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 start.

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, 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 following operations:
  - *insertProc*: add the PCB of a process into the ready queue.
  - *removeHighestProc*: remove and return the PCB with the highest priority from the queue
  - *size*: return the number of elements in the queue
  - *displayQueue*: 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 Ready_Queue. However, you cannot directly use any exsiting "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. Your program should first print your name when it starts.

1. In the first test, you make a pcb_table of 20 entries with PCB ID from 1 to 20. Assume the priority values range from 1 to 50, where the lower value means the higher priority, i.e. priority = 1 means the highest prority and 50 means the lowest. Assume in this test the process of PID i have its initial priority = i, i.e. the lower PID means higher 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.
   (a) add processes 5, 1, 8, 11 to $q1$. Display the content of $q1$
   (b) remove the process with the highest priority from $q1$ and display $q1$.

   (c) remove the process with the highest priority from $q1$ and display $q1$
   (d) Insert processes 3, 7, 2, 12, 9 to $q1$
   (e) One by one remove the process with the highest priority from the queue $q1$ and display it after each removal.

2. The second test evaluates the performance of your implementation. Use the ReadyQueue $q1$ and the pcb_table from the first test. First randomly select 10 process from the pcb_table and add them into $q1$, but assign each process a random initial priority between 1 and 50. Then You need to repeat the following steps 1,000,000 times and measure the total time of running the loop.

   (a) Randomly decide to remove a process from or add a process to the ready queue with equal probability (50% chance).
      i. If choose to remove a process, remove the one with the highest priority from $q1$ using the *removeHighestProc* function. If there are more than one processes with the 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.
      ii. If choose to add a process, select one process from the pcb_table that isn't currently in $q1$ and insert it into $q1$ using the *insertProc* function, with a random initial priority value between 1 and 50. 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 UNIX `gettimeofday` function to measure the time. type "`man gettimeofday`" in a Unix shell to get more information or look it up online.

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 provide your source code and Makefile so that I can compile you program with the make command. I provide you a dummy "test.c" file and a Makefile for students who haven't used Makefile before. You can download them as a zip file from the class website or copy them form directory /cs433/example on the cs433.cs.csusm.edu server. You should modify the makefile for your own project. 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 "-O" or "-O2" to generate optimized code for timing test 2. The group that implements test2 correctly and has the fastest time will get 10 points extra credit.