- # What is an Event

  - It is an action that occurs at an instant in time and changes the state of a system. For example, in our homework, the arrival of a new job is an event. When it occurs the state of the system changes and we have to take a number of actions, such as: if the server is busy, the job waits in the server's queue, otherwise it starts service immediately on the server. Another event is the completion of service at the server. At this completion event, we decide where the job goes next. This is a routing decision for a job like a switch. The routing decision is an integral part of what is done for the event when the job leaves the preceding server. Note also that there is a distinction between "Event type" and "Event instance". "Arrival of a job" is an Event type. "Arrival of job 7 at …" is an Event instance.
    Events are associated with jobs. For each job there is a unique event describing what will happen next to that job.

- # What is Discrete Event Simulation (DES)

  - DES is a way of simulating [i.e. of analyzing models of] systems. It can be done for systems whose behavior can be expressed in terms of what they do in correspondence to a certain number of event types. For example, the system in your homework can be described in terms of what it does when there is an Arrival event, or a Service completion event, or the task termination.
    Discrete event simulation is used extensively when trying to predict or analyze the behavior of computer systems. Systems such as chemical plants, where the process being carried out is continuous [i.e. does not have discrete times where the state changes], cannot be simulated this way.

- # Simulation and Global Time

  - The simulation begins at some time G1, equal to INITTIME, usually 0. The first event that can occur is the arrival of the first job. When will this arrival take place? For simplicity we assume that it occurs exactly at G1. Now that the first event, the first arrival, has occurred, what will happen next? Two events are going to happen:
    - There will be a next arrival. When? At time G1 plus a random value A2 between AMIN and AMAX.
    - The first job will terminate service at SERVER_0 after a random time interval S0, that is in the range S_0MIN .. S_0MAX.

    These two events are going to happen, respectively, at time G2 = G1+A2 and G1+S0. Say that G2 is the smaller of these two values [if the two values are equal one is chosen in some fashion, possibly random]. Then the next event occurs at time G2, and the global time jumps from G1 to G2. No change of state is possible between two successive events.

    This is how the simulation proceeds. We consider what events are going to happen next, we choose as “next event” the one that will occur first, and we advance the global clock to the time of this next event. When an event occurs, we do all the activities implied by that event.

Event Queue and Server Queues

We have seen that we keep in a queue the jobs whose events are expected to occur next. And we have seen that jobs wait in a queue to be served by a server. BUT the two kinds of queues are very different.

The queue for jobs whose events are expected to occur next is called the **Event Queue** or **Agenda**, and it is a priority queue where job events are kept in sorted order for ascending timestamps of expected occurrence of their events. New events will be enqueued in the agenda at the place indicated by the time stamp of the event. We dequeue the event with the smallest time stamp.

Server queues are FIFOs where we store "jobs" in the order of their arrival to that server.

Generating Random Numbers

Random numbers are used extensively in programming. Thus, each system, application, and language have some means to generate random number. The random number generators that we usually find produce numbers that are uniformly distributed in some specified range. For example, we may find a generator of real numbers uniformly distributed in the interval (0.0, 1.0) or a generator of integer numbers uniformly distributed in the range (0, RAND_MAX), where RAND_MAX is some specified value, usually 2^{31}-1.

If in Unix you execute

- **man rand**

you will be told about the functions **rand**, **rand_r**, and **srand**.
If you execute

- **man random**

you will be told about the functions **random**, **srandom**, **initstate**, and **setstate**.
Here is an example of use of random and initstate:

```
/* myran.c - Testing random and initstate */

#include < stdio.h >
#define SIZE 1000

void initrandom(unsigned seed){
  /* This function must be called before we can generate random
numbers.
   * You can choose as seed any number, say, 1, 123, 357, ...
   */
```

```
#define RANDSIZE 256
  static char randstate[RANDSIZE];
  initstate(seed, randstate, RANDSIZE);
}

int myrandom(int low, int high){
  /* It returns a random number in the interval [low,high]
   */
  return (low + random()%(high-low+1)); /* this assumes that all the
bits
                        of the number generated by random are
                        equally random, which is not true in
                        old implementations of random */
}

int main() {
  /* Generates and prints SIZE random numbers in the interval
[10,20].
   * Then prints out their average (it should be close to 15).
   */
    int i;
    int table[SIZE];
    int sum = 0;

    initrandom(1357);
    for (i=0; i<SIZE; i++) {
      table[i] = myrandom(10,20);
      sum += table[i];
      printf("%2d\n", table[i]);
    }
    printf("the average is %f\n", sum/(float)SIZE);
    exit(0);
}
```

If one is interested in generating random real numbers uniformly distributed in range 0.0 .. 1.0, assuming the inclusion of stdlib.h , one can use the function

```
  /* returns a random real uniformly distributed in range 0 .. 1 */
  double rand01(void)
  {
     return ((random())/(double)RAND_MAX);
  }
```