CS 571
Operating Systems

Deadlock
What is a Deadlock?

- Processes that acquire multiple resources are dependent on those resources
  - E.g., locks, semaphores, monitors, etc.

- What if one process tries to allocate a resource that a second process holds, and vice-versa
  - Neither can ever make progress!

- We call this situation **Deadlock**, and we’ll look at:
  - Definition and conditions necessary for deadlock
  - Representation of deadlock conditions
  - Approaches to dealing with deadlock
Deadlock Definition

- **Deadlock** is a problem that can arise:
  - When processes compete for access to limited resources
  - When processes are incorrectly synchronized

- **Definition:**
  - Deadlock exists among a set of processes if every process is waiting for an event that can be caused only by another process in the set.

```
Process 1
lockA->Acquire();
...
lkB->Acquire();

Process 2
lockB->Acquire();
...
lkB->Acquire();
```
Conditions for Deadlock

- **Deadlocks** can exist if and only if four conditions hold:

1) **Mutual exclusion**: At least one resource must be held in a non-sharable mode. (*i.e.*, only one instance)

2) **Hold and wait**: There must be one process holding one resource and waiting for another resource

3) **No preemption**: Resources cannot be preempted (*i.e.*, critical sections cannot be aborted externally)

4) **Circular wait**: There must exist a set of processes \( P_1, P_2, P_3, \ldots, P_n \) such that \( P_1 \) is waiting for a resource held by \( P_2 \), \( P_2 \) is waiting for \( P_3 \), \ldots, and \( P_n \) for \( P_1 \)
Deadlock can be described using a resource allocation graph (RAG).

The RAG consists of sets of vertices $P = \{P_1, P_2, \ldots, P_n\}$ of processes and $R = \{R_1, R_2, \ldots, R_m\}$ resources.

- A directed edge from a process to a resource, $P_i \rightarrow R_j$, implies that $P_i$ has requested $R_j$.
- A directed edge from a resource to a process, $R_j \rightarrow P_i$, implies that $R_j$ has been acquired by $P_i$.
- Each resource has a fixed number of units.

If the graph has no cycles, deadlock cannot exist.

If the graph has a cycle, deadlock may exist.
Resource Allocation Graph (cont)

DEADLOCK

NO DEADLOCK
Handling Deadlock

There are four ways to deal with deadlock:

- **Ignore it**
  - How lucky do you feel?

- **Prevention**
  - Make it impossible for deadlock to happen

- **Avoidance**
  - Control allocation of resources

- **Detection and recovery**
  - Look for a cycle in dependencies
Deadlock Prevention

Prevent at least one condition from happening:

- Mutual exclusion
  - Make resources sharable (not generally practical)
- Hold and wait
  - Process cannot hold one resource when requesting another
  - Process requests, releases all needed resources at once
- Preemption
  - OS can preempt resource (costly)
- Circular wait
  - Impose an ordering (numbering) on the resources and request them in order (popular implementation technique)
Deadlock: Example of Avoidance

Assume that system has 12 available backup units

<table>
<thead>
<tr>
<th></th>
<th>maximum needs</th>
<th>current needs</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>P1</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>TOTAL</td>
<td>23</td>
<td>9</td>
</tr>
</tbody>
</table>

Sequence **P1, P0, P2** is a safe sequence
Deadlock: Example of Avoidance

P2 requests one more tape drive:

<table>
<thead>
<tr>
<th></th>
<th>maximum needs</th>
<th>current needs</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>P1</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>TOTAL</td>
<td>23</td>
<td>10</td>
</tr>
</tbody>
</table>

- No safe sequence P1,... because P2 and P0 could deadlock
- For avoidance with multiple instances: use Banker's Algorithm
The Banker’s Algorithm is the classic approach to deadlock avoidance for resources with multiple units

1) Assign a *credit limit* to each customer (process)
   - Maximum credit claim must be stated in advance

2) Reject any request that leads to a *dangerous state*
   - A dangerous state is one where a sudden request by any customer for the full credit limit could lead to deadlock
   - A recursive reduction procedure recognizes dangerous states

3) In practice, the system must keep resource usage well below capacity to maintain a *resource surplus*
   - Rarely used in practice due to low resource utilization
Deadlock Avoidance

**SAFE:** R1 is assigned to P1 with priority

**UNSAFE:** (P1 requests R2) -> cycle
Deadlock Detection and Recovery

- Detection and recovery
  - If we don’t have deadlock prevention or avoidance, then deadlock may occur
  - In this case, we need to detect deadlock and recover from it

- To do this, we need two algorithms
  - One to determine whether a deadlock has occurred
  - Another to recover from the deadlock

- Possible, but expensive (time consuming)
  - Implemented in VMS
  - Run detection algorithm when resource request times out
Deadlock Detection

- Detection
  - Traverse the resource graph looking for cycles
  - If a cycle is found, preempt resource (force a process to release)

- Expensive
  - Many processes and resources to traverse

- Only invoke detection algorithm depending on
  - How often or likely deadlock is
  - How many processes are likely to be affected when it occurs
Deadlock Recovery

Once a deadlock is detected, we have two options…

1. Abort processes
   - Abort all deadlocked processes
     - Processes need start over again
   - Abort one process at a time until cycle is eliminated
     - System needs to rerun detection after each abort

2. Preempt resources (force their release)
   - Need to select process and resource to preempt
   - Need to rollback process to previous state
   - Need to prevent starvation
Deadlock Summary

- Deadlock occurs when processes are waiting on each other and cannot make progress
  - Cycles in Resource Allocation Graph (RAG)

- Deadlock requires four conditions
  - Mutual exclusion, hold and wait, no resource preemption, circular wait

- Four approaches to dealing with deadlock:
  - Ignore it – Living life on the edge
  - Prevention – Make one of the four conditions impossible
  - Avoidance – Banker’s Algorithm (control allocation)
  - Detection and Recovery – Look for a cycle, preempt or abort