05. Scheduling Algorithms

9th ed: Ch. 6
10th ed: Ch. 5
Objectives

• To understand how to apply several common scheduling algorithms
  • FCFS, SJF, SRTF
  • Round Robin
  • Priority
  • Multilevel Queues

• To understand use of measurement and prediction for unknown scheduling parameters
Outline

• First-Come First-Served (FCFS)
• Shortest Job First (SJF)
• Shortest Remaining Time First (SRTF)
• Round Robin (RR)
• Priority scheduling
• Multilevel queues
Outline

• First-Come First-Served (FCFS)
  • Convoy effect
• Shortest Job First (SJF)
• Shortest Remaining Time First (SRTF)
• Round Robin (RR)
• Priority scheduling
• Multilevel queues
First-Come First-Served (FCFS)

• Schedule depends purely on the order in which processes arrive
• Simplest possible scheduling algorithm
• Not terribly robust to different arrival processes

• E.g., suppose processes with the following burst times arrive in the order $P_1$, $P_2$, $P_3$

<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>
First-Come First-Served (FCFS)

• Then the Gantt chart is

```
P1  | P2  | P3
0   | 24  | 27  | 30
```

• The waiting times are

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
<th>Waiting Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>P3</td>
<td>3</td>
<td>27</td>
</tr>
</tbody>
</table>

• This gives an average per-process waiting time of \( \frac{0 + 24 + 27}{3} = 17 \)
The Convoy Effect

• Now suppose the same processes arrive in the order \( P_2, P_3, P_1 \)
• Then the Gantt chart and waiting times are:

\[
\begin{array}{c|c|c}
 P_2 & P_3 & P_1 \\
\hline
 0 & 3 & 6 \\
\hline
 30 & & \\
\end{array}
\]

• Gives an average per-process waiting time of \( \frac{6 + 0 + 3}{3} = 3 \)

• First case is an example of the **Convoy Effect**
  • Short-run processes getting stuck behind long-run processes
  • Consider one CPU-bound and many IO-bound processes
Outline

• First-Come First-Served (FCFS)
• Shortest Job First (SJF)
• Shortest Remaining Time First (SRTF)
• Round Robin (RR)
• Priority scheduling
• Multilevel queues
Shortest Job First (SJF)

- Associate length of next CPU burst with each process
- Schedule the process with the shortest next burst
- Optimality: SJF gives the least possible waiting time for a given set of processes
Shortest Job First (SJF)

• Consider the following arrivals process and resulting Gantt chart:

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_1</td>
<td>6</td>
</tr>
<tr>
<td>P_2</td>
<td>8</td>
</tr>
<tr>
<td>P_3</td>
<td>7</td>
</tr>
<tr>
<td>P_4</td>
<td>3</td>
</tr>
</tbody>
</table>

• Gives an average per-process waiting time of \[
\frac{3 + 16 + 9 + 0}{4} = 7
\]
Outline

• First-Come First-Served (FCFS)
• Shortest Job First (SJF)
• Shortest Remaining Time First (SRTF)
  • Predicting the future
  • Exponential averaging
• Round Robin (RR)
• Priority scheduling
• Multilevel queues
Shortest Remaining Time First (SRTF)

• Simply a pre-emptive version of SJF
  • Pre-empt current process if a new one arrives with a shorter burst length than the remaining time of the current process

• Distinguish arrival time and burst length, e.g.,

• Gives Gantt chart

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Burst Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>P₂</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>P₃</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>P₄</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>P₁</th>
<th>P₂</th>
<th>P₄</th>
<th>P₁</th>
<th>P₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>1</td>
<td>5</td>
<td>10</td>
<td>17</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>26</td>
</tr>
</tbody>
</table>

• Average waiting time now \(\frac{|10-1| + |1-1| + |17-2| + |5-3|}{4} = \frac{26}{4} = 6\frac{1}{2}\)
Optimality in the future

• If SJF is optimal given a known set of processes (demand), then surely SRTF is optimal in the face of new runnable processes arriving?

• No! Why?

• Context switches are not free, so if short burst processes keep arriving the OS will start thrashing the CPU, so no useful work gets done

• More fundamentally,

  how can we know the length of a future burst?

  (Ask the user? Ask the developer? Measure and predict?)
Predicting burst lengths

• Assume the next burst will not be too different from the previous
• Then
  • measure burst lengths as processes are scheduled,
  • predict next burst length, and
  • choose the process with the shortest predicted burst length
• E.g., exponential averaging on length of previous bursts
  • Set $t_n$ to be the measured length of the $n^{th}$ CPU burst
  • Define $\tau_{n+1}$, predicted length of $(n+1)^{th}$ burst as $\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$
Examples of exponential averaging

• Expanding this formula gives, for $\tau_0$ some constant

$$\tau_{n+1} = \alpha t_n + \ldots + (1 - \alpha)^j \alpha t_{n-j} + \ldots + (1 - \alpha)^{n+1} \tau_0$$

• As both $\alpha, 1-\alpha \leq 1$, each term has less weight than its predecessor

• Choose value of $\alpha$ according to our belief about the system, e.g,
  • If we believe past history irrelevant, choose $\alpha \approx 1$ and then get $\tau_{n+1} \approx t_n$
  • If we believe recent history irrelevant, choose $\alpha \approx 0$ and then get $\tau_{n+1} \approx \tau_n$

• Exponential averaging is often a good predictor if the variance is small
  • ...if the variance is not changing “too fast” with respect to the size of time slot
  • Also consider system load, else (counter-intuitively) priorities increase with load
Examples of exponential averaging

CPU burst ($t_i$) | 6 | 4 | 6 | 4 | 13 | 13 | 13 | …
"guess" ($\tau_i$) | 10 | 8 | 6 | 6 | 5 | 9 | 11 | 12 | …
Outline

• First-Come First-Served (FCFS)
• Shortest Job First (SJF)
• Shortest Remaining Time First (SRTF)
• Round Robin (RR)
• Priority scheduling
• Multilevel queues
Round Robin

• A pre-emptive scheduling scheme for time-sharing systems
  • Give each process a **quantum** (or time-slice) of CPU time e.g., 10—100 milliseconds
  • Once quantum elapsed, process is pre-empted and appended to the ready queue
  • Timer interrupts every quantum to schedule next process

• Can be tricky to choose correctly
  • $q$ too large degenerates into a FIFO queue (∼ FCFS)
  • $q$ too small makes the context switch overhead too great

• $q$ usually 10ms to 100ms, while context switch < 10 μsec
Round Robin

• Consider the first example again

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>24</td>
</tr>
<tr>
<td>P₂</td>
<td>3</td>
</tr>
<tr>
<td>P₃</td>
<td>3</td>
</tr>
</tbody>
</table>

• For quantum $q$ and $n$ processes ready,
  • **Fair**: each process gets $1/n$ CPU time in chunks of at most $q$ time units, and
  • **Live**: no process ever waits more than $(n-1)q$ time units

• Typically
  • higher average turnaround time than SRTF, but
  • better average response time
Outline

• First-Come First-Served (FCFS)
• Shortest Job First (SJF)
• Shortest Remaining Time First (SRTF)
• Round Robin (RR)
• Priority scheduling
  • Dynamic priorities
  • Computed priorities
• Multilevel queues
Priority scheduling

• Associate integer priority with process, and schedule the highest priority (~ lowest number) process, e.g.,

<table>
<thead>
<tr>
<th>Process</th>
<th>Priority</th>
<th>Burst Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>$P_2$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$P_3$</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>$P_4$</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>$P_5$</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>6</th>
<th>16</th>
<th>18</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_2$</td>
<td>$P_5$</td>
<td>$P_1$</td>
<td>$P_3$</td>
<td>$P_4$</td>
<td></td>
</tr>
</tbody>
</table>

• Average waiting time now

$$\frac{|1+5|+0+|1+5+10|+(1+5+10+2)+1}{5} = \frac{41}{5} = 8 \frac{1}{5}$$

• Consider: SJF as priority scheduling using inverse of predicted burst length
Dynamic priority scheduling

• **Starvation** can occur if low priority processes never execute

• Urban legend?
  • When the IBM 7074 at MIT was shut down in 1973, low-priority processes were found that had been submitted in 1967 and had not yet been run...

• This is the biggest problem with static priority systems!
  • A low priority process is not guaranteed to run — ever!

• Solve by making priorities **dynamic**
  • E.g., **aging** increases priority starting from a static base as time passes without process being scheduled
Computed Priority

• E.g., UNIX scheduler
  • Priorities 0–127; user processes ≥ Base = 50
  • Round robin within priority queue, quantum = 100ms
  • Priority recalculated every 4 ticks (typically, 40ms) it is found running

• Kernel mode process scheduling
  • Fixed priority, non-preemptive
  • Modified by reasons for process waiting
  • E.g., waiting for disk I/O < waiting for terminal input

• User mode process scheduling
  • Dynamically computed, pre-emptive
  • Per-tick (10ms), if there is a higher-priority process, switch to it
  • Per-quantum (10 ticks = 100ms), if there is a process in the same priority queue, switch to it
Computing the priority

• Priority of process $j$ at start of interval $i$ is based on
  • $\text{base}_j$, the base priority of a user mode process (50)
  • $\text{nice}_j$, a user controllable parameter between -20 and 20 (default = 0)
  • $\text{load}_j$, the sampled (1 minute) average length of the run queue
  • $\text{CPU}_j$, incrementing counter if process $j$ was observed running this tick

• Every 100 ticks,
  • Age the $\text{CPU}_j$ counter: $\text{CPU}_j(i) = \frac{2 \times \text{load}_j}{(2 \times \text{load}_j) + 1} \text{CPU}_j(i-1)$

  • Compute the new priority: $P_j(i) = \text{Base}_j + \frac{\text{CPU}_j(i)}{4} + 2 \times \text{nice}_j$
Outline

• First-Come First-Served (FCFS)
• Shortest Job First (SJF)
• Shortest Remaining Time First (SRTF)
• Round Robin (RR)
• Priority scheduling

• Multilevel queues
  • Multilevel queues
  • Multilevel feedback queues
Multilevel Queues

• Partition Ready queue into many queues for different types of process, e.g.,
  • Foreground/interactive processes
  • Background/batch processes
• Each process is permanently assigned a given queue
• Each queue runs its own scheduling algorithm, e.g.,
  • Foreground runs Round Robin
  • Background runs First-Come First-Served

05. Scheduling Algorithms
Multilevel Feedback Queues

• Now scheduling must be done between the queues:
  • **Fixed priority**, e.g., serve all from foreground then from background, permits starvation
  • **Time slice**, each queue gets a certain amount of CPU time which it can schedule amongst its processes, e.g., 80% to foreground in RR, 20% to background in FCFS

• A process can move between the various queues
  • Aging can be implemented this way

• Multilevel-feedback-queue scheduler defined by the following parameters:
  • number of queues
  • scheduling algorithms for each queue
  • method used to determine when to upgrade a process
  • method used to determine when to demote a process
  • method used to determine which queue a process will enter when it needs service
Multilevel Feedback Queues

• Three queues:
  • $Q_0$ – RR with time quantum 8 milliseconds
  • $Q_1$ – RR time quantum 16 milliseconds
  • $Q_2$ – FCFS

• Scheduling
  • A new job enters queue $Q_0$ which is served FCFS
  • When it gains CPU, job receives 8 milliseconds
  • If it does not finish in 8 milliseconds, job is moved to queue $Q_1$
  • At $Q_1$ job is again served FCFS and receives 16 additional milliseconds
  • If it still does not complete, it is pre-empted and moved to queue $Q_2$
Summary

• First-Come First-Served (FCFS)
  • Convoy effect
• Shortest Job First (SJF)
• Shortest Remaining Time First (SRTF)
  • Predicting the future
  • Exponential averaging

• Round-Robin (RR)
• Priority scheduling
  • Dynamic priorities
  • Computed priorities
• Multilevel queues
  • Multilevel feedback queues