CS433 Programming Assignment 1

Priority Queue of Processes (100 points)

Overall Goal

This assignment aims 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 will 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. For this assignment, just think of PCB as a simple class with three attributes: an integer ID, an integer priority, 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.
- **ReadyQueue**: Ready queue is a queue of PCBs that are in the READY state, which are to be scheduled to run on CPU. It is a priority queue such that the process (PCB) with the highest priority can be selected next. It should support at least the following member functions:
  - ***addPCB***: add a PCB into the ready queue.
  - ***removePCB***: 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 the data structure for ReadyQueue implementation is critical for the performance of your program. In the report, you should discuss your choice of data structure, the expected time complexity of your implementation, and how the timing results match 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 will test the correctness of the ReadyQueue implementation by adding and removing some PCBs. Assume the priority values range from 1 to 50, where the higher value means the higher priority, i.e., priority = 1 represents the lowest priority and 50 means the highest. For simplicity, assume in this test the process of PID i has its initial priority = i; for example, process 1 is assumed to have a priority of 1. When a process is created, it is in the NEW state; when the process is added to the ReadyQueue, its state should be changed to READY; you should change its state to RUNNING when a process is removed from the ReadyQueue.

   Create an empty ReadyQueue $q1$. Then do the following tests:

   1) add processes 15, 6, 23, 39 and 8 to $q1$. Display the content of $q1$
   2) remove the process with the highest priority from $q1$ and display $q1$.
   3) add processes 47, 1, 37 and 5 into $q1$, and display $q1$.
   4) remove the process with the highest priority from $q1$ and display $q1$.
   5) add processes 43, 17, 32, 12 and 19 to $q1$ and display $q1$.
   6) 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. Create a ReadyQueue $q2$, and initially add 100 PCBs with IDs from 1 to 100 into $q2$, assigning each process a random priority between 1 and 50. 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 $q2$ 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 $q2$ 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, create a new process with a unique ID and insert it into $q2$ using the `add` function. You can assign an ID to the new process using an incremental counter to make sure that each PCB has a different ID; while the priority of a new process is set randomly between 1 and 50.

   Measure the total time of running the 1,000,000 iterations and print out the final content and the size of the ReadyQueue (don't print in each of the iterations). Hint: Measure execution time with high precision in C/C++. 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.

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 test 2 correctly and has the fastest time will get 10 points extra credit.