CPU Scheduling

Chapter 9
CPU Scheduling

- We concentrate on the problem of scheduling the usage of a single processor among all the existing processes in the system.
- The goal is to achieve:
  - High processor utilization
  - High throughput
    - number of processes completed per unit time
  - Low response time
    - time elapse from the submission of a request to the beginning of the response
Classification of Scheduling Activity

- **Long-term**: which process to admit
- **Medium-term**: which process to swap in or out
- **Short-term**: which ready process to execute next
Queuing Diagram for Scheduling
Long-Term Scheduling

- Determines which programs are admitted to the system for processing
- Controls the degree of multiprogramming
- If more processes are admitted
  - less likely that all processes will be blocked
    - better CPU usage
  - each process has less fraction of the CPU
- The long term scheduler may attempt to keep a mix of processor-bound and I/O-bound processes
Medium-Term Scheduling
- Whether to add a process to those that are at least partially in main memory and therefore available for execution
- Part of the swapping function

Short-Term Scheduling
- Determines which process is going to execute next (also called CPU scheduling)
- The short-term scheduler is known as the dispatcher
A scheduling decision takes place when an event occurs:
- That leads to the blocking of a current process
- That provides an opportunity to preempt a currently running process

Possible events (a quick recall):
- Clock interrupts
- I/O interrupts
- Operating system calls
- Signals (e.g., for synchronization)
Short-Term Scheduling

- Implementation
  - A process dispatcher / scheduler will select a process (or a thread when threading is used) from the “Ready” queue for execution
  - Processes are scheduled according to a scheduling algorithm
Process Behavior

CPU bursts: the amount of time the process uses the processor before it is no longer ready

- Processes usually alternate bursts of computation on a CPU with waiting for completion of I/O Requests
- Processes use a CPU in bursts and usually do not compute continuously

Types of CPU bursts:
- Long bursts -- process is CPU bound (i.e. array work)
- Short bursts -- process I/O bound (i.e. vi)
Process Behavior

- Alternating bursts of processing on the CPU and waiting for I/O
  - CPU is not always used by a process
  - Scheduler can schedule another process

![Graph showing frequency vs. burst duration (milliseconds)]
Scheduler Decision

- Scheduling can be preemptive or non-preemptive

- Non-preemptive
  - Once a process is scheduled, it continues to execute on the CPU, until it is finished
  - It releases the CPU voluntarily (cooperative scheduling behavior, then goes back to Ready queue)
  - It blocks due to I/O interrupts or because it waits for signal from another process
  - No scheduling decision is made during clock interrupts (at the end of each time slice), process is resumed after this interruption
Scheduler Decision

- **Preemptive:**
  - A scheduled process executes, until its time slice is used up
  - Clock interrupt returns control of CPU back to scheduler at end of time slice
    - Current process is suspended and placed in the Ready queue
    - New process (even with the same priority) is selected from Ready queue and executed on CPU
  - Used by most operating systems
Scheduling Criteria

- **User-oriented**
  - *Turnaround Time*:Elapsed time from the submission of a process (in the ready queue) to its completion
  - *Response Time*: Elapsed time from the submission of a request to the beginning of response
  - *Waiting time*: Total amount of time a process has been waiting in the Ready queue

- **System-oriented**
  - *Processor utilization*: Keep the CPU as busy as possible
  - *Fairness* (consider the starvation of a process)
  - *Throughput*: number of processes completed per unit time
## Scheduling Criteria

<table>
<thead>
<tr>
<th></th>
<th>User-Oriented</th>
<th>System-Oriented</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Performance</strong></td>
<td>Response Time</td>
<td>Throughput</td>
</tr>
<tr>
<td></td>
<td>Turnaround Time</td>
<td>Processor Utilisation</td>
</tr>
<tr>
<td></td>
<td>Waiting Time</td>
<td></td>
</tr>
<tr>
<td><strong>Other</strong></td>
<td>Predictability</td>
<td>Fairness</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Priorities</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Resource Balance</td>
</tr>
</tbody>
</table>
Scheduling Metrics

- We want to
  - Maximize CPU utilization
  - Maximize throughput
  - Minimize turnaround time
  - Minimize waiting time
  - Minimize response time

Scheduling criteria can conflict with each other.
Scheduling Algorithms

- First Come First Serve scheduling (FIFO)
- Shortest Job first scheduling
- Priority scheduling
- Round-Robin Scheduling
- Multilevel Queue scheduling
- Multilevel Feedback Queue scheduling
Scheduling Algorithms

- Arrival time $t_a$: time when process became “ready”
- Wait time $T_w$: time spent ready waiting for CPU
- Service time $T_s$: time spent executing on the CPU
- Turnaround time $T_r$: total time spent waiting + executing

![Diagram showing arrival, execution, wait, and service times](image)

$T_r = T_w + T_s$
Running Example

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Service Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>
First Come First Serve (FCFS)

- Selection function: the process that has been waiting the longest in the ready queue (hence, FCFS)
- Decision mode: **non-preemptive**: a process run until it blocks itself

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Service Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>

Gantt Chart
First Come First Serve (FCFS)

- Processes are scheduled for execution according to their arrival sequence.
- When a process becomes ready, it is put at the end of the queue.
- No other considerations such as how long a process may need to finish its execution (and other priorities).
- Other processes with shorter execution time may have to wait, also impact on the responsiveness of a computer system.
FCFS: Critique

- FCFS favors long CPU-bound processes
  - Short CPU-bound processes have to wait until long process completes
  - They may have to wait even when I/O operations are completed (poor device utilization)

- Disadvantages
  - Average waiting time is highly variable
    - Short jobs may wait behind large ones
  - Lack of preemption is not suited in a time sharing environment
Computing Performance Metrics

- Arrival time $t_a$: time when process became “ready”
- Wait time $T_w$: time spent ready waiting for CPU
- Service time $T_s$: time spent executing on the CPU
- Turnaround time $T_r$: total time spent waiting + executing

\[ T_r = T_w + T_s \]
Shortest Process Next (SPN)

- **Non-preemptive** scheduling policy that tries to schedule the “shortest” process first
- Process with shortest expected processing time is selected next

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Service Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>

![Graph showing process scheduling]

**Graph:**
- Process A starts at 0, lasts until 3.
- Process B starts at 2, lasts until 8.
- Process C starts at 4, lasts until 5.
- Process D starts at 6, lasts until 14.
- Process E starts at 8, lasts until 20.

---

23
Shortest Process Next (SPN)

- Process with shortest expected (total) service time jumps ahead of longer processes

- Difficulty
  - The necessity to estimate the required CPU service time of each process
    - For batch jobs: past experience, specified manually
    - For interactive jobs: analyze bursts of CPU time
Shortest Process Next (SPN): Critique

- Possibility of **starvation** for longer processes as long as there is a steady supply of shorter processes.
- Lack of preemption is not suited in a time sharing environment.
- SPN implicitly incorporates priorities: shortest jobs are given preferences.
- The algorithm penalizes longer jobs.
Highest Response Ratio Next (HRRN)

- Is a **non-preemptive** scheduling policy based on a ratio between time spent waiting and expected service time of a process

\[
Ratio = \frac{time \ spent \ waiting + expected \ service \ time}{expected \ service \ time}
\]

- Avoids starvation with **aging**
  - Accounts for the age of the process
  - While shorter jobs are favored, aging eventually increases the ratio for longer running processes that pushes them past the shorter processes
Recall: Non-preemptive Scheduling

- First Come First Serve scheduling (FIFO)
- Shortest Process Next scheduling (SPN)
- Highest Response Ratio Next (HRRN)
Shortest Remaining Time (SRT)

- **Preemptive** version of the SPN scheduling policy
  - Scheduler tries to select the process whose remaining execution time is the shortest
- Same selection function as SPN (with a different priority criteria)
  - SPN: based on total execution time
  - SRT: based on remaining execution time
Shortest Remaining Time (SRT)

- **Preemptive** version of Shortest Process Next (SPN)
- At each time point, must estimate remaining execution time and choose the shortest

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Service Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>

Timeline:
- B arrives
- C arrives
- D arrives
- E arrives
Shortest Remaining Time (SRT)

- Process currently executing on CPU is preempted if a job with shorter estimated CPU time becomes ready.

- Better turnaround performance
  - Shorter job is given *immediate* preference to a longer running job.
  - Reduces waiting times.
Priority Scheduling

- A priority number is associated with each process
- The CPU is allocated to the process with the highest priority
- Preemptive / non-preemptive form of priority scheduling
  - SPN is a non-preemptive priority scheduling algorithm where priority is the execution time
  - SRT is a preemptive priority scheduling algorithm where priority is the remaining time
Priority Scheduling

- **Starvation**
  - Low-priority processes may never execute
  - Both preemptive and non-preemptive scheduler can suffer from starvation

- **Solution to Starvation**
  - Introduce concept of “aging”: as time progresses, increase the priority of the process
  - E.g., Priority is based on estimated CPU time and waited time so far
Round-Robin (RR, Time Slicing)

- Selection function: same as FCFS
- Decision mode: preemptive
- Time quantum
  - a process is allowed to run with a fixed time slice of CPU, called quantum (typically from 10 to 100 ms)
  - After time quantum has elapsed, the process is preempted and added to the end of the Ready queue
  - Processes scheduled in a cyclical fashion (always same sequence of processes) – round robin
Use **preemption** based on a clock (i.e., quantum)

Also called time slicing, because each process is given a slice of time before being preempted

### Round-Robin (RR, Time Slicing)

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Service Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>

Need to track the process order in the ready queue at each time point.
Round-Robin (RR, Time Slicing)

- **Fairness**: Each process gets an equal time quantum
  - If the time quantum is $q$ units and if there are $n$ processes in the Ready queue, each process gets $1/n$ of the CPU time, and...
  - No process waits more than $(n-1)q$ time units: After the other $(n-1)$ processes are executed (with $q$ time units each), the last process will be executed for sure.
Round-Robin (RR, Time Slicing)

- **Performance**
  - If $q$ is large: round-robin becomes FCFS
  - If $q$ is small: fair, starvation-free, high responsiveness
  - $q$ still has to be considerably larger than the time needed for a context switch, otherwise the dispatcher consumes more CPU time than the actual processes it dispatches

- **Disadvantage:** I/O-bound processes are overtaken by CPU-bound processes
  - I/O intensive processes block early for I/O, are not using their full time quantum, will be overtaken by CPU intensive processes
Recall: CPU Scheduling

Non-preemptive:
- First Come First Serve scheduling (FIFO)
- Shortest Process Next scheduling (SPN)
- Highest Response Ratio Next (HRRN)

Preemptive:
- Shortest Remaining Time (SRT)
- Round Robin (RR)
Multilevel Queue Scheduling

- Multiple queues are used to hold processes with different priorities
- Processes always go back to the same queue

Example

- Highest Priority
- System Processes
- Interactive Processes
- Interactive Edit Processes
- Batch Processes
- User Processes
Multilevel Feedback Queue

- Processes can be moved (promote or demote) to another queue with a different priority.
A multilevel feedback queue scheduler is defined by the following parameters:

- Number of queues
- Scheduling algorithms for each queue
- Method used to determine when to promote or demote a process (change to a different queue)
- Method used to determine which queue a process will enter when that process comes into the system for service
Algorithm Comparison

Which one is the best? The answer depends on:

- on the system workload (job distributions)
- relative weighting of performance criteria (response time, CPU utilization, throughput...)
- hardware support for the dispatcher

<table>
<thead>
<tr>
<th>Process</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrival Time</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Service Time ($T_s$)</td>
<td>3</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Finish Time</td>
<td>3</td>
<td>9</td>
<td>13</td>
<td>18</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Turnaround Time ($T_R$)</td>
<td>3</td>
<td>7</td>
<td>9</td>
<td>12</td>
<td>12</td>
<td>8.60</td>
</tr>
<tr>
<td>$T_R/T_s$</td>
<td>1.00</td>
<td>1.17</td>
<td>2.25</td>
<td>2.40</td>
<td>6.00</td>
<td>2.56</td>
</tr>
<tr>
<td>Finish Time</td>
<td>4</td>
<td>18</td>
<td>17</td>
<td>20</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Turnaround Time ($T_R$)</td>
<td>4</td>
<td>16</td>
<td>13</td>
<td>14</td>
<td>7</td>
<td>10.80</td>
</tr>
<tr>
<td>$T_R/T_s$</td>
<td>1.33</td>
<td>2.67</td>
<td>3.25</td>
<td>2.80</td>
<td>3.50</td>
<td>2.71</td>
</tr>
<tr>
<td>Finish Time</td>
<td>3</td>
<td>9</td>
<td>15</td>
<td>20</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>Turnaround Time ($T_R$)</td>
<td>3</td>
<td>7</td>
<td>11</td>
<td>14</td>
<td>3</td>
<td>7.60</td>
</tr>
<tr>
<td>$T_R/T_s$</td>
<td>1.00</td>
<td>1.17</td>
<td>2.75</td>
<td>2.80</td>
<td>1.50</td>
<td>1.84</td>
</tr>
<tr>
<td>Finish Time</td>
<td>3</td>
<td>15</td>
<td>8</td>
<td>20</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Turnaround Time ($T_R$)</td>
<td>3</td>
<td>13</td>
<td>4</td>
<td>14</td>
<td>2</td>
<td>7.20</td>
</tr>
<tr>
<td>$T_R/T_s$</td>
<td>1.00</td>
<td>2.17</td>
<td>1.00</td>
<td>2.80</td>
<td>1.00</td>
<td>1.59</td>
</tr>
<tr>
<td>Finish Time</td>
<td>3</td>
<td>9</td>
<td>13</td>
<td>20</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Turnaround Time ($T_R$)</td>
<td>3</td>
<td>7</td>
<td>9</td>
<td>14</td>
<td>7</td>
<td>8.00</td>
</tr>
<tr>
<td>$T_R/T_s$</td>
<td>1.00</td>
<td>1.17</td>
<td>2.25</td>
<td>2.80</td>
<td>3.50</td>
<td>2.14</td>
</tr>
<tr>
<td>Finish Time</td>
<td>4</td>
<td>20</td>
<td>16</td>
<td>19</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>Turnaround Time ($T_R$)</td>
<td>4</td>
<td>18</td>
<td>12</td>
<td>13</td>
<td>3</td>
<td>10.00</td>
</tr>
<tr>
<td>$T_R/T_s$</td>
<td>1.33</td>
<td>3.00</td>
<td>3.00</td>
<td>2.60</td>
<td>1.5</td>
<td>2.29</td>
</tr>
</tbody>
</table>