|1|0|3
4|97|5|2499|0
3|1770|2|822|2
```

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).

Note that the priority of each process is used only in the PWA algorithm.

Your simulator must read the input file, then, given these initial conditions, simulate the three separate scheduling algorithms (i.e., FCFS, SRT, and PWA). Note that after you simulate each of these 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.

In general, you must add processes to the queue based on the scheduling algorithm or in the process order shown. In the above example input file, processes for FCFS will initially be on the queue in the order `1`, `2`, `4`, `3` (with process `1` at the front of the queue). For PWA, the order will be `4`, `1`, `3`, `2`.

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.

As with Project 1, 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 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.

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.

Required Output

Your simulator should keep track of elapsed time `t` (measured in milliseconds) for each scheduling algorithm. Initially, `t` is set to `0`. As your simulation proceeds based on the input file and the scheduling algorithm, `t` advances to each "interesting" event that occurs, displaying a line of output describing each event.

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 (including what scheduling algorithm is being simulated)
- Process starts using the CPU
- Process is preempted
- Process 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 shown below. As with Project 1, when your simulator ends, display that event and skip the very last context switch.

```
time 0ms: Simulator started for FCFS [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]
 .
 .
 .
time 11125ms: P4 terminated [Q]
time 11125ms: Simulator for FCFS ended

time 0ms: Simulator started for SRT [Q 4 1 2 3]
time 13ms: P4 started using the CPU [Q 1 2 3]
time 110ms: P4 completed its CPU burst [Q 1 2 3]
time 110ms: P4 performing I/O [Q 1 2 3]
time 123ms: P1 started using the CPU [Q 2 3]
 .
 .
 .
time 1000ms: P3 preempted by P4 [Q ...]
time 1013ms: P4 started using the CPU [Q ... P3]
 .
 .
 .
time #####ms: Simulator for SRT ended

time 0ms: Simulator started for PWA [Q 4 1 3 2]
time 13ms: P4 started using the CPU [Q 1 3 2]
time 110ms: P4 completed its CPU burst [Q 1 2 3]
time 110ms: P4 performing I/O [Q 1 2 3]
time 123ms: P1 started using the CPU [Q 2 3]
 .
 .
 .
time 1000ms: P3 preempted by P4 [Q ...]
time 1013ms: P4 started using the CPU [Q ... P3]
 .
 .
 .
time #####ms: Simulator for PWA ended
```

In addition to the above output (which should be sent to `stdout`), generate a `simout.txt` file that contains statistics for each simulated algorithm. The file format is shown below (with `#` as placeholders for numerical data). Note that all 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 SRT
-- average CPU burst time: #.## ms
-- average wait time: #.## ms
-- average turnaround time: #.## ms
-- total number of context switches: #

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

Please match the above example output and file formats **exactly**, as testing and grading will be automated to the extent possible.

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.