Lecture 26
Chapter # 7 (contd.)
SIMULATION MODELING

3) Discrete-Event Simulations

• A simulation using a discrete-state model of the system that changes as a function of time. However, the time values could be either discrete or continuous.
• Components of a discrete-event simulator:

a. An event scheduler
b. Simulation Clock and a Time-advancing Mechanism
c. Event processing routines
d. Initialization Routines
e. Event-generation
f. Recording and summarization of data

a) Event Scheduler

you can actually consider it as a data structure that actually maintains a linked…
• It maintains a linked list of all pending events in their global time order.
It will actually maintain a list of all the events that have not yet been answered or such events that have not yet been provided services, such events are the pending events & they are maintained in a linked list according to their arrival times.

• Processes the next event by removing it from the list and dispatching it to the appropriate event-processing routine.
• May be called several times during one event to schedule other new events.
• The scheduler also inserts new events into the appropriate point in the list based on their execution time. So now it depends on the policy that scheduler is maintaining, it may insert new events according to some priority value or let's suppose it may be serving the shorter events first so it can anticipate their execution time & first pick those events who are actually taking lesser execution time.

• Events can be manipulated in various ways as follows:
  • Schedule event X at time T.
  • Cancel a previously scheduled event X.
  • Hold event X indefinitely (until it is scheduled by another event).

  So the event scheduler can either schedule an event or cancel a previously schedule event or it may hold some event indefinitely until it is schedule by any other event.

b) Simulation Clock and a Time-advancing Mechanism

• A global time variable records current simulation time.
• It can be updated by the scheduler using one of two approaches:
  • In the *fixed-increment approach*, the scheduler increments the global clock by some fixed amount.
  • It then checks the pending events.
  • If the scheduled time for any of the pending events matches the current time, all of these events are dispatched for execution.
  • After all these events have been processed, the scheduler again increments the global time variable.
  • The alternative ***event-driven approach*** allows the global time to jump to the value of the next event at the head of the pending-event list.
  • Here the value of the clock will change non-uniformly.
  In event driven approach, a scheduler will be notified if there is any pending event & the time value can actually be jumped to the head of the pending event list. It means that the clock is not been incremented by some uniformed or fixed value & all depends on the measure of events.
  • The event-driven approach is probably the most common in simulations of computer systems.

c) Event Processing Routines

• Each event in the system is simulated by its own event-processing routine. For example, routines to handle the two events of job arrivals and job departure.
• These routines may update the global state and they may generate additional events that must be inserted into the pending-event list.
• For example, a memory-access event in a simulation of a processor may result in two possible outcomes.
  • Cache-hit: the event may simply return the stored value.
  • Cache-miss: the memory-access event routine may generate a new event that returns the corresponding data value t_{miss} time units in the future, where t_{miss} is the time required to service a cache miss.
  So a simple cache memory access event can result in the generation of other events as well.

d) Initialization Routines

These set the initial state of the system state variables and initialize various random-number generation streams.

e) Event-Generation

Three possible techniques of Event-generation are as follows:

i. Execution Driven
ii. Trace Driven
iii. Distribution Driven

i) Execution Driven

• The simulator executes a benchmark program.
• It models the necessary details of the system being tested.
• To simulate a program that does floating-point-arithmetic operations or an input/output operation, for instance,
  • the simulator must provide mechanisms for performing the necessary arithmetic and input/output operations.

  It means that the generation of event part is also coded in the program & whenever part of some code is executed some event is generated & that is to be serviced or cater.

ii) Trace Driven

• A trace is a time-ordered record of events on a real system.
• Trace-driven simulations are generally used in analyzing or tuning resource management algorithms like Paging algorithms, CPU scheduling algorithms, deadlock prevention algorithms etc.
• In this technique, trace of resource demand patterns of key programs are obtained on a system.
• This trace can then be used as input to the simulation which models different algorithms. As OS is responsible to manage the computer system's resources & different processes & different programs actually demand those resources from time to time. First of all, we need to actually get a time order record of demands from the users & once we obtained such a pattern for a system, we can use it to simulate the usage model of different resources may be.

iii) Distribution Driven

• A distribution driven simulation is like a trace-driven simulation, except that input events are generated by the simulator itself to follow some predefined probabilistic function.
• For example, sending messages over a communication network could be modeled by using an exponential distribution to determine the amount of time that elapses between each message.
• The simulator then produces an output that would occur if the real system were driven by an application program that produced the same sequence of inputs.

Such simulator will produces an output which would actually look like real response because we are actually following some random probabilistic distribution function for the input events generation & most of the real-world examples are random & probabilistic in nature. So whatever output the simulator will produce would seem like if the real system would driven by an application program & they have produce the sequence of inputs.

f) Recording and Summarization of Data

• The simulator maintains appropriate event counts and time measurements.
• These values will be used at the end of simulation to calculate appropriate statistics.
• For example, if a memory system is being modeled, the simulator would probably count
• the total number of memory references and the number of those references that result in cache misses.
• These values can be used to estimate the cache-miss ratio.

4) Continuous-Event Simulations

• The state of the system varies continuously with time.
• Used in chemical simulations where the state of the system is described by the concentration of a chemical substance.

As we know that the concentration value for the per chemical thing a changed continuously with time. So, such applications can be modeled using continuous event simulations.

The Simulation Algorithm

we can continue our simulation work if there are no more event & if current time is less than the maximum allotted simulation

```
Initialize global state variables
        Initialize the global time to 0
        Obtain the first input event
        while ((no more events) AND (time < maximum simulation time limit))
        {
           Advance the global time
           Remove the next event from the pending-event list
           Process the event by processing routine
           {
                 Perform event-specific operations
                 Update global variables
                 Update simulation statistics
                 Generate new events triggered by this event

           }

        } when all the events are being served then
        Print the simulation results
```

GENERATION OF RANDOM NUMBERS

• Implementing a distribution-driven discrete-event simulation and Monte-Carlo simulation requires a supply of random numbers from probability distributions.
• A ***random number generator*** is an algorithm that produces sequences of random numbers that follow a specified probability distribution.

METHODS FOR GENERATION

Three methods for generating successive random samples these samples are generated one after other on specific time unit, since we are talking about the discrete event simulation so these samples will be generated at discrete point in time represented as (t = t_1, t_2, ...) from a probability distribution f(t):

1. Inverse Transformation method.
2. Convolution method.
3. Acceptance-rejection method.

The Inverse Transformation Method

• Suppose it is desired to obtain a random sample t from the (continuous or discrete) PDF f(t).
• The inverse method first determines a closed-form expression of the Cumulative Density Function, CDF, F(t) = P{y ≤ t}, where 0 ≤F(t) ≤ 1, for all defined values of y.
• Given that R is a random value obtained from a uniform (0, 1) distribution, and assuming that F^{-1} is the inverse of F, the steps of the method are as follows:

• Step 1. Generate a (0, 1) uniform random number, R.
• Step 2. Set F(x) = R and solve for x as x = F^{-1}(R).

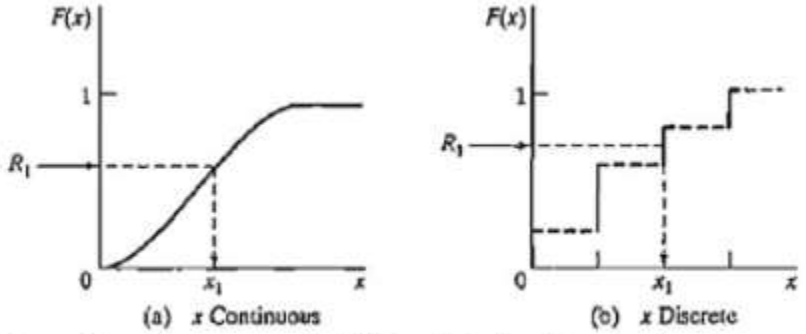

Fig 1: Sampling from a probability distribution by the inverse method

➢ For certain *continuous* distributions, the inverse transformation method can be implemented on a computer by first solving the equation F(x) = R *analytically* for x.
➢ The CDF for the **exponential distribution** is

$$F(t) = 1 - e^{-\lambda t}, \text{ for } x \geq 0,$$

➢ Setting F(t) = R thereby yields

$$1 - e^{-\lambda t} = R \Rightarrow \lambda e^{-\lambda t} = 1 - R,$$

• Therefore, taking the natural logarithm of both sides gives

$$\ln e^{-\lambda t} = \ln (1 - R), \Rightarrow -\lambda t = \ln(1 - R),$$

which yields

$$t = \frac{\ln(1-R)}{-\lambda}$$

• Note that 1 – R is itself a uniform random number. & we are generating this random no. using a certain range, using a uniform distribution. Here in this particular case we have selected this range between 0 & 1. Sometimes in order to save a subtraction, it is common in practice to use the original R directly in place of 1-R.

• In terms of simulation, the result means that arrivals are spaced t time units apart. Note that we have used the CDF of exponential distribution function.

• For example, for λ = 4 customers per hour and R = .9, the time period until the next arrival occurs is computed as:

$$t = \frac{\ln(1 - 0.9)}{-4} = 0.577\ hour = 34.5\ minutes$$

• The values of R used to obtain successive samples must be selected randomly from a uniform (0, 1) distribution.