A Non-Preemptive Priority Scheduler

Out: 9/8
Due: 9/21 by 11:59 PM

Learning Objectives

- To implement a priority queue ADT using the heap
- Manipulating an extensible array
- Simulating and Modeling a Queuing System

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 will write a program that schedules simulated CPU jobs on a single-core processor. 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 *burst* value, which is a random integer between 1 and 100, inclusive, indicating the number of time slices that are needed to complete 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 burst time without being interrupted. Also, each process will have a process identification number 1 . . . n, where n is the number of simulated processes created.

Some Basic Terms

Definition 1. **Arrival time** is the time when a process enters into a state in which it is ready for its execution.

Definition 2. **Burst time** is the total time required for a process to be executed on the CPU.

Definition 3. **Exit time** is the time when a process completes its execution.

Definition 4. Response time is the time spent by a process in the ready state until it gets the CPU for the first time.

Definition 5. **Waiting time** is the total time spent by the process in the ready state waiting for CPU. This may be different from the response time if the scheduling is non-preemptive. In non-preemptive scheduling a process may relinquish a CPU several times before its burst time is completed. In that case its waiting time will be the sum of all the times that it was in a ready state and not being executed. On the other hand, its response time will only be when it was in a ready state and gets the CPU for the first time.

Definition 6. Throughput is the number of processes executed by the CPU per a given unit of time.

Definition 7. **Turnaround time** is the total amount of time spent by a process from entering its ready state for the first time to its completion. That is, $Turnaround\,time = Burst\,time + Waiting\,time = Exit\,time - Arrival\,time.$

Modes of Operations of the Simulator

The simulator runs in two modes: *random*, denoted by the command line argument **-r** or **-R**, or *file*, denoted by the command line argument **-f** or **-F**. In either mode, the simulator runs with three command line arguments. In either mode, the first command argument is a positive integer representing the number of CPU cycles for the simulation. In random mode the remaining command line arguments are the -r or -R flag followed a positive real number less than or equal to 1.0 representing the probability that a process is created during a CPU cycle. In file mode the remaining command line arguments are the -f or -F flag followed by the file name of the file containing information about the simulated jobs. The main method/function of your program should display usage information and terminate the program if the number of command line tokens. It should also display usage information and exit the program if the flag used as a command line token is invalid.

Listing 1: Activities During a Cycle

```
A. if (readyq is empty)
      print idle CPU message
   else
      if (readyQ head process is not running)
         execute the head process
      endif
      if (readyQ head process is finished exeuting)
         terminate the process
         print process termination message
      else
         print process running message
      endif
   endif

B. if (process is created)
      add the process to the readyQ
      print process created message
   else
      print no process created message
   endif
```

First, you will implement a priority queue ADT. Then you will use the ADT in writing a simple CPU scheduling simulator. At time zero the ready queue is empty. As described above, to run the simulator in random mode you will enter as a third command line argument p, a value between 0.01 and 1.00 inclusive. Your application will generate a random probability value q between 0 and 1 and for $q \leq p$ a simulated job will be 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 your analysis: A PCB has an *arrived* field, the arrival time of the process, *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. Similarly, to run the simulator in file mode, you will enter a third command line argument representing the name of a text file containing information about the simulated jobs. Each line of this file will contain four integers. The first integer is the process identification number. The second number is an integer in $[-19, 20]$, the priority value of this process. The smaller this number, the higher the priority. The third number is an integer in $[1, c]$, where c is the number of cycles that the simulator will run. The third number represents the cycle during which the process was created. The last number is a positive integer that represents the burst time of the process. These values are used to set the relevant fields of the simulated jobs. In either mode, a simplifying assumption is that only one simulated job may be created during a cycle.

Your application should trace these activities during each time slice:

1. **** Cycle #: c **** at the beginning of each cycle.
2. If the ready queue is empty, the message *The CPU is idle* should be displayed.
3. Whenever a process is finished executing, the message *Process #* **pid** *has just terminated* should be displayed.
4. If a process is still executing, the message *Process #* **pid** *is executing.* should be displayed.
5. Whenever a new process is created the message *Adding job with pid # x and priority p and burst t.* should be displayed.
6. 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 number of processes created per cycle and the average turnaround time per process. It should also display 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*. I have also provided a sample input file, *simulatedjobs.txt* and its corresponding output. When you run the simulator in file mode using this file, the simulator should display the output shown in *simulatedjobsoutput.txt* on the screen. See starter code that I have provided for additional details. Modify the code only where indicated.

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)
 * @see LIST OF FILES THAT THIS FILE REFERENCES
 * </pre>
 * Date: TYPE THE DATE LAST MODIFIED
 * Course: CS3102
 * Programming Project #: 1
 * Instructor: Dr. Duncan
 * </pre>
 */
```

2. Verify that your code has no syntax error and that it is ISO C++ 11 or Java^{TM} *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, **PQueueAPI.java**, **PQueue.java**, **PCB.java** and **SingleCoreScheduler.java**

   (b) for C++ programmers, **PQueue.h**, **PQueue.cpp**, **PCB.h** and **SingleCoreScheduler.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.