SWE 501 Project: Discrete Event Simulation, Spring 2017

Due: 5 June 2017, 9:00

Introduction

First of all, what is a discrete event simulation?

A discrete-event simulation (DES) models the operation of a system as a discrete sequence of events in time. Each event occurs at a particular instant in time and marks a change of state in the system. Between consecutive events, no change in the system is assumed to occur; thus the simulation can directly jump in time from one event to the next (wikipedia). In other words, DES is the representation of a certain process by simulating certain events which occur at certain times. Hence in DES, time is not continuous and can be split into intervals according to the events. As a very simple example a day of a barber can be simulated like this:

- TIME: 09:00 Open the shop.
- TIME: 09:30 Wash the grounds.
- TIME: 11:20 Shave Baker Ali Abi.
- TIME: 12:30 Play backgammon with pharmacist.
- ⋮

Note that in DES the only thing that counts is events and everything else is ignored. In the case of the barber, till 12:31 only 4 things happened.

In this project, you are expected to simulate the two algorithms of task scheduling for the CPU.

Task scheduling is the concept of delivering processes(tasks,events) to CPU to be handled in an order, according to a certain rule. Let's think of the barber case again. Assume that two customer is waiting for shaving in the queue. First customer arrived early and it would take 1 hour to shave him -he is a hairy guy-, while the other one is a boy and shaving would take only 10 minutes. So the barber has to make a decision about who to shave first!

And this decision making is actually called as **task-scheduling**.

Description

For the case of this project there will be events that will be needed to processed by CPUs and output devices(sound, display etc). Since the number of events will be more than the number of CPUs and output devices, one needs task scheduling algorithms for both CPU and output devices -otherwise every task can be processed simultaneously-. So we can examine the project in two parts, First the CPU, second the output devices.

1 CPU part

The events that needs to be processed will first visit the CPU. However, since there will not be enough CPU for each event, the events should be distributed among the CPUs according to a rule: Shortest Job First Algorithm (will be explained below). If every CPU is occupied, the events will wait in a priority queue and once one of the CPU is freed the event that is popped from the queue will be processed. This procedure will continue until every event is processed and delivered to the output part. Note that you will be provided with the information of frequency of each CPU, each task's arrival time and time needed to process that specific task in the CPU.

2 Output part

This part is where the events will be processed after CPU part and presented to the user as an output -display,sound etc.- Just like in the CPU part, since there will be more tasks than the number of the output devices, the tasks will be scheduled according to a rule: Round Robin Algorithm(will be explained below). Likewise the CPU part, if the devices are free, the event will move into the output process, otherwise they will wait in the priority queue. Note that you will also be given the time needed to process an event in the output part.

3 Shortest Job First Algorithm

In this algorithm, the event that would take the least time to process, is processed first. In real world, it is nearly impossible to guess the time of process, but for our purposes every event will be processed certain amount time which will be given. Therefore, according to this algorithm, in the barber shop case, the kid will be shaved first, no matter when the hairy guy has arrived!

4 Round Robin Algorithm

In Round Robin, every task is processed for a specific time(quantum) then popped out of CPU even if it has not finished and pushed back to the priority queue. Since this will cause circular processing of tasks and it is a 'just' system, it is called Round Robin Algorithm. Hence if round robin is applied in the barber shop, the barber would shave the kid for a while (a quantum) then he would shave the hairy guy, then the kid again and so on.. (It is not a good algorithm for a barber I suppose.)

For further reading and comprehension of the task scheduling algorithms please visit this link and please see the the flow chart below for visualization of the system.

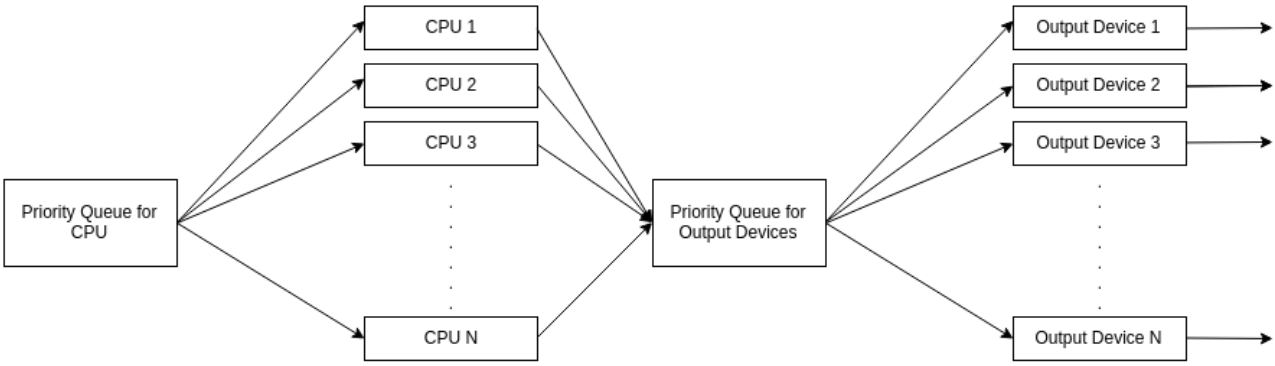

Figure 1: Flow chart of the system

To sum up, you are going to simulate a system that consists of CPU and output devices and keep track of certain statistics (explained in the Output section) during the simulation to print them as output.

Input&Output

Input

As input, you are going to have the number of the CPUs and output devices. The frequency of CPUs and quantum values of the output devices and number of tasks with details of the tasks in the following format:

- In the first line: An integer N (Number of CPUs).
- Next N lines contain doubles that represent frequency of the each CPU. (For a task, time to spend in the CPU can be calculated as: *CPU work of task / frequency of CPU*)
- Next line contains an integer, the number of output devices units (M).
- Next M lines contain a double, quantum value of each output unit.
- Next line contains an integer, the number of tasks to process
- Remaining lines contain 3 doubles, arrival time, CPU work and output device work of the task.

Output

And you are expected to output the statistics you kept track of each in a new line which should be:

1. Maximum length of first priority queue
2. Maximum length of second priority queue

3. ID of CPU which has the greatest active time
4. ID of output device which has the greatest active time (Break the ties by choosing the one with smaller ID. Note that IDs start from 1.)
5. Average wait time of all tasks (Wait time: Time passed in the queues as total.)
6. Longest wait time of all tasks
7. Average time a task spent in the system(It can be calculated as $Time_{exit} - Time_{arrival}$ for a single task.)

So an example the input file in the table can be visualized as in figure below.

Sample Input File	Sample Output File
2	2
1	2
2	2
2	1
5	6.625000
10	15.500000
4	32.500000
0 10 5	
1 22 20	
2 5 2	
3 3 50	

Table 1: Sample Input and Output files

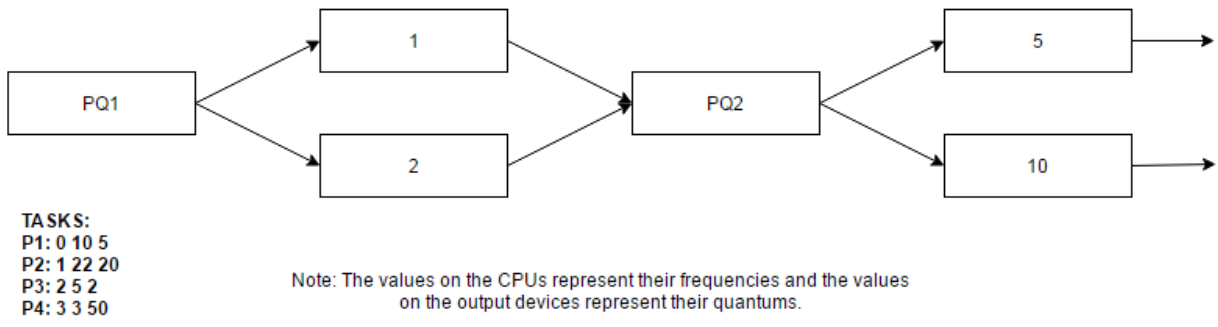

Figure 2: Visualization of the example

Some heads up about project

1. If two events occurs at the same time, process the one closer to the end first and if they are at the same distance, process the one at the upper side of the system. (i.e. If one event will pop up from output device and one event will pop op from CPU handle the event at the output device first. If two events will pop up from CPUs pop up the event from the CPU with smaller ID.)

2. For the output devices part, do not pop up the task if no task is waiting in the queue to be processed even if the quantum has finished.

3. Whenever a unit finishes its job, it immediately fetches the task waiting in the priority queue connected to the unit. If that queue is empty, the unit goes idle.

4. Whenever more than one idle units are available, the unit with the smallest ID gets the job. For example, if a task arrives in the CPU and $CPU1$ and $CPU2$ are idle, $CPU1$ starts processing.

5. Try to comprehend the project with all definitions and concepts before start implementing. **DO NOT START CODING RIGHT AWAY!** - Learned by experience -. First read the definitions a couple of times, use links we provided. When you are comfortable with project in every aspect, simulate some test cases by hand and think of edge cases. Then start coding.

6. For a better code try to code object oriented. Think of what components can be classes or structs and what objects you can make use of(For example: Heap). We ensure you it will lead to much easier and less complicated project.

Submission Details

You are supposed to use the Git system provided to you for all projects. No other type of submission will be accepted. Also pay attention to the following points:

- Your code must be compiled by the simple `cmake CMakeLists.txt` and `make` commands and your code must read the arguments from the command line. Your code will be tested with the command:
  `./project inputFile outputFile`

- All source codes are checked automatically for similarity with other submissions and exercises from previous years. Make sure you write and submit your own code.

- Your program will be graded based the correctness of your output and the clarity of the source code. Correctness of your output will be tested automatically so make sure you stick with the format described above.

- There are several issues that makes a code piece 'quality'. In our case, you are expected to use C++ as powerful, steady and flexible as possible. Use mechanisms that affects these issues positively.

- Make sure you document your code with necessary inline comments, and use meaningful variable names. Do not over-comment, or make your variable names unnecessarily long.
- Try to write as efficient (both in terms of space and time) as possible.