CS433 Programming Assignment 1

Priority Queue of Processes (100 points)

Overall Goal

The goal of this assignment is to help you review C/C++ data structures and familiarize you with Unix/Linux programming. You should carefully read the programming policies and submission guidelines before starting.

Overview

In this assignment, you need to implement some commonly used data structures in operating systems, including:

- **PCB**: Process control block (PCB) is a data structure representing a process in the system. A process should have at least an ID and a state (i.e. NEW, READY, RUNNING, WAITING or TERMINATED). It may also have other attributes (*discussed more in later chapters, not needed for this assignment*), such as Program Counter, CPU Registers, scheduling information (e.g. priority), Memory-management information and I/O status information, etc.
- **PCB Table**: An array(list) of PCB elements. It contains all processes currently in the system.
- **ReadyQueue**: This is a queue of processes that are in the READY state to be scheduled to run. It needs to be a priority queue such that the process with the highest priority can be selected next. It should support at least the following member functions:
  - *add*: add the PCB of a process into the ready queue.
  - *removeHighest*: remove and return the PCB with the highest priority from the queue
  - *size*: return the number of elements in the queue
  - *display*: Display the IDs and priorities of processes in the queue.

  You have the freedom of choosing the data structure, for example, linked-list, array, binary tree or heap, used for implementing the **ReadyQueue**. However, you shouldn't directly use any existing "priority queue" data structure, for example, the one from STL. The choice of your implementation is critical for the performance of your program. In the report, you should discuss your choice of data structure, the time complexity of your implementation, and how the timing results match with your expectations.

Required Output

Read and follow the programming policies and submission guidelines.

You need to write a driver(main) program to test your data structures as follows. As a good program structure, the main program should be in a separate file from the classes.

1. Your program should always first print your names and a short description when it starts.
2. In the first test, you make a PCB_table of 30 entries with PCB ID from 1 to 30. Assume the priority values range from 1 to 50, where the higher value means the higher priority, i.e. priority = 1 means the lowest priority and 50 means the highest. Assume in this test the process of PID i has its initial priority = i, i.e. the lower PID means lower priority. When you add a process to the ReadyQueue you should change its state in the PCB_table to READY; you should change its state to RUNNING if you remove a process from the ReadyQueue. Create a ReadyQueue $q1$. Then do the following tests:
   1) add processes 15, 6, 23, 29 and 8 to $q1$. Display the content of $q1$
   2) remove the process with the highest priority from $q1$ and display $q1$.
   3) remove the process with the highest priority from $q1$ and display $q1$.
   4) add processes 3, 17, 22, 12, 19 to $q1$ and display $q1$.
   5) One by one remove the process with the highest priority from the queue $q1$ and display the queue after each removal.
3. The second test evaluates the performance of your implementation. Using a PCB_table of 30 PCB's with ID from 1 to 30 as in the first test but assign each process a random initial priority between 1 and 50. Initially randomly select 15 processes from the PCB_table and add them into $q1$. Then you need to repeat the following steps 1,000,000 times and measure the total time of running the loop.
   (a) With equal probability (50% chance), randomly decide either to remove a process from or insert a process to the ReadyQueue.
      a. If choosing to remove a process, remove the one with the highest priority from $q1$ using the removeHighest function. If there are more than one processes with the same highest priority, it's your design option to choose which one. For example, you may choose one randomly or the one that was inserted into the queue the earliest. Notice only processes currently in the queue can be removed. If $q1$ is empty, then no process should be removed. Again, the state of a process should be changed to RUNNING when removed from the ReadyQueue.
      b. If choosing to insert a process, select one process from the PCB_table that isn't currently in $q1$ and insert it into $q1$ using the add function. If all processes in the PCB_table are already in $q1$, no process should be added.

   Measure the total time of running the 1,000,000 iterations and print out the final content of the ReadyQueue (don't print in each of the iteration). Hint: You may use the C Time Library (http://www.cplusplus.com/reference/ctime/). The timing results of your program should be measured on the class server. You may run your program a few times and take the average. In the report, you should present and discuss your timing results. You also

need to print out the number of times of each PCB was added and removed from the ReadyQueue, for example

```
PCB ID 1 added 15787 times, removed 15787 times

PCB ID 2 added 14916 times, removed 14915 times

PCB ID 3 added 11743 times, removed 11743 times

PCB ID 4 added 14067 times, removed 14067 times

PCB ID 5 added 14980 times, removed 14980 times

PCB ID 6 added 16157 times, removed 16157 times

PCB ID 7 added 14579 times, removed 14579 times
```

Useful Things to Start

You must submit your source code and Makefile so that we can compile your program with the make command. Some starter code and a Makefile are provided under the directory "/cs433/example_code/assignments-starter-code/assign1" on the server to help you get started. You are not required to use the starter code for the programming assignments but can have your own design for the program. You should modify the makefile for your own project. If you used the starter code for your program, please state that in your report. **Your submission must be your own code and carefully read the academic honest policy in the syllabus.** If you used any other code from other resources, even from your previous classes, you must refer to it in your report as well.

If you are new to Linux/Unix programming environment and Makefile, please read the "Unix programming tools" doc and links to additional resources for more information. Use "-g" compilation flag for g++ when debugging your program and "-O2" or "-O3" to generate optimized code for timing in test 2. The group that implements test2 correctly and has the fastest time will get 10 points extra credit.