Event-Driven Simulation

Textbook: Chapter 2 and 3

CIS*2460 - Fall 2007

Overview

- Event-driven simulation is a modelling paradigm in which flow of control within the system is driven by events rather than sequence
  - contrast: procedural vs. event-driven programming
- Sequential flow of control:
  - sequence, branching, iteration
- Event-driven flow of control:
  - events are managed dynamically by an event scheduler
    - typically represented by an event queue (separate from system)
  - advantages: flexible, handles synchronous/asynchronous models with arbitrary timing delays
  - disadvantages: performance

Overview (cont.)

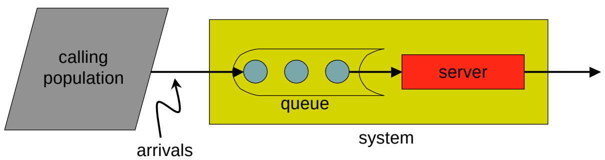
- Event queue
  - every time step:
    - remove items from the event queue due for processing
    - perform appropriate action/updates/etc.
    - as future events are determined, they are added to the event queue (and will be processed at the appropriate time-step)
- Alternative: event loop
  - a thread/process monitors key resources for change, and performs an associated action when it is detected
  - NOTE: this is uncommon in computer simulation
    - unusual to desire completely asynchronous real-time behaviour in a simulated system (consider games, or graphics though…)

Overview (cont.)

- Measures (this list is not exhaustive):
  - # customers processed (N)
  - total service/wait/system time ($W = W_S + W_Q$)
    - cumulative measures
  - total idle/clock time (T_0/T_{max})
  - average wait time ($w_q = W_Q/N$)
  - probability of waiting $p(w_q^n > 0) = \frac{\#\,wait}{N}$
  - probability of idle server ($P_0=T_0/T_{max}$)
  - probability of busy server ($1-P_0$)
  - average service time ($w_s = W_S/N$)
  - average service rate ($1/w_s$)

Example

- Single server queue
  - we'll deal with more realistic situations shortly, however this simple example illustrates the basic concepts



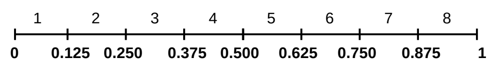
- Assume:
  - customers of one type arrive according to some random process
  - service times vary from one customer to another

Example (cont.)

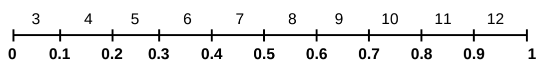
- In order to analyze this system numerically, we will need:
  - knowledge of customer population statistics
    - distribution of inter-arrival times of customers
    - nature of calling population
  - knowledge of server discipline
    - distribution of service times
  - means of generating variates from appropriate distributions
- For simplicity:
  - assume inter-arrival and service times are uniformly distributed integers
    - how would we notate this queueing system?
  - specifically:
    - inter-arrival times $\in$ [1,8], service times $\in$ [3,12]

Example (cont.)

- Since IATs and STs are uniform:
  - inter-arrival times:



  - service times:



  - use a pseudo-random number generator to obtain a sample of 20 IATs and STs for 20 customers

Example (cont.)

Cust.	U_1	IAT	U_2	ST
1	-	-	0.2910	5
2	0.6039	6	0.1321	4
3	0.2782	3	0.5318	8
4	0.2103	2	0.4716	7
5	0.9101	8	0.8662	11
6	0.5983	5	0.5414	8
7	0.1931	2	0.7358	10
8	0.8663	7	0.0137	3
9	0.3944	4	0.5955	8
10	0.7223	6	0.9751	12

Cust.	U_1	IAT	U_2	ST
11	0.0997	1	0.6344	9
12	0.8616	7	0.1848	4
13	0.3889	4	0.6166	9
14	0.7455	6	0.8282	11
15	0.2544	3	0.5563	8
116	0.1191	1	0.7329	10
17	0.4876	4	0.1478	4
18	0.6441	6	0.3731	6
19	0.5232	5	0.9697	12
20	0.1856	2	0.2419	5

Example (cont.)

Cust.	ST	IAT	t_{arrive}	service start	service end
1	5	-	0	0	5
2	4	6	6	6	10
3	8	3	9	10	18
4	7	2	11	18	25
5	11	8	19	25	36
6	8	5	24	36	44
…	…	…	…	…	…

inputs (ST, IAT) — derived (t_{arrive}, service start, service end)

Example (cont.)

- Basic Data
  - \# customers = N = 20 (N_W=18)
  - total service time = W_S = 154m
  - total wait time = W_Q = 589m
  - total system time = W = W_S+W_Q = 743m
  - total idle time = T_0 = 1m
  - total clock time = T_{max} = 155m
- Performance measures
  - avg. wait time = w_q = W_Q/N = 29.45m
  - prob. of waiting = $p(w_q^n > 0)$ = #_wait/N = 0.9
  - prob. of idle server = P_0 = T_0/T_{max} = 0.00645
  - prob. of busy server = 1-P_0 = 0.99355

Example (cont.)

- Performance measures…
  - avg. service time = w_s = W_S/N = 7.7m
  - avg. service rate = 1/w_s = 0.12987
    - NOTE: E(1/w_s) = 1/10(3+4+…+12) = 7.5m
  - avg. inter-arrival time = total_IAT / N-1 = 4.3158m
  - inter-arrival rate = 1/avg_IAT = 0.2317cust./m
    - NOTE: E(inter-arrival rate) = 1/8(1+2+…+8) = 4.5m
  - avg. time in system = w = W/N = w_q+(1/w_s) = 37.15m
  - etc.

Simpler Model

- Don’t need explicit servers or customers
  - Server is a boolean variable that says whether a customer is being served (LS)
  - Queue is an integer variable that says how many customers is lined up (LQ)
  - Events Lists (FEL)
    - Holds events clock-time pairs
      - A = arrival
      - D = departure
      - E = end
    - Will hold 1, 2 or 3 events only
      - (E t_e)
      - (A t_a) (E t_e)
      - (A t_a)(D t_d)(E t_e) or (D t_d)(A t_a)(E t_e)

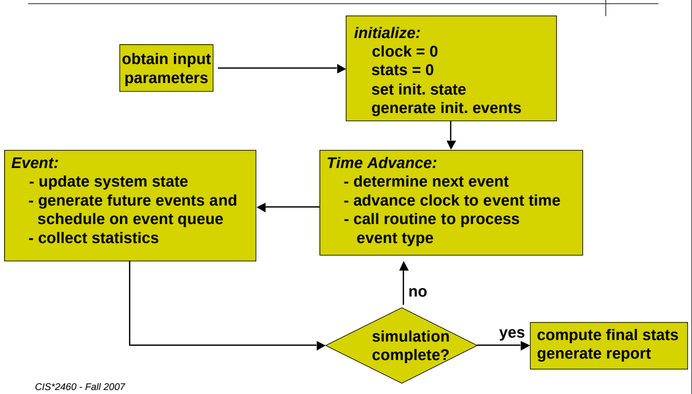
Simpler Model

- For every event processed
  - Generate a service time and hence a new departure event each time LS is set
    - Can occur during an arrival event (LQ is empty)
    - Or a departure event (LQ is not empty)
  - Generate a new arrival event each time an arrival event is processed
    - This occurs whether the arriving 'customer' is
      - serviced immediately (LQ is empty and LS is false)
      - added to LQ (LQ is not empty or LS is true)
    - i.e there is always a new arrival

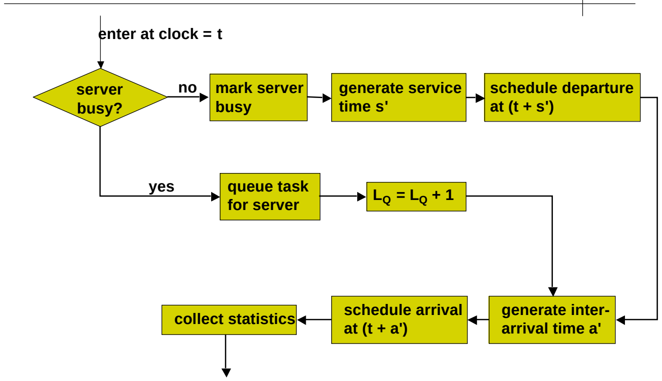
Example (cont.)

	Clock	Server busy	# in system queue	event queue
	init.	no	0	A0, E100
A	0	yes	0	D5, A6, E100
D	5	no	0	A6, E100
A	6	yes	0	A9, D10, E100
A	9	yes	1	D10, A11, E100
D	10	yes	0	A11, D18, E100
…	…	…	…	…

Event-Driven Simulation Flowchart



Arrival Event Execution Flowchart



Departure Event Execution Flowchart