Operating Systems
CMPSC 473
CPU Scheduling
February 10, 2009 - Lecture 8
Instructor: Trent Jaeger
• Last class:
  – Threads
• Today:
  – CPU Scheduling
Resource Allocation

- In a multiprogramming system, we need to share resources among the running processes
  - What are the types of OS resources?
- Question: Which process gets access to which resources?
  - To maximize performance
Resources Types

- **Memory**: Allocate portion of finite resource
  - Virtual memory tries to make this appear infinite
  - Physical resources are limited
- **I/O**: Allocate portion of finite resource and time with resource
  - Store information on disk
  - A time slot to store that information
- **CPU**: Allocate time slot with resource
  - A time slot to run instructions
- We will focus on *CPU scheduling* in the section
CPU Scheduling Examples

• Single process view
  – GUI request
    • Click on the mouse
  – Scientific computation
    • Long-running, but want to complete ASAP

• System view
  – Get as many tasks done as quickly as possible
  – Minimize waiting time for processes
  – Utilize CPU fully
Process Scheduling

- Running
  - Dispatched (CPU assigned)
  - Pre-empted (CPU yanked)
  - Process Terminates
- Ready
  - New process creation
  - Event Occurred
- Blocked
  - Wait For Event (e.g. I/O)
Scheduling Problem

• Choose the *ready/running* process to run at any time
  – Maximize “performance”
• Model/estimate “performance” as a function
  – System performance of scheduling each process
    • $f(process) = y$
  – What are some choices for $f(process)$?
• Choose the process with the best $y$
  – Estimating overall performance is intractable
    • E.g., scheduling so all tasks are completed as soon as possible
Scheduling Concepts
When Can Scheduling Occur?

- 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

- Scheduling for events 1 and 4 do not preempt a process
  - Process volunteers to give up the CPU
Preemptive vs Non-preemptive

• Can we reschedule a process that is actively running?
  – If so, we have a *preemptive* scheduler
  – If not, we have a *non-preemptive* scheduler

• Suppose a process becomes ready
  – E.g., new process is created or it is no longer waiting

• It may be better to schedule this process
  – So, we preempt the running process

• *In what ways could the new process be better?*
Bursts

• A process runs in CPU and I/O Bursts
  – Run instructions (CPU Burst)
  – Wait for I/O (I/O Burst)

• Scheduling is aided by knowing the length of these bursts
  – More later…
Bursts

- load store
- add store
- read from file

- store increment index
- write to file

- wait for I/O

- CPU burst
- I/O burst
- CPU burst
- I/O burst

- load store
- add store
- read from file

- wait for I/O

- -
CPU Burst Duration
Dispatcher

• Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this 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 Loop

• How a system runs
  – From a scheduling perspective
    • Don’t care about what the process is actually doing…

• Sequence of:
  – Run
  – Scheduling event
  – Schedule
    • Latency
  – Dispatch (if necessary)
    • Latency
  – Rinse, repeat…
Scheduling Criteria

- **Utilization/efficiency**: keep the CPU busy 100% of the time with useful work.
- **Throughput**: maximize the number of jobs processed per hour.
- **Turnaround time**: from the time of submission to the time of completion.
- **Waiting time**: Sum of times spent (in Ready queue) waiting to be scheduled on the CPU.
- **Response Time**: time from submission till the first response is produced (mainly for interactive jobs).
- **Fairness**: make sure each process gets a fair share of the CPU.
Scheduling Algorithms
One Algorithm

- First-Come, First-Served (FCFS)
  - Serve the jobs in the order they arrive.
  - Non-preemptive
  - Simple and easy to implement: When a process is ready, add it to tail of ready queue, and serve the ready queue in FCFS order.
  - Very fair: No process is starved out, and the service order is immune to job size, etc.
First-Come, First-Served (FCFS)

<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></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$

• Average waiting time: $(0 + 24 + 27)/3 = 17$
Reducing Waiting Time

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>3</td>
<td>6</td>
<td>30</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:
  – Non-preemptive – 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
Example of Non-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 (non-preemptive)

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

```
  0  2  4  5  7  11  16
P1  P2  P3  P2  P4  P1
```

- Average waiting time = $(9 + 1 + 0 + 2)/4 = 3$
Determining Next CPU Burst

• Can only estimate the length
• Can be done by using the length of previous CPU bursts, using exponential averaging
  1. $t_n = \text{actual length of } n^{th} \text{ CPU burst}$
  2. $\tau_{n+1} = \text{predicted value for the next CPU burst}$
  3. $\alpha, 0 \leq \alpha \leq 1$
  4. Define: $\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$. 
Determining Next CPU Burst

• If $\alpha = 0$, no weightage to recent history
• If $\alpha = 1$, no weightage to old history
• Typically, choose $\alpha = 1/2$ which gives more weightage to newer information compared to older information.

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: $\tau_{n+1} = \alpha t_n + (1 - \alpha) \tau_n$. 
Exponential Averaging

• If we expand the formula, we get:

\[ \tau_{n+1} = \alpha t_n + (1 - \alpha)\alpha t_{n-1} + \ldots + (1 - \alpha)^i \alpha t_{n-i} + \ldots + (1 - \alpha)^{n+1} \tau_0 \]

• Since both \( \alpha \) and \( (1 - \alpha) \) are less than or equal to 1, each successive term has less weight than its predecessor
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  ...
Summary

• CPU Scheduling
  – Choose the process to assign to the CPU
    • To maximize “performance”
  – Hard problem in general
  – Goal: minimize average waiting time
    • CPU bursts
    • Can devise optimal algorithms
      – If we can only predict the next CPU burst
  – Algorithms
    • FCFS
    • SJF
• Next time: More CPU Scheduling Algorithms and Systems