

# Week 7 – Deadlocks & Resource Allocation

## Objective

The goal of Week 7 was to understand how deadlocks occur in operating systems, how to represent them using Resource Allocation Graphs (RAGs), and how to apply Banker's Algorithm to determine whether a system is in a safe or unsafe state.

## Steps Taken

### 1. Deadlock Conditions

A deadlock can only occur when **all four** of these conditions hold:

- **Mutual Exclusion** Only one process can use a resource at a time.
- **Hold and Wait** A process holds one resource while waiting for another.
- **No Preemption** Resources cannot be forcibly taken away; they must be released voluntarily.
- **Circular Wait** A cycle exists where each process waits for a resource held by the next process.

If **any one** of these conditions is broken, deadlock cannot occur.

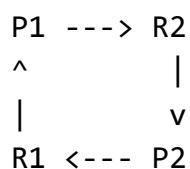
### 2. Resource Allocation Graph (RAG)

#### Scenario

- $P_1$  holds  $R_1$  and requests  $R_2$
- $P_2$  holds  $R_2$  and requests  $R_1$

#### ASCII Diagram (Deadlock Cycle)

Code



## Conclusion

There is a cycle ( $P_1 \rightarrow R_2 \rightarrow P_2 \rightarrow R_1 \rightarrow P_1$ ), so the system is in **deadlock**.

## 3. Detection vs Prevention vs Avoidance

- **Deadlock Detection** The system allows deadlocks to occur and then detects them (e.g., by finding cycles).
- **Deadlock Prevention** The system ensures at least one of the four deadlock conditions never holds.
- **Deadlock Avoidance (Banker's Algorithm)** The system checks each request and only grants it if the system remains in a **safe state**.

## 4. Banker's Algorithm – Worked Example

### Given Resources

- Total resources:  $A = 7, B = 2, C = 6$

### Allocation Table

Process	Allocation	Max	Need (Max – Allocation)
P0	1 0 2	3 1 3	2 1 1
P1	2 1 1	4 1 3	2 0 2
P2	0 0 2	3 1 4	3 1 2

### Available Resources

Total allocated =  $(1+2+0, 0+1+0, 2+1+2) = (3, 1, 5)$  Available = Total – Allocated =  $(4, 1, 1)$

### Safe Sequence Check

1. **Check P0 Need (2,1,1)  $\leq$  Available (4,1,1)**  $\rightarrow$  YES
  - a. Run P0
  - b. Release (1,0,2)
  - c. New Available = (5,1,3)
2. **Check P1 Need (2,0,2)  $\leq$  (5,1,3)**  $\rightarrow$  YES
  - a. Run P1
  - b. Release (2,1,1)
  - c. New Available = (7,2,4)
3. **Check P2 Need (3,1,2)  $\leq$  (7,2,4)**  $\rightarrow$  YES
  - a. Run P2

- b. Release (0,0,2)
- c. New Available = (7,2,6)

## Final Safe Sequence

{ P0, P1, P2 } The system is in a **safe state**.

## Reflection

This week helped me understand how deadlocks occur and why they can completely stop a system from making progress. By learning the four deadlock conditions, I saw that deadlocks are not random—they happen only when specific rules are all true at the same time. Drawing a Resource Allocation Graph made it easier to visualize how processes and resources interact, and how a simple cycle can cause a complete standstill.

Working through Banker's Algorithm gave me hands-on experience with checking safe states. Calculating Need, Available, and finding a safe sequence showed me how the operating system decides whether it is safe to grant a resource request. This made the idea of "safe" and "unsafe" states much clearer.

Overall, Week 7 strengthened my understanding of how operating systems prevent system-wide failures. I now feel more confident explaining deadlocks, detecting them, and using Banker's Algorithm to avoid unsafe states.