How to Write a Simulation Program: an application of queue data structure

Professor Yeali S. Sun
Department of Information Management
National Taiwan University
Simulation

- A technique for *modeling the behavior* of both natural and human-made systems

Goal

- Generate *statistics* that summarize the *performance* of an existing system
- *Predict* the performance of a proposed system
- Obtain in-depth understanding (close-up) of the behavior of the system

Example

- A simulation of the behavior of a bank
Performance Modeling and Evaluation: Theory and Simulation
Introduction

- If a “system” is to be used as intended, it must have “acceptable” performance (e.g., small response time).
- It is common, but unfortunate, that performance is not seriously considered until the later stages of system evolution.
- Many systems have unacceptable performance when completed.
- By then, there will be relatively few avenues available to improve performance.
- The most frequently chosen taking is to acquire additional hardware.
Performance as a System Requirement

- Performance to be considered:
  - in the design and development stages
  - during the operational stages

- **Modeling** must be used.
  - Finding the performance bottleneck.
  - Design to meet performance targets

- The system is **not** yet operational.

- Performance is not measurable.
Making Modeling Fun and Useful

- Modeling in general requires a lot of mathematical background and is not widely understood.

- To make modeling methodology accessible to system designers, system developers, and system managers, so to benefit from modeling.

- Avoid sophisticated mathematics and do modeling on an intuitive basis.
Performance modeling and analysis

- Has been and continues to be of great *practical* and *theoretical* importance in the design, development, and optimization of systems and applications (e.g., computer and communication).
- This includes a broad spectrum of activities from the use of more empirical methods
- “Simple models”
- Prototype implementations
- The use of simulation to more sophisticated mathematical methods.
A Queueing System

Arrival process:
- customers to be served
- Inter-arrival time distribution

Queue: finite or infinite capacity

Queueing discipline
- Service time distribution
- Number of servers

Nobody likes to wait in line.
Queueing discipline

A means for choosing which customer in the queue is to be serviced next.

Decision can be based on any or all of the following

Some measure related to the relative arrival times for those customers in the queue;
  • E.g., first-come-first-serve (FCFS), last-come-first-serve (LCFS).
The Model (2/2)

- Some measure (exact value, estimate, probability density function) of the service time required or the services so far received.
  - E.g., shortest-job-first (SJF), longest-job-first (LJF), similar rules based on average, etc.

- Or some function of group membership (i.e. the order of service based on an externally imposed priority class structure)
  - e.g., priority queueing
A Simple Queueing System

A sequence of arrival events:

\(<c_1, t_1, s_1>: \quad (<\text{customerIndex}, \text{arrivalTime}, \text{serviceTime}>)
\)<c_2, t_2, s_2>
\(<c_3, t_3, s_3>
\cdot
\cdot

- System
  - Server
  - Waiting queue
- Behavior (events)
  - Customer arrival
  - Customer departure
**A System**

- **System**
  - Server
  - Waiting queue

- **Behavior (events)**
  - Customer arrival
  - Customer departure

*Figure 7-14* A bank line at

time (a) 0; (b) 12; (c) 20;
(d) 38

Copyright 2008 Yeali S. Sun. All rights reserved. No part of this document may be reproduced, stored in a retrieval system, or transmitted in any form, or by any means without the prior written permission of the author.
Busy Period (1/3)
Event-Driven Simulation

- **Simulation clock**
  - advances by one time unit
- **A system behavior is described by**
  - a sequence of events
    - e.g., customer “arrival” event, “departure” event
    - An event is described by **event time** (the time instance the event takes place) and associated **action**
    - **An event may generate next event**
- **Simulation clock **advances** to time of next event**

**System state transition**
**is based on**
**the event list**
**(the sequence of events ordered by the event time).**
Data Structure: Event

1. typedef struct Event {
2.     int type; /* customerArrival, customerDeparture */
3.     float event_time;
4.     float arrival_time;
5.     float service_time;
6.     struct Event * next;
7. } EVENT;
Event List

1. Ordered by “event_time”
2. EVENT *elist_head;
3. void add_event(EVENT * event_ptr);
4. EVENT *get_event();
Generate Customer Arrivals

Example: Poisson Process

i.e. customer inter-arrival time follows exponential distribution

\[ f(x) = \lambda e^{-\lambda x}, \quad x \geq 0 \]

\[ \Pr ob[\tilde{x} \leq X] = 1 - e^{-\lambda X} \]
Generate Customer Arrivals
(cont’d)

1. // generate an arrival
2. int k = rand();
3. float x = k;
4. x = log(x/MAXNUM);
5. x = -x/\lambda; // customer interarrival time
6. new_event = new (struct event *)
7. new_event->event_time = current_event->event_time + x;
8. new_event->arrival_time = new_event -> event_time;
9. new_event->next = NULL;

\[ \Pr(\tilde{x} \leq X) = 1 - e^{-\lambda x} \]
Data Structures: Simulation Clock

float sim_clock, simRun_duration;
Main Program

1. for (curnt_event = \textit{get_event}(); (sim_clock < simrun_duration) \&\& (curnt_event != NULL); curnt_event = \textit{get_event})
2. 
3. sim_clock = curnt_event->event_time;
4. curnt_event->next = NULL;
5. switch (curnt_event->type) {
6. case customerArrival:
7. if (server_busy == TRUE)
8. q.enqueue();
Main Program (cont’d)

8. else {
9.     change2DepartureEvent(curnt_event);
10.    e.addEvent(curnt_event);
11. }
12.    gen_next_arrival(curnt_event);
13.    update_statistics(curnt_event);
14.    break;
Main Program (cont’d)

15. case customerDeparture:
16.     update_statistics(curnt_event);
17.     if (q.empty() != TRUE) {
18.         q.dequeue(nextSvc_event);
19.         change2DepartureEvent(nextSvc_event);
20.         e.addEvent(nextSvc_event);
21.     }
22.     break;
23. } // switch
24. } // for
25. // Simulation Stops
26. calculate_statistics();
Data Structures: Statistics

- Performance metrics: waiting time, system time
- int total_customers;
- float total_waitTime;
- float total_sysTime;
Summary

- Event-driven simulation uses an event list to:
  - Keep track of arrival and departure events that will occur but have not occurred yet
  - Contain at most one arrival event and one departure event
- Events are generated by using a mathematical model based on statistics and probability