Operating Systems: Deadlocks

Neerja Mhaskar

Department of Computing and Software, McMaster University, Canada

Acknowledgements: Material based on the textbook Operating Systems Concepts (Chapters 5)

System Model

- System consists of resources
- Resource types $R_1, R_2, . . ., R_m$
  - *CPU cycles, memory space, I/O devices*
- Each resource type R_i has W_i instances.
  - Eg: You can have two instances of a printer available
- Each process utilizes a resource as follows:
  - **request**
  - **use**
  - **Release**
- if the resources not available, the process enters a *waiting state*.

Deadlock

- **Deadlock –** when a waiting process has requested a resource held by other waiting processes.
- In a deadlock processes never finish executing
  - System resources are tied up.
  - Thus, preventing other processes from starting.

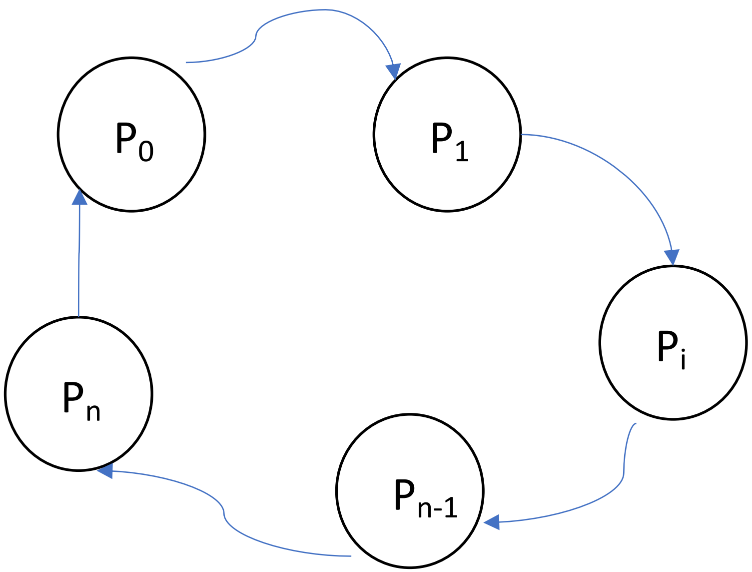
Deadlock Characterization

Deadlock ⇔ below **four conditions hold** at the same time.

- **Mutual exclusion:** only one process at a time can use a resource.
- **Hold and wait**: a process holding at least one resource is **waiting to acquire additional resources** held by other processes.
- **No preemption**: a resource can be **released only voluntarily** by the process holding it, after that process finishes its task.

Deadlock Characterization contd..

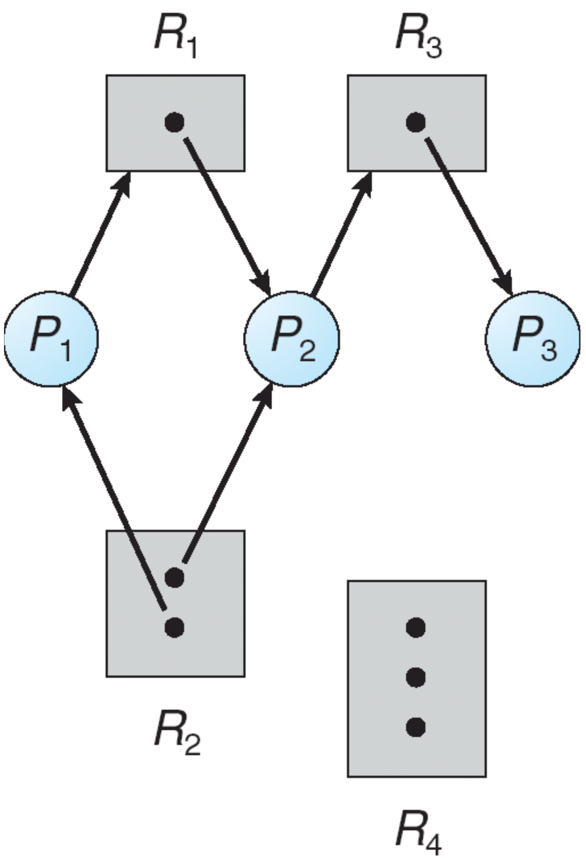
- **Circular wait:** there exists. a set $\{P_0, P_1, \ldots, P_n\}$ of **waiting processes** such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by P_2, …, P_{n-1} is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0.



circular-wait condition => the hold-and-wait condition.

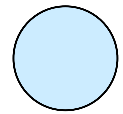
Resource-Allocation Graph

- **System resource-allocation graph** (directed graph G = (V, E)) used to describe deadlocks.
- V is partitioned into two sets:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system
- E has two types of edges:
  - **Request edge** – directed edge $P_i \rightarrow R_j$: Process P_i requested an instance of resource R_j and is waiting.
  - **Assignment edge** – directed edge $R_j \rightarrow P_i$: Resource R_j assigned to process P_i

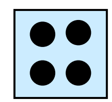


Resource-Allocation Graph (Cont.)

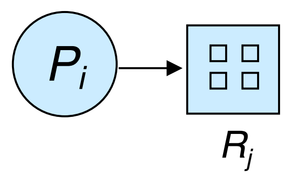
- Process



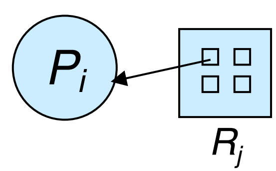
- Resource Type with 4 instances



- P_i requests instance of R_j

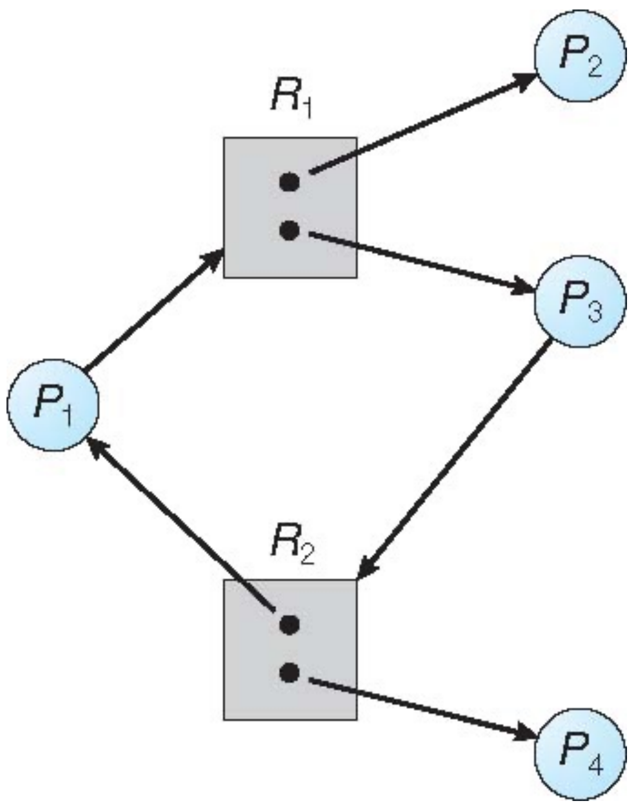


- P_i is holding an instance of R_j

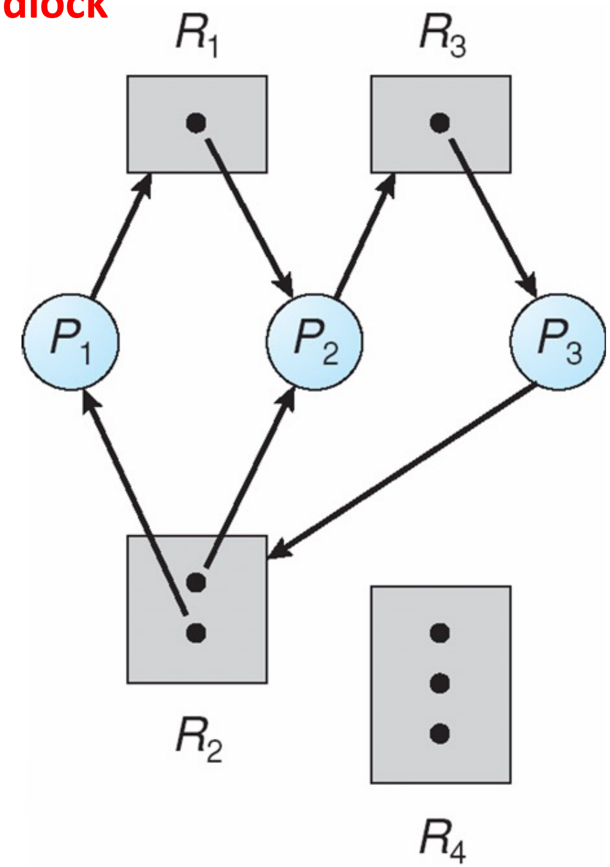


Resource Allocation Graph Examples

Resource allocation graph with a cycle but no Deadlock



Resource allocation graph with a cycle and a Deadlock



Cycles with circular wait condition satisfied

- *P1 → R1 → P2 → R3 → P3 → R2 → P1*
- *P2 → R3 → P3 → R2 → P2*

Basic Facts

- If a graph contains no cycles $\Rightarrow$ no deadlock
- If graph contains a cycle $\Rightarrow$
  - if only one instance per resource type, then deadlock
  - if several instances per resource type, possibility of deadlock

Methods for Handling Deadlocks

- Ensure that the system will ***never*** enter a deadlock state:
  - **Deadlock prevention:** ensure that atleast one of the necessary conditions (stated in slide #4) cannot hold.
    - Only the Circular Wait condition is a practical option for deadlock prevention.
  - **Deadlock avoidance**: requires that OS be given additional information in advance about the type and no. of resources a process will request. Based on this data OS decides the waiting strategy for the process.

Methods for Handling Deadlocks

- **Deadlock detection:** Allow the system to enter a deadlock state and then recover.
- **IGNORE!** the problem and pretend that deadlocks never occur; used by most operating systems such as UNIX, Linux, and Windows!

Deadlock Avoidance Strategy

- Requires that the OS be given **additional information in advance** about the type and no. of resources a process will request.
- Each process declares its ***maximum need = number* of resources** of each type that the process may need.
- The deadlock-avoidance algorithm dynamically examines the *resource-allocation state* to ensure that there can never be a **circular-wait condition.**

Resource allocation state

Resource-allocation state is defined by

- Number of available resources
- Number of allocated resources
- the maximum need of all the processes in the system.

Example: Consider a system with a single resource type – magnetic tapes= 12.

Resource allocation state of the system at time T_0:

	Maximum Need	Allocated resources	Available
P0	10	5	12-9 = 3
P1	4	2	
P2	9	2	

Safe State

- System is in **safe state** if there exists a sequence $<P_1, P_2, \ldots, P_n>$ of ALL the processes in the system such that for each $P_{i:}$
  - Resources P_i needs $\leq$ total available resources + resources held by/allocated to all $P_1, P_2, \ldots, P_{i-1}$.
- We compute the needs of a process P_i as below

P_is Need = Max. Need of Process P_i – Allocated resources for process P_i

Safe State Example

Is the below system at time t_0 in safe state?

- Consider a system with 12 magnetic tapes.
- **Resource allocation state** of the system at time t_0:

	Maximum Needs	*Allocated resources*	*Available*
*P*0	10	5	12-9 = 3
*P*1	4	2	
*P*2	9	2	

Safe State Example – with Need vector

- ***P_is Need = Max. Need of Process P_i – Allocated resources for process P_i***
- **Resource allocation state** of the system at time t_0 with need vector:

	Maximum Needs	*Allocated resources*	*Need*	*Available*
*P*0	10	5	5	3
*P*1	4	2	2	
*P*2	9	2	7	

Is there a sequence consisting of ALL processes ($P_1, P_2, \ldots, P_n$) in the system such that every process satisfies the safe state requirement?

Safe state requirement is

Resources P_i needs $\leq$ total available resources + resources held by/allocated to all $P_1, P_2, \ldots, P_{i-1}$

YES! The sequence is **<P_1, P_0, P_2>**

Safe State Example Cont.

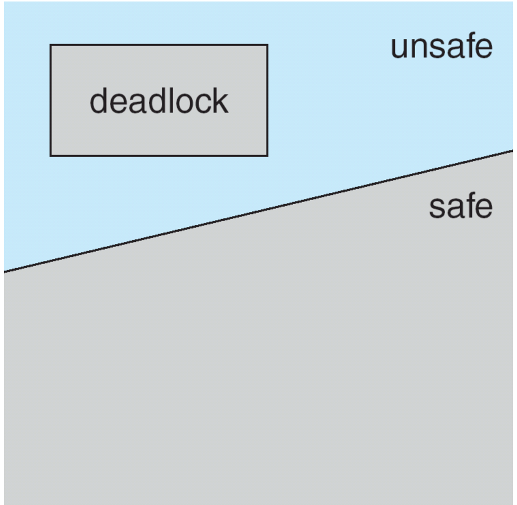
- Suppose at time t_1 process P_2 *requests an additional tape drive* and is assigned as well.
- **Resource allocation state** of the system at time t_1:

Process	*Maximum Needs*	*Allocated resources*	*Need*	*Available*
*P*0	10	5	5	2
*P*1	4	2	2	
*P*2	9	3	6	

  - Available resources at time t_1= 2
- Is the system at time t_1 in safe state?

Basic Facts

- If a system is in safe state $\Rightarrow$ no deadlocks
- If a system is in unsafe state $\Rightarrow$ possibility of deadlock
- Avoidance $\Rightarrow$ ensure that a system will never enter an unsafe state.



Deadlock Avoidance algorithm Outline

- Given the resource allocation state of a system,
  - The algorithm checks if the system is in a *safe state*.
  - If the system is in a safe state, whenever a process requests an instance of a resource type,
    - It checks to see if allocating the resources continues to have the system in a safe state.
      - If yes -- allocate the resources.
      - If no -- have the process wait.