JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

Lecture-21 By Vivek Dubey

COMPUTER SCIENCE AND ENGINEERING

CS403PC

COMPUTER SCIENCE AND ENGINEERING (ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING)

CS403PC

BHARAT INSTITUTE OF ENGINEERING AND TECHNOLOGY

Scheme of Operating System

S. No.	Course Code	Course Title	L	T	P	Credits
3	CS403PC	Operating Systems	3	0	0	3
6	CS406PC	Operating Systems Lab	0	0	3	1.5

Lecture-21 By Vivek Dubey

UNIT-3:

a) **Deadlocks** - System Model, Deadlocks Characterization, Methods for Handling Deadlocks, Deadlock Prevention, Deadlock Avoidance, Deadlock Detection, and Recovery from Deadlock

b) **Process Management and Synchronization** - The Critical Section Problem, Synchronization Hardware, Semaphores, and Classical Problems of Synchronization, Critical Regions, Monitors

c) **Interprocess Communication Mechanisms:** IPC between processes on a single computer system, IPC between processes on different systems, using pipes, FIFOs, message queues, shared memory.

Lecture-21 By Vivek Dubey

UNIT-3a:

- **Deadlocks** –
  - System Model,
  - Deadlocks Characterization,
  - Methods for Handling Deadlocks,
  - Deadlock Prevention,
  - Deadlock Avoidance,
  - Deadlock Detection, and
  - Recovery from Deadlock

Lecture-21 By Vivek Dubey

System Model

- System consists of resources
  - *A system consists of a finite number of resources to be distributed among a number of competing processes.*
- Resource types $R_1, R_2, \ldots, R_m$ - *CPU cycles, memory space, I/O devices*
  - *The resources may be partitioned into several types (or classes), each consisting of some number of identical instances. CPU cycles, files, and I/O devices (such as printers and DVD drives) are examples of resource types. If a system has two CPUs, then the resource type CPU has two instances. Similarly, the resource type printer may have five instances.*
- Each resource type R_i has W_i instances.

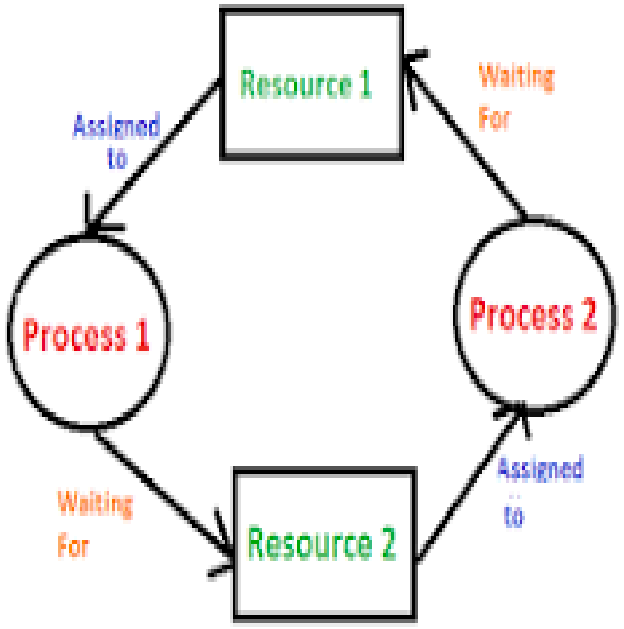
Contd..

- Under the normal mode of operation, a process may utilize a resource in only the following sequence:
- **1. Request**. The process requests the resource. If the request cannot be granted immediately (for example, if the resource is being used by another process), then the requesting process must wait until it can acquire the resource.
- **2. Use**. The process can operate on the resource (for example, if the resource is a printer, the process can print on the printer).
- **3. Release**. The process releases the resource. The request() & release() device, open() & close() file, allocate() & free() memory are system calls.

Define Deadlock in OS

- Deadlock is a situation where a group of processes are permanently blocked waiting for the resources held by each other in a group. For Example:

Consider a situation where, a process - P1, request for a resource. If the requested resource is not available at that time, the process enters a waiting state. If the requested resource is held by other waiting process- P2, then P1 which is waiting can never change from its waiting state.



- In a deadlock, processes never finish executing, and system resources are tied up, preventing other jobs from starting.

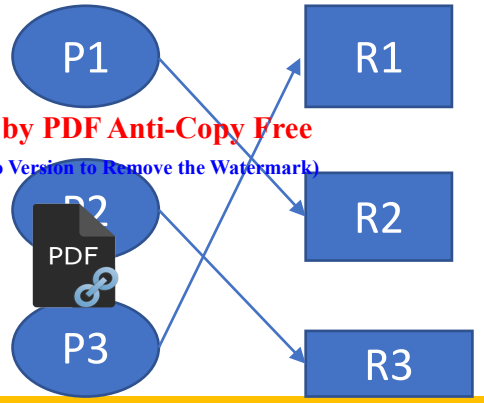
Deadlocks Characterization

What are the four conditions of deadlock?

Deadlocks are normally because of the following **four conditions held simultaneously** in a system:

1. Mutual Exclusion
2. Hold and Wait
3. No Pre-emption
4. Circular Wait

Mutual Exclusion



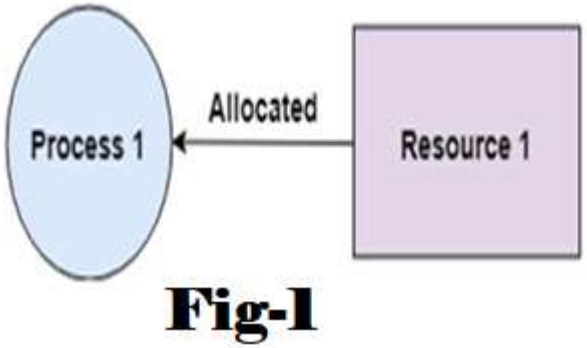


Fig-1

- Only one process ***can use*** a resource at any given time i.e. the Ri are ***non-sharable and One-to-One relationship*** with Pi.
- In the Fig-1, *there is a single instance of Resource-1 and it is held/allocated by Process-1 only.*

Hold and Wait

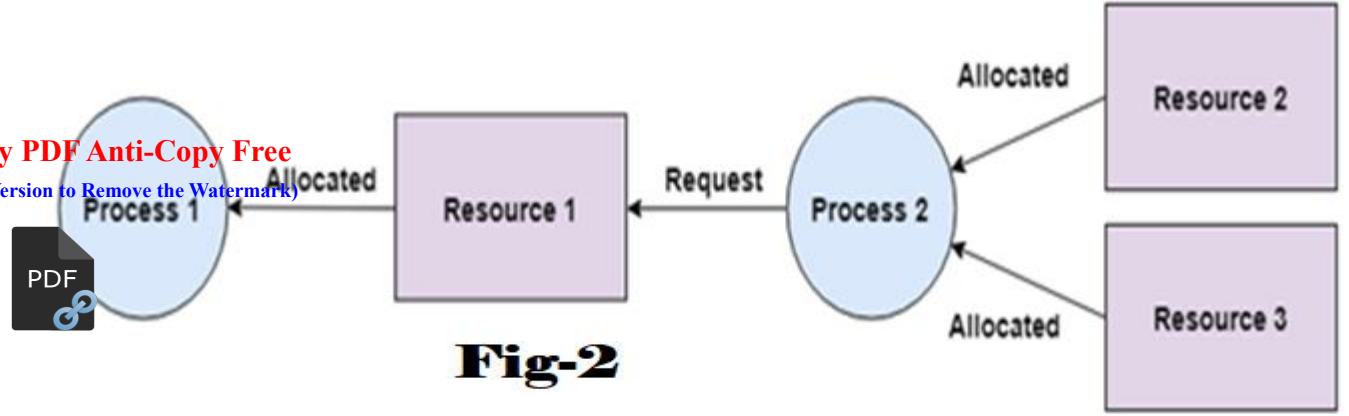


Fig-2

- A process (P2) can hold multiple resources (R1 and R2) and still request more resources ***from other processes*** -> which are holding them.
- In the Fig-2, *Process 2 holds Resource 2 and Resource 3 and is requesting the Resource 1 which is held by Process 1.*

No-Pre-emption

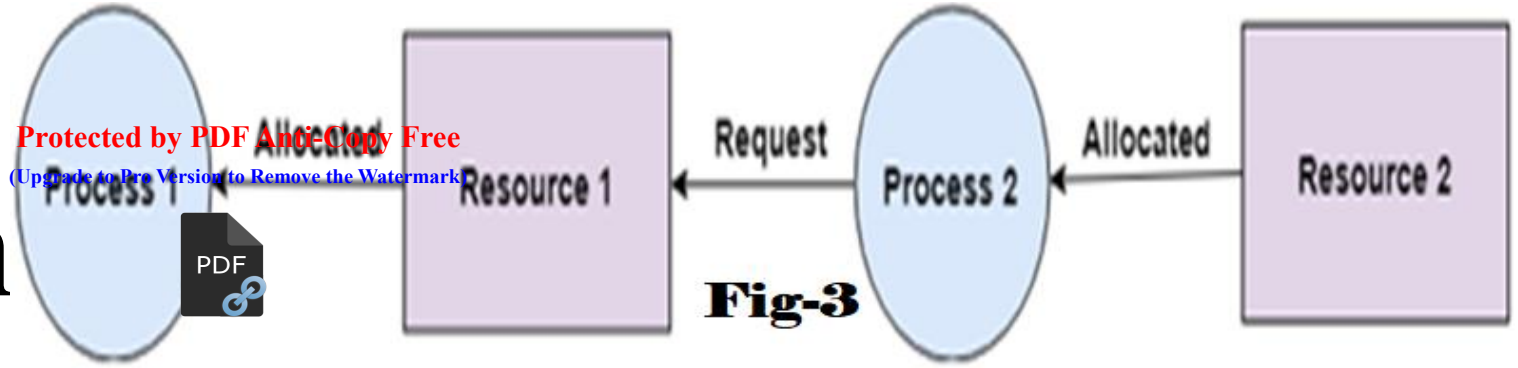


Fig-3

- A resource cannot be preempted *from a process by force.*
- A process can only release *a resource voluntarily*.
- In the Fig-3, Process-2 cannot preempt Resource-1 from Process-1.
- It will only be released when Process-1 give up it voluntarily after its execution is complete.

Circular Wait

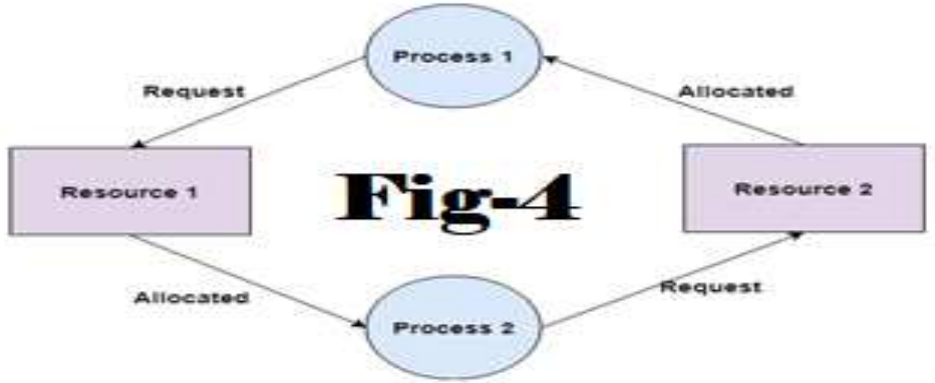


Fig-4

- A set pf processes ***are waiting*** each other in a circular.
- For example: Process 1 is allocated Resource2 and it is requesting Resource 1. Similarly, Process 2 is allocated Resource 1 and it is requesting Resource 2. This forms a circular wait loop.

Methods for Handling Deadlocks

Handling deadlock problem using one of the following methods:

- Using a protocol ***to prevent or avoid*** deadlocks, ***ensuring*** that the system ***will never*** enter a deadlocked state.
- Allowing the system ***to enter*** a deadlocked state, ***detect*** it, and ***recover*** it.
- ***Ignoring*** the deadlock problem completely and ***pretending*** as if deadlocks ***had never occurred*** in the system. It is then up to the application developer ***to write*** programs that handle deadlocks.

Contd..

- To ensure that deadlocks never occur, ***the system can use either***
  1. a deadlock prevention or
  2. a deadlock-avoidance scheme.
- **Deadlock prevention** ***provides*** a set of methods ***to ensure*** that at least one of the necessary conditions cannot hold. These methods ***prevent*** deadlocks ***by constraining how requests*** for resources can be made.
- **Deadlock avoidance** ***requires*** that the operating system ***be given*** additional information in advance concerning which resources a process will request ***and use*** during its lifetime.

Deadlock Prevention

- Following methods employed to prevent deadlock:
  1. Mutual Exclusion
  2. Hold and Wait
  3. No Pre-emption
  4. Circular Wait

Mutual Exclusion:

- The mutual exclusion condition ***must hold*** indicating that at least one resource is non-sharable.
- A process ***never needs to wait*** for a sharable resource.
- Sharable resource, ***do not require*** mutually exclusive access and thus ***cannot be involved*** in a deadlock.
- Read-only files ***are*** sharable resources ***which attempted*** by many processes ***to open*** a readonly file simultaneously, they ***can be granted*** simultaneous access to the file.
- A mutex lock ***is*** a non-sharable resource that ***cannot be*** simultaneously shared by several processes.

Hold and Wait

- ***In order to prevent*** from the occurrence of the hold-and-wait condition in a system, ***it must be ensured*** that, **a process** ***requesting*** for a resource, ***must not hold*** any other resources.
- ***A process may request*** some resources and ***use*** them.
- ***Before it can request*** any additional resources, ***it must release*** all the resources ***that it is*** currently allocated.
- ***This can be implemented*** by two methods:
  - ***Ensuring that system calls requesting resources for a process*** precede all other system calls.
  - ***Allowing a process to request*** resources only when it has none.
- **Disadvantages:**
  - Resource utilization ***may be low.***
  - Starvation ***is possible making at least one*** of the resources that a process ***needs is*** always allocated to some other process.

No Pre-emption

A necessary condition for deadlocks **is that there be** *no preemption of resources that* **have already been allocated**. *To ensure that* **this condition does not hold**, *the following methods can be adopted.*

Method 1:

If a process ***is holding some resources and requests*** another resource that ***cannot be immediately allocated to it,*** then all resources the process ***is currently holding are implicitly released***.

These preempted resources ***are added to the list of resource***s for which the process ***is waiting***. The process ***will be restarted*** only if it can regain its old resources, as well as the new ones ***that it has requested.***

Method 2:

If a process ***requests some resources***, check for its availability and allocate it if available. If not, ***check if it is allocated*** to some other process that is waiting for additional resources.

- If so, ***preempt the desired resources*** from the waiting process and allocate them to the requesting process.
- If the resources ***are neither available nor held*** by a waiting process, the requesting process must wait.
- A process can be restarted only ***when it is allocated*** the requested new resources and ***recovers any resources*** that were preempted while it was waiting.

Circular Wait

- ***To ensure*** that circular wait condition never holds, ***enforce*** a total ordering of all resource types ***and require*** that each process ***requests*** resources in an increasing order of enumeration.
- ***Ensuring*** that resources ***are acquired*** in the proper order is the responsibility of application developers.
- Software can also be used ***to verify*** that locks are acquired in the proper order ***and to give*** appropriate warnings when locks are acquired out of order.

Deadlock Avoidance

- In order to avoid deadlock the system ***must consider the following***:
  - the resources currently available,
  - the resources currently allocated to each process,
  - the future ***requests and releases*** of each process.
- System models ***require a priori information*** about the number of resources it may need.
- With this prior information, ***it is possible to construct*** an algorithm that ***ensures that the system will never*** enter a deadlocked state.

Contd..

- The resource allocation state ***is defined*** by the number of available and ***allocated*** resources and the maximum demands of the processes.
- A state space ***can be in one of the following states***:
  - Deadlocked State
  - Safe State
  - Unsafe State
- A state is *safe* if the system ***can allocate resources to each process*** in some order and still avoid a deadlock.
- A deadlock-avoidance algorithm dynamically ***examines*** the resource allocation state ***to avoid deadlock***.
- **Two important deadlock avoidance algorithms** *are:*
  - Resource allocation Graph algorithm
  - Bankers Algorithm

RECOVERY FROM DEADLOCK

The two methods employed for recovering from deadlocked state are,

1. Process Termination
2. Resource Pre-emption

Process termination: *This techniques* **adopts** *one of following the strategies*:

1. ***Abort all deadlocked processes***: **Will break** the deadlock cycle, with high computational overhead. The processes that **are computed** for a long time, as well as the results of these partial computations **are discarded** which **needs** recomputation at a later stage.
2. ***Abort one process at a time until the deadlock cycle is eliminated:*** After aborting each process, the deadlock-detection algorithm **is invoked to find out** if there are other processes in deadlock state.

Contd..

The factors that affect in choosing the process for aborting includes:

- the number of processes **to be terminated**
- the type of the process (interactive / batch) **priority of the process**
- the time needed to complete the chosen task for the types of resources **used by the process**
- the required resources for the process **to complete it**

Resource Preemption:

This method **adopts** the following strategies:

a. ***Selecting a victim:*** the process **must be chosen considering parameters** *like:*

- the number of resources a deadlocked process is holding and
- the amount of time the process has consumed so far.

b. ***Rollback:*** **To recover the process** with pre-empted resources, **roll back** the process **to some safe state** and **restart it** from that state. Rolling back the process only as far as necessary requires the system **to keep** information about the state of all running processes. Therefore, a total rollback **will be best method** that **will abort** the process and then **restart** it.

c. ***Starvation:*** Resources **should not always be preempted** from the same process.

If done so, that process **will never complete** its task, **leading to a starvation situation**,

To avoid such a situation, ***it must be ensured*** that a process **can be picked** as a victim few times only.

Note: The number of rollbacks **is** a solution.