Modeling Complex Systems

Chapter 2

Jiang Li, Ph.D.
Department of Computer Science

2.1 Introduction

- List processing
  - An activity that takes place in most simulations
- simlib
  - A group of ANSI-standard C support functions
    - Includes common simulation activities
  - Complete source code can be downloaded from www.mhhe.com/law

Jiang Li, Ph.D.
Department of Computer Science

2.2 List Processing in Simulation

- Examples in chapter 1 contained either one or no lists of records
  - Other than the event list
  - Always processed as first-in, first-out (FIFO)
- Most complex simulations require many lists
  - May not be processed in a FIFO manner

Jiang Li, Ph.D.
Department of Computer Science

Approaches to Storing Lists

- Sequential-allocation
  - Records are stored in physically adjacent memory
  - Relatively simple
  - First element may be at fixed or dynamic index
  - How to prepend, append, insert, delete?
    - Could be time-expensive

Jiang Li, Ph.D.
Department of Computer Science

Approaches to Storing Lists (cont'd)

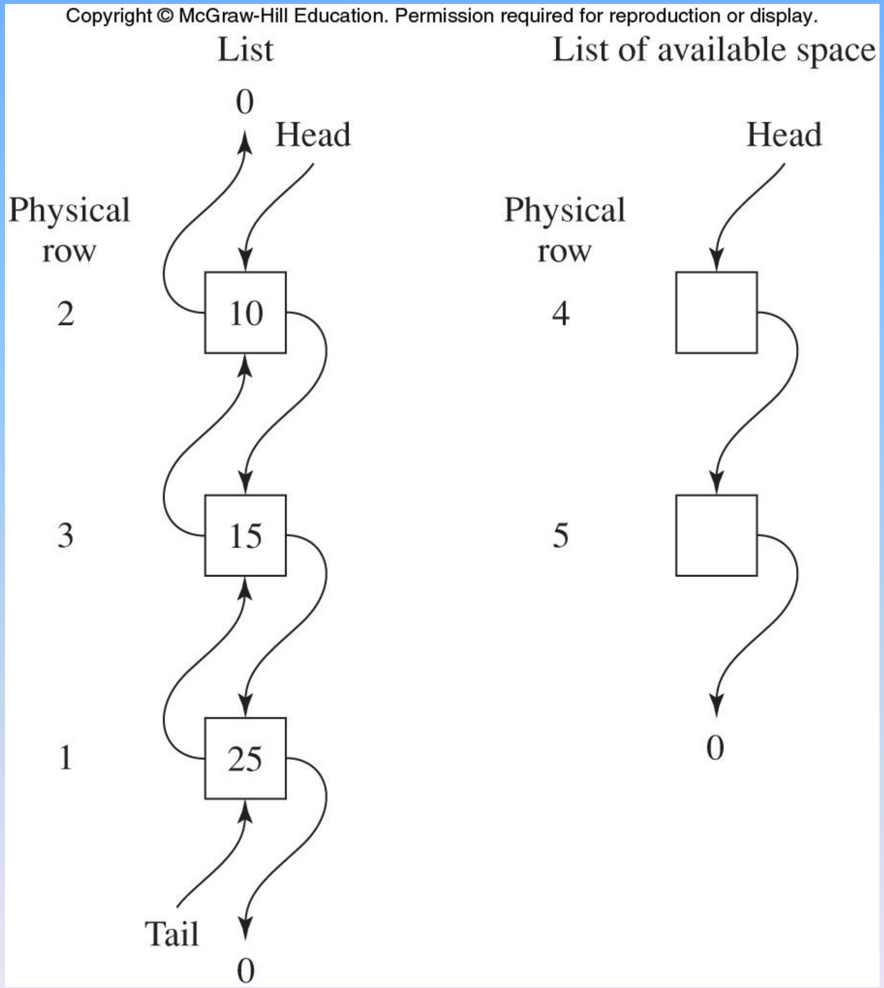
- Linked-allocation
  - Each record contains pointers that specify its logical relationship to other records in the list
  - Singly linked
    - Successor (forward) link
    - Head pointer
  - Doubly linked
    - Successor (forward) link
    - Predecessor (backward) link
    - Head pointer
    - Tail pointer

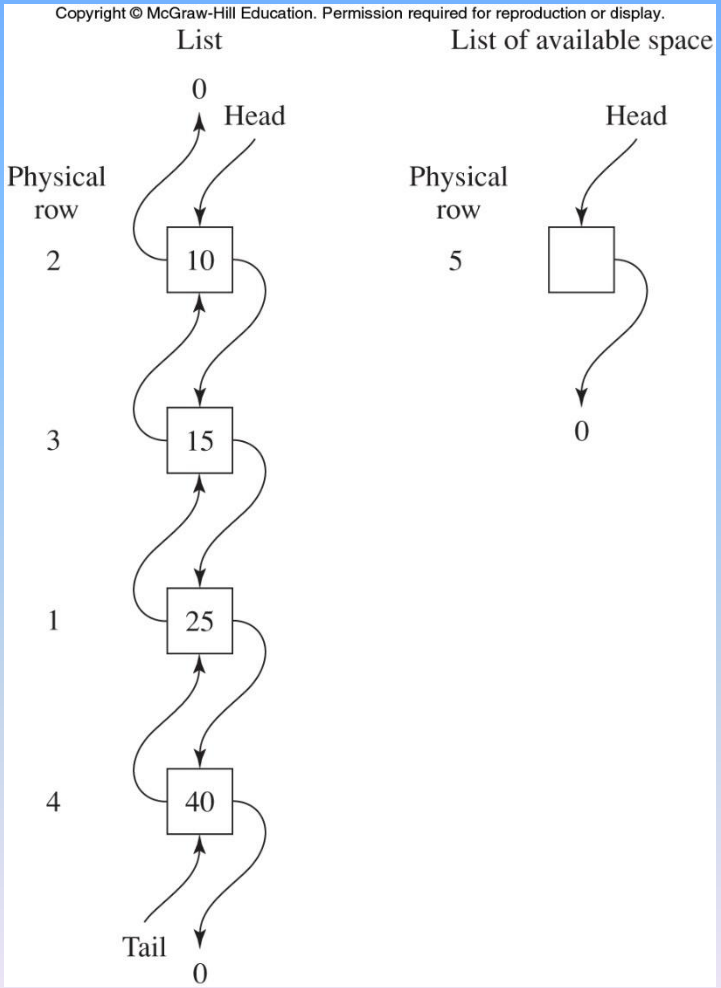
Approaches to Storing Lists (cont'd)

- Linked-allocation
  - Allocate memory with OS-provided API
  - Allocate memory from a self-maintained array
    - Use index instead of pointers
    - Maintain a List of available space

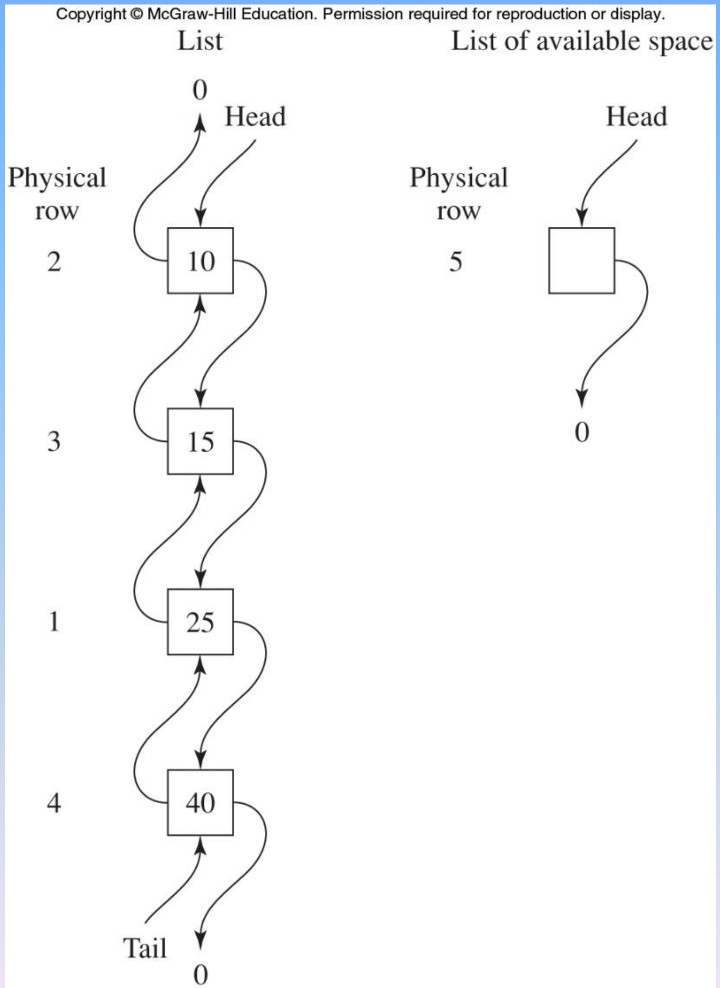
Jiang Li, Ph.D.
Department of Computer Science

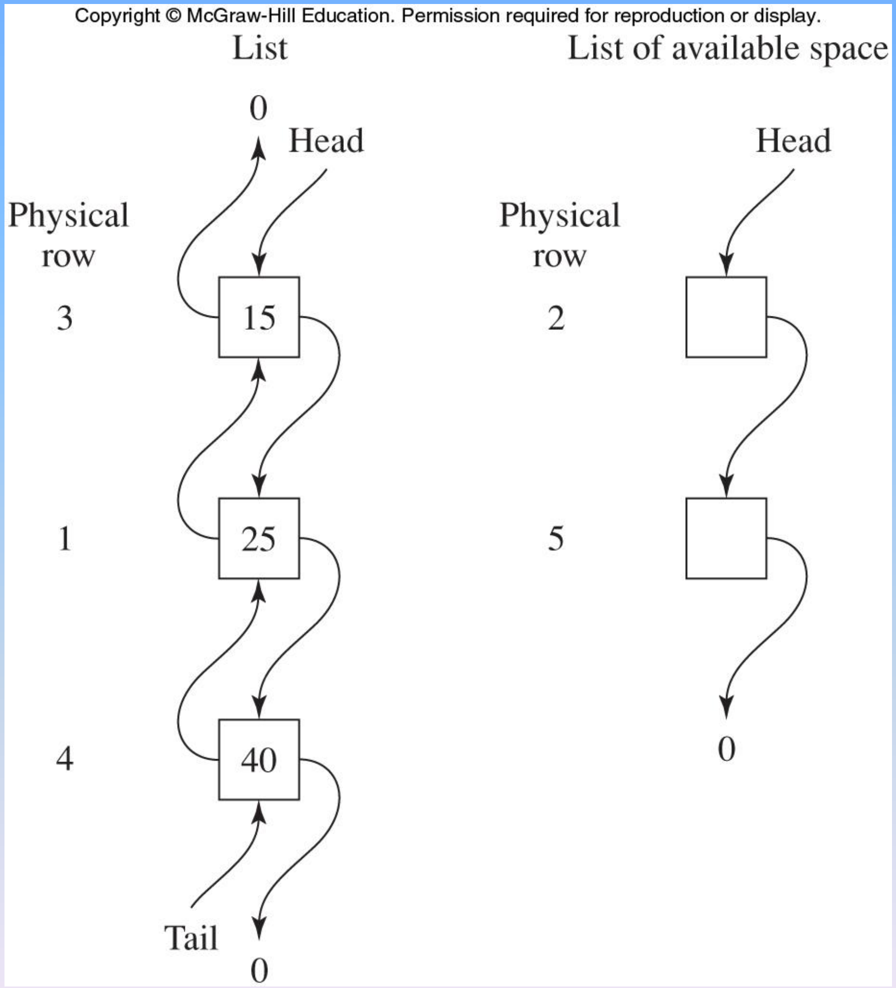
Use of Lists in Simulations





Use of Lists in Simulations

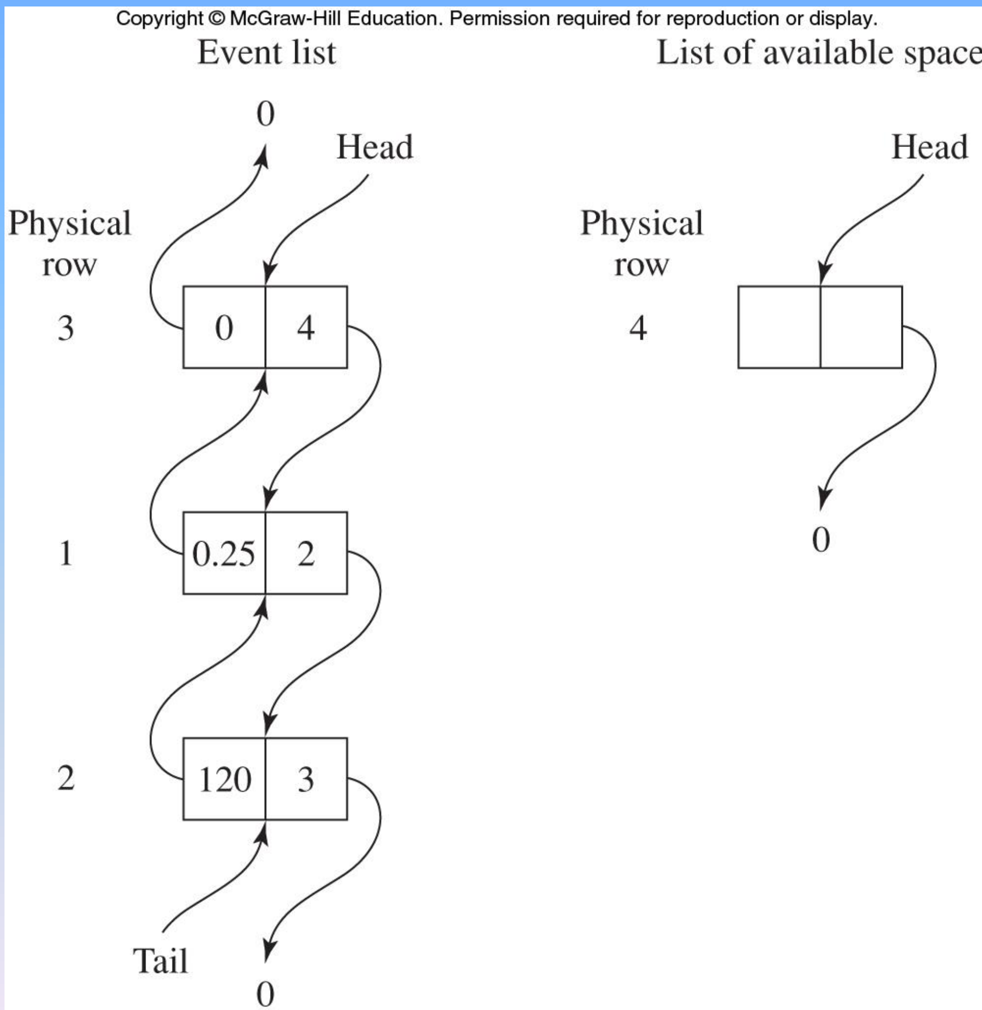


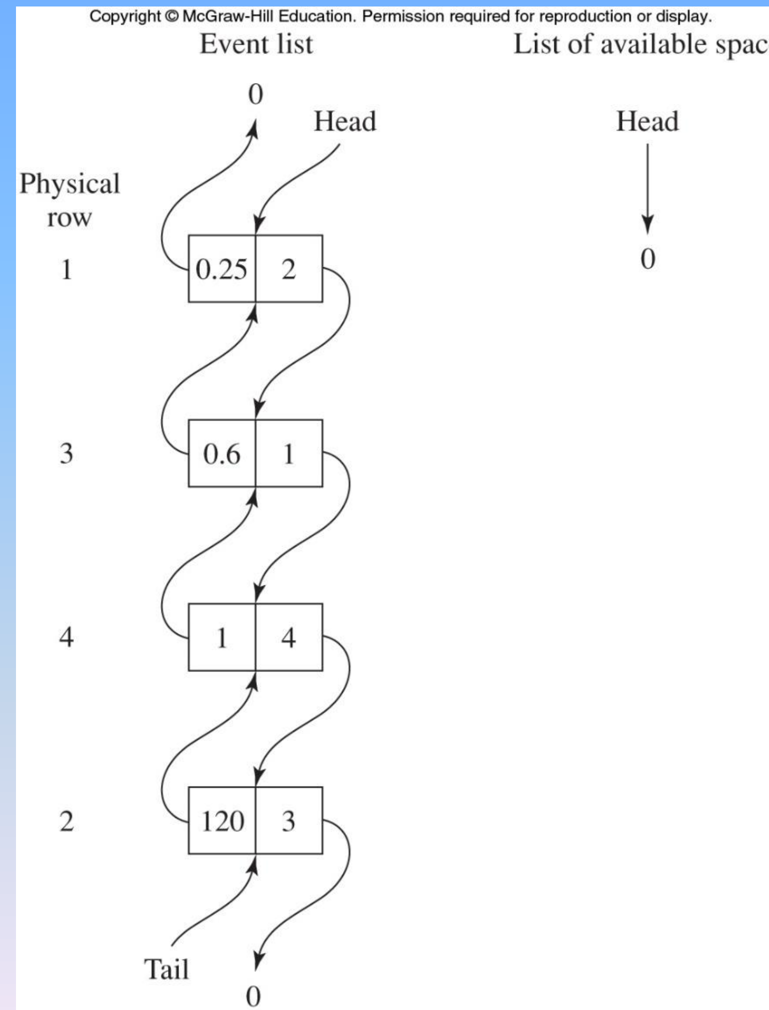


HOWARD UNIVERSITY

Jiang Li, Ph.D.
Department of Computer Science

Use of Lists in Simulations





HOWARD UNIVERSITY

Jiang Li, Ph.D.
Department of Computer Science

2.3 A Simple Simulation Language (Lib): simlib

- C-based
- Implements linked storage allocation
  - Provides 25 lists (25^{th} reserved for events)
  - Each list element has up to 10 float attributes
- Includes 19 functions
  - Each designed to perform a frequently-occurring simulation activity

Jiang Li, Ph.D.
Department of Computer Science

Functions in simlib (1)

- init_simlib()
- list_file(option, list_index)
  - Insert a record (in predefined transfer[]) into list
  - option: FIRST,LAST,INCREASING,DECREASING
- list_remove(option, list_index)
  - option: FIRST,LAST
- timing()
  - Update sim_time to next event
  - Maintain event list

Functions in simlib (2)

- event_schedule(event_time, event_type)
- event_cancel(event_type)
  - Cancel the first event of event_type
- sampst(value,var_index)
  - 20 sampst variables
  - Provide mean/num. of values/max/min when called with negative var_index

Functions in simlib (3)

- timest(value,var_index)
  - 20 timest variables
  - Provide time avg./max/min when called with negative var_index
- filest(list_index)
  - Provide time-avg./max/min number of records in list
- out_sampst(file,low_var_index, high_var_index)
  - Write summary statistics to file for sampst variables from low index to high index

Jiang Li, Ph.D.
Department of Computer Science

Functions in simlib (4)

- out_timest(file,low_var_index, high_var_index)
- out_filest(file,low_list_index,high_list_index)
- expon(mean,stream)
  - stream: select a stream of random numbers, use two different streams for different sets of random values (e.g. inter-arrival time and service time)
- random_integer(prob_dist[], stream)
  - Specify cumulative probability of 1 ~ 25 in prob_dist[]
- uniform(a,b,stream)

Functions in simlib (5)

- erlang(m,mean,stream)
- lcgrand(stream)
  - Return a uniformly distributed r.v. in [0, 1]
- lcgrandst(random_seed, stream)
  - Sets random seed for stream
- lcgrandgt(stream)
  - Get the next random underlying integer in stream

2.4 Single-Server Queuing Simulation with simlib

- Identify the events
  - Arrival: type 1 event
  - Departure: type 2 event
- Define the simlib lists and the attributes in their records

List	Attribute 1	Attribute 2
1, queue	Time of arrival to queue	—
2, server	—	—
25, event list	Event time	Event type

Single-Server Queuing Simulation with simlib

- Identify all sampst and timest variables used

sampst variable number	Meaning
1	Delays in queue

- Identify separate random number streams for interarrival times and service times

Stream	Purpose
1	Interarrival times
2	Service times

Jiang Li, Ph.D.
Department of Computer Science

Single-Server Queuing Simulation with simlib


```
/* 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);
```


Jiang Li, Ph.D.
Department of Computer Science

Single-Server Queuing Simulation with simlib


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

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



```
main()  /* Main function. */
{
    /* Open input and output files. */

    infile  = fopen("mm1smlb.in",  "r");
    outfile = fopen("mm1smlb.out", "w");

    /* Read input parameters. */

    fscanf(infile, "%f %f %d", &mean_interarrival, &mean_service,
           &num_delays_required);

    /* Write report heading and input parameters. */

    fprintf(outfile, "Single-server queueing system using simlib\n\n");
    fprintf(outfile, "Mean interarrival time%11.3f minutes\n\n",
            mean_interarrival);
    fprintf(outfile, "Mean service time%16.3f minutes\n\n", mean_service);
    fprintf(outfile, "Number of customers%14d\n\n\n", num_delays_required)

    /* Initialize simlib */

    init_simlib();

    /* Set maxatr = max(maximum number of attributes per record, 4) */

    maxatr = 4;  /* NEVER SET maxatr TO BE SMALLER THAN 4. */

    /* Initialize the model. */

    init_model();

    /* Run the simulation while more delays are still needed. */

    while (num_custs_delayed < num_delays_required) {

        /* Determine the next event. */

        timing();

        /* Invoke the appropriate event function. */

        switch (next_event_type) {
            case EVENT_ARRIVAL:
                arrive();
                break;
            case EVENT_DEPARTURE:
                depart();
                break;
        }
    }

    /* Invoke the report generator and end the simulation. */

    report();

    fclose(infile);
    fclose(outfile);

    return 0;
}
```

Single-Server Queuing Simulation with simlib


```
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);
    }
}
```


Jiang Li, Ph.D.
Department of Computer Science

Single-Server Queuing Simulation with simlib


```
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);
    }
}
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);
}
```


Jiang Li, Ph.D.
Department of Computer Science

Single-Server Queuing Simulation with simlib


```
Single-server queueing system using simlib

Mean interarrival time        1.000 minutes

Mean service time             0.500 minutes

Number of customers            1000


Delays in queue, in minutes:

 SAMPST                              Number
variable                               of
 number         Average              values          Maximum          Minimum
-------------------------------------------------------------------------------

    1        0.5248728E+00    0.1000000E+04    0.5633087E+01    0.0000000E+00
-------------------------------------------------------------------------------


Queue length (1) and server utilization (2):

  File           Time
 number         average           Maximum          Minimum
-------------------------------------------------------------

    1        0.5400774E+00    0.8000000E+01    0.0000000E+00

    2        0.5106925E+00    0.1000000E+01    0.0000000E+00
-------------------------------------------------------------


Time simulation ended:      971.847 minutes
```


Jiang Li, Ph.D.
Department of Computer Science

Single-Server Queuing Simulation with simlib

- Output is different from the output obtained for the same system in Chapter 1
  - Average delay in queue changed by over 20 percent
    - Reason: now using dedicated random number stream
  - Both programs are correct

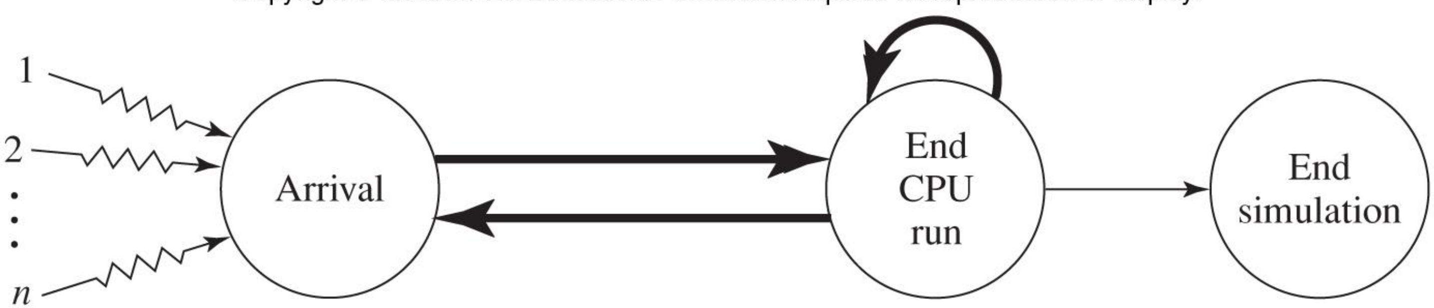
Jiang Li, Ph.D.
Department of Computer Science

2.5 Time-Shared Computer Model

- Simulate a model of a time-shared computer facility
  - Single CPU with *n* terminals
  - Operators send jobs to CPU from terminal after thinking
    - No more than one outstanding job per terminal
  - Arriving jobs join a single queue
  - Jobs are served in a round-robin manner
    - Not FIFO
- Want to know how many terminals can a system have while keeping average response time <= 30 seconds.
  - Also estimate average amount of jobs in queue, CPU utilization

Jiang Li, Ph.D.
Department of Computer Science

Event Graph



Time-Shared Computer Model

- Events for this model

Event description	Event type
Arrival of a job to the CPU from a terminal, at the end of a think time	1
End of a CPU run, when a job either completes its service requirement or has received the maximum processing quantum *q*	2
End of the simulation	3

- Lists of records

List	Attribute 1	Attribute 2
1, queue	Time of arrival of job to computer	Remaining service time
2, CPU	Time of arrival of job to computer	Remaining service time after the present CPU pass (negative if the present CPU pass is the last one needed for this job)
25, event list	Event time	Event type

Jiang Li, Ph.D.
Department of Computer Science

Time-Shared Computer Model

- One discrete-time statistic of interest
  - Response times

sampst variable number	Meaning
1	Response times

- Two types of random variables

Stream	Purpose
1	Think times
2	Service times

Time-Shared Computer Model

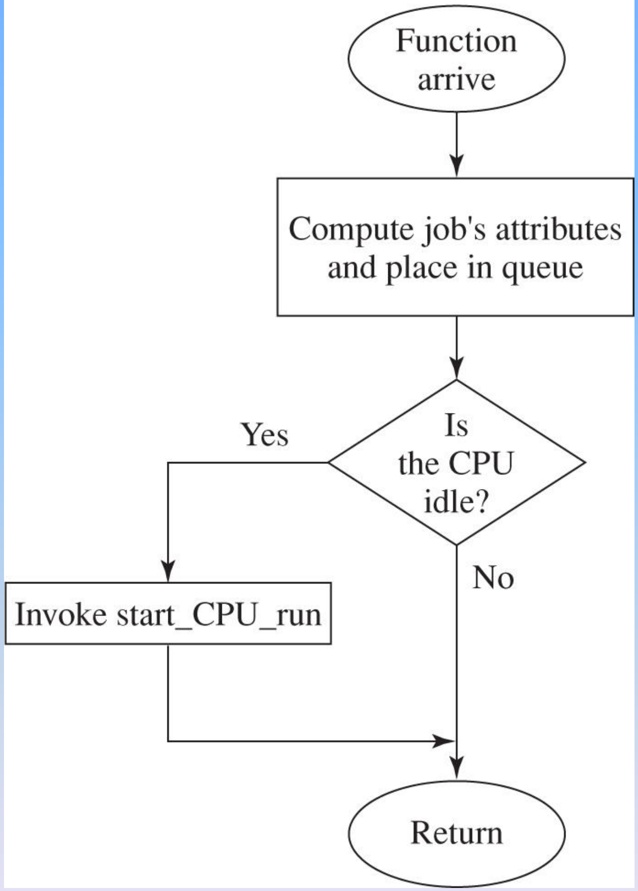

```
/* External definitions for time-shared computer model. */

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

#define EVENT_ARRIVAL          1  /* Event type for arrival of job to CPU. */
#define EVENT_END_CPU_RUN      2  /* Event type for end of a CPU run. */
#define EVENT_END_SIMULATION   3  /* Event type for end of the simulation. */
#define LIST_QUEUE             1  /* List number for CPU queue. */
#define LIST_CPU               2  /* List number for CPU. */
#define SAMPST_RESPONSE_TIMES  1  /* sampst variable for response times. */
#define STREAM_THINK           1  /* Random-number stream for think times. */
#define STREAM_SERVICE         2  /* Random-number stream for service times. */

/* Declare non-simlib global variables. */

int   min_terms, max_terms, incr_terms, num_terms, num_responses,
      num_responses_required, term;
float mean_think, mean_service, quantum, swap;
FILE  *infile, *outfile;

/* Declare non-simlib functions. */

void arrive(void);
void start_CPU_run(void);
void end_CPU_run(void);
void report(void);
```

Jiang Li, Ph.D.
Department of Computer Science

Time-Shared Computer Model


```
main()  /* Main function. */
{
    /* Open input and output files. */

    infile  = fopen("tscomp.in",  "r");
    outfile = fopen("tscomp.out", "w");

    /* Read input parameters. */

    fscanf(infile, "%d %d %d %d %f %f %f %f",
           &min_terms, &max_terms, &incr_terms, &num_responses_required,
           &mean_think, &mean_service, &quantum, &swap);

    /* Write report heading and input parameters. */

    fprintf(outfile, "Time-shared computer model\n\n");
    fprintf(outfile, "Number of terminals%9d to%4d by %4d\n\n",
            min_terms, max_terms, incr_terms);
    fprintf(outfile, "Mean think time  %11.3f seconds\n\n", mean_think);
    fprintf(outfile, "Mean service time%11.3f seconds\n\n", mean_service);
    fprintf(outfile, "Quantum          %11.3f seconds\n\n", quantum);
    fprintf(outfile, "Swap time        %11.3f seconds\n\n", swap);
    fprintf(outfile, "Number of jobs processed%12d\n\n\n",
            num_responses_required);
    fprintf(outfile, "Number of      Average         Average");
    fprintf(outfile, "       Utilization\n");
    fprintf(outfile, "terminals   response time  number in queue     of CPU");

    /* Run the simulation varying the number of terminals. */

    for (num_terms = min_terms; num_terms <= max_terms;
         num_terms += incr_terms) {

        /* Initialize simlib */

        init_simlib();

        /* Set maxatr = max(maximum number of attributes per record, 4) */

        maxatr = 4;  /* NEVER SET maxatr TO BE SMALLER THAN 4. */

        /* Initialize the non-simlib statistical counter. */

        num_responses = 0;

        /* Schedule the first arrival to the CPU from each terminal. */

        for (term = 1; term <= num_terms; ++term)
            event_schedule(expon(mean_think, STREAM_THINK), EVENT_ARRIVAL);

        /* Run the simulation until it terminates after an end-simulation event
           (type EVENT_END_SIMULATION) occurs. */

        do {

            /* Determine the next event. */

            timing();

            /* Invoke the appropriate event function. */

            switch (next_event_type) {
                case EVENT_ARRIVAL:
                    arrive();
                    break;

                case EVENT_END_CPU_RUN:
                    end_CPU_run();
                    break;
                case EVENT_END_SIMULATION:
                    report();
                    break;
            }

            /* If the event just executed was not the end-simulation event (type
               EVENT_END_SIMULATION), continue simulating.  Otherwise, end the
               simulation. */

        } while (next_event_type != EVENT_END_SIMULATION);
    }

    fclose(infile);
    fclose(outfile);

    return 0;
```

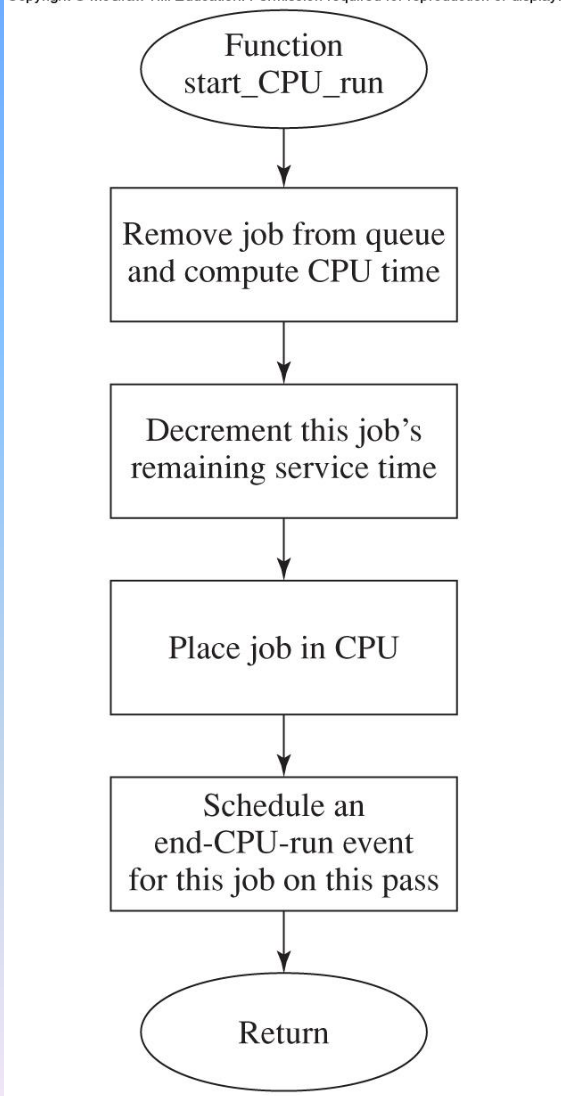
Time-Shared Computer Model




```
void arrive(void)  /* Event function for arrival of job at CPU after think
                      time. */
{

    /* Place the arriving job at the end of the CPU queue.
       Note that the following attributes are stored for each job record:
            1. Time of arrival to the computer.
            2. The (remaining) CPU service time required (here equal to the
               total service time since the job is just arriving). */

    transfer[1] = sim_time;
    transfer[2] = expon(mean_service, STREAM_SERVICE);
    list_file(LAST, LIST_QUEUE);

    /* If the CPU is idle, start a CPU run. */

    if (list_size[LIST_CPU] == 0)
        start_CPU_run();
}
```


Jiang Li, Ph.D.
Department of Computer Science

Time-Shared Computer Model

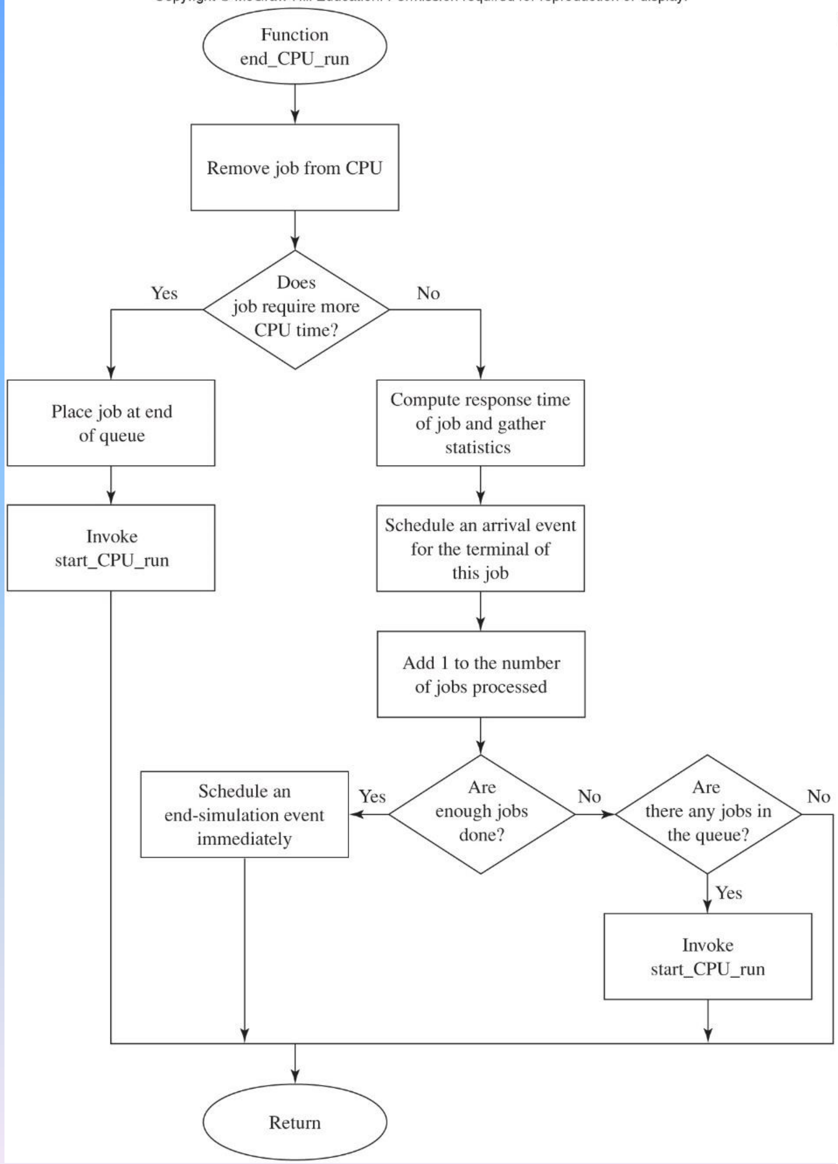



```
void start_CPU_run(void)  /* Non-event function to start a CPU run of a job. */
{
    float run_time;

    /* Remove the first job from the queue. */

    list_remove(FIRST, LIST_QUEUE);

    /* Determine the CPU time for this pass, including the swap time. */

    if (quantum < transfer[2])
        run_time = quantum + swap;
    else
        run_time = transfer[2] + swap;

    /* Decrement remaining CPU time by a full quantum.  (If less than a full
       quantum is needed, this attribute becomes negative.  This indicates that
       the job, after exiting the CPU for the current pass, will be done and is
       to be sent back to its terminal.) */

    transfer[2] -= quantum;

    /* Place the job into the CPU. */

    list_file(FIRST, LIST_CPU);

    /* Schedule the end of the CPU run. */

    event_schedule(sim_time + run_time, EVENT_END_CPU_RUN);
}
```

HOWARD UNIVERSITY

Jiang Li, Ph.D.
Department of Computer Science

Time-Shared Computer Model



Copyright © McGraw-Hill Education. Permission required for reproduction or display.

```
void end_CPU_run(void)  /* Event function to end a CPU run of a job. */
{
    /* Remove the job from the CPU. */

    list_remove(FIRST, LIST_CPU);

    /* Check to see whether this job requires more CPU time. */

    if (transfer[2] > 0.0) {

        /* This job requires more CPU time, so place it at the end of the queue
           and start the first job in the queue. */

        list_file(LAST, LIST_QUEUE);
        start_CPU_run();
    }

    else {

        /* This job is finished, so collect response-time statistics and send it
           back to its terminal, i.e., schedule another arrival from the same
           terminal. */

        sampst(sim_time - transfer[1], SAMPST_RESPONSE_TIMES);

        event_schedule(sim_time + expon(mean_think, STREAM_THINK),
                       EVENT_ARRIVAL);

        /* Increment the number of completed jobs. */

        ++num_responses;

        /* Check to see whether enough jobs are done. */

        if (num_responses >= num_responses_required)

            /* Enough jobs are done, so schedule the end of the simulation
               immediately (forcing it to the head of the event list). */

            event_schedule(sim_time, EVENT_END_SIMULATION);

        else

            /* Not enough jobs are done; if the queue is not empty, start
               another job. */

            if (list_size[LIST_QUEUE] > 0)
                start_CPU_run();
    }
}
```

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

    fprintf(outfile, "\n\n%5d%16.3f%16.3f%16.3f", num_terms,
            sampst(0.0, -SAMPST_RESPONSE_TIMES), filest(LIST_QUEUE),
            filest(LIST_CPU));
}
```

Time-Shared Computer Model

- Output
  - Congestion worsens as number of terminals rises
  - System could handle about 60 terminals
    - Response time degrades to 30 seconds at that point


```
Time-shared computer model

Number of terminals          10 to   80 by    10

Mean think time          25.000 seconds

Mean service time         0.800 seconds

Quantum                   0.100 seconds

Swap time                 0.015 seconds

Number of jobs processed          1000
```

Number of terminals	Average response time	Average number in queue	Utilization of CPU
10	1.324	0.156	0.358
20	2.165	0.929	0.658
30	5.505	4.453	0.914
40	12.698	12.904	0.998
50	24.593	23.871	0.998
60	31.712	32.958	1.000
70	42.310	42.666	0.999
80	47.547	51.158	1.000

2.8 Efficient Event-List Management

- For complex systems with a large number of events
  - Much of the computer time is used on event-list processing
- One solution: use more efficient data structure and search technique
  - Median-pointer linked list
  - Other approaches
    - Heaps and trees, calendar or ladder queues, etc.