Week 5: Deadlock Avoidance and Prevention

Sherif Khattab

http://www.cs.pitt.edu/~skhattab/cs1550

(slides are from Silberschatz, Galvin and Gagne ©2013)
• Project 1 out and due on 2/21 @11:59pm
Agenda

• System Model
• Deadlock Avoidance
• Deadlock Detection
Deadlock

“When two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other has gone.”

- A set of processes is in a deadlocked state when every process in the set is waiting for an event that can be caused only by another process in the set.
- deadlocked vs. frozen state
System Model

- System consists of resources
- 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
Necessary Conditions

- **Mutual exclusion**: only one process at a time can use a resource.
- **Hold and wait**: a process holding at least one resource is waiting to acquire additional resources held by other processes.
- **No preemption**: a resource can be released only voluntarily by the process holding it, after that process has completed its task.
- **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 \( P_0 \).

Deadlock can arise if four conditions hold simultaneously.
Resource-Allocation Graph

- A set of vertices $V$ and a set of edges $E$.
- $V$ is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set of all the processes in the system
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set of all resource types in the system
- **request edge** – directed edge $P_i \rightarrow R_j$
- **assignment edge** – directed edge $R_j \rightarrow P_i$
Resource Allocation Graph With A Deadlock

![Resource Allocation Graph](image-url)
Graph With A Cycle But No Deadlock
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
    - necessary and sufficient condition
  - if several instances per resource type, possibility of deadlock
    - necessary condition
How to handle deadlocks?

- Prevention
- Avoidance
- Detection and Recovery
- Ignore(!) (most common due to some sort of risk analysis)
Deadlock Avoidance

- System state is defined by
  - number of available and allocated resources, and
  - the maximum demands of the processes (a priori knowledge)

- If a system is in safe state $\implies$ no deadlocks

- If a system is in unsafe state $\implies$ possibility of deadlock

- Avoidance $\implies$ ensure that a system will never enter an unsafe state.
Suppose that process $P_i$ requests a resource $R_j$. The request can be granted only if converting the request edge to an assignment edge does not result in the formation of a cycle in the resource allocation graph.
Banker’s Algorithm

- Multiple resource instances
- Each process must *a priori* claim maximum use
- When a process requests a resource it may have to wait
- When a process gets all its resources it must return them in a finite amount of time
**Available:** Vector of length $m$. If available[$j$] = $k$, there are $k$ instances of resource type $R_j$ available.

**Max:** $n \times m$ matrix. If Max[i,$j$] = $k$, then process $P_i$ may request at most $k$ instances of resource type $R_j$.

Let $n$ = number of processes, and $m$ = number of resources types.

**Allocation:** $n \times m$ matrix. If Allocation[i,$j$] = $k$ then $P_i$ is currently allocated $k$ instances of $R_j$.

**Need:** $n \times m$ matrix. If Need[i,$j$] = $k$, then $P_i$ may need $k$ more instances of $R_j$ to complete its task.

$$\text{Need}[i,j] = \text{Max}[i,j] - \text{Allocation}[i,j]$$
Checking if the system is in a safe state

1. Let Work and Finish be vectors of length $m$ and $n$, respectively. Initialize:

   Work = Available
   Finish[i] = false for $i = 0, 1, \ldots, n-1$

2. Find an $i$ such that both:
   (a) Finish[i] = false
   (b) Need[i] $\leq$ Work

   If no such $i$ exists, go to step 4

3. Work = Work + Allocation$_i$
   Finish[i] = true
   go to step 2

4. If Finish[i] == true for all $i$, then the system is in a safe state
Resource-Request Algorithm for Process $P_i$

$\text{Request}_i = \text{request vector for process } P_i$. If $\text{Request}_i[j] = k$ then process $P_i$ wants $k$ instances of resource type $R_j$

1. If $\text{Request}_i \leq \text{Need}_i$ go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.

2. If $\text{Request}_i \leq \text{Available}$, go to step 3. Otherwise $P_i$ must wait, since resources are not available.

3. Pretend to allocate requested resources to $P_i$ by modifying the state as follows:

   \[
   \text{Available} = \text{Available} - \text{Request}_i; \\
   \text{Allocation}_i = \text{Allocation}_i + \text{Request}_i; \\
   \text{Need}_i = \text{Need}_i - \text{Request}_i; \\
   \]

   - If safe $\Rightarrow$ the resources are allocated to $P_i$
   - If unsafe $\Rightarrow$ $P_i$ must wait, and the old resource-allocation state is restored
A sequence of processes $<P_1, P_2, ..., P_n>$ is a safe sequence for the current allocation state if,

- for each $P_i$, the resource requests that $P_i$ can still make can be satisfied by the currently available resources plus the resources held by all processes before it in the sequence ($P_j$, with $j<i$).
Example of Banker’s Algorithm

- 5 processes $P_0$ through $P_4$;

3 resource types:

\[ A \text{ (10 instances)}, \quad B \text{ (5 instances)}, \quad \text{and } C \text{ (7 instances)} \]

- Snapshot at time $T_0$:

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Max</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>$P_0$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$P_1$</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>$P_2$</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>$P_3$</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$P_4$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
• The content of the matrix \textit{Need} is defined to be \textit{Max} – \textit{Allocation}

\begin{align*}
\text{Need} \\
\begin{array}{ccc}
A & B & C \\
P_0 & 7 & 4 & 3 \\
P_1 & 1 & 2 & 2 \\
P_2 & 6 & 0 & 0 \\
P_3 & 0 & 1 & 1 \\
P_4 & 4 & 3 & 1
\end{array}
\end{align*}

• The system is in a safe state since the sequence \(< P_1, P_3, P_4, P_2, P_0 >\) satisfies safety criteria
Example: $P_1$ Request $(1,0,2)$

- Check that Request $\leq$ Available (that is, $(1,0,2) \leq (3,3,2)) \Rightarrow true$

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Need</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C</td>
<td>A B C</td>
<td>A B C</td>
</tr>
<tr>
<td>$P_0$ 0 1 0</td>
<td>7 4 3</td>
<td>2 3 0</td>
</tr>
<tr>
<td>$P_1$ 3 0 2</td>
<td>0 2 0</td>
<td></td>
</tr>
<tr>
<td>$P_2$ 3 0 2</td>
<td>6 0 0</td>
<td></td>
</tr>
<tr>
<td>$P_3$ 2 1 1</td>
<td>0 1 1</td>
<td></td>
</tr>
<tr>
<td>$P_4$ 0 0 2</td>
<td>4 3 1</td>
<td></td>
</tr>
</tbody>
</table>

- Executing safety algorithm shows that sequence $< P_1, P_3, P_4, P_0, P_2 >$ satisfies safety requirement

- Can request for $(3,3,0)$ by $P_4$ be granted?

- Can request for $(0,2,0)$ by $P_0$ be granted?
• **Available:** A vector of length \( m \) indicates the number of available resources of each type.

• **Allocation:** An \( n \times m \) matrix defines the number of resources of each type currently allocated to each process.

• **Request:** An \( n \times m \) matrix indicates the current request of each process. If \( \text{Request}[i][j] = k \), then process \( P_i \) is requesting \( k \) more instances of resource type \( R_j \).
1. (a) \( \text{Work} = \text{Available} \)
   
   (b) For \( i = 1, 2, \ldots, n \), if \( \text{Allocation}_i \neq 0 \), then \( \text{Finish}[i] = \text{false} \); otherwise, \( \text{Finish}[i] = \text{true} \)

2. Find an index \( i \) such that both:
   
   (a) \( \text{Finish}[i] == \text{false} \)
   
   (b) \( \text{Request}_i \leq \text{Work} \)

   If no such \( i \) exists, go to step 4

3. \( \text{Work} = \text{Work} + \text{Allocation}_i \)
   
   \( \text{Finish}[i] = \text{true} \) (optimistic attitude)

   go to step 2

4. If \( \text{Finish}[i] == \text{false} \), for some \( i, 1 \leq i \leq n \), then the system is in deadlock state. Moreover, if \( \text{Finish}[i] == \text{false} \), then \( P_i \) is deadlocked

Algorithm requires \( O(m \times n^2) \) operations
Example of Detection Algorithm

- Five processes $P_0$ through $P_4$; three resource types A (7 instances), B (2 instances), and C (6 instances)

- Snapshot at time $T_0$:

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Request</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C</td>
<td>A B C</td>
<td>A B C</td>
</tr>
<tr>
<td>$P_0$</td>
<td>0 1 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>$P_1$</td>
<td>2 0 0</td>
<td>2 0 2</td>
</tr>
<tr>
<td>$P_2$</td>
<td>3 0 3</td>
<td>0 0 0</td>
</tr>
<tr>
<td>$P_3$</td>
<td>2 1 1</td>
<td>1 0 0</td>
</tr>
<tr>
<td>$P_4$</td>
<td>0 0 2</td>
<td>0 0 2</td>
</tr>
</tbody>
</table>

- Sequence $<P_0, P_2, P_3, P_1, P_4>$ will result in $Finish[i] = true$ for all $i$
Example (Cont.)

- $P_2$ requests an additional instance of type $C$

  **Request**
  
  A B C
  
  $P_0$ 0 0 0
  $P_1$ 2 0 2
  $P_2$ 0 0 1
  $P_3$ 1 0 0
  $P_4$ 0 0 2

- State of system?
  - Can reclaim resources held by process $P_0$, but insufficient resources to fulfill other processes; requests
  - Deadlock exists, consisting of processes $P_1$, $P_2$, $P_3$, and $P_4$
Detection-Algorithm Usage

- When, and how often, to invoke depends on:
  - How often a deadlock is likely to occur?
  - How many processes will need to be rolled back?
    - one for each disjoint cycle

- If detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph and so we would not be able to tell which of the many deadlocked processes “caused” the deadlock.
Recovery from Deadlock: Process Termination

• Abort all deadlocked processes

• Abort one process at a time until the deadlock cycle is eliminated

• In which order should we choose to abort?
  1. Priority of the process
  2. How long process has computed, and how much longer to completion
  3. Resources the process has used
  4. Resources process needs to complete
  5. How many processes will need to be terminated
  6. Is process interactive or batch?
Recovery from Deadlock: Resource Preemption

- **Selecting a victim** – minimize cost

- **Rollback** – return to some safe state, restart process for that state

- **Starvation** – same process may always be picked as victim
  - include number of rollback in cost factor