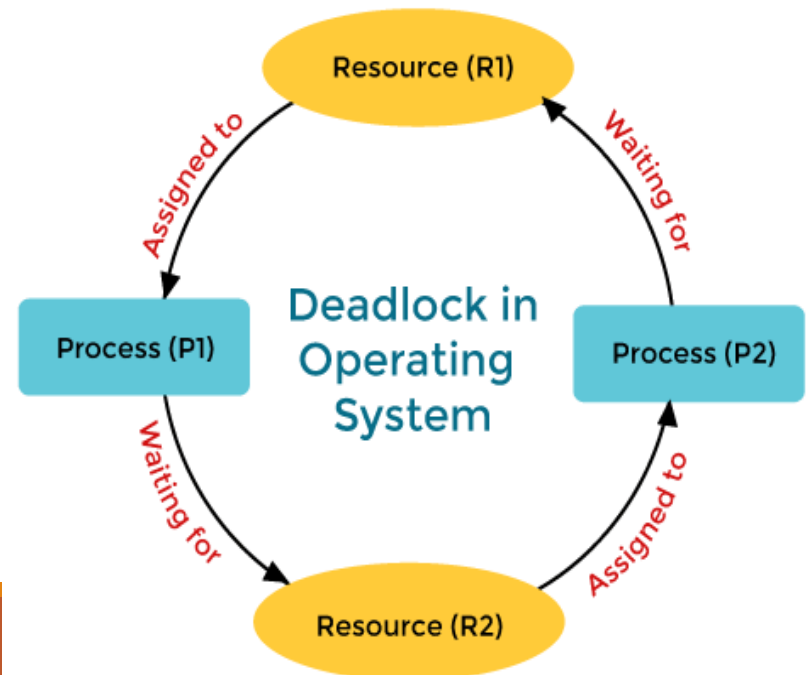


DEADLOCK

IN

OPERATING SYSTEM

PRESENTED BY-
HIRA SIDDIQUI
212123022



DEADLOCKS

EXAMPLES:

"It takes money to make money".

" You can't get a job without experience; you can't get experience without a job ".

BACKGROUND:

The cause of deadlocks: Each process needs what another process has. This results from sharing resources such as memory, devices, and links.

DEFINITION:

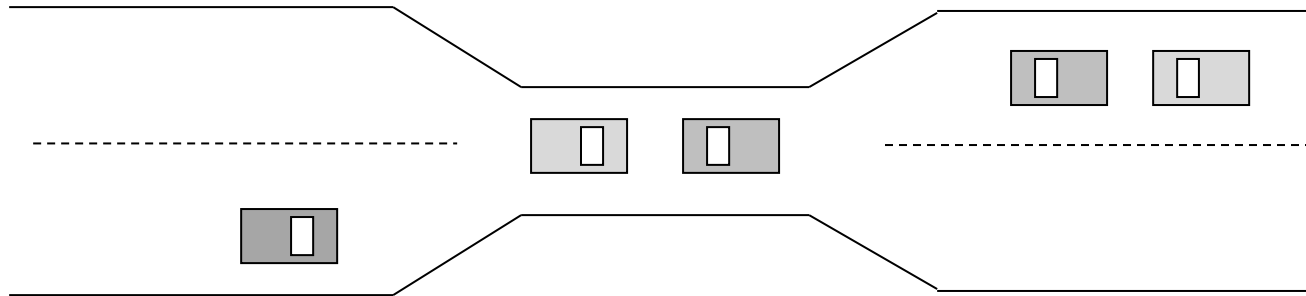
A *deadlock* is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

SYSTEM MODEL

- System consists of resources.
- Resource types R_1, R_2, \dots, R_m .
- Each resource type R_i has W_i instances.
- Under normal operation, Each process utilizes a resource as follows:
 1. Request a resource.
 2. Use the resource.
 3. Release the resource.

Resource types examples= CPU cycles, memory space, I/O devices..

Bridge Crossing Example



- Traffic is only in one direction where cars are driving in opposite direction.
- None of the cars can move once they are in front of each other.
- Each section of a bridge can be viewed as a resource.
- A situation in operating systems when two or more processes hold some resources and wait for resources held by other(s).
- If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback).
- Several cars may have to be backed up if a deadlock occurs.

DEADLOCK CHARACTERISATION

NECESSARY AND SUFFICIENT CONDITIONS

ALL of these four **must** happen simultaneously for a deadlock to occur:

Mutual exclusion

One or more than one resource must be held by a process in a non-sharable (exclusive) mode.

Hold and Wait

A process holds a resource while waiting for another resource.

No Preemption

There is only voluntary release of a resource - nobody else can make a process give up a resource.

Circular Wait

Process A waits for Process B waits for Process C waits for Process A.

RESOURCE ALLOCATION GRAPH

A visual (mathematical) way to determine if a deadlock has, or may occur.

G = (V, E) The graph contains nodes and edges.

V Nodes consist of processes = { P1, P2, P3, ... } and resource types
 { R1, R2, ... }

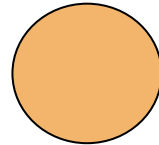
E Edges are (P_i, R_j) or (R_i, P_j)

An arrow from the **process** to **resource** indicates the process is **requesting** the resource.

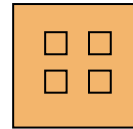
An arrow from **resource** to **process** shows an instance of the resource has been **allocated** to the process.

NOTATIONS

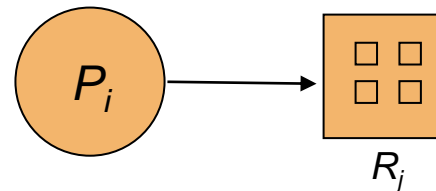
- Process



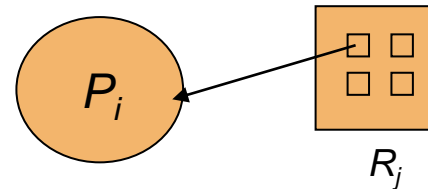
- Resource Type with 4 instances



- P_i requests an instance of R_j



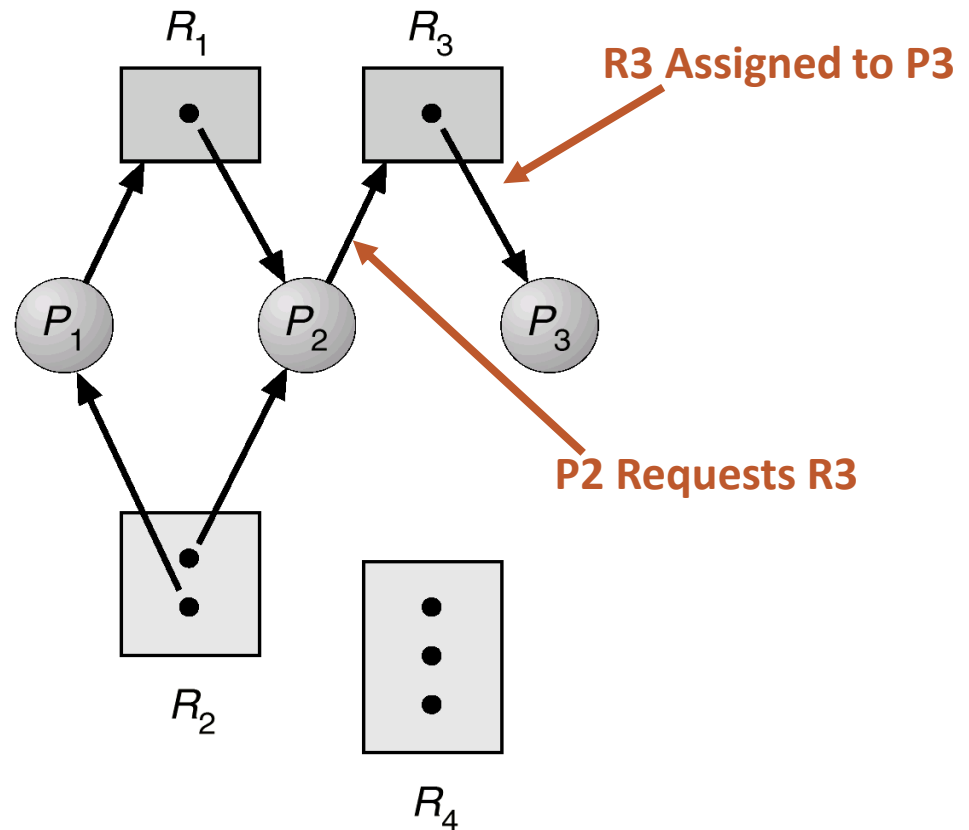
- P_i is holding an instance of R_j



RESOURCE ALLOCATION GRAPH

An illustration to show a resource allocation graph.

- P1 holds an instance of resource R2 and is requesting resource R1
- P2 holds R1 and an instance of R2, and requests R3
- P3 holds R3



SOME FACTS

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

RESOURCE ALLOCATION GRAPH WITH DEADLOCK

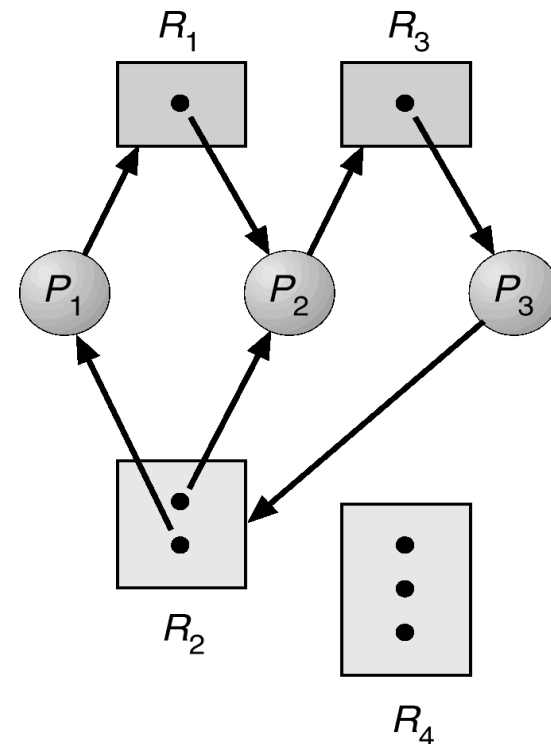
– Deadlock occurs!

- P3 requests R2, which is held by P2, which requests R3, which is held by P3 - this is a loop – $P_3 \rightarrow R_2 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3$

– If P1 could somehow release an instance of R2, then we could break the deadlock.

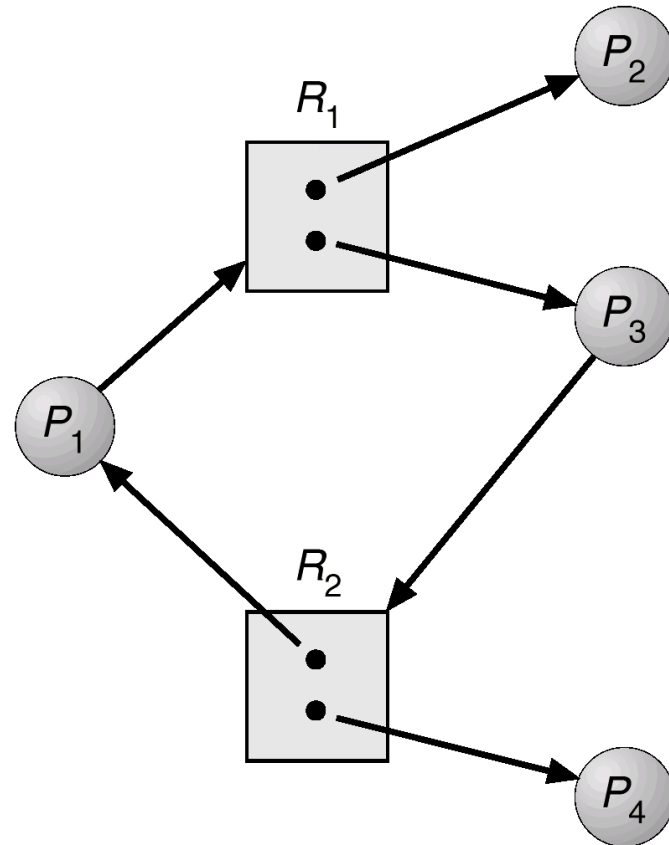
- But P1 is part of a second loop:

– $P_3 \rightarrow R_2 \rightarrow P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3$
– So P1 can't release its instance of R2



RESOURCE ALLOCATION GRAPH WITH CYCLE BUT NO DEADLOCK

- There is a loop:
 - $P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$
- In this case, there is no deadlock
 - Either P_2 can release an instance of R_1 or P_4 can release an instance of R_2
- This breaks any possible deadlock cycle



HOW TO HANDLE DEADLOCKS : GENERAL STRATEGIES

There are three methods:

1. Ignore Deadlocks. (Used by most Operating Systems)

2. Ensure deadlock **never** occurs using either:

Prevention Prevent any one of the 4 conditions from happening.

Avoidance Allow all deadlock conditions but calculate cycles about to happen and stop dangerous operations.

3. Allow deadlock to **happen**. This requires using both:

Detection Know a deadlock has occurred.

Recovery Regain the resources.

DEADLOCK IGNORANCE

- Stick your head in the sand and pretend there is no problem at all, this method of solving any problem is called Ostrich Algorithm.
- The Ostrich algorithm is the most widely used technique in order to ignore the deadlock.
- If there is a deadlock in the system, then the OS will reboot the system in order to function well.
- The performance of the system decreases if it uses a deadlock handling mechanism all the time.
- **Scientists** all over the world believe that the most efficient method to deal with deadlock is deadlock prevention. But the **Engineers** that deal with the system believe that deadlock prevention should be paid less attention as there are very less chances for deadlock occurrence.

DEADLOCK PREVENTION

Do not allow one of the four conditions to occur.

Mutual exclusion:

- a) Mutual exclusion means that multiple processes can't use the same resource at the same time.
- b) There will be no process waiting for the resource if we may use the same resource for several processes at the same time.
- c) Automatically holds for printers and other non-sharable.
- d) Shared entities (read-only files) don't need mutual exclusion (and aren't susceptible to deadlock).
- e) Prevention not possible, since some devices are intrinsically non-sharable.

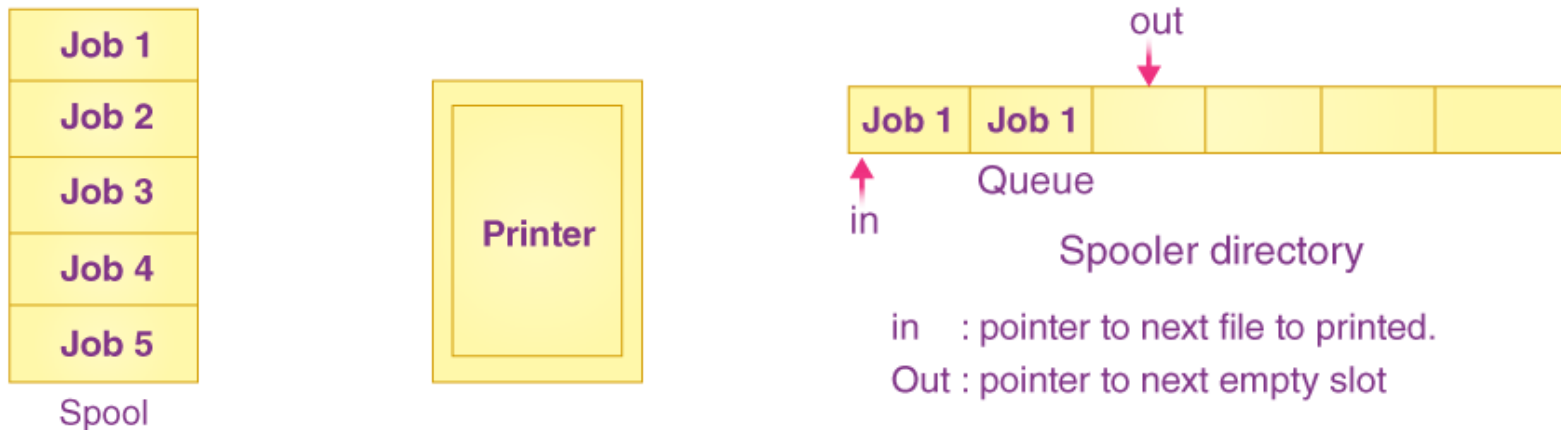
SPOOLING

- Simultaneous Peripheral Operations Online is what Spooling stands for. Spooling is a process in which data is temporarily saved for use and execution by a device, program, or system.
- Data is sent and stored in memory or other volatile storage until a program or computer requests execution.
- The spools are processed in ascending order and work based on a FIFO (First In First Out) algorithm.
- Spooling is used for devices like the printer, mouse, and keyboard. Using spooling, we can prevent mutual exclusion.

DEADLOCK PREVENTION

Let's look at how a printer's spooling procedure works:

- Memory is connected to a printer.
- We can keep all the jobs stored in all processes with the assistance of this memory.
- The printer then collects all of the jobs and prints each one in an FCFS (First Come, First Serve) manner. No process will have to wait for the printer if we follow this technique.
- Finally, the printer collects the outputs when they have been created.

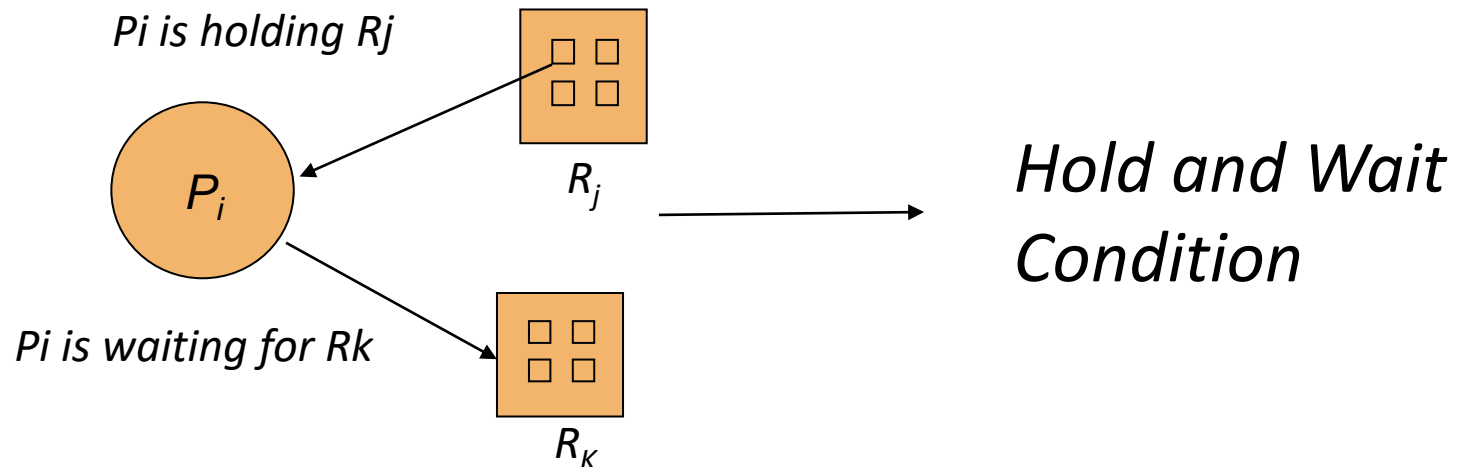


BUT Spooling cannot be used for all resources. As a result, we might conclude that a violation of mutual exclusion is nearly impossible.

DEADLOCK PREVENTION

Hold and wait:

- a) A hold and wait condition occurs when a process holds a resource while waiting for other resources to complete its task.
- b) We'll need to come up with a procedure that doesn't hold any resources or wait for any resources.
- c) This means that before beginning the process, we should assign all of the resources that it will require.
- d) The process is then started without having to wait for any resources, which is highly impractical because a process can't determine necessary resources initially.



DEADLOCK PREVENTION

Do not allow one of the four conditions to occur.

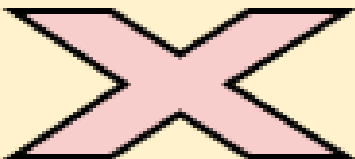
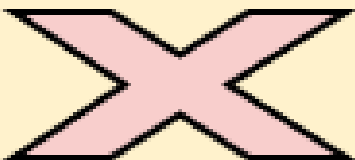
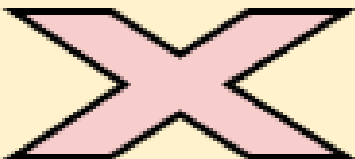
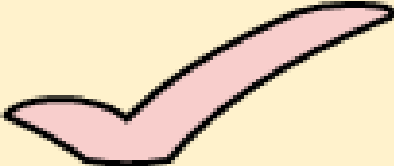
No preemption:

- a) Release any resource already being held if the process can't get an additional resource.
- b) Allow preemption - if a needed resource is held by another process, which is also waiting on some resource, steal it. Otherwise, wait.

Circular wait:

- a) Number resources and only requests in ascending order.
- b) EACH of these prevention techniques may cause a decrease in utilization and/or resources. For this reason, prevention isn't necessarily the best technique.
- c) Prevention is generally the easiest to implement.

DEADLOCK PREVENTION

Condition	Approach	Is Practically Possible?
Mutual Exclusion	Spooling	
Hold and Wait	Request for all the resources initially	
No Preemption	Snatch all the resources	
Circular Wait	Assign priority to each resources and order resources numerically	

SUMMARY

- A deadlock in OS is a situation in which more than one process is blocked because it is holding a resource and also requires some resource that is acquired by some other process.
- The four necessary conditions for a deadlock situation are mutual exclusion, no preemption, hold and wait and circular set.
- There are methods of handling deadlocks - deadlock ignorance, deadlock avoidance, deadlock prevention, deadline detection and recovery.
- We can prevent a deadlock by preventing any one of the four necessary conditions for a deadlock.
