Serial Model Execution

- Two primary approaches to executing a system model on a sequential: time slicing, event scheduling
- The key attribute of time slicing over the event scheduling method is in the way that time is incremented: through addition

Time Slicing

- Time slicing involves incrementing time using a global clock by adding a certain amount to the clock value over time
- A virtual clock is updated in intervals of some arbitrary size which can change throughout the simulation
- The system is normally defined as a group of functions each of which must be evaluated at the end of each interval.
- The time-slicing method can be inefficient in circumstances where the values of inputs and outputs do not change regularly.
- The time-slicing method is appropriate for continuous system with continually varying inputs, state values, and the outputs.

Algorithm for time-slice execution

Begin Main
  Initialize state variables
  Loop through all functions
    While not end of simulation
      UpdateFunctions
    End while
End Main
Procedure UpdateFunctions
  For all functions
    Switch on function type
      Case type 1: apply function type 1 to state and inputs
Case type 2: apply function type 2 to state and inputs

Case type n: apply function type n to state and inputs

End For

Increment time using time slice
End UpdateFunctions

Event Scheduling

- Event scheduling is defined as executing code that contains an event loop where events are posted and then checked according to minimum time
- The event on the future event list with the minimum time tag executes first
- Event scheduling can be likened to a “calendar” or an “alarm clock” in that events are scheduled to occur at a specific date or time

Example of Event Scheduling: Single-Server Queue

- The next arrival time is always relative to the last arrival time
- The next departure time depends on whether there is an entity already in the system besides the entity that is departing
  - If there is no other entity, the next departure time is relative to the next arrival time
  - If there is another entity, the next departure time is relative to the last departure time

\[
D_0 = 0 \\
D_i = \max(D_{i-1}, A_i) + \sigma_i
\]

$D_i$ refers to the departure time of the customer $i$, $A_i$ is the arrivals, and $\sigma_i$ represents the time to service customer $i$. Table 1 displays the example of a single-server queue and five customers.

<table>
<thead>
<tr>
<th>i</th>
<th>$A_i$</th>
<th>$\sigma_i$</th>
<th>$D_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>4</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>22</td>
<td>1</td>
<td>23</td>
</tr>
</tbody>
</table>
A simple piece of code for basic scheduling follows. Note the following variable definitions: $ua$ and $us$ are mean times for interarrival and service; $ta$ and $td$ are time of next arrival and time of next departure; $te$ is “end of simulation” time; $time$ is the global simulation clock time; $n$ is the number of entities in the system; and $dist()$ is a random number generator for an arbitrary probability distribution.

```c
float ua, us, te, ta, td, time;
int n;
n=0; ta=0.0; td = ta + dist(us); time=0.0;
while (time < te) {
  if (ta < td) {
    /* arrival event */
    time = ta; n++; ta = time + dist(ua); }
  else { /* departure event */
    time = td; n--;
    if (n==0) td = ta + dist(us);
    else td = td + dist(us);
  } /* end if */
} /* end while */
```