Deadlocks

Topics
- System Model
- Deadlock characterization
- Methods for handling deadlocks
- Deadlock prevention, avoidance
- Deadlock detection and recovery

The Deadlock Problem
- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set.
- Example: System has two tape drives. Process P_1 and P_2 each holds one and needs the other tape.
  - Example: P1: wait(B); wait(A);
  - P2: wait(A); wait(B);

Bridge Crossing Example
- Traffic only in one direction
- Each direction of a bridge can be viewed as a resource.
- If a deadlock occurs, it can be resolved if one car backs up (preempt and rollback)
- Backing up may be impossible
- Starvation is possible
**System Model**

Resource types $R_1, R_2, ..., R_{m-1}$
- 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 and release

---

**Deadlock Characterization**

Deadlock can arise if four conditions hold simultaneously.
- Mutual Exclusion
- Hold and Wait
- No preemption
- Circular Wait

---

**Resource-Allocation Graph**

A set of vertices $V$ and edges $E$
- $V$ is partitioned into two types: $P = \{P1, P2, ..., Pn\}$, the set consisting of all processes in the system and $R = \{R1, R2, ..., Rm\}$, the set consisting of all resources in the system.
- Request edge - directed edge from $Pi$ to $Rj$
- Assignment edge - directed edge from $Rj$ to $Pi$.
Resource-Allocation Graph

- Process
  - Resources type with 4 instances
  - Pi Requests an instance of Rj

Example of a Graph with no cycle

Example of a graph with a cycle
Basic Facts
- If a graph contains no cycle implies no deadlock.
- If a graph contains a cycle: if only one instance per resource type then there is deadlock.
- If several instances per resource type possibility of deadlock.

Methods for Handling Deadlocks
- Ensure that the system will never enter a deadlock state.
- Allow the system to enter a deadlock state and then recover.
- Ignore the problem and pretend that deadlocks never occur in the system.

Deadlock prevention
- Restrained the ways resource requests are made.
- Hold and wait - must guarantee that whenever a process requests a resource, it does not hold any other resources - either request before a process begins execution or when the process has none.
- Low resource utilization - starvation possible.
Deadlock prevention

No preemption - if a process requests a resource that cannot be allocated, all currently held resources are released.

Circular wait - imposes a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration.

Deadlock Avoidance

Requires that the system a priori information available maximum number of resources needed.

Dead-lock avoidance algorithm dynamically examines the resource-allocation to ensure that there is no circular wait.

Resource allocation state is defined by the number of available and allocated resources and the maximum demands.

Safe State

When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state.

System is in a safe state if there exists a safe sequence for all processes.

Sequence <P1,P2,...,Pn> is safe for each Pi, if the requests of Pi can still request can be satisfied by currently available resources and resources held by Pj with j < i.
Basic Facts
- 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.

Banker’s Algorithm
Multiple 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.

Data Structures for the Banker’s Algorithm
N= number of processes, m= available resources
Available- vector Available[i]=k means that there are k instances of resource types \( R_i \).
Allocation -n x m matrix If Allocation[i,j]=k means process \( P_i \) has been allocated k units of resource type \( R_j \).
Data Structures for the Banker’s Algorithm (continued)

Need - n x m matrix.

If Need[i,j]=m means process Pi needs m units of resource type Rj.

Need[i,j]=Max[i,j]-Allocation[i,j]

Safety Algorithm

1) Let Work and Finish be vectors of length m and n respectively; initialize Work=Available and Finish=false
2) Find an i such that both Finish[i]=false and Need[i]<Work. If no such i exists go to step 4.
3) Work=Work+Allocation[i]; Finish=true; goto step 2.
4) If Finish[i] is true for all i, then the system is in a safe state.

Resource-Request Algorithm for Process P_i

1. If Request[i] <= Need[i] goto step 2. Otherwise error condition.
2. If Request[i] <= Available, goto step 3. Otherwise, P_i must wait.
3. Pretend to allocate requested resource by modifying the state. Available=Available-Request[i]; Allocation_i=Allocation_i+Request[i]; Need_i=Need_i-Request[i]; If Safe requests are allocated else P_i waits.
Example

5 processes P0 to P4; 3 Resource types A (10 instances), B(5 instances) and C(7 instances)

Snap slot at time T0

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Max</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>0 1 0</td>
<td>7 5 3</td>
<td>3 3 2</td>
</tr>
<tr>
<td>2 0 0</td>
<td>3 2 2</td>
<td></td>
</tr>
<tr>
<td>3 0 2</td>
<td>9 0 2</td>
<td></td>
</tr>
<tr>
<td>2 1 1</td>
<td>2 2 2</td>
<td></td>
</tr>
<tr>
<td>0 0 2</td>
<td>4 3 3</td>
<td></td>
</tr>
</tbody>
</table>

Example

P1 Requests (1,0,2)
Can a request of (3,3,0) of P4 be granted
Can a request (0,2,0) by P0 be granted.

Deadlock Detection

Allow system to enter deadlock state.
Detection Algorithm
Recovery scheme
### Single Instance of Resource Type

Maintain wait-for graph (nodes are processes, an edge \( P_i \) to \( P_j \) denotes that \( P_i \) is waiting for \( P_j \)).

Periodically invoke an algorithm for finding cycle.

### Multiple Instances of Resource Type

Data Structures Available, Allocation and Request as for Banker’s Algorithm

Algorithm:
1) Let \( \text{Work} \) and \( \text{Finish} \) be vectors of length \( m \) and \( n \) respectively. Initialize: \( \text{Work} = \text{Available} \). If \( \text{Allocation}_I \neq \) then \( \text{Finish}_I = \text{false} \) else \( \text{Finish}_I = \text{true} \);
2) Find an index \( I \) such that \( \text{Finish}[I] = \text{false} \) and \( \text{Request}[I] \leq \text{Work} \) - if no such \( I \) exists goto 4.
3) \( \text{Work} = \text{Work} + \text{Allocation} \), \( \text{Finish}[I] = \text{true} \); goto step 2.
4) If \( \text{Finish}[I] = \text{false} \) for some \( I \), then the system is in a deadlock state. Moreover, if \( \text{Finish}[I] = \text{false} \), then \( P_i \) is deadlocked.
Example
5 Processes three resource types A(4), B(2) and C (6).
Snapshot at t0.
<table>
<thead>
<tr>
<th>Allocation Request Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

Detection Algorithm Usage
When and how to invoke depends on how often a deadlock is likely to occur and how many processes will be needed to rollback.
We also have to know which processes caused deadlock.

Recovery from Deadlock
Abort all processes
Abort one process at a time until the deadlock cycle is eliminated.
Which process to abort - priority, how long computation has been done, resources, interactive or batch.
Recovery from Deadlock

Selecting a victim - minimize cost
Rollback - return to safe state.
Starvation

Combined Approach to Deadlock Handling

Combine the three basic approaches - Prevention, Avoidance and detection
Partition the resources into hierarchically ordered classes
Use most appropriate technique for handling deadlocks within each class.