Deadlock in OS

Prof.Babahenini Mohamed Chaouki

Mohamed Khider Biskra University
Computer science departement

L3 - SE2 (2024/2025)

December 1, 2024

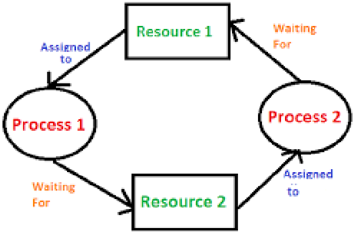
Outline

What is a Deadlock?

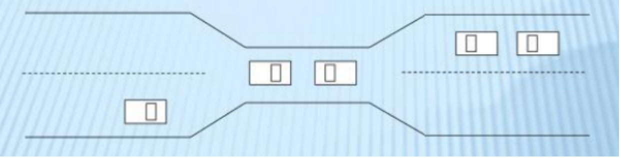
- Deadlock occurs when processes block each other indefinitely while waiting for resources.
- Examples of resources: CPU cycles, memory, files, printers.
- Common in multi-tasking and distributed systems.

Example Scenario:

- Process 1 holds Resource 1 and requests Resource 2.
- Process 2 holds Resource 2 and requests Resource 1.



Bridge crossing example



- Traffic only in one direction.
- Each section of a bridge can be viewed as a resource.
- 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 if a deadlock occurs.
- Starvation is possible

Necessary Conditions for Deadlock

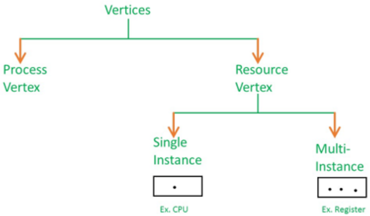
Deadlock can arise if four conditions hold simultaneously.

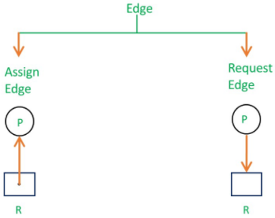
1. **Mutual Exclusion:** only one process can use a resource at a time. Resources cannot be shared.
2. **Hold and Wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes.
3. **No Preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task.
4. **Circular Wait:** A set of processes form a cycle, each waiting for a resource held by the next.

All four conditions must hold simultaneously for a deadlock to occur.

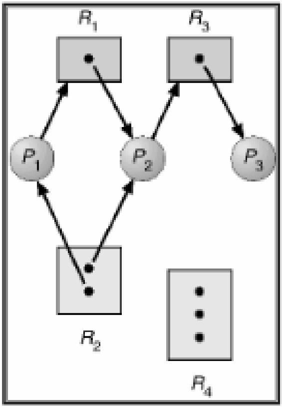
Resource Allocation Graph (RAG)

- A directed graph representing processes and resources.(A set of vertices V and a set of edges E).
- Nodes:
  - Circular nodes represent processes.
  - Rectangular nodes represent resources.
- Edges:
  - Request edge: process $\rightarrow$ resource.
  - Allocation edge: resource $\rightarrow$ process.





Example of a RAG



Approaches to Deadlock Detection and Recovery

- **Detection and Recovery:**
  - When deadlocks occur despite preventive measures, the operating system must identify and resolve them.
  - Detection algorithms like the Wait-For Graph analyze system states to pinpoint deadlocks.
  - Once detected, recovery methods like Process Rollback or Process Termination are employed.
  - The recovery algorithm releases the resources held by one or more processes, allowing the system to progress.
- **Prevention:**
  - The operating system proactively ensures that deadlocks cannot occur by always maintaining the system in a safe state.
  - This involves carefully controlling resource allocation to processes.
  - Techniques like the Banker's Algorithm are commonly used, where the system evaluates requests to guarantee that granting them will not lead to an unsafe state.

Detecting Deadlocks

Steps to Detect Deadlocks:

1. Construct a Resource Allocation Graph (RAG).
2. Look for cycles in the graph.
3. For multiple instances of resources, use a detection algorithm.

Example Detection Algorithm:

- Work array tracks available resources.
- Finish array flags if processes can complete.
- Simulate resource allocation and check for blocked processes.

Recovery Methods

Strategies for Recovery:

1. **Terminate Processes:**
   - Abort all deadlocked processes.
   - Abort one process at a time until the deadlock is resolved.
2. **Preempt Resources:**
   - Select a process to preempt resources from.
   - Rollback the process if necessary.

Challenges:

- High overhead for detecting deadlocks.
- Risk of starvation for some processes.

Safe and Unsafe States

Safe State:

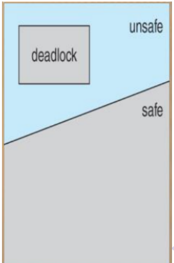
- If a system is in the safe state, there are no deadlocks.

Unsafe State:

- If a system is in an unsafe state, there is a possibility of deadlock.

Avoidance deadlock:

- Avoidance ensures that a system will never enter an unsafe state.



Banker's Algorithm

Introduction the Banker's Algorithm:

1. Multiple instances.
2. Each process must a priori claim maximum use.
3. When a process requests a resource, it may have to wait.
4. When a process gets all its resources it must return them in a finite amount of time

Steps in the Banker's Algorithm:

1. Check if the request is less than the current Need.
2. Check if resources are available to fulfill the request.
3. Temporarily allocate resources and check if the system remains in a safe state.
4. If the system is safe, grant the request; otherwise, deny it.

Steps of the Banker's Algorithm

1. Initialize:
   - Define available resources, allocated resources, and maximum resources for each process.
2. Request Resources:
   - Check if the request is valid:
     - Request $\leq$ Need
     - Request $\leq$ Available
3. Simulate Allocation:
   - Temporarily allocate the resources.
4. Check Safe State:
   - Ensure that the system remains in a safe state after allocation.
5. Approve or Deny:
   - Approve if safe, deny otherwise.

Detailed Steps of the Banker's Algorithm

- **Initialization**:
  - Define the following matrices:
    - **Available**: Vector of resources currently available.
    - **Maximum**: Maximum demand of each process for each resource.
    - **Allocation**: Resources currently allocated to each process.
    - **Need**: Resources each process still requires, calculated as:

$$\text{Need}[i,j] = \text{Maximum}[i,j] - \text{Allocation}[i,j]$$

- **Resource Request**:
  - A process P_i requests resources $Request[i]$.
  - Check:

$$\text{Request[i]} \leq \text{Need[i]}$$

$$\text{Request[i]} \leq \text{Available}$$

  - If the above conditions are not met, deny the request.

Detailed Steps of the Banker's Algorithm

- **Pretend Allocation**:
  - Temporarily allocate the resources to P_i by updating:

$$\text{Available} = \text{Available} - \text{Request[i]}$$

$$\text{Allocation[i]} = \text{Allocation[i]} + \text{Request[i]}$$

$$\text{Need[i]} = \text{Need[i]} - \text{Request[i]}$$

Detailed Steps of the Banker's Algorithm

- **Safety Check**:
  - Perform a **safety algorithm** to determine if the system is in a safe state:
    1. Initialize **Work = Available** and **Finish[i] = false** for all i.
    2. Find a process P_i such that:
       $$\text{Need[i]} \leq \text{Work}$$
    3. If such a process is found: Update Work = Work + Allocation[i] and Mark Finish[i] = true.
    4. Repeat until all processes are marked finished (Finish[i] = true for all i).
  - If all processes finish, the state is safe; otherwise, it's unsafe.

Detailed Steps of the Banker's Algorithm

- **Decision**:
  - If the system is in a safe state, grant the request.
  - If unsafe, rollback the allocation:

$$\text{Available} = \text{Available} + \text{Request}[i]$$

$$\text{Allocation}[i] = \text{Allocation}[i] - \text{Request}[i]$$

$$\text{Need}[i] = \text{Need}[i] + \text{Request}[i]$$

Example

5 processes P_0 through P_4; 3 resource types A (10 instances), B (5 instances), and C (7 instances). Snapshot at time T_0

- Available: [3, 3, 2]
- Max:

$$\begin{bmatrix} 7 & 5 & 3 \\ 3 & 2 & 2 \\ 9 & 0 & 2 \\ 2 & 2 & 2 \\ 4 & 3 & 3 \end{bmatrix}$$

- Allocation:

$$\begin{bmatrix} 0 & 1 & 0 \\ 2 & 0 & 0 \\ 3 & 0 & 2 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix}$$

Example (cont)

- The content of the matrix. Need is defined to be Max − Allocation.

	Need
	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

The system is in a safe state since the sequence < P_1, P_3, P_4, P_2, P_0> satisfies safety criteria.

- Check that Request ≤ Available (that is, (1,0,2) ≤ (3,3,2) ⇒ *true*.

	Allocation	*Need*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 1	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

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?

Advantages and disadvantages of the Banker's Algorithm

Advantages

- Prevents deadlocks by ensuring safe resource allocation.
- Allows systems to handle dynamic resource requests.
- Provides a systematic way to manage resources.

Disadvantages

- Requires prior knowledge of maximum resource needs.
- High computational overhead for checking safe states.
- Impractical for systems with many processes or resources.

Preventive Techniques

Eliminating Deadlock Conditions:

1. **Mutual Exclusion:** Use resources that can be shared.
2. **Hold and Wait:** Request all resources at once.
3. **No Preemption:** Allow resources to be preempted.
4. **Circular Wait:** Impose an ordering on resource requests.

Trade-offs:

- May reduce resource utilization.
- Increases complexity of system design.

Real-World Example: Printer and File

- Process A holds the printer and requests the file.
- Process B holds the file and requests the printer.

Resolution:

- Apply deadlock prevention by eliminating circular wait.
- Assign priorities to the printer and file requests.

Summary

- Deadlocks occur when processes block each other indefinitely.
- Characterized by mutual exclusion, hold and wait, no preemption, and circular wait.
- Techniques for handling deadlocks:
  - Detection and recovery.
  - Avoidance using safe states.
  - Prevention by breaking deadlock conditions.

Key Takeaway: Understand the system requirements and choose the appropriate strategy to handle deadlocks.