Scheduling

Due: Thursday, December 8th 11:59pm

1 Description

You will write a discrete event simulation to test various scheduling algorithms. The simulation will be driven by events such as a processes arriving, a process performing IO, etc. , and will make use of discrete time. It will be a single program with two arguments. The first will be the name of a file specifying the scheduling algorithm to use and the second the name of a file that specifies the processes to run. The program will simulate the execution of the processes on a single processor. Once the last simulated process is complete the program will print out statistics about the run.

2 Details

A (simulated) process will be divided into a series of activities (this can be represent by a queue). Each activity will have a duration. The activity can either be a CPU activity, meaning it needs to run on the cpu, or an IO activity, meaning it will be blocked for the duration. These activities will alternate between CPU and IO. So, the first activity is a CPU activity, the second will be an IO activity, and so on. The final activity will be a CPU activity. This means there will always be an odd number of activities.

The simulation will comprise of a clock and a sequence of events (which are different from activities). The clock is an integer that keeps track of simulated time, and starts at zero. Each event will have a time stamp representing when it should happen. The possible events are described below. You should maintain a priority queue of events, sorted by time stamp. Ties are broken by Event types (using the order presented in this document). Further ties will simply used the process Id. So an arrival event for process 0 will have higher priority than an arrival event for process 1.

The simulation follows the following steps.

1. Load scheduler information from scheduler file
2. Load processes from process file.
3. Initialize event queue with the arrival events for the processes
4. Look at the time stamp of the next event, and set the clock to be equal to it.
5. While the next event has a time stamp equal to the clock
   - Pop the event from the queue
   - Process the event. (See below)
6. If there is no process currently running, and there is at least one ready process, dispatch a process

7. Go to to step 4 unless all processes have exited (event queue is empty)

When a process is selected to be dispatched its next activity should be a CPU activity. This should generate a BLOCK, EXIT, or TIMEOUT event depending on the scheduling algorithm, and the rest of the activity queue. As the activities are processes they are removed from the activity queue (or otherwise ignored). If a scheduling algorithm uses a time quantum and the duration of a CPU activity is less than the time quantum, then reduce the duration of that activity by the time quantum instead of removing it.

During the execution the simulation should keep track of statistics for each processes and the system. Once the simulation ends the statistics should be printed to screen; one line per process.

2.1 Events

Each event should have associated with it the time it happens and the process the event affects. The easiest way to keep track of the events, is to store them in a priority queue based on the time.

- ARRIVE – The Process is launched, admit it to the system.
- UNBLOCK – The Process is currently in the block state, waiting on IO and will unblock when this event happens
- TIMEOUT – The Process is currently running on the cpu and will timeout when this event happens
- BLOCK – The Process is currently running on the cpu and will block when this event happens
- EXIT – The Process is currently running on the cpu and will terminate when this event happens

2.2 Statistics

For each process:

- Arrival Time
  - Time process enters the system. This time is given in the process file.
- Service Time (T_s)
  - The amount of time the processes spent in possession of the CPU.
  - This should be the sum of the durations for all CPU events for that process.
- Start Time
  - The time the processes first gains control of the cpu.
- Finish Time

  - The time when the last event of the process is complete.

- Turnaround Time (T_r)
  - The duration between the arrival time and finish time
- Normalized Turnaround Time (T_r/T_s)
- Average Response time
  - The response time is the time a processes must wait for the CPU. That is the time from when a processes is placed in the ready queue(when it is launched or after an I/O event or preemption) until the processes is selected to run on the CPU
  - Most processes will have several response times, so you need to compute the average

Afterwards, your program should print out the mean turnaround time, mean normalized turnaround time, and mean average response time(for all processes).

2.3 Scheduler File Format

Some scheduling algorithm requires some configuration. As a simple example of this, consider round robin. It requires a time quantum. This is handled through the scheduler file, which determines the scheduling algorithm, and with which configuration, to use. The very first line will contain the name of the scheduling algorithm as a string that can be read with readLine (or getline).

Each line after that will contain a key/value pair of the form: key=value. The available keys will be determined by the scheduling algorithm. As an example, the time quantum for round robin will be specified by the "quantum" key:

```
quantum=5
```

The details are in Section 2.5.

2.4 Process File Format

Each line represents a process and will have the following format:

ARRIVAL TIME	ACTIVITY	ACTIVITY	···

Each activity is represented by an integer specifying the duration. Here is an example.

```
3 4 3 2
```

The process arrives at time 3, will run for 4 time units once it is given the cpu and wait for I/O for 3 time units. Then, when it receives the cpu a second time it will run for 2 more time units.

2.5 Scheduling Algorithms

The following schedule algorithms must be supported.

2.5.1 First Come First Serve

Name: FCFS

2.5.2 Virtual Round Robin

Name: VRR
quantum: An integer representing the time quantum (in number of time units)

2.5.3 Shortest Remaining Time

Name: SRT
service_given: "true" or "false"
alpha: A number between 0 and 1 representing the weight factor in exponential averaging

When the service_given key is true, the required service time is computed by adding the durations of the CPU events for the processes. If service_given is false then the exponential average is used to estimate the required service time. You may use the first CPU event as S_0.

2.5.4 Highest Response Ratio Next

Name: HRRN
service_given: "true" or "false"
alpha: A number between 0 and 1 representing the weight factor in exponential averaging

When the service_given key is true, the required service time is computed by adding the durations of the CPU events for the processes. If service_given is false then the exponential average is used to estimate the required service time. You may use the first CPU event as S_0.

2.5.5 Feedback

Name: FEEDBACK
quantum: An integer representing the time quantum (in number of time units)
num_priorities: An integer representing the number of levels of priorities

num_priorities specifies the number of dynamic priorities.

3 What to Turn in

Upload your submission as a zip archive containing the following:

- Source code (c, c++, java, or python files)
  - Source code should not require a particular IDE to compile and run.
  - Should work on the cs1 and cs2 machines
- Readme (Plain text document)
  - List the files included in the archive and their purpose
  - Explain how to compile and run your project
  - Include any other notes that the TA may need
- Write-up (Microsoft Word or pdf format)
  - How did you approach the project?
  - What problems did you encounter?
  - How did you fix them?
  - What did you learn doing this project?
  - If you did not complete some feature of the project, why not?
    - What unsolvable problems did you encounter?
    - How did you try to solve the problems?
    - Where do you think the solution might lay?
      - What would you do to try and solve the problem if you had more time?

4 Grading

The grade for this project will be out of 100, and broken down as follows:

Followed Specifications	50
Scheduling Algorithms	20
Correct Output	10
Write-up	20

If you were not able to complete some part of the program discussing the problem and potential solutions in the write-up will reduce the points deducted for it. For example, suppose there is a bug in your code that sometimes allows two customers to approach the same worker, and could not figure out the problem before the due date. You can write 2-3 paragraphs in the write-up to discuss this issue. Identify the error and discuss what you have done to try to fix it/find the problem point, and discuss how you would proceed if you had more time. Overall, inform me and the TA that you know the problem exists and you seriously spend time trying to fix the problem. Normally you may lose 5 points (since it is a rare error) but with the write-up you only lose 2. These points can make a large difference if the problem is affecting a larger portion of the program.