A Non-Preemptive Priority Scheduler

Out: 2/2
Due: 2/16 by 11:59 PM

Learning Objectives

- To implement a priority queue using the heap ADT.
- Manipulating an extensible array

CPU scheduling is the basis of multiprogramming. Whenever a computer CPU becomes idle, the operating system must select a process in the ready queue to be executed. One application of priority queues in operating systems is scheduling jobs on a CPU. In this project you write a program that schedules simulated CPU jobs. Your program should run in a loop, each iteration of which corresponds to a time slice, one CPU cycle. Each job is assigned a *priority*, which is a random integer between -20 (highest priority) and 19 (lowest priority), inclusive. From among all jobs waiting to be processed in a time slice, the CPU must work on a job with the highest priority. When two jobs have the same priority the one created earliest will be processed. In this simulation each job comes with a *length* value, which is a random integer between 1 and 100, inclusive, indicating the number of time slices that are needed to process this job. For simplicity, we will assume that the CPU is non-preemptive; that is, once a job is scheduled on the CPU, a job runs for a number of time slices equal to its length without being interrupted. Also, each process will have a process identification number 1 . . . n, where n is the number of simulated processes created.

First, you will implement a heap ADT. Then you will use the ADT in writing a simple CPU scheduling simulator. At time zero the ready queue is empty. You will enter as a command line argument p, a value between 0.01 and 1.00 inclusive, and s, the number of time slices the simulator will run. p is the probability that a new process is created during any given cycle. Your application will generate a random probability value q between 0 and 1 and for $q \leq p$ than a simulated job has been created. You will then set the relevant instance fields of the simulated jobs with appropriate values. Each process control block also has additional fields to facilitate our analysis: A PCB has an *arrived* field, the time slice during which the process was created, *start*, the time slice when the process begins running, *wait*, the length

of time from the process creation to when it begins running, and *running*, which is 0 if the process is waiting and 1 if the process is executing.

Your application should print the activities which occur during each time slice:

1. **** Cycle #: c **** at the beginning of each cycle.
2. Whenever the process's quantum expires, the message *Process # x has just terminated.* should be displayed.
3. If the process is still executing, the message *Process # x is executing.* should be displayed.
4. Whenever a new process is created the message *Adding job with pid # x and priority p and length t.* should be displayed.
5. If no new process is created during a cycle, the message *No new job this cycle.* should be displayed.

At the end of the simulation, your program should also display the average turnaround (number of processes per cycle) and the average wait time per process.

To run your simulation for 1000 time slices (cycles) where the probability that a new process is created during each cycle is 0.2, your program will be executed with these values as command line tokens. Be sure to seed the random number generator using time of day. Do so at the beginning of the *main*. See project files for details on assigned tasks.

Submitting Your Work

1. All source code files you submit must have header comments with the following:

```
/**
 * Describe what the purpose of this file
 * @author Programmer(s)
 * @since 9999-99-99
 * Course: CS3102.01
 * Programming Project #: 1
 * Instructor: Dr. Duncan
 * @see list of files that this file references
 */
```

2. Verify that your code has no syntax error and that it is ISO C++11 or JDK 8 compliant prior to uploading it to the drop box on Moodle. Be sure to provide missing documentation, where applicable. Also, add your name after the @author tag when you augment the starter code that I have provided. Put the last date modified in the header comments for each source code file.

3. Locate your source files -

   (a) for Java programmers, **HeapException.java**, **HeapAPI.java**, **Heap.java**, **PCB.java** and **CPUScheduler.java**

   (b) for C++ programmers, **Heap.h**, **Heap.cpp**, **PCB.h** and **CPUScheduler.cpp**

   - and enclose them in a zip file, YOURPAWSID_proj01.zip, and submit your programming project for grading using the digital drop box on the course Moodle.