Lecture 7: CPU Scheduling
Chapter 5
Schedulers

- **Scheduler**: a module in OS to execute scheduling decisions.
- **Long-term scheduler** (or job scheduler) – selects which processes should be brought into the ready queue.
- **Medium-term scheduler** – selects which processes should be swapped in/out the memory.
- **Short-term scheduler** (or CPU scheduler) – selects which process should be executed next and allocates CPU.
When is scheduling done?

- **new** → **admitted** → **ready**
- **ready** → **scheduled** → **running**
- **running** → **interrupt/yield** → **blocked**
- **blocked** → **wait for event** → **ready**
- **ready** → **exit, kill** → **terminated**
Big Picture

- Long CPU burst
- Short CPU burst
- Waiting for I/O

CPU not needed.
Process goes to blocked/waiting state.

Interrupt: back from I/O operation, ready to use the CPU.
Terminology: Preemptive vs. non-Preemptive

- Preemptive: A Process can be suspended and resumed
- Non-preemptive: A process runs until it voluntarily gives up the CPU (waiting on I/O or terminate).
- Most modern OSs use preemptive CPU scheduling, implemented via timer interrupts.
- Non-preemptive is used when suspending a process is impossible or very expensive: e.g., can’t “replace” a flight crew in middle of flight.
Scheduling Performance Metrics

- CPU utilization
- Throughput
- Turnaround time
- Waiting time
- Response time
- Predictability (real-time systems, interactive systems)
- Fairness
- Meeting deadlines
- …
Scheduling Policies

- Batch systems:
  - First Come First Served
  - Shorted Job First
  - Shortest Remaining Time Next

- Interactive systems:
  - Round Robin
  - Priority Scheduling
  - Multiple Queues
  - Guaranteed Scheduling
  - Lottery Scheduling
  - Fair-share Scheduling

- Real-time systems:
  - Static vs. dynamic
First-Come, First-Served (FCFS) Scheduling

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>24</td>
</tr>
<tr>
<td>$P_2$</td>
<td>3</td>
</tr>
<tr>
<td>$P_3$</td>
<td>3</td>
</tr>
</tbody>
</table>

- Suppose that the processes arrive in the order: $P_1$, $P_2$, $P_3$

  The Gantt Chart for the schedule is:

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td>27</td>
<td>30</td>
</tr>
</tbody>
</table>

- Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Average waiting time: $(0 + 24 + 27)/3 = 17$
Suppose that the processes arrive in the order $P_2, P_3, P_1$.

The Gantt chart for the schedule is:

<table>
<thead>
<tr>
<th></th>
<th>P_2</th>
<th>P_3</th>
<th>P_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Waiting time for $P_1 = 6; P_2 = 0; P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = 3$
- Much better than previous case
- *Convoy effect* short process behind long process
Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- SJF is optimal – gives minimum average waiting time for a given set of processes.
  - The difficulty is knowing the length of the next CPU request.
### Example of SJF

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>0.0</td>
<td>6</td>
</tr>
<tr>
<td>$P_2$</td>
<td>2.0</td>
<td>8</td>
</tr>
<tr>
<td>$P_3$</td>
<td>4.0</td>
<td>7</td>
</tr>
<tr>
<td>$P_4$</td>
<td>5.0</td>
<td>3</td>
</tr>
</tbody>
</table>

- **SJF scheduling chart**

- Average waiting time = $(3 + 16 + 9 + 0) / 4 = 7$
Determining Length of Next CPU Burst

- Can only estimate the length
- Can be done by using the length of previous CPU bursts, using exponential averaging

1. \( t_n = \) actual length of \( n^{th} \) CPU burst
2. \( \tau_{n+1} = \) predicted value for the next CPU burst
3. \( \alpha, 0 \leq \alpha \leq 1 \)
4. Define: \( \tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n. \)
Prediction of the Length of the Next CPU Burst

| CPU burst ($t_i$) | 6 | 4 | 6 | 4 | 13 | 13 | 13 | ... |
| "guess" ($\tau_i$) | 10 | 8 | 6 | 6 | 5 | 9 | 11 | 12 | ... |
Examples of Exponential Averaging

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - Recent history does not count
- $\alpha = 1$
  - $\tau_{n+1} = \alpha t_n$
  - Only the actual last CPU burst counts
- If we expand the formula, we get:
  $$\tau_{n+1} = \alpha t_n + (1 - \alpha)\alpha t_{n-1} + \ldots + (1 - \alpha)^{j} \alpha t_{n-j} + \ldots + (1 - \alpha)^{n+1} \tau_0$$
- Since both $\alpha$ and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor.
Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer = highest priority)
  - Preemptive
  - Non-preemptive
- SJF is a priority scheduling where priority is the predicted next CPU burst time
- Problem = Starvation – low priority processes may never execute
- Solution = Aging – as time progresses increase the priority of the process
Round Robin (RR)

- Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.

- If there are $n$ processes in the ready queue and the time quantum is $q$, then each process gets $1/n$ of the CPU time in chunks of at most $q$ time units at once. No process waits more than $(n-1)q$ time units.

- Performance
  - $q$ large $\Rightarrow$ FIFO
  - $q$ small $\Rightarrow$ $q$ must be large with respect to context switch, otherwise overhead is too high
The Gantt chart is:

```
<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P1</th>
<th>P1</th>
<th>P1</th>
<th>P1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>7</td>
<td>10</td>
<td>14</td>
<td>18</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>26</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>30</td>
</tr>
</tbody>
</table>
```

- Typically, higher average turnaround than SJF, but better response
Time Quantum and Context Switch Time

**Diagram**

- Process time = 10
- Quantum: 12, Context switches: 0
- Quantum: 6, Context switches: 1
- Quantum: 1, Context switches: 9
Turnaround Time Varies With The Time Quantum

<table>
<thead>
<tr>
<th>process</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>6</td>
</tr>
<tr>
<td>$P_2$</td>
<td>3</td>
</tr>
<tr>
<td>$P_3$</td>
<td>1</td>
</tr>
<tr>
<td>$P_4$</td>
<td>7</td>
</tr>
</tbody>
</table>