# CMPT 300
# Operating System I
## Deadlock

**Dr. Hazra Imran**

**Summer 2022**

# Example

5 processes and 3 resources R1 (10 instances), R2(5 instances) and R3 (7 instances)

| Process | Allocated | | | Max | | | Available | | | Need | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | R1 | R2 | R3 | R1 | R2 | R3 | R1 | R2 | R3 | R1 | R2 | R3 |
| P1 | 0 | 1 | 0 | 7 | 5 | 3 | 3 | 3 | 2 | | | |
| P2 | 2 | 0 | 0 | 3 | 2 | 2 | | | | | | |
| P3 | 3 | 0 | 2 | 9 | 0 | 2 | | | | | | |
| P4 | 2 | 1 | 1 | 2 | 2 | 2 | | | | | | |
| P5 | 0 | 0 | 2 | 4 | 3 | 3 | | | | | | |

Step 1: Compute Need Matrix

Need [i] = Max[i] - Allocated[I]

Step 2: Check if system is in safe state?

# Example

5 processes and 3 resources R1 (10 instances), R2(5 instances) and R3 (7 instances)

| Process | Allocated | | | Max | | | Available | | | Need (Max - Allocation) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | R1 | R2 | R3 | R1 | R2 | R3 | R1 | R2 | R3 | R1 | R2 | R3 |
| P1 | 0 | 1 | 0 | 7 | 5 | 3 | 3 | 3 | 2 | 7 | 4 | 3 |
| P2 | 2 | 0 | 0 | 3 | 2 | 2 | | | | 1 | 2 | 2 |
| P3 | 3 | 0 | 2 | 9 | 0 | 2 | | | | 6 | 0 | 0 |
| P4 | 2 | 1 | 1 | 2 | 2 | 2 | | | | 0 | 1 | 1 |
| P5 | 0 | 0 | 2 | 4 | 3 | 3 | | | | 4 | 3 | 1 |

- Check P1

P1 need <7,4,3> but available is <3,3,2>. Check for other processes

- Check P2

P2 needs <1,2,2> and available is <3,3,2> so resources will be allocated.

After finishing P2 will release its allocated resources as well.

# Example

5 processes and 3 resources R1 (10 instances), R2(5 instances) and R3 (7 instances)

| Process | Allocated | | | Max | | | Available | | | Need (Max - Allocation) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | R1 | R2 | R3 | R1 | R2 | R3 | R1 | R2 | R3 | R1 | R2 | R3 |
| P1 | 0 | 1 | 0 | 7 | 5 | 3 | | | | 7 | 4 | 3 |
| <P2> | | | | | | | 5 | 3 | 2 | | | |
| P3 | 3 | 0 | 2 | 9 | 0 | 2 | | | | 6 | 0 | 0 |
| P4 | 2 | 1 | 1 | 2 | 2 | 2 | | | | 0 | 1 | 1 |
| P5 | 0 | 0 | 2 | 4 | 3 | 3 | | | | 4 | 3 | 1 |

<P2>

- Check P3, resources cannot be allocated. Next process
- Check P4. We can allocate resources.

# Example

5 processes and 3 resources R1 (10 instances), R2(5 instances) and R3 (7 instances)

| Process | Allocated | | | Max | | | Available | | | Need (Max - Allocation) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | R1 | R2 | R3 | R1 | R2 | R3 | R1 | R2 | R3 | R1 | R2 | R3 |
| P1 | 0 | 1 | 0 | 7 | 5 | 3 | | | | 7 | 4 | 3 |
| | | | | | | | | | | | | |
| P3 | 3 | 0 | 2 | 9 | 0 | 2 | | | | 6 | 0 | 0 |
| | | | | | | | 7 | 4 | 3 | | | |
| P5 | 0 | 0 | 2 | 4 | 3 | 3 | | | | 4 | 3 | 1 |

- Check P3, resources cannot be allocated. Next process
- Check P4. We can allocate resources.

<P2, P4>

# Example

5 processes and 3 resources R1 (10 instances), R2(5 instances) and R3 (7 instances)

| Process | Allocated | | | Max | | | Available | | | Need (Max - Allocation) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | R1 | R2 | R3 | R1 | R2 | R3 | R1 | R2 | R3 | R1 | R2 | R3 |
| P1 | 0 | 1 | 0 | 7 | 5 | 3 | | | | 7 | 4 | 3 |
| | | | | | | | | | | | | |
| P3 | 3 | 0 | 2 | 9 | 0 | 2 | | | | 6 | 0 | 0 |
| | | | | | | | 7 | 4 | 3 | | | |
| P5 | 0 | 0 | 2 | 4 | 3 | 3 | | | | 4 | 3 | 1 |

- Check P5. We can allocate resources.

<P2, P4>

# Example

5 processes and 3 resources R1 (10 instances), R2(5 instances) and R3 (7 instances)

| Process | Allocated | | | Max | | | Available | | | Need (Max - Allocation) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | R1 | R2 | R3 | R1 | R2 | R3 | R1 | R2 | R3 | R1 | R2 | R3 |
| P1 | 0 | 1 | 0 | 7 | 5 | 3 | | | | 7 | 4 | 3 |
| | | | | | | | | | | | | |
| P3 | 3 | 0 | 2 | 9 | 0 | 2 | | | | 6 | 0 | 0 |
| | | | | | | | | | | | | |
| | | | | | | | 7 | 4 | 5 | | | |

- Check P5. We can allocate resources.

<P2, P4, P5>

# Example

5 processes and 3 resources R1 (10 instances), R2(5 instances) and R3 (7 instances)

| Process | Allocated | | | Max | | | Available | | | Need (Max - Allocation) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | R1 | R2 | R3 | R1 | R2 | R3 | R1 | R2 | R3 | R1 | R2 | R3 |
| P1 | 0 | 1 | 0 | 7 | 5 | 3 | | | | 7 | 4 | 3 |
| | | | | | | | | | | | | |
| P3 | 3 | 0 | 2 | 9 | 0 | 2 | | | | 6 | 0 | 0 |
| | | | | | | | | | | | | |
| | | | | | | | 7 | 4 | 5 | | | |

- Check P1
- We can allocate resources

<P2, P4, P5>

# Example

5 processes and 3 resources R1 (10 instances), R2(5 instances) and R3 (7 instances)

| Process | Allocated | | | Max | | | Available | | | Need (Max - Allocation) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | R1 | R2 | R3 | R1 | R2 | R3 | R1 | R2 | R3 | R1 | R2 | R3 |
| P1 | | | | | | | 7 | 5 | 5 | | | |
| | | | | | | | | | | | | |
| P3 | 3 | 0 | 2 | 9 | 0 | 2 | | | | 6 | 0 | 0 |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |

- Check P1
- We can allocate resources

<P2, P4, P5, P1>

# Example

5 processes and 3 resources R1 (10 instances), R2(5 instances) and R3 (7 instances)

| Process | Allocated | | | Max | | | Available | | | Need (Max - Allocation) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | R1 | R2 | R3 | R1 | R2 | R3 | R1 | R2 | R3 | R1 | R2 | R3 |
| P1 | | | | | | | 7 | 5 | 5 | | | |
| | | | | | | | | | | | | |
| P3 | 3 | 0 | 2 | 9 | 0 | 2 | | | | 6 | 0 | 0 |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |

- Check P3
- We can allocate resources

Now available will be
<10,5, 7>

<P2, P4, P5, P1, P3>

# Detect and Recover

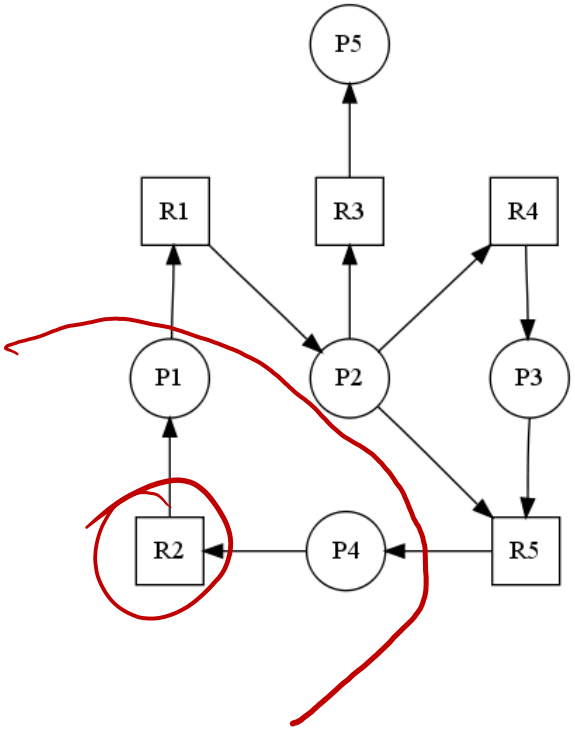
**Allow deadlock to occur, then take actions to resolve it**

- Employ deadlock detection and recovery techniques
  - A deadlock detector runs periodically
- One possible approach: maintain a graph that summarizes dependencies and check it for cycles
  - Cycle → potential deadlock → restart system or preempt some threads (aka “victims”)
- Common in database systems

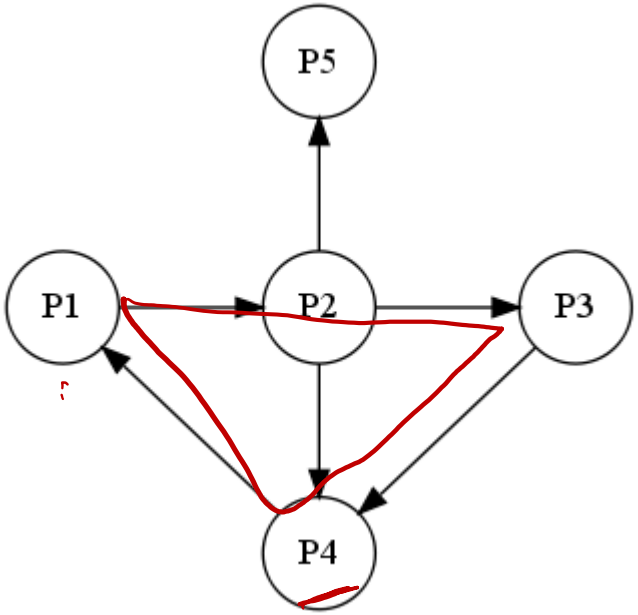
# Single instance

Resource-allocations graphs for deadlock detection

resource-allocation graph:




corresponding wait-for graph:



# Multiple instances

- Use an algorithm similar to Banker's, which simply investigates *every* possible allocation sequence for the processes which remain to be completed.

# Summary

- Deadlock: A set of processes each holding a resource and waiting for a resource held by another process in the set
- Four necessary conditions
  - Mutual exclusion, no preemption, hold and wait, circular wait
- Deadlock handling
  1. Prevention: ensure that at least one of the necessary conditions does not hold
  2. Avoidance: rely on global knowledge to schedule threads/processes so that no deadlock could occur
  3. Detection and Recovery: allow deadlocks to form, then resolve them
     - Need to track dependencies for detecting deadlocks
     - Common in database systems