**Dealing With Deadlock**

**Deadlock avoidance**

- Examine each resource request and determine whether or not granting the request can lead to deadlock.

Define a set of vectors and matrices that characterize the current state of all resources and processes:

- **resource allocation state matrix**
  
  \[ \text{Alloc}_j = \text{the number of units of resource } j \text{ held by process } i \]

- **maximum claim matrix**
  
  \[ \text{Max}_j = \text{the maximum number of units of resource } j \text{ that the process } i \text{ will ever require simultaneously} \]

- **available vector**
  
  \[ \text{Avail} = \text{the number of units of resource } j \text{ that are unallocated} \]

**Deadlock Avoidance**

- The OS will examine each resource request and determine whether or not granting the request can lead to deadlock.
  
  - If, after we grant this request, all processes simultaneously make their maximum claim, will the system deadlock?
  
  - Can we satisfy the maximum claims of processes in some order?

**Deadlock Avoidance Algorithm**

**State definitions**

- A resource allocation state is **safe** if the system can allocate resources to each process up to its maximum claim such that the system can not deadlock.

There must be an ordering of the processes \( P_1, P_2, ..., P_n \) such that for all processes \( P_i \):

\[ \text{MAX}_i = \text{ALLOCATION}_i \leq \text{AVAIL} + \sum_{j<i} \text{ALLOCATION}_j \]

The largest request for resources that process \( P_i \) can make

The resources available to process \( P_i \) after processes \( P_1 \) through \( P_{i-1} \) terminate

This ordering of processes is called a **safe sequence**.

If a safe sequence exists then there exists a process \( P_k \) that can execute to completion in the current state, and \( P_i \) can execute to completion at worst after processes \( P_1 \) through \( P_{i-1} \) complete.
A computer system with 5 processes and 4 resources

<table>
<thead>
<tr>
<th>ALLOCATION</th>
<th>MAX CLAIM</th>
<th>AVAILABLE</th>
<th>MAX_REQUEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1 0 0 1 2</td>
<td>0 0 1 2</td>
<td></td>
<td>0 0 1 2</td>
</tr>
<tr>
<td>P1 0 0 0</td>
<td>1 2 5 6</td>
<td></td>
<td>1 2 5 6</td>
</tr>
<tr>
<td>P1 0 3 2</td>
<td>0 6 3 2</td>
<td></td>
<td>0 6 3 2</td>
</tr>
<tr>
<td>P1 0 1 4</td>
<td>0 6 5</td>
<td></td>
<td>0 6 5</td>
</tr>
</tbody>
</table>

Is this system in a safe state?

Does there exist a safe sequence?

\[ \text{MAXCLAIM}_i - \text{ALLOCATION}_i \leq \text{AVAIL} \leq \text{MAXCLAIM}_i \]
Deadlock Avoidance
Banker’s algorithm (Dijkstra and Habermann)

- When a process makes a request for resources...
  - Simulate the effect of granting the process’s request
  - Then check to see if a safe sequence exists

<table>
<thead>
<tr>
<th>ALLOCATION</th>
<th>MAX_CLAIM</th>
<th>AVAILABLE</th>
<th>MAX_REQUEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1, R_2, R_3$</td>
<td>$R_1, R_2, R_3$</td>
<td>$R_1, R_2, R_3$</td>
<td>$R_1, R_2, R_3$</td>
</tr>
<tr>
<td>$P_1$</td>
<td>0 0 1 2</td>
<td>0 0 1 2</td>
<td>0 0 0</td>
</tr>
<tr>
<td>$P_2$</td>
<td>1 7 5 0</td>
<td>1 7 5 0</td>
<td>0 7 5 0</td>
</tr>
<tr>
<td>$P_3$</td>
<td>3 5 3 2</td>
<td>3 5 3 2</td>
<td>1 0 0 3</td>
</tr>
<tr>
<td>$P_4$</td>
<td>0 6 3 2</td>
<td>0 6 3 2</td>
<td>0 0 2 0</td>
</tr>
<tr>
<td>$P_5$</td>
<td>0 6 1 4</td>
<td>0 6 1 4</td>
<td>0 6 4 2</td>
</tr>
</tbody>
</table>

Dealing With Deadlock
Deadlock detection & recovery

- Deadlock prevention and avoidance:
  - Develop and use resource allocation mechanisms and protocols that prohibit deadlock

- Deadlock detection and recovery:
  - Let the system deadlock and then deal with it
  - Detect that a set of processes are deadlocked
  - Recover from the deadlock

Deadlock Detection & Recovery
Detecting deadlock

- Run Banker’s algorithm and see if a safe sequence exists
  - Replace MAX_REQUEST with a "REQUEST" matrix
  - If a safe sequence does not exist then the system is deadlocked

<table>
<thead>
<tr>
<th>ALLOCATION</th>
<th>MAX_CLAIM</th>
<th>AVAILABLE</th>
<th>REQUEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1, R_2, R_3$</td>
<td>$R_1, R_2, R_3$</td>
<td>$R_1, R_2, R_3$</td>
<td>$R_1, R_2, R_3$</td>
</tr>
<tr>
<td>$P_1$</td>
<td>0 0 1 2</td>
<td>0 0 1 2</td>
<td>0 0 0</td>
</tr>
<tr>
<td>$P_2$</td>
<td>1 7 5 0</td>
<td>1 7 5 0</td>
<td>0 2 1 0</td>
</tr>
<tr>
<td>$P_3$</td>
<td>3 5 3 2</td>
<td>3 5 3 2</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>$P_4$</td>
<td>0 6 3 2</td>
<td>0 6 3 2</td>
<td>0 0 2 0</td>
</tr>
<tr>
<td>$P_5$</td>
<td>0 6 1 4</td>
<td>0 6 1 4</td>
<td>0 2 2 2</td>
</tr>
</tbody>
</table>
**Deadlock Detection & Recovery**

*Recovering from deadlock*

- Abort all deadlocked processes & reclaim their resources
- Abort one process at a time until all cycles in the RAG are eliminated
- Where to start?
  - Select low priority process
  - Processes with most allocation of resources
- Caveat: ensure system is in consistent state (e.g., transactions)
- Optimization:
  - Checkpoint processes periodically; rollback processes to checkpointed state