Operating Systems
Lecture 12
CPU Scheduling and Deadlocks

April 29, 2013
http://web.uettaxila.edu.pk/CMS/SP2013/seOSbs/
Agenda for Today

- Multi-Level Feed Back Queue Scheduling
- Unix System V Scheduling
- DEADLOCKS
- Deadlock characterization
- Deadlock handling
- Deadlock prevention
- Deadlock avoidance
Multilevel Feedback Queues

• 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
Multilevel Feedback Queues

- 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 that process needs service
Multilevel Feedback Queues

quantum = 8

quantum = 16

FCFS
Consider the following set of processes, with the length of the CPU burst given in milliseconds:

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

The processes are assumed to arrived in the order $P_1, P_2, P_3, P_4, P_5$ all at time 0.
a. Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, Nonpreemptive priority (a smaller priority number implies a higher priority), and RR (Quantum=1).

b. What is the turn around time of each process for each of these scheduling algorithms?

c. What is the waiting time of each process for each of these scheduling algorithms?

d. Which of the algorithms results in the minimum averaging waiting time (over all processes)?
UNIX System V Scheduling Algorithm

- Multilevel feedback priority queues with round robin within each queue
- Quantum = 1 second
- Priorities are divided into two groups/bands:
  - Kernel Group
  - User Group
• Priorities in the Kernel Group are assigned in a manner to minimize bottlenecks: processes waiting in a lower-level routine get higher priorities than those waiting at relatively higher-level routines.
In decreasing order of priority, the bands are:

- Swapper
- Block I/O device control processes
- File manipulation
- Character I/O device control processes
- User processes
UNIX System V Scheduling Algorithm

- Priorities of processes in the Kernel Group remain fixed.
- Priorities of processes in the User Group are recalculated every second.
- Inside the User Group, the CPU-bound processes are penalized at the expense of I/O-bound processes.
UNIX System V Scheduling Algorithm
UNIX System V Scheduling Algorithm

- The CPU usage of each process is updated every clock tick by the clock ISR.
- Every second, a decay function is applied to the CPU usage of each process in the ready queue:
  - Decay (CPU Usage) = (CPU Usage) / 2
- Priority number of each process is recomputed every second
UNIX System V Scheduling Algorithm

- Every second, the priority number of all those processes that are in the main memory and ready to run is updated by using the following formula:
  \[ \text{Priority#} = \frac{\text{Recent CPU Usage}}{2} + \text{Thr. Pri.} + \text{nice} \]
- Threshold priority and nice values are always positive to prevent a user from migrating out of its assigned group
**UNIX System V Example**

<table>
<thead>
<tr>
<th>Time</th>
<th>PA Priority</th>
<th>PA CPU Count</th>
<th>PB Priority</th>
<th>PB CPU Count</th>
<th>PC Priority</th>
<th>PC CPU Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>60</td>
<td>0</td>
<td>60</td>
<td>0</td>
<td>60</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>30</td>
<td>60</td>
<td>0</td>
<td>60</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>0</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td>0</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>67</td>
<td>15</td>
<td>75</td>
<td>30</td>
<td>60</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td></td>
<td>30</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>63</td>
<td>7</td>
<td>67</td>
<td>15</td>
<td>75</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td></td>
<td>30</td>
<td></td>
<td>60</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>4</td>
<td>76</td>
<td>67</td>
<td>63</td>
<td>7</td>
<td>67</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>33</td>
<td></td>
<td>8</td>
<td></td>
<td>15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>5</td>
<td>68</td>
<td>16</td>
<td>76</td>
<td>67</td>
<td>63</td>
<td>71</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>33</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Consider the following set of processes, with the length of the CPU burst given in milliseconds:

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

The processes are assumed to arrived in the order $P_1$, $P_2$, $P_3$, $P_4$, $P_5$ all at time 0.
a. Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, Nonpreemptive priority (a smaller priority number implies a higher priority), and RR(Quantum=1).

b. What is the turn around time of each process for each of these scheduling algorithms?

c. What is the waiting time of each process for each of these scheduling algorithms?

d. Which of the algorithms results in the minimum averaging waiting time (over all processes)?
Resources

- Resources – passive entities needed by Processes to do their work
  - CPU time, disk space, memory
- Two types of resources:
  - Preemptable – can take it away
    » CPU, Embedded security chip
  - Non-preemptable – must leave it with the process
    » Disk space, printer, chunk of virtual address space
    » Critical section
- Resources may require exclusive access or may be sharable
  - Read-only files are typically sharable
  - Printers are not sharable during time of printing
- One of the major tasks of an operating system is to manage resources
Starvation vs Deadlock

• **Starvation vs. Deadlock**
  - **Starvation:** Process waits indefinitely
    » Example, low-priority process waiting for resources constantly in use by high-priority processes
  - **Deadlock:** circular waiting for resources
    » Process A owns Res 1 and is waiting for Res 2
    » Process B owns Res 2 and is waiting for Res 1

- Deadlock ⇒ Starvation but not vice versa
  » Starvation can end (but doesn't have to)
  » Deadlock can’t end without external intervention
The Deadlock Problem

- Example
  - System has 2 tape drives.
  - P1 and P2 each hold one tape drive and each needs another one.
• Each segment of road can be viewed as a resource
  - Car must own the segment under them
  - Must acquire segment that they are moving into
• For bridge: must acquire both halves
  - Traffic only in one direction at a time
  - Problem occurs when two cars in opposite directions on bridge: each acquires one segment and needs next
• If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback)
  - Several cars may have to be backed up
• Starvation is possible
  - East-going traffic really fast $\Rightarrow$ no one goes west
System Model

- Resource types $R_1, R_2, \ldots, R_m$ (CPU cycles, memory space, I/O devices)
- Each resource type $R_i$ has $W_i$ instances.
- Each process utilizes a resource as follows:
  - request → use → release
Deadlock can arise if four conditions hold simultaneously.

- Mutual exclusion: only one process at a time can use a resource.
Deadlock Characterization

- Hold and wait: a process holding at least one resource is waiting to acquire additional resources held by other processes.
Deadlock Characterization

- No preemption: a resource can be released only voluntarily by the process holding it, after that process has completed its task.
Deadlock Characterization

- **Circular wait:** There exists a set \{P_0, P_1, \ldots, P_n\} of waiting processes such that \(P_0\) is waiting for a resource that is held by \(P_1\), \(P_1\) is waiting for a resource that is held by \(P_2\), \ldots, \(P_{n-1}\) is waiting for a resource that is held by \(P_n\), and \(P_n\) is waiting for a resource that is held by \(P_0\).

\[ P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow \ldots \rightarrow P_n \rightarrow P_0 \]
• System Model
  - A set of Processes $P_1, P_2, \ldots, P_n$
  - Resource types $R_1, R_2, \ldots, R_m$
    - CPU cycles, memory space, I/O devices
  - Each resource type $R_i$ has $W_i$ instances.
  - Each process utilizes a resource as follows:
    - Request() / Use() / Release()

• Resource-Allocation Graph:
  - $V$ is partitioned into two types:
    - $P= \{P_1, P_2, \ldots, P_n\}$, the set processes in the system.
    - $R = \{R_1, R_2, \ldots, R_m\}$, the set of resource types in system
  - request edge - directed edge $P_1 \rightarrow R_j$
  - assignment edge - directed edge $R_j \rightarrow P_i$
Resource Allocation Graph Examples

- Recall:
  - request edge - directed edge $P_1 \rightarrow R_j$
  - assignment edge - directed edge $R_j \rightarrow P_i$
Basic Facts

• If graph contains no cycles $\Rightarrow$ no deadlock.

• If graph contains a cycle $\Rightarrow$
  • if only one instance per resource type, then deadlock.
  • if several instances per resource type, possibility of deadlock.
Methods for Handling Deadlocks

• Allow system to enter deadlock and then recover
  - Requires deadlock detection algorithm
  - Some technique for forcibly preempts resources and/or terminating tasks

• Ensure that system will never enter a deadlock
  - Need to monitor all lock acquisitions
  - Selectively deny those that might lead to deadlock

• Ignore the problem and pretend that deadlocks never occur in the system
  - Used by most operating systems, including UNIX
Deadlock Handling

- Deadlock prevention
- Deadlock avoidance
- Deadlock detection and recovery
Deadlock Prevention

- Restrain the ways resource allocation requests can be made to insure that at least one of the four necessary conditions is violated.
• Mutual Exclusion

  Cannot be prevented for all resources. Some resources are inherently non-sharable because their states cannot be saved and restored without ill effects, such as a printer.
• Hold and Wait – we must guarantee that whenever a process requests a resource, it does not hold any other resources.
Deadlock Prevention

- Require a process to request and be allocated all its resources before it begins execution, or allow a process to request resources only when the process has none.
- Low resource utilization; starvation possible.
Deadlock Prevention

• No Preemption

If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released.
Deadlock Prevention

- Preempted resources are added to the list of resources for which the process is waiting for.
- Process will be restarted only when it can regain its old resources as well as the new ones that it has requested.
Deadlock Prevention

- Circular Wait – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration.
Deadlock Prevention

- We assign a unique number to each resource type by using function 
  \[ F: R \rightarrow \mathbb{N} \]
  and make sure that processes request resources in an increasing order of enumeration.
- For example, tape drive = 1, disk drive = 5, and printer = 12.
Proof

Let's assume that there is a cycle

\[ P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow \ldots \rightarrow P_k \rightarrow P_0 \]

\[ R_0 \quad R_1 \quad R_2 \quad R_k \quad R_0 \]

\[ \Rightarrow F(R_0) < F(R_1) < \ldots F(R_k) < F(R_0) \]

\[ \Rightarrow F(R_0) < F(R_0), \text{ which is impossible} \]

\[ \Rightarrow \text{There can be no circular wait.} \]