Lecture - V

CPU Scheduling - I

Tevfik Koşar

University at Buffalo
September 17th, 2013
Roadmap

- CPU Scheduling
  - Basic Concepts
  - Scheduling Criteria & Metrics
  - Different Scheduling Algorithms
    - FCFS
    - SJF
    - Priority
    - RR
  - Preemptive vs Non-preemptive Scheduling
  - Gantt Charts & Performance Comparison
Basic Concepts

- Multiprogramming is needed for efficient CPU utilization
- **CPU Scheduling:** deciding which processes to execute when
- Process execution begins with a **CPU burst**, followed by an **I/O burst**
- **CPU-I/O Burst Cycle** - Process execution consists of a *cycle* of CPU execution and I/O wait
Alternating Sequence of CPU And I/O Bursts

- load store
- add store
- read from file

wait for I/O

store increment index
write to file

wait for I/O

load store
add store
read from file

wait for I/O

CPU burst

I/O burst

CPU burst

I/O burst

CPU burst

I/O burst
Histogram of CPU-burst Durations

![Histogram of CPU-burst Durations](image-url)
Process State

- As a process executes, it changes state
  - new: The process is being created
  - ready: The process is waiting to be assigned to a process
  - running: Instructions are being executed
  - waiting: The process is waiting for some event to occur
  - terminated: The process has finished execution
CPU Scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them
  - short-term scheduler
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to ready
  4. Terminates
  5. A new process arrives
- Scheduling under 1 and 4 is nonpreemptive/cooperative
  - Once a process gets the CPU, keeps it until termination/switching to waiting state/release of the CPU
- All other scheduling is preemptive
  - Most OS use this
  - Cost associated with access to shared data
  - i.e. time quota expires
Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler;
- Its function involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- **Dispatch latency** - time it takes for the dispatcher to stop one process and start another running
Scheduling Criteria

- **CPU utilization** - keep the CPU as busy as possible
  --> maximize

- **Throughput** - # of processes that complete their execution per time unit
  --> maximize

- **Turnaround time** - amount of time passed to finish execution of a particular process
  --> minimize
  - i.e. execution time + waiting time

- **Waiting time** - total amount of time a process has been waiting in the ready queue
  --> minimize

- **Response time** - amount of time it takes from when a request was submitted until the first response is produced
  - not output (for time-sharing environment)
  --> minimize
Optimization Criteria

- Maximize CPU utilization
- Maximize throughput
- Minimize turnaround time
- Minimize waiting time
- Minimize response time
Scheduling Metrics

- **Scheduling metrics**
  - arrival time $T_a = \text{time the process became "Ready" (again)}$
  - wait time $T_w = \text{time spent waiting for the CPU}$
  - service time $T_s = \text{time spent executing in the CPU}$
  - **turnaround time** $T_r = \text{total time spent waiting and executing}$

\[
T_r = T_w + T_s
\]

Arrival times:
- #5 arrived

Execution times:
- #5 executed

$T_r / T_s = 2.5$
First-Come, First-Served (FCFS) Scheduling

- processes are assigned the CPU in the order they request it
- when the running process blocks, the first “Ready” is run next
- when a process gets “Ready”, it is put at the end of the queue

Arrival times

First-Come-First Served (FCFS)

<table>
<thead>
<tr>
<th>Process</th>
<th>Finish Time</th>
<th>Turnaround Time ($T_r$)</th>
<th>$T_r/T_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>3</td>
<td>1.00</td>
</tr>
<tr>
<td>B</td>
<td>9</td>
<td>7</td>
<td>1.17</td>
</tr>
<tr>
<td>C</td>
<td>13</td>
<td>9</td>
<td>2.25</td>
</tr>
<tr>
<td>D</td>
<td>18</td>
<td>12</td>
<td>2.40</td>
</tr>
<tr>
<td>E</td>
<td>20</td>
<td>12</td>
<td>6.00</td>
</tr>
</tbody>
</table>

Mean: 8.60

**FCFS Scheduling - Example**

<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:

- Waiting time for $P_1 = 0$; $P_2 = 24; P_3 = 27$
- Average waiting time: $(0 + 24 + 27)/3 = 17$
FCFS Scheduling - Example

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>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

• Two schemes:
  - nonpreemptive - once CPU given to the process it cannot be preempted until completes its CPU burst
  - preemptive - if a new process arrives with CPU burst length less than remaining time of current executing process, preempt.
    -->This scheme is know as the Shortest-Remaining-Time-First (SRTF)

• SJF is optimal - gives minimum average waiting time for a given set of processes
Non-Preemptive SJF

- nonpreemptive, assumes the run times are known in advance
- among several equally important “Ready” jobs (or CPU bursts), the scheduler picks the one that will finish the earliest

**Arrival times**

**Shortest Job First (SJF)**

<table>
<thead>
<tr>
<th>SJF</th>
<th>Finish Time Turnaround Time ($T_r$)</th>
<th>$T_r/T_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>1.17</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>15</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>2.75</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>20</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>2.80</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>1.50</td>
<td></td>
</tr>
</tbody>
</table>

**Mean**

- 7.60
- 1.84

Non-Preemptive SJF - Example

<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>7</td>
</tr>
<tr>
<td>$P_2$</td>
<td>2.0</td>
<td>4</td>
</tr>
<tr>
<td>$P_3$</td>
<td>4.0</td>
<td>1</td>
</tr>
<tr>
<td>$P_4$</td>
<td>5.0</td>
<td>4</td>
</tr>
</tbody>
</table>

- SJF (non-preemptive) **Gantt Chart**

- Average waiting time $= (0 + 6 + 3 + 7)/4 = 4$
Preemptive SJF (SRT)

Shortest Remaining Time (SRT)

- preemptive version of SJF, also assumes known run time
- choose the process whose remaining run time is shortest
- allows new short jobs to get good service

Example of Preemptive 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>7</td>
</tr>
<tr>
<td>$P_2$</td>
<td>2.0</td>
<td>4</td>
</tr>
<tr>
<td>$P_3$</td>
<td>4.0</td>
<td>1</td>
</tr>
<tr>
<td>$P_4$</td>
<td>5.0</td>
<td>4</td>
</tr>
</tbody>
</table>

- SJF (preemptive) Gantt Chart
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
  - nonpreemptive
- 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
### Example of Priority

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Burst Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>0.0</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>$P_2$</td>
<td>2.0</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>$P_3$</td>
<td>4.0</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>$P_4$</td>
<td>5.0</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

- **Priority (non-preemptive)**
  - $P_1$ --> $P_2$ --> $P_4$ --> $P_3$

- **Priority (preemptive)**
  - ??
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
Round Robin (RR)

✓ **preemptive** FCFS, based on a timeout interval, the **quantum** $q$
✓ the running process is interrupted by the clock and put last in a FIFO “Ready” queue; then, the first “Ready” process is run instead

Arrival times

Round-Robin (RR), $q = 1$

<table>
<thead>
<tr>
<th>RR $q = 1$</th>
<th>Finish Time</th>
<th>Turnaround Time ($T_r$)</th>
<th>$T_r/T_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>4</td>
<td>1.33</td>
</tr>
<tr>
<td>B</td>
<td>18</td>
<td>16</td>
<td>2.67</td>
</tr>
<tr>
<td>C</td>
<td>17</td>
<td>13</td>
<td>3.25</td>
</tr>
<tr>
<td>D</td>
<td>20</td>
<td>14</td>
<td>2.80</td>
</tr>
<tr>
<td>E</td>
<td>15</td>
<td>7</td>
<td>3.50</td>
</tr>
</tbody>
</table>

Mean: 10.80, 2.71

Round Robin (RR)

- a crucial parameter is the quantum \( q \) (generally \( \sim 10\text{–}100\text{ms} \))
  - \( q \) should be big compared to context switch latency (~10 \( \mu \text{s} \))
  - \( q \) should be less than the longest CPU bursts, otherwise RR degenerates to FCFS

RR \( q = 4 \) scheduling policy

Round-Robin (RR), \( q = 4 \)

<table>
<thead>
<tr>
<th>RR ( q = 4 )</th>
<th>Finish Time Turnaround Time ( (T_r) )</th>
<th>( T_r/T_s )</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3, 17</td>
<td>3, 1.00</td>
</tr>
<tr>
<td>B</td>
<td>15, 7</td>
<td>2.5, 1.75</td>
</tr>
<tr>
<td>C</td>
<td>14, 8</td>
<td>2.80</td>
</tr>
<tr>
<td>D</td>
<td>19</td>
<td>5.50</td>
</tr>
<tr>
<td>E</td>
<td>10.00</td>
<td>2.71</td>
</tr>
</tbody>
</table>

Mean

Example of RR with Time Quantum = 20

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>53</td>
</tr>
<tr>
<td>$P_2$</td>
<td>17</td>
</tr>
<tr>
<td>$P_3$</td>
<td>68</td>
</tr>
<tr>
<td>$P_4$</td>
<td>24</td>
</tr>
</tbody>
</table>

For $q=20$, the Gantt chart is:

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>$P_4$</th>
<th>$P_1$</th>
<th>$P_3$</th>
<th>$P_4$</th>
<th>$P_1$</th>
<th>$P_3$</th>
<th>$P_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>20</td>
<td>37</td>
<td>57</td>
<td>77</td>
<td>97</td>
<td>117</td>
<td>121</td>
<td>134</td>
<td>154</td>
<td>162</td>
</tr>
</tbody>
</table>

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

- Process time: 10
- Quantum: 12
- Context switches: 0
- Process time: 6
- Quantum: 6
- Context switches: 1
- Process time: 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>
Exercise

<table>
<thead>
<tr>
<th>Process ID</th>
<th>Arrival Time</th>
<th>Priority</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>B</td>
<td>5</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>C</td>
<td>10</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>D</td>
<td>15</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

- Draw gantt charts, find average turnaround, waiting, and response times for above processes, considering:
  - 1) First Come First Served Scheduling
  - 2) Shortest Job First Scheduling (non-preemptive)
  - 3) Shortest Job First Scheduling (preemptive)
  - 4) Round-Robin Scheduling
  - 5) Priority Scheduling (non-preemptive)
  - 6) Priority Scheduling (preemptive)
Summary

- CPU Scheduling
  - Basic Concepts
  - Scheduling Criteria & Metrics
  - Different Scheduling Algorithms
    - FCFS
    - SJF
    - Priority
    - RR

- Next Lecture: Continue CPU Scheduling

- Reading Assignment: Chapter 5 from Silberschatz.
Acknowledgements

• “Operating Systems Concepts” book and supplementary material by A. Silberschatz, P. Galvin and G. Gagne

• “Operating Systems: Internals and Design Principles” book and supplementary material by W. Stallings

• “Modern Operating Systems” book and supplementary material by A. Tanenbaum

• R. Doursat and M. Yuksel from UNR