LECTURE 11
Deadlock Prevention

Introduction :The Deadlock problem

- In a computer system deadlocks arise when members of a group of processes which hold resources are blocked indefinitely from access to resources held by other processes within the group.

Deadlock example

- P_i requests one I/O controller and the system allocates one.
- P_j requests one I/O controller and again the system allocates one.
- P_i wants another I/O controller but has to wait since the system ran out of I/O controllers.
- P_j wants another I/O controller and waits.

Strategies for handling deadlocks(Deadlock preliminaries)

- Deadlock prevention. Prevents deadlocks by restraining requests made to ensure that at least one of the four deadlock conditions cannot occur.
- Deadlock avoidance. Dynamically grants a resource to a process if the resulting state is safe. A state is safe if there is at least one execution sequence that allows all processes to run to completion.
- Deadlock detection and recovery. Allows deadlocks to form; then finds and breaks them.

Deadlock Prevention

- 1. A process acquires all the needed resources simultaneously before it begins its execution, therefore breaking the hold and wait condition.
- E.g. In the dining philosophers' problem, each philosopher is required to pick up both forks at the same time. If he fails, he has to release the fork(s) (if any) he has acquired.
- Drawback: over-cautious.

Cont..

- 2. All resources are assigned unique numbers. A process may request a resource with a unique number I only if it is not holding a resource with a number less than or equal to I and therefore breaking the circular wait condition.
- E.g. In the dining philosophers problem, each philosopher is required to pick a fork that has a larger id than the one he currently holds. That is, philosopher P5 needs to pick up fork F5 and then F1; the other philosopher Pi should pick up fork Fi followed by Fi-1.
- Drawback: over-cautions.

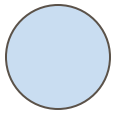
Resource-Allocation Graph(Model of deadlock, resources)

A set of vertices V and a set of edges E.

- V is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, 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.
- request edge – directed edge $P_i \rightarrow R_j$
- assignment edge – directed edge $R_j \rightarrow P_i$

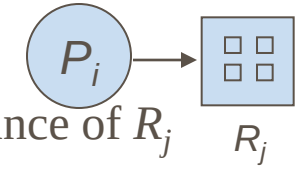
Resource-Allocation Graph (Cont.)

- Process

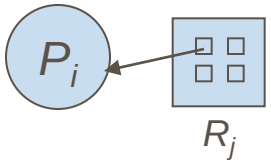


- Resource Type with 4 instances

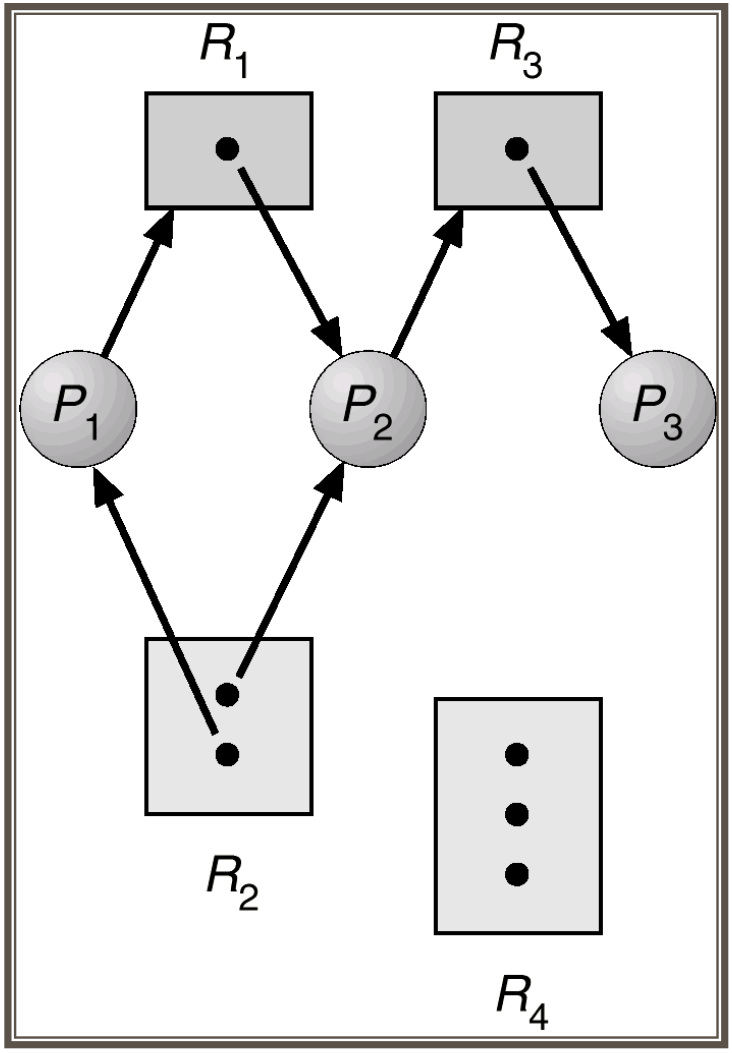
- P_i requests an instance of R_j



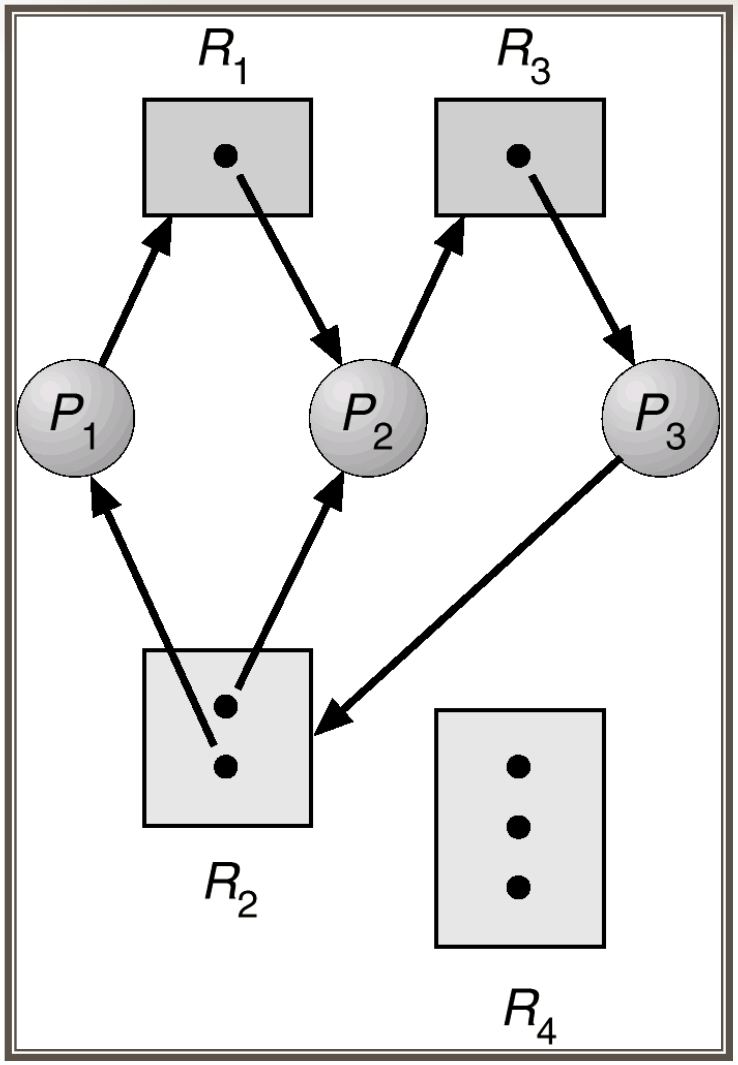
- P_i is holding an instance of R_j



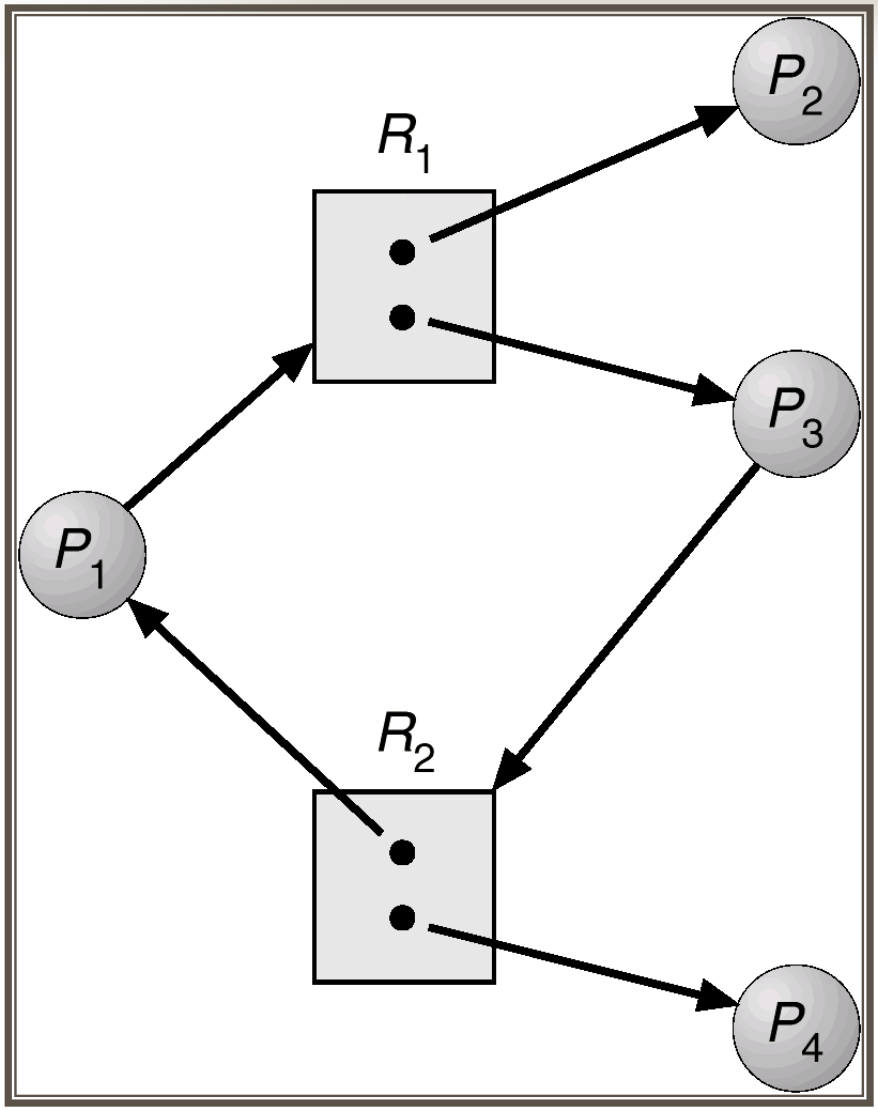
Example of a Resource Allocation Graph



Resource Allocation Graph With A Deadlock



Resource Allocation Graph With A Cycle But No Deadlock



Basic Facts(Necessary & sufficient condition for deadlock)

- If the resource allocation graph contains no cycles $\Rightarrow$ no deadlock.
- If the resource allocation graph contains a cycle $\Rightarrow$
  - if only one instance per resource type is available in the system, then there is a deadlock.
  - if several instances per resource type, possibility of deadlock exists.

Deadlock Avoidance

Introduction

- Possible side effects of preventing deadlock are low device utilization and reduce system throughput

- An alternative method for avoiding deadlocks is to require additional information about how resources are to be required.
- With this knowledge of the complete sequence of requests and releases for each process the system can decide for each request whether or not the process should wait in order to avoid possible future deadlock.
- A deadlock avoidance algorithm dynamically examines the resource-allocation state to ensure that a **circular wait** condition can never happen

Safe State

- A state is **safe** if the system can allocate resources to each processes in some order **(safe sequence)** and still avoid a deadlock.
- If no such sequence exists, then the system state is said to be **unsafe.**
- If we have prior knowledge of how resources will be requested, it's possible to determine if we are entering an "unsafe" state.

Cont…

Possible states are:

Deadlock No forward progress can be made.

Unsafe state A state that **may** allow deadlock.

Safe state A state is safe if a sequence of processes exist such that there are enough resources for the first to finish, and as each finishes and releases its resources there are enough for the next to finish.

The rule is simple: If a request allocation would cause an unsafe state, do not honor that request.

Safe State Example

- Let's assume a very simple model: each process declares its maximum needs. In this case, algorithms exist that will ensure that no unsafe state is reached. *Maximum needs* does NOT mean it *must* use that many resources – simply that it *might* do so under some circumstances.
- There exists a total of 12 resources. Each resource is used exclusively by a process. The current state looks like this:

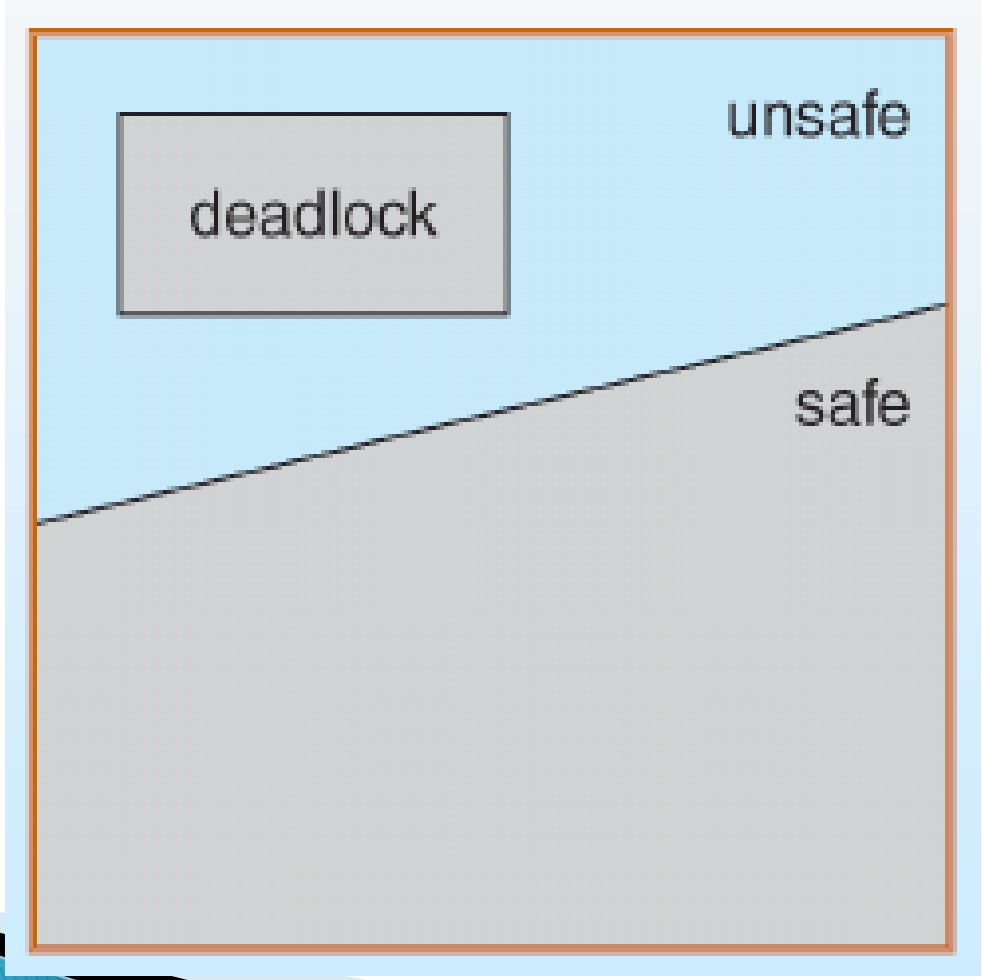
	Max needs	Current needs
P0	10	5
P1	4	2
P2	9	2

At T0, the system is in safe state, since <P1,P0,P2> satisfied safe state condition

What if P2 currently ask for one more tape and has that one?

Safe, Unsafe, Deadlock

Note: All deadlocks are unsafe, but all unsafes are NOT deadlocks.



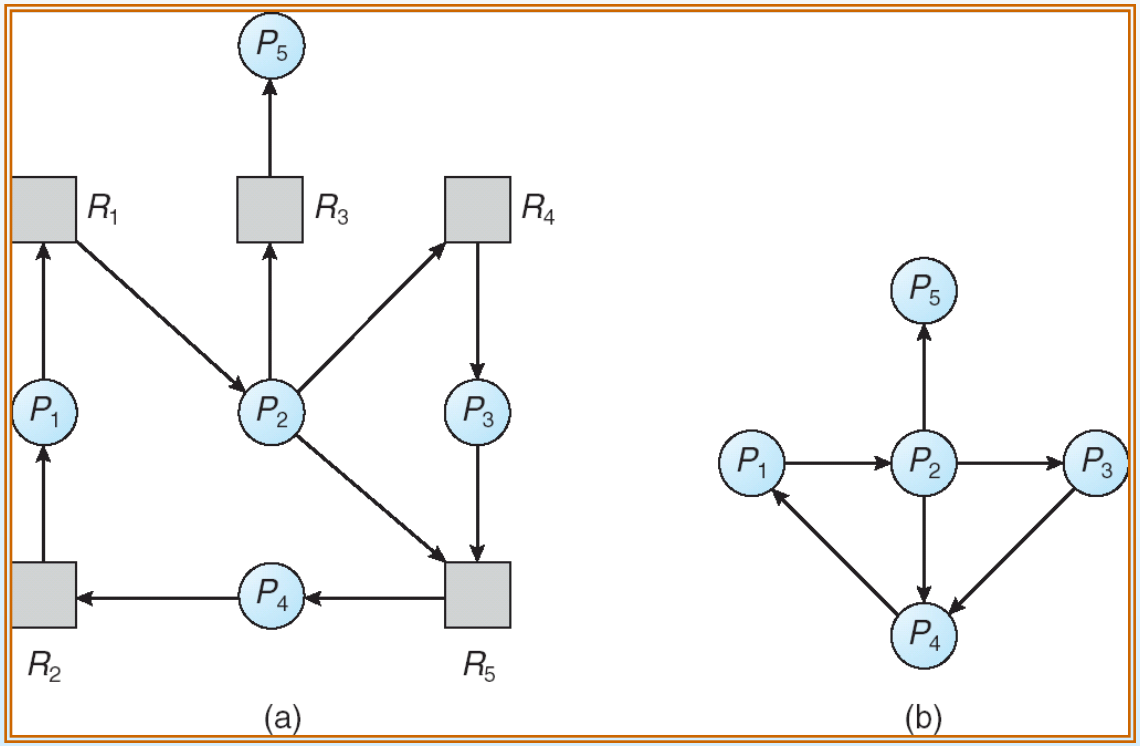
Deadlock Detection

- Allow system to enter deadlock state
- Detection algorithm
- Recovery scheme

System with Single unit request

- Maintain *wait-for* graph
  - Nodes are processes.
  - $P_i \rightarrow P_j$ if P_i is waiting for P_j.
- Periodically invoke an algorithm that searches for a cycle in the graph. If there is a cycle, there exists a deadlock.
- An algorithm to detect a cycle in a graph requires an order of n^2 operations, where n is the number of vertices in the graph.

Resource-Allocation Graph and Wait-for Graph



Resource-Allocation Graph

Corresponding wait-for graph

Consumable resource, reusable resource(Several Instances of a Resource Type)

- ***Available**:* A vector of length *m* indicates the number of available resources of each type.

- ***Allocation**:* An *n* x *m* matrix defines the number of resources of each type currently allocated to each process.

- ***Request**:* An *n* x *m* matrix indicates the current request of each process. If *Request* $[i_j] = k$, then process P_i is requesting *k* more instances of resource type. R_j.

Detection Algorithm

1. Let *Work* and *Finish* be vectors of length *m* and *n*, respectively Initialize:

   (a) *Work* = *Available*

   (b) For i = 1,2, …, n, if $Allocation_i \neq 0$, then *Finish*[i] = false;otherwise, *Finish*[i] = *true*.

2. Find an index *i* such that both:

   (a) *Finish*[*i*] == *false*

   (b) $Request_i \leq Work$

   If no such *i* exists, go to step 4.

Detection Algorithm (Cont.)

3. *Work* = *Work* + *Allocation*$_i$
   Finish[*i*] = *true*
   go to step 2.

4. If *Finish*[*i*] == false, for some *i*, $1 \leq i \leq n$, then the system is in deadlock state. Moreover, if *Finish*[*i*] == *false*, then P_i is deadlocked.

Algorithm requires an order of O(m x $n^{2)}$ operations to detect whether the system is in deadlocked state.

Example of Detection Algorithm

- Five processes P_0 through P_4; three resource types A (7 instances), *B* (2 instances), and *C* (6 instances).
- Snapshot at time T_0:

	Allocation	*Request*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

- Sequence <P_0, P_2, P_3, P_1, P_4> will result in *Finish*[*i*] = true for all *i*.

Example (Cont.)

- P_2 requests an additional instance of type *C*.

	Request
	A B C
P_0	0 0 0
P_1	2 0 1
P_2	0 0 1
P_3	1 0 0
P_4	0 0 2

- State of system?
  - Can reclaim resources held by process P_0, but insufficient resources to fulfill other processes; requests.
  - Deadlock exists, consisting of processes P_1, P_2, P_3, and P_4.

Detection-Algorithm Usage

- When, and how often, to invoke depends on:
  - How often a deadlock is likely to occur?
  - How many processes will need to be rolled back?
    - One for each disjoint cycle
- If detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph and so we would not be able to tell which of the many deadlocked processes “caused” the deadlock.

Recovery from Deadlock: Process Termination

- Abort all deadlocked processes.
- Abort one process at a time until the deadlock cycle is eliminated.
- In which order should we choose to abort?
  - Priority of the process.
  - How long process has computed, and how much longer to completion.
  - Resources the process has used.
  - Resources process needs to complete.
  - How many processes will need to be terminated.
  - Is process interactive or batch?

Recovery from Deadlock: Resource Preemption

- Selecting a victim – minimize cost.
- Rollback – return to some safe state, restart process for that state.
- Starvation – same process may always be picked as victim, include number of rollback in cost factor.