19.2.1 Simulating the M/M/1 Queue and Some Extensions

We now consider a number of examples in more detail, beginning with the simple $M/M/1$ queue. This queue was treated analytically in Part III of this text, and although it is rather straightforward to simulate (and indeed to treat analytically), it is adequate for illustrating all the most important aspects of designing and implementing a simulation. To recap, the $M/M/1$ queue is a single server queue with first-come first-served scheduling of customers. The arrival process is Poisson with

parameter λ, i.e., interarrival times have an exponential distribution with mean $1/\lambda$; the service time is also exponentially distributed and the mean service time is $1/\mu$.

The different variables that we shall employ are now specified:

- A single integer n is sufficient to completely characterize a state of the system and thus is taken as the state descriptor vector.
- There are only two events: the arrival of a new customer to the system and the departure of a customer who has just completed service.
- Integer variables n_a and n_d count the number of arrivals and departures, respectively.
- We shall run the simulation until a total of N customers have been served and then compute the mean interarrival time, the mean service time and the mean time spent waiting in the queue.
- The variable t will be used to denote the internal clock time. The variables t_a and t_d denote the scheduled times of the next arrival and the next departure.
- We use t_λ and t_μ to denote generated interarrival times and service times, respectively. During the simulation run, tot_λ and tot_μ are running totals of arrival times and service times.
- Since we require the mean time spent in the system, we keep the arrival time of each customer. Element i of array T will hold the arrival time of customer i, for $i \leq N$. Subtracting $T[i]$ from the time customer i departs gives customer i's total system time (the variable *response*). The average queueing time (variable *wait*) is found by subtracting the sum of all service times from the sum of all response times and dividing by N.
- At simulation initialization, we shall set $t = 0$ and $t_d = \infty$ where ∞ is taken to be a number so large that there is little possibility of t ever reaching this value during the simulation. The initialization will also generate the time of the first arrival and initialize t_a to this value.

The following actions must be performed by the program modules that handle events:

1. An arrival:
   - Advance the internal clock to the time of arrival.
   - An additional customer has arrived so modify system state n appropriately ($n = n+1$).
   - Increment n_a, the number of arrivals so far and, if $n_a \leq N$, set $T[n_a] = t$.
   - Generate t_λ, the time until the next arrival; compute the next arrival instant $t_a = t + t_\lambda$ and add t_λ into tot_λ.
   - If $n = 1$, signifying that the server had been idle prior to this arrival, generate t_μ, the service time for this customer, compute its departure time $t_d = t + t_\mu$ and add t_μ into tot_μ.
2. A departure:
   - Advance the internal clock to time of departure.
   - A customer has left so modify system state n appropriately ($n = n - 1$).
   - Increment n_d, the number of departures so far. Compute the time this departing customer spent in the system and add into *response*.
   - If $n = 0$, set $t_d = \infty$. Otherwise generate t_μ, the service time of the next customer; compute the next departure instant $t_d = t + t_\mu$ and add t_μ into tot_μ.

All that remains is to specify the main module. This module must

- Perform all initialization functions as described above.
- If the number of departures is less than N:
  – initiate an arrival event if $t_a < t_d$: otherwise initiate a departure event ($t_a \geq t_d$).
- Generate terminal statistics and print report.

Some Extensions to the *M/M/*1 Queue

1. Non-exponential interarrival and service time distributions.
   This can be handled quite easily. The previous code can be used with the exponential random number generator replaced by a random number generator for whatever type of distribution is required.
2. Limited waiting room ($M/M/1/K$).
   In this case, some customers will not be able to enter the queue and a counter is needed to keep track of the number of lost customers. Only the arrival event module needs to be modified. After the count of lost customers is incremented and the internal clock updated, the only remaining task is to generate the time of the next arrival, and update the sum of interarrival times.
3. Multiple servers (*M/M/c*).
   Both arrival and departure event modules need to be modified. If, upon an arrival, the number in the system is less than c, that arriving customer can go immediately into service—the condition $n == 1$ should be replaced with $n <= c$. Within the departure event module, the mechanism for computing the queueing time must be modified, since departures need not

   be in the same order as arrivals. One possibility is to use an additional array TS whose i^{th} elements keeps the time at which customer i enters service. Then $TS[i] - TA[i]$ is the time that customer i waits in the queue.
4. The *M/M/*1 queue with probability α of feedback.
   Only the departure event module will need to be changed. It is necessary to generate a uniformly distributed random number $u \in (0, 1)$ to determine whether a departure is fedback or not. If $u < \alpha$ then the system state must be left unaltered. Everything else remains unchanged.
5. Different customer classes with nonpreemptive scheduling.
   In this case, the system state must be replaced by a vector of length K where K is the maximum number of possible classes. At each arrival instant, the arrival module must first generate a uniformly distributed random number $u \in (0, 1)$ and use it to select the class of the next arrival. When this has been accomplished, it must then generate the time of the next arrival for a customer of that class and increment the system state according. If upon arrival the system is empty, a service time specific to the class of the arriving customer is generated and the arriving customer goes immediately into service.
   At each departure instant, the departure event module must select the next customer to enter service (highest priority first), modify the appropriate component of the state descriptor vector and generate the service time of the next customer of the same class, if such a customer is present. Since departures are not necessarily in the same order as arrivals, the comments made with respect to the *M/M/c* queue and the array *TS* also apply here.

11.1.4 Graphical Representations of Queues

There exist a number of ways of graphing queueing systems and these can frequently provide insight into the behavior of the system. For a FCFS, single server queue, one possibility is shown in Figure 11.6. We let C_n denote the n^{th} customer to enter the queueing system. If the service discipline is not FCFS, then C_n may not be the n^{th} customer to depart. We let τ_n denote the time at which C_n arrives in the system and t_n the interarrival time between the arrival of C_{n-1} and C_n (i.e., $t_n = \tau_n - \tau_{n-1}$). We shall assume that the t_n are drawn from a distribution such that $\text{Prob}\{t_n \leq t\} = A(t)$, i.e., independent of n, where $A(t)$ is a completely arbitrary interarrival time distribution. Similarly, we shall define x_n to be the service time for C_n and $\text{Prob}\{x_n \leq x\} = B(x)$, where $B(x)$ is a completely arbitrary (but independent) service time distribution. We are now in a position to graph $N(t)$, the number of customers in the system at time t. In Figure 11.6, w_n represents the waiting time (in queue) for C_n, and s_n represents the system time (queue plus service) for C_n. (Note that $s_n = w_n + x_n$).

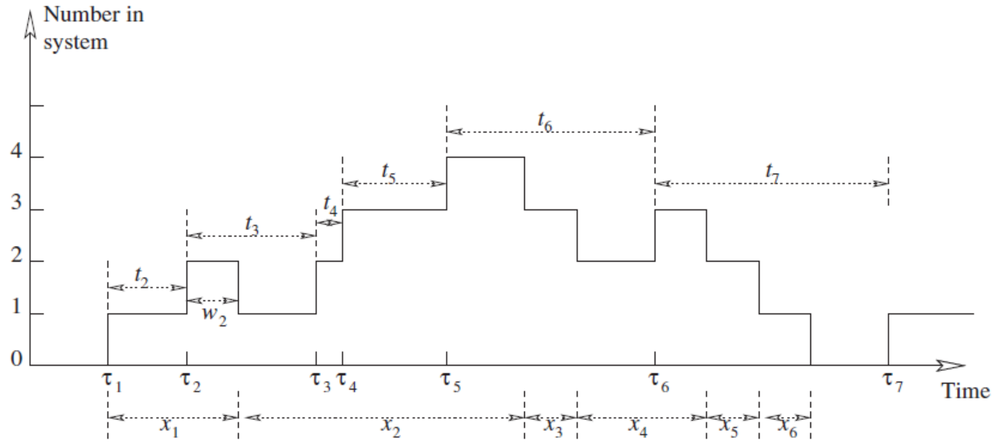

Figure 11.6. Graphical representation No. 1.

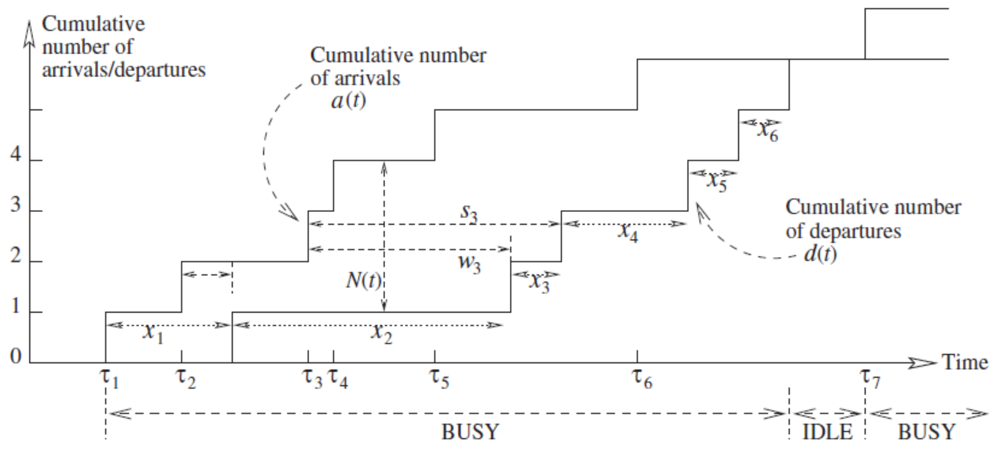

Figure 11.7. Graphical representation No. 2.

An alternative graphical representation is given in Figure 11.7. In this graph $N(t) = a(t) - d(t)$, where $a(t)$ denotes the number of arrivals in $[0, t)$ and $d(t)$ denotes the number of departures in this same time period.

There is also a double-time-axis graphical representation which is shown in Figure 11.8 (for the single-server, FCFS case). This particular variant is relatively easy to generalize to many servers and to disciplines other than FCFS.

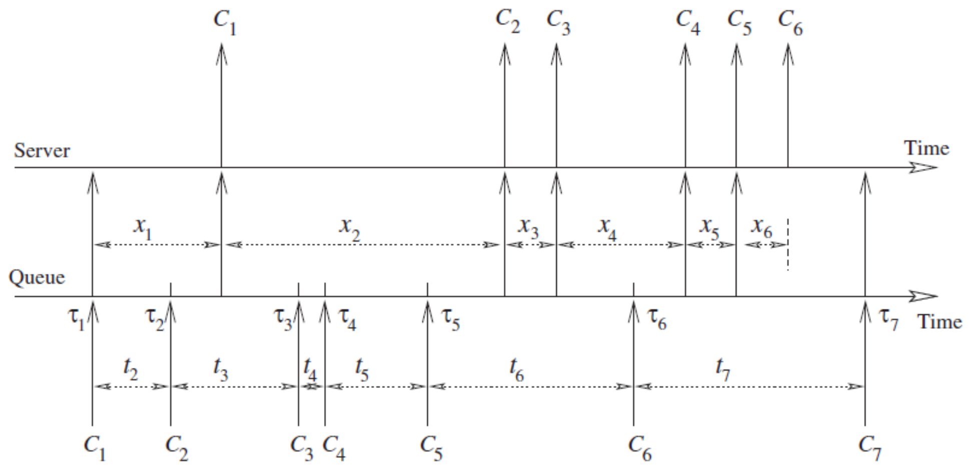

Figure 11.8. Double-time-axis notation.