CSIM: Part I

Instructor: Dr. Sunho Lim (Ph.D., Assistant Professor)

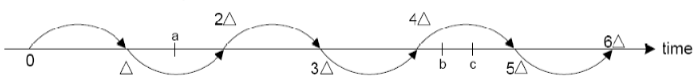
Lecture 04

sunho.lim@ttu.edu

Adapted partially from CSIM Document, http://www.mesquite.com/documentation

Introduction

- CSIM 20
  - A library of routines, for use with C or C++ programs,
  - Create **process-oriented**, and **discrete-event driven models**
    - ns-3, GloMoSim, OMNeT++, QualNet, etc.
- Time-Driven Simulation
  - Observe the system at a fixed interval
  - Event occurs within an interval is assumed to occur at the end of the interval
  - e.g., Suppose an interval = Δ seconds. Then the simulation proceeds as follows:

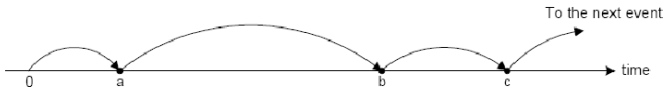


a, b, c, and d are events!!
a is assume to occurred at t= 2Δ

  - The simulation finishes at a pre specified time

Introduction (cont.)

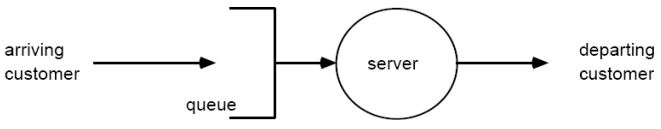
- Discrete-event simulation
  - Each operation in system is an event
  - A sequence of events (e.g., event queue)
  - Each event changes the state of system
  - Clock
    - ***Not* real time**
    - Keep track of the current simulation time in whatever measurement unit
    - Due to the discrete events, the clock skips to the next event start time



  - Simulation finishes,
    - At a pre-specified time
    - When there is no more event

HelloWorld CSIM

- M/M/1,
  - A single-server queue model
- Key parameters,
  - The intervals of time between customer arrivals
  - The intervals of server usage
- Results,
  - The average customer **response time** (time of arrival to time of departure)
  - The customer **throughput** rate (customers served per unit time)
  - The server **utilization** (percentage of elapsed time that the server is busy)
  - The average queue length (number of customers at the facility)



```
/*this CSIM program simulates an M/M/1 service center*/

#include <csim.h>                  /*include the CSIM library*/

FACILITY f;                        /*the service center*/

sim()                              /*sim process*/
{
    create("sim");                 /*make this a process*/
    f = facility("f");             /*create the service center - f*/
    while(simtime() < 5000.0) {    /*loop until end of simulation*/
        hold(exponential(1.0));    /*delay between customer arrivals*/
        customer();                /*generate new customer*/
    }
    report();                      /*produce statistics report*/
}

customer()
{
    create("customer");            /*make this a process*/
    use(f, exponential(0.5));      /*obtain needed amount of service*/
}
```


HelloWorld CSIM (cont.)

Modeling parallel activities is a major feature of process-oriented models

```
/*this CSIM program simulates an M/M/1 service center*/

#include <csim.h>                  /*include the CSIM library*/

FACILITY f;                        /*the service center*/

sim()                              /*sim process*/
{
    create("sim");                 /*make this a process*/
    f = facility("f");             /*create the service center - f*/
    while(simtime() < 5000.0) {    /*loop until end of simulation*/
        hold(exponential(1.0));    /*delay between customer arrivals*/
        customer();                /*generate new customer*/
    }
    report();                      /*produce statistics report*/
}

customer()
{
    create("customer");            /*make this a process*/
    use(f, exponential(0.5));      /*obtain needed amount of service*/
}
```

CSIM Objects: Process

- In CSIM,
  - A process is a **C procedure** that executes the "**create()**" statement
  - Every time a "create()" statement is executed, a new instance of that process is created
- For example,

```
...
for(i = 0; I < 100; I++)
    customer(i);
hold(1000.0);
...

customer()
{
    create("customer");          /*make this a process*/
    use(f, exponential(0.5));    /*obtain needed amount of service*/
}
```

  - 100 instances of the process named "customer" will be created
  - None of them will start to execute until the calling process executes the **hold** statement.

CSIM Objects: Event

- Synchronize and control interactions between different processes
- A CSIM event has two states:
  - occurred (OCC)
  - not occurred (NOT_OCC)
- A process can either wait for an event to occur or queue on the event
- When a process "waits" for an event:
  - If the event is in the **not occurred state**, the process is suspended and placed in a **queue** of processes waiting for the event to happen (occur)
  - When some other process does a **"set"** operation on that event, **all of the waiting processes** are reactivated (allowed to proceed) and the event is reset to not occurred
- If the event is in the occurred state,
  - The process continues to execute and the event state is changed to **not occurred**

CSIM Objects: Event (cont.)

- When a process queues on an event,
  - If the event is in the **not occurred state**, the process is suspended and placed in a queue of processes queued for the event to happen (occur)
  - When some other process does a "set" operation on that event, **only the first queued process** is reactivated (allowed to proceed) and the event is reset to not occurred
- If the event is in the occurred state, and there are no other processes queued on that event,
  - The process continues to execute and the event state is changed to **not occurred**

CSIM Objects: Event (cont.)

- To declare, initialize, and use an event:

```
EVENT ev;              /*declare event variable ev */
...
ev = event("ev");      /*initialize an event named ev */
....
wait(ev);              /*wait for event to occur before proceeding */
...
queue(ev);  /*wait for event to occur, for processes to respond before proceeding*/
set(ev);               /*indicate that an event has occurred */
...
```

- To declare, initialize, and use an array of twenty-five events:

```
#define NUM_EVENTS 25                        /*set number of events in array */
EVENT ev_arr[NUM_EVENTS];                    /*declare event array */
....
event_set(ev_arr, "ev arr", NUM_EVENTS);     /*initialize array of 25 events*/
....
wait(ev_arr[5]);          /*wait for sixth event to occur before proceeding */
...
set(ev_arr[5]);           /*indicate that sixth event has occurred*/
...
```


CSIM Objects: Event (cont.)

- To wait for an event only if it occurs within a given length of time:

```
…
st = timed_wait(ev, 50.0);          /*wait for a maximum of 50 time units */
if(st ! = TIMED_OUT) {              /*did not timed out */
…..
}
```


CSIM Objects: Facility

- Entities that processes **"use"** or occupy,
  - A single server facility (can only service one process at a time)
  - A multi-server facility (can service n processes at once, where n is the number of servers defined for the facility)
  - An array of single server facilities
- Processes are ranked in the queue of waiting processes in order of their process priorities,
  - In the case of equal priorities, the scheduling policy (discipline) is **first-come, first-served (or FIFO – first in, first out)**.

CSIM Objects: Facility (cont.)

- A single queue and a single server:

```
FACILITY single_server;                 /* declare facility variable/
...
single_server = facility("sngle srvr"); /* initialize facility named sngle srvr*/
...
use(single_server, service_time);       /* use facility for length of service_time*/
...
reserve(single_server);                 /* reserve (use) facility */
...
hold(service_time);
...
release(single_server);                 /* release the facility  */
...
```

- "use()" , when the process will be "using" the facility
- "reserve", when the process will acquire exclusive use of the facility and then **do something** other than a "hold" statement.

CSIM Objects: Facility (cont.)

- A single queue and three servers:

```
#define NUM_SRVRS 3                     /* set number of servers to 3 */
FACILITY multi_server;                  /* declare facility variable */
...
multi_server = facility_ms("multi srvr", NUM_SRVRS); /*initialize srvr with 3 srvrs*/
...
use(multi_server, service_time);        /*use facility for length of service_time*/
....
```


CSIM Objects: Facility (cont.)

- An array of ten single server facilities

```
#define NUM_FACS 10                    /*set number of facilities in array to 10 */
FACILITY facs[NUM_FACS];               /* declare facility array */
...
facility_set(facs, "facs", NUM_FACS);  /*initialize set of 10 facilities */
i = random(0, NUM_FACS-1);             /*select the facility to be used next*/
use(facs[i], service_time);            /*use facility[i] for length of service_time*/
```

- To reserve a facility only if it can be obtained within a given length of time:

```
#define TIME_OUT 5.0                   /*length of time to wait for facility*/
.....
st = timed_reserve(single_server,  TIME_OUT);  /*reserve facility in 5 time units*/
if(st != TIMED_OUT) {                  /*if facility was, in fact, reserved in time*/
     hold(service_time);               /*simulate servicing customer for service_time*/
     release(single_server);           /*release facility since service is now complete*/
} else {                               /*request timed out */
....
     }
```