A Case Study

Dr. Xiao Qin

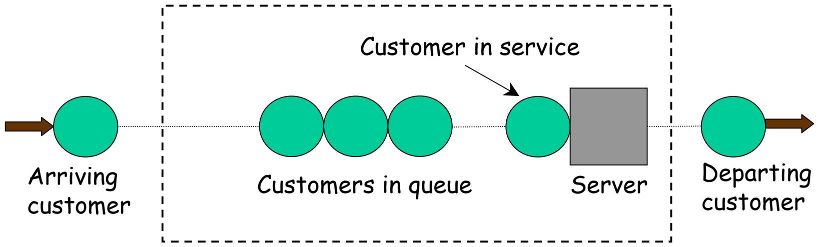
New Mexico Tech
http://www.cs.nmt.edu/~xqin
xqin@cs.nmt.edu

Fall, 2006

These slides are adapted from notes by Dr. Anirban Mahanti (U. of Calgary)

Objective

- Learn the basics of a simple simulation lang.
- We will focus on a simple single-server service center. Develop this using “simlib”



Introduction

- Complex systems usually require complex models – difficult to code from scratch in general-purpose language
- Need some help, more simulation-oriented software
- Develop simulations using "simlib" – a rudimentary simulation tool that provides tools for several "routine" simulation tasks
  - List processing, event-list management, random-number and variate generation, statistics collection, output reporting
  - Use C programming
- **`simlib`** is not a "real" simulation language
  - Doing it here to illustrate what's behind commercial simulation software, enable modeling of more complex systems

Overview of simlib

- Capabilities
  - List processing (pointers, file a record, remove a record)
  - Processes event list
  - Tallies statistical counters
  - Generate random numbers, variates from some distributions
  - Provide "standard" output (optional)
- Not a "real" simulation language – incomplete, inefficient
  - But illustrates what's in real simulation software, how it works

Overview of simlib (cont.)

- Heart of **`simlib`** is a collection of doubly linked lists
  - Lists use dynamic memory – space allocated as new records are filed in lists, freed when records are removed from lists
  - Maximum of 25 lists
  - Records in lists can have a maximum of 10 attributes
  - Data are stored as type **`float`**
  - Uses dynamic storage allocation, so total number of records in all lists is limited only by hardware
  - List 25 always reserved for event list
    - Always: attribute 1 = (future) event time, attribute 2 = event type
    - attributes 3-10 can be used for other event attributes
    - Kept sorted in increasing order on event time – next event is always on top
- User must **`#include simlib.h`** for required declarations, definitions
  - **`simlib.h`** in turn **`#include`**s **`simlibdefs.h`**

Simlib – variables and constants

- **`sim_time`** = simulation clock; updated by **`simlib`**
- **`next_event_type`** = type of next event; determined by **`simlib`**
- **`transfer[i]`**: float array indexed on **`i`** = 1, 2, …, 10 for transferring attributes of records into and out of lists
- **`maxatr`** = max number of attributes in any list; defaults to 10, but user can initialize to < 10 for improved efficiency (cannot be set to < 4 due to the way **`simlib`** works)

Simlib – variables and constants (cont.)

- **`list_size[list]`** = current number of records in list **`list`**; maintained by **`simlib`**
- **`list_rank[list]`** = attribute number (if any) on which list **`list`** is to be ranked (incr. or decr.); must be initialized by user
- **`FIRST`**, **`LAST`**, **`INCREASING`**, **`DECREASING`**: symbolic constants for options of filing a record into a list
- **`LIST_EVENT`**, **`EVENT_TIME`**, **`EVENT_TYPE`**: symbolic constants for event-list number, attribute number of event time, attribute number of event type

Simlib functions

- **`init_simlib`**: Invoke at beginning of each simulation run from the (user-written) main function to allocate storage for lists, initialize all pointers, set clock to 0, sets event list for proper ranking on event time, defaults **`maxatr`** to 10, sets all statistical counters to 0
- **`list_file(option, list)`**: File a record (user must pre-load attributes into **`transfer`** array) in list **`list`**, according to **`option`**:
  - 1 or **`FIRST`** File record as the new beginning of the list
  - 2 or **`LAST`** File record as the new end of the list
  - 3 or **`INCREASING`** File record to keep list in increasing order on attribute **`list_rank[list]`** (as pre-set by user)
  - 4 or **`DECREASING`** File record to keep list in increasing order on attribute **`list_rank[list]`**

Simlib functions (cont.)

- **`list_remove(option, list)`**: Remove a record from list **`list`** and copy its attributes into the **`transfer`** array, according to **`option`**:
  - 1 or **`FIRST`** Remove the first record from the list
  - 2 or **`LAST`** Remove the last record from the list

- **`timing`**: Invoke from main function to remove the first record from the event list (the next event), advance the clock to the time of the next event, and set **`next_event_type`** to its type; if attributes beyond 1 and 2 are used in the event list their values are copied into the corresponding spots in the **`transfer`** array

Simlib functions (cont.)

- **`event_schedule(time_of_event, type_of_event)`**: Invoke to schedule an event at the indicated time of the indicated type; if attributes beyond 1 and 2 are used in the event list their values must be pre-set in the **`transfer`** array
- **`event_cancel(event_type)`**: Cancel (remove) the first (most imminent) event of type **`event_type`** from the event list, if there is one, and copy its attributes into the **`transfer`** array

Simlib functions (cont.)

- **`sampst(value, variable)`**: Accumulate and summarize discrete-time process data. Can maintain up to 20 separate "registers" (**`sampst`** variables); 3 functions:
  - During the simulation: record a value already placed in **`value`** in **`sampst`** variable **`variable`**: **`sampst (value, variable)`**
  - At end of simulation: invoke **`sampst`** with the *negative* of the variable desired (**`value`** doesn't matter); get in **`transfer`** array the mean (1), number of observations (2), max (3), min (4); name **`sampst`** also has mean
  - To reset all **`sampst`** variables: **`sampst (0.0, 0)`**; normally done at initialization, but could be done at any time during the simulation

Simlib functions (cont.)

- **`timest(value, variable)`**: Accumulate and summarize continuous-time process data. Can maintain up to 20 separate "registers" (**`timest`** variables), separate from **`sampst`** variables; 3 functions:
  - During the simulation: record a *new* value (after the change in its level) already placed in **`value`** in **`timest`** variable **`variable`**: **`timest (value, variable)`**
  - At end of simulation: invoke **`timest`** with the *negative* of the variable desired (**`value`** doesn't matter); get in **`transfer`** array the mean (1), max (2), min (3); name **`timest`** also has mean
  - To reset all **`timest`** variables: **`timest (0.0, 0)`**; normally done at initialization, but could be done at any time during the simulation

Simlib functions (cont.)

- **`filest(list)`**: Produces summary statistics on number of records in list **`list`** up to time of invocation
  - Get in **`transfer`** array the mean (1), max (2), min (3) number of records in list list; name **`filest`** also has mean
  - Why? List lengths can have physical meaning (queue length, server status)
- **`out_sampst(unit, lowvar, highvar)`**: Write to file **`unit`** summary statistics (mean, number of values, max, min) on **`sampst`** variables **`lowvar`** through **`highvar`**
  - Get "standard" output format, 80 characters wide
  - Alternative to final invocation of **`sampst`** (still must initialize and use **`sampst`** along the way)

Simlib functions (cont.)

- **`out_timest(unit, lowvar, highvar)`**: Like **`out_sampst`** but for **`timest`** variables
- **`out_filest(unit, lowfile, highfile)`**: Like **`out_sampst`** but for summary statistics on number of records in lists **`lowfile`** through **`highfile`**
- **`expon(mean, stream)`**: Returns in its name a variate from an exponential distribution with mean **`mean`**, using random-number "stream" **`stream`**
  - Uses random-number generator **`lcgrand`**

Simlib functions (cont.)

- **`random_integer(prob_distrib[], stream)`**: Returns a variate from a discrete probability distribution with *cumulative* distribution in the array **`prob_distrib`**, using stream **`stream`**
  - Assumes range is 1, 2, …, k, with $k \le 25$
  - User prespecifies **`prob_distrib[i]`** to be $P(X \le$ **`i`**$)$ for **`i`** = 1, 2, …, k
  - Note that **`prob_distrib[`***k***`]`** should be specified as 1.0
- **`uniform(a, b, stream)`**: Returns a variate from a continuous uniform distribution on [**`a`**, **`b`**], using stream **`stream`**
- **`erlang(m, mean, stream)`**: Returns a variate from an **`m`**-Erlang distribution with mean **`mean`**, using stream **`stream`**

Simlib functions (cont.)

- **`lcgrand(stream)`**: Random-number generator, returns a variate from the (continuous) U(0, 1) distribution, using stream **`stream`**
  - **`stream`** can be 1, 2, …, 100
  - When using **`simlib`**, no need to **`#include lcgrand.h`** since **`simlib.h`**, already **`#include`**d, has the definitions needed for **`lcgrand`**
- **`lcgrandst(zset, stream)`**: Sets the random-number "seed" for stream **`stream`** to **`zset`**
- **`lcgrandgt(stream)`**: Returns the current underlying integer for stream **`stream`**

Using Simlib

- Still up to user to determine events, write main function and event functions (and maybe other functions), but **`simlib`** functions makes this easier
- Determine what lists are needed, what their attributes are
- Determine **`sampst`**, **`timest`** variables
- Determine and assign usage of random-number streams
- **`simlib`** variables take the place of many state, accumulator variables, but user may still need to declare some global or local variables

Writing a simulation using Simlib

1. Read/write input parameters
2. Invoke **`init_simlib`** to initialize **`simlib`**'s variables
3. (Maybe) Set **`lrank_list[list]`** to attribute number for ranked lists
4. (Speed option) Set **`maxatr`** to max number of attributes per list
5. (Maybe) **`timest`** to initialize any **`timest`** variables to nonzero values

Writing a simulation using Simlib

6. Initialize event list via **`event_schedule`** for each event scheduled at time 0
   - If using more than first two attributes in event list (time, type), must set **`transfer[3]`**, **`transfer[4]`**, … before invoking **`event_schedule`**
   - Events not initially to be scheduled are just left out of the event list
7. Invoke timing to determine **`next_event_type`** and advance clock
8. Invoke appropriate event function, perhaps with **`case`** statement
9. At end of simulation, invoke user-written report generator that usually uses **`sampst`**, **`timest`**, **`filest`**, **`out_sampst`**, **`out_timest`**, **`out_filest`**

More on using Simlib

- Things to do in your program
  - Maintain lists via **`list_file`**, **`list_remove`**, together with **`transfer`** to communicate attribute data to and from lists
  - Gather statistics via **`sampst`** and **`timest`**
  - Update event list via **`event_schedule`**
- Error checking – cannot check for all kinds of errors, but some opportunities to do some things in simulation "software" – so **`simlib`** checks/traps for:
  - Time reversal (scheduling an event to happen in the past)
  - Illegal list numbers, variable numbers
  - Trying to remove a record from an empty list

Single Server Simulation using Simlib

- Same model as discussed earlier
  - Original 1000-delay stopping rule
  - Same events (1 = arrival, 2 = departure)
- `simlib` lists, attributes:

  1 = queue, attributes = [time of arrival to queue]

  2 = server, no attributes (dummy list for utilization)

  25 = event list, attributes = [event time, event type]
- `sampst` variable: 1 = delays in queue
- `timest` variables: none (use `filest` or `out_filest`)
- Random-number streams:
  1 = interarrivals, 2 = service times

External Definitions

```
/* External definitions for single-server queueing system using simlib. */

#include "simlib.h"            /* Required for use of simlib.c. */

#define EVENT_ARRIVAL        1  /* Event type for arrival. */
#define EVENT_DEPARTURE      2  /* Event type for departure. */
#define LIST_QUEUE           1  /* List number for queue. */
#define LIST_SERVER          2  /* List number for server. */
#define SAMPST_DELAYS        1  /* sampst variable for delays in queue. */
#define STREAM_INTERARRIVAL  1  /* Random-number stream for interarrivals. */
#define STREAM_SERVICE       2  /* Random-number stream for service times. */

/* Declare non-simlib global variables. */

int   num_custs_delayed, num_delays_required;
float mean_interarrival, mean_service;
FILE  *infile, *outfile;

/* Declare non-simlib functions. */

void init_model(void);
void arrive(void);
void depart(void);
void report(void);
```

The main function

```
main()  /* Main function. */
{
    /* Open input and output files. */
   /* Read input parameters. */
   /* Write report heading and input parameters. */
    init_simlib();   /* Initialize simlib */
    maxatr = 4;  /* NEVER SET maxatr TO BE SMALLER THAN 4. */
    init_model(); /* Initialize the model. */
    while (num_custs_delayed < num_delays_required) {
        timing(); /* Determine the next event. */
        switch (next_event_type) {/* Invoke the appropriate event
       function. */
            case EVENT_ARRIVAL:
                arrive();
                break;
            case EVENT_DEPARTURE:
                depart();
                break;
        }
    }
    report(); /* Invoke the report generator and end the simulation. */
   /* close files */
return 0;
}
```

The init Module

```
void init_model(void)  /* Initialization function. */
{
    num_custs_delayed = 0;

    event_schedule(sim_time + expon(mean_interarrival,
      STREAM_INTERARRIVAL),
                EVENT_ARRIVAL);
}
```

The arrive Module

```
void arrive(void)  /* Arrival event function. */
{
   /* Schedule next arrival. */
   event_schedule(sim_time + expon(mean_interarrival, STREAM_INTERARRIVAL),
               EVENT_ARRIVAL);
   /* Check to see whether server is busy (i.e., list SERVER contains a
      record). */
   if (list_size[LIST_SERVER] == 1) {
      /* Server is busy, so store time of arrival of arriving customer at end
         of list LIST_QUEUE. */
      transfer[1] = sim_time;
      list_file(LAST, LIST_QUEUE);
   }
   else {
     /* Server is idle, so start service on arriving customer, who has a
        delay of zero.  (The following statement IS necessary here.) */
      sampst(0.0, SAMPST_DELAYS);
      /* Increment the number of customers delayed. */
      ++num_custs_delayed;
     /* Make server busy by filing a dummy record in list LIST_SERVER. */
      list_file(FIRST, LIST_SERVER);
      /* Schedule a departure (service completion). */
      event_schedule(sim_time + expon(mean_service, STREAM_SERVICE),
                  EVENT_DEPARTURE);
   }
}
```

The depart Module

```
void depart(void)  /* Departure event function. */
{
   /* Check to see whether queue is empty. */
   if (list_size[LIST_QUEUE] == 0)
      /* The queue is empty, so make the server idle and leave the departure
         (service completion) event out of the event list. (It is currently
         not in the event list, having just been removed by timing before
         coming here.) */
      list_remove(FIRST, LIST_SERVER);
else {
      /* The queue is nonempty, so remove the first customer from the queue,
         register delay, increment the number of customers delayed, and
         schedule departure. */
      list_remove(FIRST, LIST_QUEUE);
      sampst(sim_time - transfer[1], SAMPST_DELAYS);
      ++num_custs_delayed;
      event_schedule(sim_time + expon(mean_service, STREAM_SERVICE),
                  EVENT_DEPARTURE);
   }
}
```

The report Module

```
void report(void)  /* Report generator function. */
{
   /* Get and write out estimates of desired measures of performance. */

   fprintf(outfile, "\nDelays in queue, in minutes:\n");
   out_sampst(outfile, SAMPST_DELAYS, SAMPST_DELAYS);
   fprintf(outfile, "\nQueue length (1) and server utilization (2):\n");
   out_filest(outfile, LIST_QUEUE, LIST_SERVER);
   fprintf(outfile, "\nTime simulation ended:%12.3f minutes\n",
    sim_time);
}
```

Your Tasks

- Play with simlib
- Try introducing a timest variable for keeping track of the continuous time quantity “server utilization”