Operating Systems

Karim Lounis

Queen’s University, Canada

Winter’19, Jan-Apr 2019
Deadlocks

Deadlocks, Deadlocks prevention, detection, and avoidance
Deadlocks

Definition

**Deadlock**: Is a system state in which processes or threads are waiting for some events (availability of some resources) that will never occur.

A set of processes $P = \{p_1, \ldots, p_n\}$ is deadlocked if: $\forall p_i \in P : p_i$ is waitn’ for an event that can only be caused by another process $p_j \neq i \in P$.
Deadlocks

Three ways to deal with deadlocks:

1. Deadlock ignorance.
2. Deadlock avoidance or prevention.
3. Deadlock detection and recovery.
Resource Allocation Graph

**Definition**

**RAG (Resource Allocation Graph).** is a directed graph where the vertices \( V = P \cup R \) represent the set of processes \( P = \{P_1, \ldots, P_n\} \) and system resources \( R = \{R_1, \ldots, R_m\} \), whereas the edges \( E \) express which process \( P_i \in P \) has requested or is holding a given resource \( R_j \in R \).

- An edge from process \( P_i \in P \) to a resource \( R_j \in R \); signifies that process \( P_i \in P \) has requested an instance of resource type \( R_j \in R \) and its currently waiting for it (\( P_i \rightarrow R_j \) is called request edge).

- An edge from a resources \( R_i \in R \) to a process \( P_j \in P \); signifies that process \( P_j \in P \) has been allocated an instance of resource type \( R_j \in R \) (\( R_i \rightarrow P_i \) is called assignment edge).

- Each process \( P_i \in P \) is depicted by a circle whereas resources \( R_i \in R \) are represented by rectangles.
Resource Allocation Graph

- Process $P_1$ requests resource $R_1$ held by $P_2$.
- Process $P_2$ requests resource $R_2$ held by $P_1$.
- Deadlocks can occur in this graph.

Karim Lounis (Queen’s University, Canada)
We can use a RAG (Resource Allocation Graph) to identify whether a particular resource allocation sequence $S$ is deadlock-free or not.

If the RAG does not contain any closed path (cycle), then the sequence is **deadlock free**.

Deadlock $\implies$ Directed cycle inside the RAG

Else **deadlock may or may not happen**. Why?

Directed cycle inside the RAG $\nRightarrow$ Deadlock

But for resource types with a single instance:

Deadlock $\iff$ Directed cycle inside the RAG
“No deadlock now” does not guarantee anything about “no deadlock in the future”
A third type of edges was introduced, called **Claim Edges**.

![Resource Allocation Graph](image)

**Process Synchronization**  
**Deadlocks**

---

Karim Lounis  (Queen's University, Canada)  
Operating Systems  
Winter’19, Jan-Apr 2019  
9 / 17
A third type of edges was introduced, called **Claim Edges**.

Two possible scenarios, one safe and the other unsafe (may lead to deadlock).
Graphs with claim edges: There is potential for deadlock (if OS grants requests)
If OS grants P1 two instances of resource type R1, then: No deadlock
If OS grants P2 two instances of resource type R1, then: **Deadlock may occur** (Unsafe state).

![Resource Allocation Graph](image-url)
Safe State & Safe Sequence

**Definition**

**Safe state.** Is a state in which a system can allocate resources to processes in some order that avoid falling into a deadlock.

In a system with \( n \)-processes and \( m \)-resources where processes request different numbers and types of resources:

- There will be different sequences of processes \( < P_1, P_3, P_2, \ldots, P_k > \);
- Certain sequences leave the system in a safe state.
- Other sequences may drive the system to an unsafe state.

**Definition**

**Safe sequence.** Is a sequence \( S \) of processes: \( \forall P_i \in S \), the resources requested by \( P_i \in S \) can be allocated from the currently available resources plus the resources held by all \( P_j < i \in S \) (i.e., process \( P_i \in S \) may wait).
Safe State & Safe Sequence

**Example.** Consider a system that has 12 instances of a given resources type e.g., printer, and consider three processes $P_0$, $P_1$, and $P_2$ such that:

- Process $P_0$ needs 10 printers to complete its execution.
- Process $P_1$ needs 4 printers to complete its execution.
- Process $P_2$ needs 9 printers to complete its execution.

If at a given time $t_0$ process $P_0$ is holding 5 printers, and $P_1$ and $P_2$ are holding 2 printers each (The system is in a safe state.)

The sequence $< P_1, P_0, P_2 >$ is a **safe sequence**, whereas, the sequence $< P_0, P_1, P_2 >, < P_2, P_1, P_0 >, < P_1, P_2, P_0 >$ are not.

At $t_1$, if $P_2$ is allocated one additional printer, the system goes from a safe state to an **unsafe state**.
Safe State & Safe Sequence

The idea of **deadlock avoidance** consists of **determining** whether a given process that is requesting a given resource **must wait** or **must be immediately allocated** the resource.

Resource is allocated $\Rightarrow$ the system should remain in a safe state.

Resource is granted **only if** the allocation leaves the system in a safe state.

If a process requests a resource that is **currently available**, it may still have to wait.
End.