Computing & Information Systems
Trent University

Assignment 2

Due Monday, November 6, 2017 at 11:59pm
A late penalty of 10% per day is assessed until Saturday, November 11, 2017

Learning Objectives

1) To understand the principles behind a discrete event simulation
2) To make use of the classes for Queue and Priority Queue
3) To implement an Event class that overrides the IComparable interface

Introduction

A *discrete event simulation* monitors certain events as they arise within an overall system. These events are recorded (output) in chronological order. For example, within an elevator system, events such as passengers arriving at the elevator door or getting on/off the elevator are recorded as (discrete) events. As the simulation runs, it also keeps track of data to calculate statistics such as the average waiting time at an elevator door, the average time aboard an elevator, or the average number of passengers on the elevator. Naturally, these statistics depend on input parameters such as the passenger inter-arrival rate per floor, the choice of destination floor, the elevator speed, the number of floors, elevator capacity, and so on as well as the rules governing how the elevator system works. For example, the elevator might move to the floor where the maximum number of passengers wishes to embark/disembark or simply move in one direction if there are passengers wishing to embark/disembark in that same direction.

In short, a discrete event simulation records events in chronological order based on the input parameters (often randomized) and the system rules. The simulations are typically executed using different input parameters to see how the statistics are affected for a given set of rules. Alternatively, simulations are executed for a given set of input parameters to see how the statistics are affected under different sets of rules. Either way, it is much cheaper than building a "real" system.

How It Works

At the heart of a discrete event simulation is a Priority Queue of future events. Each event is prioritized based on time and is often triggered by a previous event. For instance, processing the arrival of one passenger at an elevator door generates an event for the next passenger to arrive.

The next event to be processed in a simulation is always found at the "top" of the Priority Queue. When an event is removed from the Priority Queue and processed, time moves forward to that of

the event and hence, only event times are recorded. Unlike a real-time simulation, the time between events is not relevant.

More information on "how it works" will be presented in class on Monday, October 16th. Of course, there is a plethora of information available on-line and in print. So, a little reading goes a long way!

Your Task

An operating system is a complex piece of software that manages the resources of a computing system, including the processor(s), file system, main memory, storage, and peripherals. In this assignment, we will develop a discrete event simulator to mimic the assignment and execution of jobs in a computing system with P processors.

Let's start by defining the Job class. Each instance of Job represents one of two types of processes: regular processes and interrupt processes. On average, 90% of the Job instances are regular processes and have the following requirements.

1) Job number (starting at 1)
2) Number of processors needed from 1 to P
3) Time (same for each processor)

The remaining 10% are interrupt processes. Once generated, each interrupt process is immediately executed on the given processor and pre-empts the current Job that is executing (if any). The interrupt reuses the same data members, but with different interpretations.

1) Job number is 0
2) Processor to be interrupted from 1 to P
3) Time (to execute the interrupt)

Note that an interrupt cannot interrupt another interrupt that is currently executing on a given processor. Hence, it is ignored and terminated with the following message.

Interrupt for processor 3 is ignored at 9:05am.

Jobs arrive for execution on average every M minutes where the inter-arrival time between Jobs is drawn from an exponential distribution with a mean of M. To calculate a random inter-arrival time which is drawn from an exponential distribution with a mean of M, the following formula is used:

$$- M \ln(u)$$

where u is an uniform random number between 0 and 1, and ln is the function for the natural logarithm (base e). The time for each process, regular or interrupt, is also drawn from an exponential distribution with a mean of T. Finally, the number of processors (or processor to be interrupted) is drawn from a uniform distribution between 1 and P, inclusively.

When an instance of Job is created and the Job requires p processors then p (identical) instances of an Event class are also created. Each Event contains the Job, the type of Event, the time it will occur, and the assigned processor (for a DEPARTURE). There are two types of Events.

1) ARRIVAL when a Job arrives to be executed in the computing system
2) DEPARTURE when a Job finishes its execution.

When an Event is created, it is placed in a Priority Queue of Events ordered by time. Therefore, the top Event of the Priority Queue is the next event to occur in the computing system. It also means that the Event class must implement the IComparable interface.

When an Event is removed from the Priority Queue, it is processed in one of three ways.

1) If the Event is an ARRIVAL and the Job is regular then the Job is assigned to a processor if one is available. If no processor is available then the Job is placed in a waiting Queue. Two possible messages are produced.

   Job 3 arrives at 9:00am and begins execution on processor 4.
   or
   Job 3 arrives and is placed in the waiting queue.

   Once a Job is assigned to a processor, a DEPARTURE event is created and entered into the Priority Queue.

2) If the Event is an ARRIVAL and the Job is an interrupt then the interrupt immediately begins execution on the given processor. The Job (if any) that is interrupted is pre-empted, placed in the waiting Queue, and removed from the Priority Queue. Its time is reduced by the amount of time that has already been devoted to its execution. Instead, a DEPARTURE event for the interrupt is created and entered into the Priority Queue. Note that the pre-empted Job can resume execution on any processor.

   Job 0 arrives and begins execution on processor 6 at 9:10am.
   or
   Job 0 arrives and begins execution on processor 6 at 9:10am, pre-empting Job 2.

3) If the Event is a DEPARTURE then it frees up the processor to select a Job from the waiting Queue if there is one, in which case, a new DEPARTURE Event is created. Otherwise, the processor is available for the next Job to arrive.

   Job 2 completes execution on processor 3 at 9:15am.
   or
   Job 2 completes execution and Job 5 begins execution on processor 3 at 9:15am.

Statistics

To measure computing system performance under different scenarios for M, P, and T, the computing system will accept Jobs for two hours. The simulation will then calculate the average time a regular Job waits for execution. The wait time for a regular Job includes the total wait time across all processors that are assigned to its execution. The total number of processes is determined by the total number of ARRIVAL events that are generated for regular Jobs. Hence, a Job that requires 5 processors represents 5 processes.

Things to Keep in Mind

1) The messages output for each arrival, departure, or interrupt will be in chronological order.

2) The Priority Queue stores *future* Events.

3) Each ARRIVAL Event triggers the next ARRIVAL Event. Therefore, the Priority Queue need only store on ARRIVAL Event.

4) The Priority Queue stores at most P DEPARTURE Events, one for each processor.

Grading Scheme

Job class	5%
Event class (including CompareTo)	10%
Modification to the Priority Queue class (to remove an Event)	5%
Main simulation program (using the Queue and Priority Queue classes)	50%
Calculation of average wait time	5%
Test results	15%
Source documentation	10%

Submit all test results and programs (source and executable) via Blackboard. A hard copy is not required. Limit the number of messages for each simulation to one page plus the average wait time.