EE/PEP 345
Modeling and Simulation
Spring 2004
Class 3
Today’s topics

• Text – Chapter 3 – General Principles
Concepts in Discrete-Event Simulation

**System**: set of objects, joined to accomplish some purpose

**Model**: Abstraction of real system, defines relationships between entities, system parameters, system state, etc.

**State of system**: collection of variables necessary to describe system at any time

**Entity**: object of interest in system

**Attribute**: property of an entity

**Event**: Instantaneous occurrence that may be associated with change of system state

**Activity**: a predefined set of actions by system objects, usually in a specified time period

**List**: a collection of associated entities, ordered in some logical fashion

**Event notice**: Record of an event to occur at some present or future time, along with associated data

**Event list**: List of event notices (Future Event List)

**Delay**: Duration of time of unspecified indefinite length, not known until it ends

**Clock**: A system variable representing simulated time
Able and Baker, in our current context

• System state:
  \( L_Q(t) \) – the number of customers waiting to be served at time, \( t \)
  \( L_A(t) = (0,1) \), the idle/busy status of Able at time, \( t \)
  \( L_B(t) = (0,1) \), the idle/busy status of Baker at time, \( t \)

• Entities:
  Able and Baker are the only entities we tracked last time

• Activities:
  Inter-arrival time
  Able’s service time
  Baker’s service time

• Delay
  Customer waiting time until serviced by Able or Baker
System Snapshots

Moving image → Series of still images, sufficient to convey recognizable motion
System Snapshots

Moving image

Series of still images, sufficient to convey recognizable motion

System simulation

Series of system descriptions, capturing
• System state
• Activities in progress
• End times
Example system snapshot at time $t$

<table>
<thead>
<tr>
<th>Clock</th>
<th>System State</th>
<th>Entities and Attributes</th>
<th>Set 1</th>
<th>Set 2</th>
<th>...</th>
<th>Future Event List (FEL)</th>
<th>Cumulative Statistics and Counters</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t$</td>
<td>$(x,y,z,\ldots)$</td>
<td>Queue memberships</td>
<td></td>
<td></td>
<td></td>
<td>$(3, t_1)$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$(1,t_2)$</td>
<td></td>
</tr>
<tr>
<td>$t_1$</td>
<td>$(x-1, y, z', \ldots)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$(1,t_2)$</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

System state = $(x, y, z, \ldots)$ means:
- state variable $1 = x$, (e.g., customers waiting = $x$)
- state variable $2 = y$, (e.g., Able is busy)
- state variable $3 = z$, \ldots (e.g., Baker is idle)

Future Event List = {$(3,t_1), (1,t_2)$} means:
- A type 3 event will occur at $t=t_1$ (e.g., Baker service completion at $t_1$)
- A type 1 event will occur at $t=t_2$ (e.g., customer arrival at $t_2$)
## Event-scheduling/time-advance algorithm

<table>
<thead>
<tr>
<th>Clock</th>
<th>System State</th>
<th>…</th>
<th>Future Event List</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>(5,1,6)</td>
<td></td>
<td>(3,t₁)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(1,t₂)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(1,t₃)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>…</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(2,tₙ)</td>
<td></td>
</tr>
</tbody>
</table>

1. Remove event notice for imminent event (at t=t₁)
2. Advance CLOCK to imminent event time
3. Execute imminent event
   - Update system state, change entity attributes, set membership, as needed
4. Generate future events, if needed, and place them on FEL, ranked by time of occurrence
5. Update cumulative statistics and counters

<table>
<thead>
<tr>
<th>Clock</th>
<th>System State</th>
<th>…</th>
<th>Future Event List</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₁</td>
<td>(5,1,5)</td>
<td></td>
<td>(1,t₂)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(4,t*)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(1,t₃)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>…</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(2,tₙ)</td>
<td></td>
</tr>
</tbody>
</table>
Actions occurring on the FEL

- Deleting the imminent event (e.g., servicing a customer in queue)
- Removing events that have already been scheduled (e.g., if customer can leave a queue)
- Adding future events as they get scheduled
- Maintaining time-ordering of events

Length and contents of FEL changes dynamically during simulation

How can you efficiently (in terms of storage and processing efficiency) maintain information about the FEL in a simulation?
Considerations in running an event-scheduling/time-advance simulation

• System snapshot at t=0
  – Preset internal values for simulation – initial conditions
  – External controls – “exogenous” event – from the system environment

• Bootstrapping
  – Used for creating an external event stream, e.g., customer arrivals.
  – Events are placed on FEL by calculating offset from present simulation time.

• Simulation stopping conditions
  – Run for a preset time (e.g., 1000 seconds of simulator time)
  – Run for a preset number of events (e.g., 1000 customers have been served)
  – Run for a preset real-world time (e.g., until the conference paper is due to be sent)
  – Run until an internal event occurs (e.g., the receiver synchronizes)
  – Run until an external event occurs (e.g., user decides to modify experiment)
World Views

• System model is developed differently, based on “world view” employed:
  – Event scheduling
    • Events (the only things that can affect system state) are the prime focus of simulation
  – Process-interaction
    • A process is the life cycle of one entity in simulation
    • Entities:
      – request resources = *events* and
      – use them = *activities*
    • Well suited to object oriented languages (e.g., C++)
      – Entities and processes = objects
  – Activity-scanning
    • Simulation runs in fixed time increments
    • Rule-based approach to decide if an activity can begin at any given point in simulated time
    • Events are quantized to the cycle time of the simulation
World Views - continued

• Activity-scanning costs considerable simulation overhead to decide if an activity can be started
• Hybrid activity-scanning/event-scheduling approach:
  – Two types of activities:
    • B activities:
      – Must occur.
      – Primary events (completion of activity) and unconditional activities
      – Can be scheduled in advance
    • C activities:
      – Conditional events or activities
      – Depend on conditions being true
      – Scan for them at the end of a time advance
Three Phase Approach to Improved Activity-Scanning

Phase A
1. Remove the imminent event from the FEL
2. Advance the clock to its event time
3. Remove any events from FEL that have same event time

Phase B
4. Execute all B-type events that were removed from FEL

Phase C
5. Scan the conditions that trigger C-type activity
6. Activate any C-type whose conditions are met
7. Rescan until complete
List Processing

• Methods to handle lists of entities and the FEL

• Properties and operations on lists:
  – A list is a set of records with an ordering (ranking) operation, accessible in order
  – One item in a list is a “record” – used to store one entity or one event notice.
  – The head
List processing

A generic list

record
record
record
record
record

ordering

Sample list records for simulation

<table>
<thead>
<tr>
<th>entity</th>
<th>attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>event notice</th>
</tr>
</thead>
</table>
List processing

List actions

Head (first record)

Remove record from list

Insert record into list

Tail (last record)
List processing

Efficient insert/deletion is easy

Efficient insert/deletion may not be easy

Efficient insert/deletion is easy
List processing
- data structures

• FORTRAN:
  – A list is an array. Successive records are in contiguous locations in memory
  – Shrinking/expanding lists may not make efficient use of memory

• C:
  – Use a structure for a linked list
  – Use malloc() for variable size structures

• C++
  – Use classes for lists
  – Allocate memory as needed

• LISP, SNOBOL, others
  – List processing is an inherent feature of language

• MatLab
  – wasn’t designed for list processing, but structured data types simplify list processing
  – memory allocation is automatic
List processing
- data structures, implementation
with linked lists

Insertion of record M after record L, before N:
Set M’s “next” to record where L pointed
Redirect “next” of record L to M

Deletion of record V, after U, before W:
Change V to point to W, not U
U’s “next” is arbitrary. Set to “null”, U,
or any other useful value

Given R, finding Q, S, such that Q<R<S
may require search of all records up to S
List processing with structures in Matlab

FEL(1).event='arrival'
FEL(1).time=5
FEL(1).next=13

FEL(13).event='arrival'
FEL(13).time=8
FEL(13).next=5

FEL(5).event='departure'
FEL(5).time=9
FEL(5).next=5
List processing enhancements

• A linked list must be searched from the beginning. The average search time to find where to put a record is on the order of N/2 for an N record list.

• A doubly linked list includes “previous” record pointers, as well as “next” record pointers.

• Adding a “middle” pointer, besides the “head” and “tail” pointers allows faster traversal of a doubly linked list from the middle element. However, there is additional bookkeeping as records get added/deleted and the list center changes.

• “Hash tables” allow fast indexing into a list with minimal searching.
Dynamic versus Static Memory-Management Tradeoffs for Simulation

• Static:
  – Allocate a fixed maximum amount of storage for simulation data
  – E.g., fixed array

• Dynamic
  – Request memory as needed for simulation data
  – E.g., malloc() or language that dynamically allocates from a “heap”

<table>
<thead>
<tr>
<th>Static memory allocation</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Faster startup</td>
<td>• Predetermined maximum list sizes</td>
</tr>
<tr>
<td></td>
<td>• More consistent simulation speed</td>
<td>• May require extra time during simulation to reorder records</td>
</tr>
<tr>
<td>Dynamic memory allocation</td>
<td>• No fixed simulation size</td>
<td>• Slower startups</td>
</tr>
<tr>
<td></td>
<td>•</td>
<td>• Memory fragmentation may require slow “garbage collection”</td>
</tr>
</tbody>
</table>
Homework 3

• Chapter 3, page 94, exercise 4.

• Continue your project started last week. Set objectives and define overall approach.