CIS 3110: Operating Systems

Assignment 2: CPU Simulation

Due Date: Feb. 27, 2017

Overview

The goal of this assignment is to develop a CPU scheduling algorithm that will complete the execution of a group of multi-threaded processes in an OS that understands threads (kernel threads). Since a real implementation of your scheduling algorithm is not feasible, you will implement a simulation of your CPU scheduling algorithm. Each process will have 1-50 threads; each of the threads has its own CPU and I/O requirements. The simulated scheduling policy is on the thread level. While your simulation will have access to all the details on the processes that need to execute, your CPU scheduling algorithm can NOT take advantage of future knowledge.

Task

Given a set of processes to execute with CPU and I/O requirements, your CPU simulator should simulate the execution of the threads based on your developed CPU scheduling policies (say FCFS or RR). Your simulation should collect the following statistics: the total time required to execute all the threads in all the processes, the CPU utilization (NOT the CPU efficiency), the average turnaround time for all the processes, and the service time (or CPU time), I/O time, and turnaround time for each individual thread.

Your simulation structure should be a next event simulation. The next event approach to simulation is the most common simulation model. At any given time, the simulation is in a single state. The simulation state can only change at event times, where an event is defined as an occurrence that may change the state of the system. Events in this CPU simulation are the following: thread arrival and the transition of a thread state (e.g., when an interrupt occurs due to a time slice, the thread moves from running state to ready state).

Each event occurs at a specified time. Since the simulation state only changes at an event, the clock can be advanced to the next most recently scheduled event. (The meaning of *next event simulation model*.)

Events are scheduled via an event queue. The event queue is a sorted queue which contains "future" events; the queue is sorted by the time of these "future" events. The event queue is initialized to contain the arrival of all threads. The main loop of the simulation consists of processing the next event, perhaps adding more future events in the queue as a result, advancing the clock, and so on until all threads have terminated.

Simulation Execution

Your simulation program will be invoked as:

simcpu [**-d**] [**-v**] [-r quantum] **<** *input_file*

where **-d** stands for detailed information, **-v** stands for verbose mode and –r indicates Round Robin scheduling with the given quantum (an integer). You can assume only these flags will be used with your program, and that they will appear in the order listed. The output for the default mode (i.e., no flags) or with flags is defined in the output format section. Note, the flags may be used separately or together.

Input Format

The simulator input includes the number_of_processes that require execution, the thread_switch overhead time (i.e., the number of time units required to switch to a new thread in the SAME process), the process_switch overhead time (i.e., the number of time units required to switch to a new thread in a DIFFERENT process), and, for each process, the number_of_threads, the arrival time for each thread, and number_of_CPU, the number of CPU execution bursts each thread requires; the CPU execution bursts are separated by time the thread does I/O.

You should assume an infinite number of I/O devices. In other words, threads do not need to wait to do I/O. You should also assume that no overhead is associated with placing a thread on, or removing a thread from, an I/O device (e.g., no overhead is associated with an "interrupt" to indicate I/O completion). Since a thread must exit from the system while in the executing state, the last task of a thread is a CPU burst. All these simulation parameters are integers. Your simulator obtains these parameters from the input file, which is in the following format. You can assume the data in an input file is valid.

number_of_processes thread_switch process_switch

process_number(1) number_of_threads(1)

thread_number(1) arrival_time(1) number_of_CPU(1)
1 cpu_time io_time
2 cpu_time io_time
.
.
number_of_CPU(1) cpu_time
thread_number(2) arrival_time(2) number_of_CPU(2)
1 cpu_time io_time

```
  2   cpu_time  io_time
  .
  .
  number_of_CPU(2)   cpu_time
  ...
  ...
  thread_number(n) arrival_time(n)  number_of_CPU(n)
  1   cpu_time  io_time
  2   cpu_time  io_time
  .
  .
  number_of_CPU(n)   cpu_time

process_number(2)   number_of_threads(2)


  thread_number(1) arrival_time(1)  number_of_CPU(1)
  1   cpu_time  io_time
  2   cpu_time  io_time
  .
  .
  number_of_CPU(1)   cpu_time
  thread_number(2) arrival_time(2)  number_of_CPU(2)
  1   cpu_time  io_time
  2   cpu_time  io_time
  .
  .
  number_of_CPU(2)   cpu_time
  ...
  ...
  thread_number(n) arrival_time(n)  number_of_CPU(n)
  1   cpu_time  io_time
  2   cpu_time  io_time
  .
  .
  number_of_CPU(n)   cpu_time
  ........
  ........
  ........
```

The following is an example input file to the CPU simulation (there are no comments in a real input file) :

```
    2 3 7              // number_of_processes  thread_switch  process_switch
```

```
1 4              // process_number(1)   number_of_threads(1)
1 0 6            // thread_number(1) arrival_time(1)  number_of_CPU(1)
1 15 400         // 1   cpu_time  io_time
2 18 200         // 2   cpu_time  io_time
3 15 100         // 3   cpu_time  io_time
4 15 400         // 4   cpu_time  io_time
5 25 100         // 5   cpu_time  io_time
6 240            // 6   cpu_time
2 12 4           // thread_number(2) arrival_time(2)  number_of_CPU(2)
1 4 150
2 30 50
3 90 75
4 15
3 27 4
1 4 400
2 810 30
3 376 45
4 652
4 28 7
1 37 100
2 37 100
```

3 37 100

4 37 100

5 37 100

6 37 100

7 37

2 2

1 0 3

1 150 2

2 4 5

3 22

2 1 1

1 50

Output Format

In default mode (i.e., no flags are given), the simulator adopts FCFS scheduling policy, the output of the simulator consists of the total time required to execute the threads in all processes to completion, the average turnaround time of the processes, and the CPU utilization. As an example:

$simcpu < input_file

FCFS:

Total Time required is 344 time units
Average Turnaround Time is 27.9 time units
CPU Utilization is 94%

In detailed information mode (i.e., **-d** flag is given), the output of the simulation consists of the total time required to execute the threads in all the processes, the average turnaround time for all the processes, the CPU utilization, and the arrival time, service time, I/O time, turnaround time, and finish time for each thread.

$simcpu -d –r 10 < input_file

Round Robin (quantum = 10 time units):

Total Time required is 140 units
Average Turnaround Time is 9.2 time units
CPU Utilization is 100%

Thread 1 of Process 1:

arrival time: 0
service time: 32 units
I/O time: 3 units
turnaround time: 45 units
finish time: 45 units

Thread 2 of Process 1:

arrival time: 5
service time: 102 units
I/O time: 15 units
turnaround time: 135 units
finish time: 140 units

Thread 1 of Process 2:

arrival time: 10
service time: 22 units
I/O time: 3 units
turnaround time: 45 units
finish time: 55 units

In verbose mode, the output of the simulation includes the scheduling decisions and the switching of the threads. Specifically, the output includes every event that occurs, the time of the event, and the state transition:

At time X: Thread {id} of Process {id} moves from {state} to {state}

For example:

At time 0: Thread 1 of Process 1 moves from new to ready

At time 7: Thread 1 of Process 1 moves from ready to running

…

Each state transition should be given as above on a separate line. (Be sure to include events that occur at the same time.) Since we do not include swapping in our simulation, the state of a thread can only be in one of five states: new, ready, running, blocked, or terminated.

Submission

- If you have any problems in the development of your programs, contact the teaching assistant (TA) assigned to this course.
- You are encouraged to discuss this project with fellow students. However, you are **not** allowed to share code with any student.
- Each student should submit a brief report that explains your CPU scheduling algorithm and any assumptions you made. (If you make any assumption because you think the problem is not well specified, make sure to include those assumptions in the report.) Your report should also give answers to the following four questions. In your discussion, explain exactly how much overhead is included (if any).
    i. Does your simulator include switch overhead for the first ready state to running state transition? Explain.
    ii. Does your simulator include switch overhead if a thread moves from ready state to running state and the CPU is idle? Explain.
    iii. Does your simulator include switch overhead if a thread moves from running state to blocked state and the ready queue is empty? Explain.
    iv. Does your simulation include switch overhead if a thread is interrupted (due to a time slice) and either the ready queue is empty or the thread has the highest priority? Explain.
- If your TA is unable to run/test your program, you should present a demo arranged by your TA's request.