Semester Project Part 4: A Preemptive Scheduling Simulation

Data Structures and Analysis of Algorithms, akk5

Objectives

- To strengthen student's knowledge of C++ programming
- To give student experience reading and parsing strings of commands
- To give student experience in writing a non-linear data structure
- To give the student experience implementing a Heap

Instructions

For this assignment you must write a program that simulates preemptive task scheduling. "In Preemptive Scheduling, the tasks are mostly assigned with their priorities. Sometimes it is important to run a task with a higher priority before another lower priority task, even if the lower priority task is still running. The lower priority task holds for some time and resumes when the higher priority task finishes its execution" (https://www.guru99.com/priority-scheduling-program.html).

The program should allow the user to schedule tasks and then simulate their order of execution on hypothetical operating system. Every task in this simulation has the following properties: a **name**, a **priority** ranging from 1 (the highest) to 100 (the lowest), a **length** measured in steps, the **elapsed** time the task has spent running, and the task's **arrival** time (the step on which the task is scheduled to begin).

The simulation will need to maintain a priority queue and a vector; both of which must store tasks. The priority queue stores only the tasks that have arrived and are waiting their turn to execute, the vector stores every task that has yet to be scheduled for execution. As the time in the simulation progresses, you will need to move tasks from the vector to the priority queue, from the priority queue into the simulation's active memory, and from the simulation's active memory into the priority queue.

The simulation performs that following actions at each time step:

1. Check to see if there are any tasks still running, waiting to be scheduled, or waiting to execute.
    a. If such a task does not exist, end the simulation.
2. Move all tasks scheduled to begin at this time step into the priority queue
3. If the currently running task has finished execution
    a. Set the running task to none
4. If there are tasks in the priority queue
    a. Peek at the next task on the priority queue
    b. If that tasks arrival time is less than or equal to the current time step
        i. Compare the next task's priority with the running task's priority
        ii. If the next task has a higher priority
            1. Remove the next task from the queue
            2. Add the running task to the queue
            3. Set the running task to the task removed from the queue
        iii. If there is no task currently running

    1. Remove the next task from the queue
    2. Set the running task to the task removed from the queue
5. If there is a running task
    a. Increment its elapsed time
6. Increment the time step by one.

Your program will need to function in two separate modes/phases: scheduling and simulation. During scheduling, your program will allow the user to create/register a list of tasks to be executed during simulation; during simulation, your program will allow the user to step through one or more increments of time and display the status of the task currently running.

Your program should implement a command prompt (text-based interface) capable of handling the following commands:

exit – exits the program

load *<file>* - parses the contents of the file as if they were entered from the command prompt.

register <task> <priority> <length> <arrives> – registers/adds the task with the given priority and length starting at time arrives to the scheduling vector

remove <task> – removes the specified task from the scheduling vector

clear – empties the scheduling vector of its current content

display – this command's behavior changes based on the program's mode. In scheduling mode, it displays a list of the tasks which have been scheduled; in simulation mode, it displays the active task, the list of tasks yet to be scheduled and the current time step.

simulate – enters simulation mode.

schedule – enters scheduling mode; this should clear the contents of the priority queue, remove the registered tasks and reset the simulation time.

step – move the simulation forward one(1) time unit and display the results

step <time> - move the simulation forward **time** time units and display the results

run - run the simulation until the last task has completed.

run til <time> - run the simulation until the specified **time** unit has been reached.

reset – resets the time to 0 and empties the priority queue.

Note: the **register, remove**, and **clear** commands should only function in schedule mode; the **step, reset** and **run** commands should only function in simulation mode; **exit**, **load**, **display**, **simulate**, and **schedule** should function in both modes.

Guidance

Make certain you implement a peek for your priority queue, while the simulation is running you will need to determine if the active task should be preempted by another task in the queue. The active task is preempted if the next task in the queue has a higher priority.

Remember to add the current task back into the priority queue if it has been preempted.

The proper handling of priority is key to the success of the simulation; a priority comparison is composed of three factors: 1. The priority of each task 2. The arrival time of each task 3. The elapsed time for each task; the simulation always favors the task with the highest priority, the lowest arrival time, and most elapsed time.

This is an individual assignment. Seeking direct help from students, tutors, and websites such as chegg or stack overflow will be construed as a violation of the honor code.

Grading Breakdown

Point Breakdown	
Structure	12 pts
The program has a header comment with the required information.	3 pts
The overall readability of the program.	3 pts
Program uses separate files for main and class definitions	3 pts
Program includes meaningful comments	3 pts
Syntax	18 pts
Implements the Heap class correctly	12 pts
Implements the Process class correctly	6 pts
Behavior	70 pts
Program handles all command inputs properly	
• The program performs all commands supported in scheduling mode	10 pts
• The program performs all commands supported in simulation mode	10 pts
• The program performs the all commands shared between modes.	10 pts
Program properly simulates Priority Scheduling	40 pts
Total Possible Points	**100pts**
Penalties	
Program does NOT compile	-100
Late up to 72 hrs	-10 / day
Late more than 72hrs	-100

Header Comment

At the top of each program, type in the following comment:

/*

Student Name: <student name>

Student NetID: <student NetID>

Compiler Used: <Visual Studio, GCC, etc.>

Program Description:

<Write a short description of the program.>

*/

Example:

/*

Student Name: John Smith

Student NetID: jjjs123

Compiler Used: Eclipse using MinGW

Program Description:

This program prints lots and lots of strings!!

*/

Assignment Information

Files Expected:

This project should require the implementation of one or more classes as well as the function necessary for parsing the various inputs. At a minimum I am expecting project.cpp, project.h, heap.cpp, heap.h, main.cpp; you might have more files depending upon your implementation.