



RAJASTHAN TECHNICAL UNIVERSITY, KOTA

Syllabus

III Year-V Semester: B.Tech. Computer Science and Engineering

5CS4-03: Operating System

Credit: 3

3L+OT+OP

Max. Marks: 150(IA:30, ETE:120)

End Term Exam: 3 Hours

SN	Contents	Hours
1	Introduction: Objective, scope and outcome of the course.	01
2	Introduction and History of Operating systems: Structure and operations; processes and files Processor management: inter process communication, mutual exclusion, semaphores, wait and signal procedures, process scheduling and algorithms, critical sections, threads, multithreading	04
3	Memory management: contiguous memory allocation, virtual memory, paging, page table structure, demand paging, page replacement policies, thrashing, segmentation, case study	05
4	Deadlock: Shared resources, resource allocation and scheduling, resource graph models, deadlock detection, deadlock avoidance, deadlock prevention algorithms Device management: devices and their characteristics, device drivers, device handling, disk scheduling algorithms and policies	15
5	File management: file concept, types and structures, directory structure, cases studies, access methods and matrices, file security, user authentication	07
6	UNIX and Linux operating systems as case studies; Time OS and case studies of Mobile OS	08
	Total	40



Swami Keshvanand Institute of Technology, Management & Gramothan, Jaipur

Programme: B.Tech. (Computer Science & Engineering)

Semester: VI

Course Name (Course Code): DISTRIBUTED SYSTEM (6CSS5-11)

Course Outcomes

After completion of this course, students will be able to –

6CSS5-11.1	List basic components of distributed system and describe various issues of DS.
6CSS5-11.2	Illustrate and analyze process management, Inter-process Communication techniques.
6CSS5-11.3	Compare Distributed File Systems, and Distributed Process Scheduling with Transaction Services and Concurrency Control.
6CSS5-11.4	Analyze Distributed Shared Memory with memory consistency models and models of distributed computation.
6CSS5-11.5	Demonstrate different algorithms and techniques for Distributed Agreement, Replicated Data Management.

Ms. Priyanka Sharma

Mr. Mayank Jain

(Signatures)

Verified by Course Coordinator

Signature
(Name: Ms. Deepa Modi)

Verified by Verification and Validation Committee, DPAQIC

Signature
(Name: Dr. Milind R. Shankar)



Swami Keshvanand Institute of Technology, Management & Gramothan, Jaipur



COURSE: Advanced Engineering Mathematics (3CS6A)

CO-PO/PSO Mapping: Formulation and Justification

The CO-PO/PSO mapping is based on the correlation of course outcome (CO) with Program Outcome Indicators. These indicators are the breakup statements of broad Program Outcome statement.

The correlation is calculated as number of correlated indicators of a PO/PSO mapped with CO divided by total indicators of a PO/PSO. The calculated value represents the correlation level between a CO & PO/PSO. Detailed formulation and mathematical representation can be seen below in equation 1:

Input: CO_i : The i^{th} course outcome of the course

PO_j : The j^{th} Program Outcome

I_{jk} : The k^{th} indicator of the j^{th} Program Outcome

$\alpha(I_{jk}, CO_i)$: level of CO-PO mapping

$$= 1, \text{ if } 0 < \alpha < 0.33$$

$$= 2, \text{ if } 0.33 \geq \alpha < 0.66$$

$$= 3, \text{ if } 0.66 \geq \alpha < 1$$

$$\alpha(I_{jk}, CO_i) = \frac{\text{count}(\lambda(I_{jk}, CO_i))}{\text{count}(I_k, PO_j)}$$

λ : Degree of correlation



**Swami Keshvanand Institute of Technology,
Management & Gramothan, Jaipur**

COURSE: Distributed System (6CSS5-11)

CO	Outcomes	Bloom's Level	PO Indicators	PSO Indicators
Upon successful completion of this course, students should be able to:				
6CSS5-11.1	List basic components of distributed system and describe various issues of DS.	1,2	1.3.1, 14.1, 2.1.1, 2.1.2, 2.4.3, 4.1.1, 4.1.2, 4.3.3, 12.1.1, 12.1.2, 12.2.1, 12.2.2	1.12, 3.1.1
6CSS5-11.2	Illustrate and analyze process management, Inter-process Communication techniques.	4	1.3.1, 14.1, 1.5, 2.1.1, 2.1.2, 2.2.2, 2.2.3, 2.2.4, 2.4.3, 4.1.1, 4.1.2, 4.3.3, 4.3.4, 12.1.1, 12.1.2, 12.2.1, 12.2.2	1.12, 3.1.1
6CSS5-11.3	Compare Distributed File Systems, and Distributed Process Scheduling with Transaction Services and Concurrency Control.	5	1.3.1, 14.1, 2.1.1, 2.1.2, 2.1.3, 2.2.2, 2.2.3, 2.2.4, 2.4.1, 2.4.2, 2.4.4, 4.1.1, 4.1.2, 4.3.2, 4.3.3, 4.3.4, 12.1.1, 12.1.2, 12.2.1, 12.2.2	1.12, 3.1.1
6CSS5-11.4	Analyze Distributed Shared Memory with memory consistency models and models of distributed computation.	4	1.3.1, 14.1, 2.1.1, 2.1.2, 2.1.3, 2.2.2, 2.2.3, 2.2.4, 2.4.1, 2.4.3, 2.4.4, 4.1.1, 4.1.2, 4.3.2, 4.3.3, 4.3.4, 12.1.1, 12.1.2, 12.2.1, 12.2.2	1.12, 3.1.1
6CSS5-11.5	Demonstrate different algorithms and techniques for Distributed Agreement, Replicated Data Management.	3	1.3.1, 14.1, 2.1.1, 2.1.2, 2.1.3, 2.2.2, 2.2.3, 2.2.4, 2.4.1, 2.4.2, 2.4.4, 4.1.1, 4.1.2, 4.3.2, 4.3.3, 4.3.4, 12.1.1, 12.1.2, 12.2.1, 12.2.2	1.12, 3.1.1



**Swami Keshvanand Institute of Technology,
Management & Gramothan, Jaipur**

CO-PO/PSO Mapping

Programme: B.Tech. (Computer Science & Engineering)
Semester: VI

Course Name (Course Code): Distributed System (6CS5-11)

	P01	P02	P03	P04	P05	P06	P07	P08	P09	P010	P011	P012	PSO 1	PSO 2	PSO 3
C01	2	1		1									3	2	2
C02	2	2		1									3	2	2
C03	2	3		2									3	2	2
C04	2	3		2									3	2	2
C05	2	3		2									3	2	2

Ms. Priyanka Sharma
(Signature)

Verified by Course Coordinator

Signature
(Name: Deepsa Modi)

Scanned with CamScanner

Mr. Mayank Jain
(Signature)

Verified by Verification and Validation Committee, DPAQIC

Signature
(Name: Syed Ali Khanfai)

COURSE NOTES

UNIT-I



Swami Keshvanand Institute of Technology, Management & Gramothan,
Rammagaria, Jagatpura, Jaipur-302017, INDIA

Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956

Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Operating System (UNIT-I)

An operating system (OS) is the software component of a computer system that is responsible for the management and coordination of activities and the sharing of the resources of the computer. An operating system acts as an intermediary between the user of a computer and computer hardware. The purpose of an operating system is to provide an environment in which a user can execute programs in a convenient and efficient manner.

An operating system is software that manages the computer hardware. The hardware must provide appropriate mechanisms to ensure the correct operation of the computer system and to prevent user programs from interfering with the proper operation of the system.

An operating system is a program that controls the execution of application programs and acts as an interface between the user of a computer and the computer hardware.

Job/functions of Operating System-

An operating system has three main functions:

- (1) manage the computer's resources, such as the central processing unit, memory, disk drives, and printers,
- (2) establish a user interface, and
- (3) execute and provide services for applications software.

Types of Operating System –

- **Batch Operating System**- Sequence of jobs in a program on a computer without manual interventions.
- **Time sharing operating System**- allows many users to share the computer resources.(Max utilization of the resources).
- **Distributed operating System**- Manages a group of different computers and make appear to be a single computer.
- **Network operating system**- computers running in different operating system can participate in common network (It is used for security purpose).
- **Real time operating system** – meant applications to fix the deadlines.

Examples of Operating System are –

- Windows (GUI based, PC)
- GNU/Linux (Personal, Workstations, ISP, File and print server, Three-tier client/Server)
- macOS (Macintosh), used for Apple's personal computers and work stations (MacBook, iMac).
- Android (Google's Operating System for smartphones/tablets/smartwatches) – TV's

(GNU's not Unix)

↳ Cupcake, Donut, Eclair, Froyo, Gingerbread,
Honey Comb, Ice cream sandwich, Jelly Bean 9.0
Kit-Kat, Lollipop, marshmallow, nougat, oreo, pie
Android 10, Android 11



Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagar, Jagatpura, Jaipur-302017, INDIA
 Approved by AICTE, Ministry of HRD, Government of India
 Recognized by UGC under Section 2(f) of the UGC Act, 1956
 Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
 E-mail: info@skit.ac.in Web: www.skit.ac.in

- iOS (Apple's OS for iPhone, iPad and iPod Touch)

- (mac OS High Sierra)

mac OS Sierra

→ Kodiak
→ Cheetah

→ Puma

→ Jaguar
→ Panther

→ Tiger

→ Leo
→ Pards

Some Operating Systems -

The Big 3

Common contemporary OS's include Microsoft Windows, Mac OS X, and Linux. Microsoft Windows has a significant majority of market share in the desktop and notebook computer markets, while the server and embedded device markets are split amongst several OSs.

Linux

Linux (also known as GNU/Linux) is one of the most prominent examples of free software and open source development which means that typically all underlying source code can be freely modified, used, and redistributed by anyone. The name "Linux" comes from the Linux kernel, started in 1991 by Linus Torvalds.

Microsoft Windows

Windows (created by Microsoft) is the most dominant OS on the market today. The two most popular versions of Windows for the desktop are XP and Vista (Vista being the latest version). There is also a mobile version of Windows as well as a server version of Windows (the latest being Windows Server 2008).

Windows is all proprietary, closed-source which is much different than Linux licenses. Most of the popular manufacturers make all of their hardware compatible with Windows which makes Windows operate and almost all kinds of new hardware.

XP

The term "XP" stands for experience. Windows XP is the successor to both Windows 2000 Professional and Windows ME. Within XP there are 2 main editions: Home and Professional. The Professional version has additional features and is targeted at power users and business clients. There is also a Media Center version that has additional multimedia features enhancing the ability to record and watch TV shows, view DVD movies, and listen to music.

Windows XP features a task-based GUI. XP analyzes the performance impact of visual effects and uses this to determine whether to enable them, so as to prevent the new functionality from consuming excessive additional processing overhead. The different themes are controlled by the user changing their preferences.

Windows XP has released a set of service packs (currently there are 3) which fix problems and add features. Each service pack is a superset of all previous service packs and patches so that only the latest service pack needs to be installed, and also includes new revisions. Support for Windows XP Service Pack 2 will end on July 13, 2010 (6 years after its general availability).



Vista

Windows Vista contains many changes and new features from XP, including an update GUI and visual style, improved searching features, new multimedia creation tools, and redesigned networking, audio, print, and display subsystems. Vista also aims to increase the level of communication between machines on a home network, using peer-to-peer technology to simplify sharing files and digital media between computers and devices.

Apple MAC-OS X

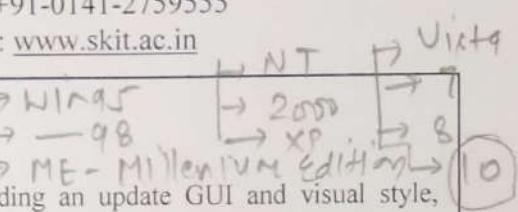
OS X is the major operating system that is created by Apple Inc. Unlike its predecessor (referred to Classic or OS 9), OS X is a UNIX based operating system.

Apple iOS

Apple's iOS is one of the most popular smartphone operating systems, second only to Android. It runs on Apple hardware, including iPhones, iPad tablets and iPod Touch media players.

Google's Android OS

Android is the most popular operating system in the world judging by the number of devices installed. Largely developed by Google, it's chiefly used on smartphones and tablets. Unlike iOS, it can be used on devices made by a variety of different manufacturers, and those makers can tweak parts of its interface to suit their own needs.





History of OS :-

(1) First Generation (1945-55) -

- * Vacuum Tubes :- Electronic tube to control current flow b/w electrodes
- Vacuum tube OS, uses Vacuum tubes logic.
- * Plugboards :- Control panels / array of 800 jacks/ sockets, into patch cords are inserted to complete an electric ckt.

(2) Second Generation (1955-65) - Transistors & Batch Systems

- * Computers become reliable
- * Machines used — mainframes
- * Located in air-conditioned rooms
- * Batch system reduced waste of time in computers.

(3) Third Generation (1965-1980) - IC's & multi-programming

- * Small-scale IC's provided price/performance advantage.
- * Multiprogramming made it possible that CPU need not be idle while a job is executing its I/O operations.



अरतो जा सद्गमय

Swami Keshvanand Institute of Technology, Management & Gramothan,

Ramnagar, Jagatpura, Jaipur-302017, INDIA

Approved by AICTE, Ministry of HRD, Government of India

Recognized by UGC under Section 2(f) of the UGC Act, 1956

Tel.: +91-0141- 5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in

- * Time - sharing was introduced
- * Computer can provide fast & interactive services to a no. of users
- * CTSS (Compatible Time Sharing System)

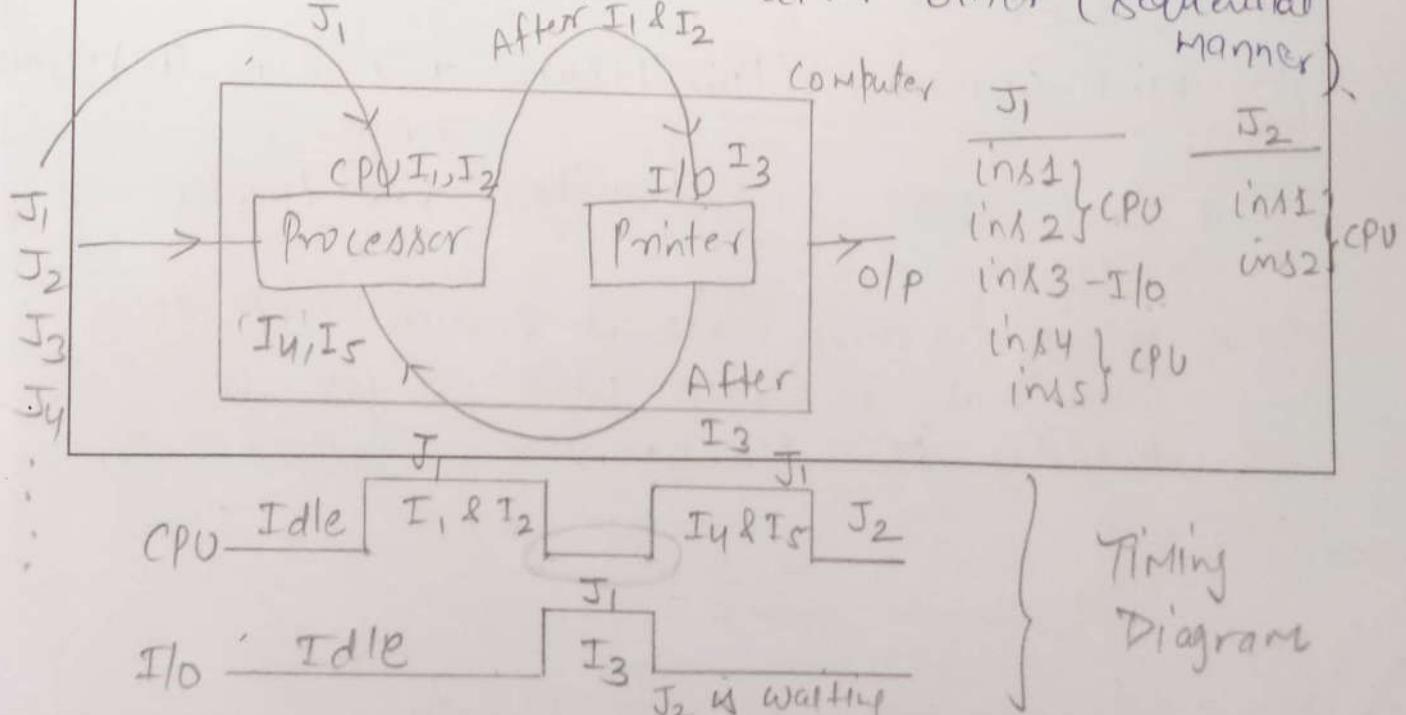
(A) Fourth Generation (1980 - Present) → Personal Computers

- * Disk D.s. (DOS)
- * MS - DOS
- * GUI - user friendly
- * Network OS & Distributed OS.

Types of OS :-

Batch Processing System :- (Sequential Processing)

- * Batch :- collection of jobs, processes will be executed one after other (sequential manner).



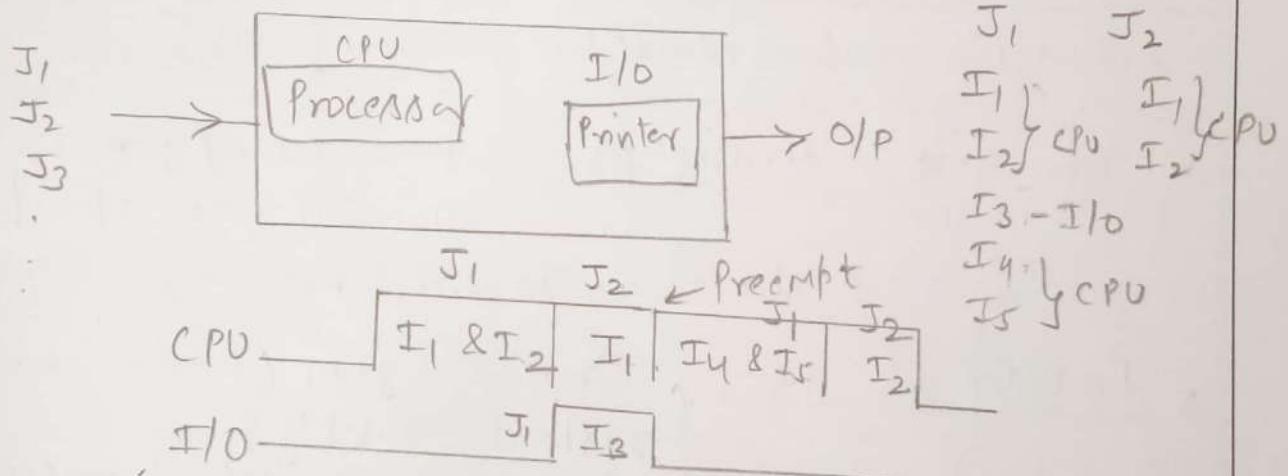


असतो मा सद्गमय

Multiprogramming / ~~multi-tasking~~ OS :- (Introducing Parallelism)

↳ maximum CPU utilization - objective

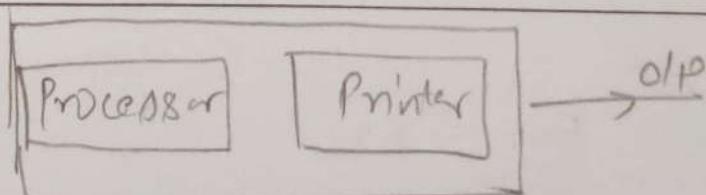
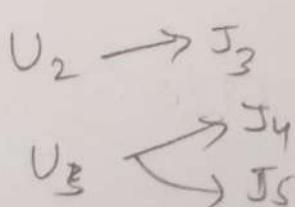
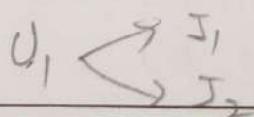
By organizing job in such a way that CPU always have one process to execute.



* When jobs want same resource, they run sequentially. When they want different resource, they run parallelly.

Problem Advantage :- Efficient Utilization of resources
→ High throughput

Disadvantage :- Poor Response time (as follows):





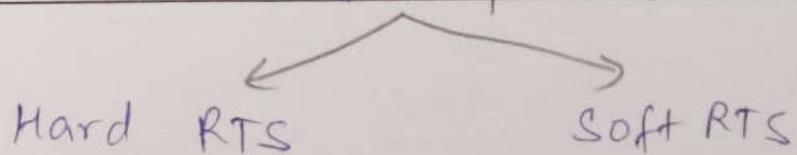
- * When users give job to computer, OS will select one of them.
 - * In above fig; If J_1 & J_2 of user 1 are lengthy, then U_2 & U_3 have to wait.
- Time-sharing & Multi-tasking OS :- Logical extension of multiprogramming OS

- * In time-sharing OS - each user/job is given a time span/ slot
eg: 1 slot \approx 50 seconds (Assume)
- * Problem in TS :- Critical jobs (which don't respond quickly).
- * Critical jobs want to be executed immediately but sometimes it may be in waiting state.

Real-time systems :- Handles 'critical Job Problem' in time sharing.

- * Most important jobs are responded quickly, on basis of:-

↳ Priority (How important the process is)
↳ interrupt vector



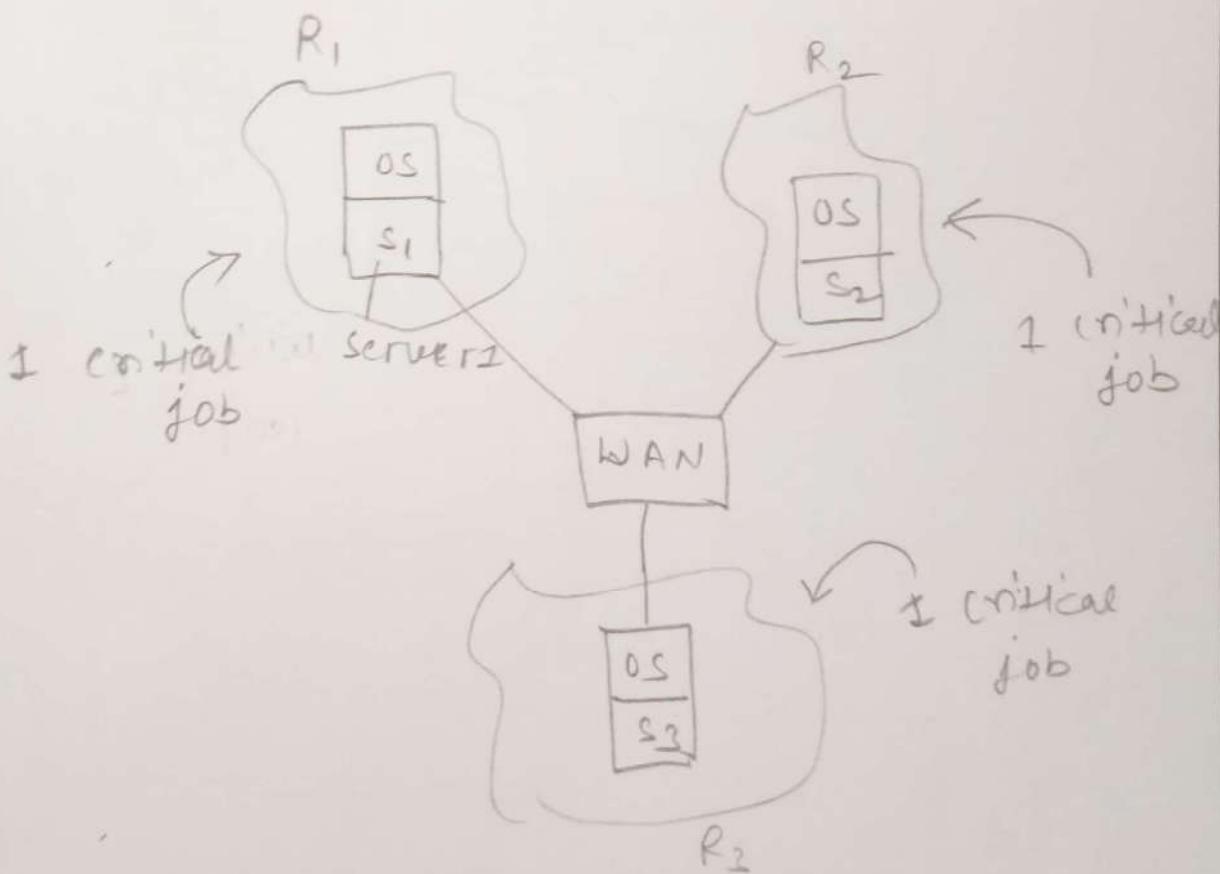
Disadvantage :-

Bottleneck Problem :-

Solution :- OS that deals with many critical jobs at a time → DOS & MOS

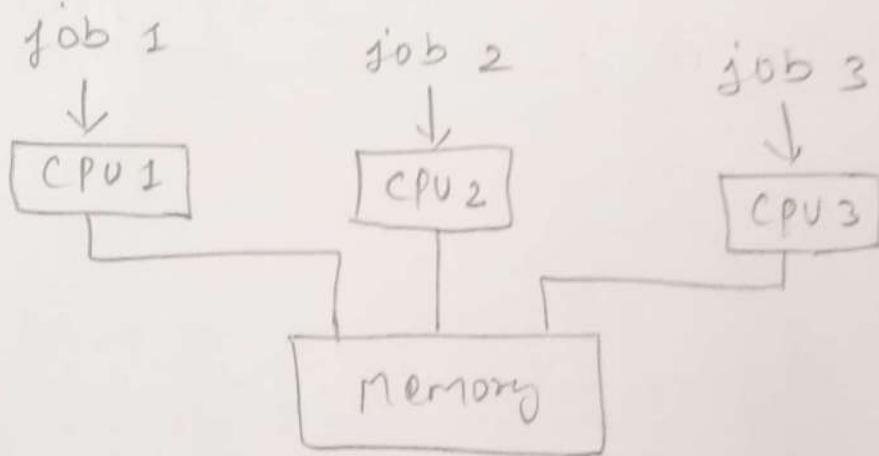
Distributed OS :- Resolves multiprocessor bottleneck problems

If many critical jobs are there it can solve only one at a time, rest must be waiting.



Multi-processing system :- [OR, Parallel or

- * Multi-processor (CPU) Tight coupling
- * All work in parallel, so [systems]
- called parallel systems
- * More Reliability, i.e. if one system fails, its load will be shifted to another one.
- * Multiple points of control.



Operating System :- A program that :-

- ↳ Manages computer Hardware
- ↳ Acts as an intermedium b/w user & hardware.

What O.S. Does ?

* Computer System :- Divided into 4 parts:-

- ↳ 1) Hardware
- ↳ Software
 - ↳ 2) System S/W → O.S.
 - ↳ 3) Application S/W
- ↳ 4) Users

* Hardware :-

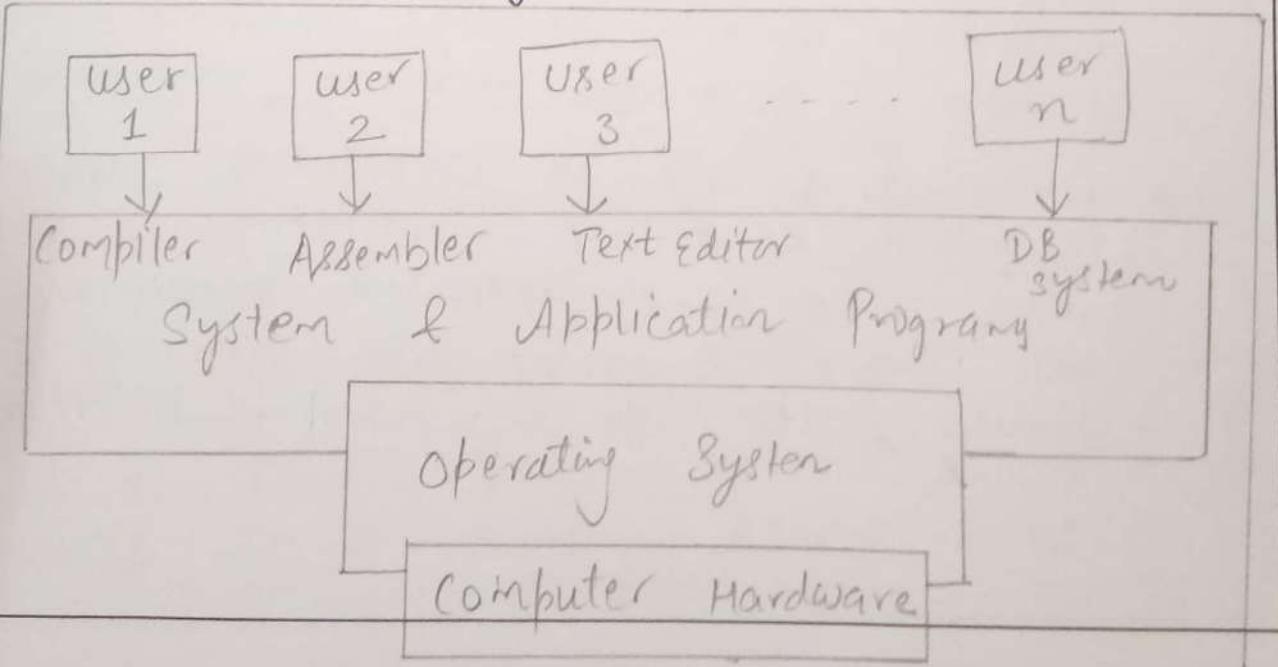
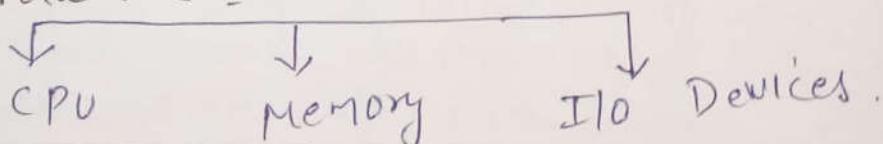


fig: "Abstract View of Computer System"
"Structure of OS"



Operating System Viewpoints :-

① User's view : - Varies according to the interface being used.

↳ When user sits in front of a PC:-
consisting of CPU, monitor, keyboard & mouse.
Such system is designed for one user,
where the goal is to maximize the work.
O.S. is designed here for ease of use only,
not for resource utilization.

↳ When user sits at a terminal connected
to a mainframe/ mini-computer : - Other
users are also accessing same computer
through other terminals. Here, users share
resources & may exchange information. Goal
is here : - maximize resource utilization.

↳ When users sits at workstations connected
to n/w's of other workstations & servers:
These users have dedicated resources, can
also share resources like networks &
servers. (also files, computers & print servers)



② System View:- from this view, O.S. is the program involved with H/W.

- ↳ As a 'Resource allocator' - Where computer system has many resources that may be required to solve a problem. O.S. acts as a manager of these resources. (CPU Time, memory space, file Space, I/O Devices etc.).
- ↳ As a 'Control Program' : - Emphasis is on the need to control various I/O devices & user programs. Control Program manages execution of user programs to prevent errors & improper use of computer.

Computer System Architecture :-

* General purpose Computer System consists of one or more CPU's & a no. of various device controllers connected through a common bus, that provides access to shared memory, as shown in figure.

Bootstrap Program:- When the system is powered up or rebooted, it needs to have an initial program to run called bootstrap program. It is stored in ROM or EEPROM (firmware). Bootstrap must know how to load O.S. & how to start executing that system.

- * For this, Bootstrap must locate OS Kernel & load it into memory. Once kernel is loaded & executing, it can start providing services to its users & system.
- * Some services are provided outside of the kernel, by the system program (which are loaded into memory at boot time) to become system processes or daemons that run for the entire time the kernel is running.

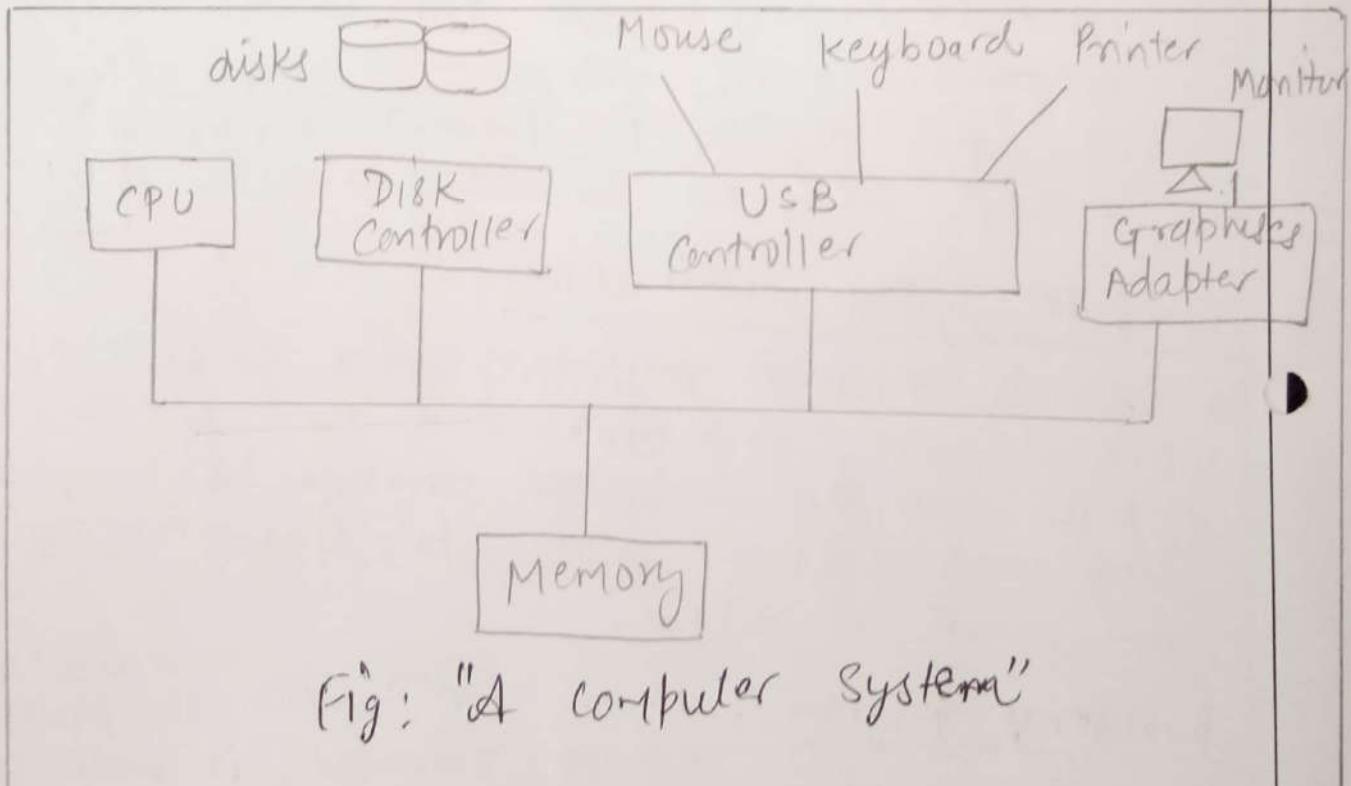


Fig: "A computer System"

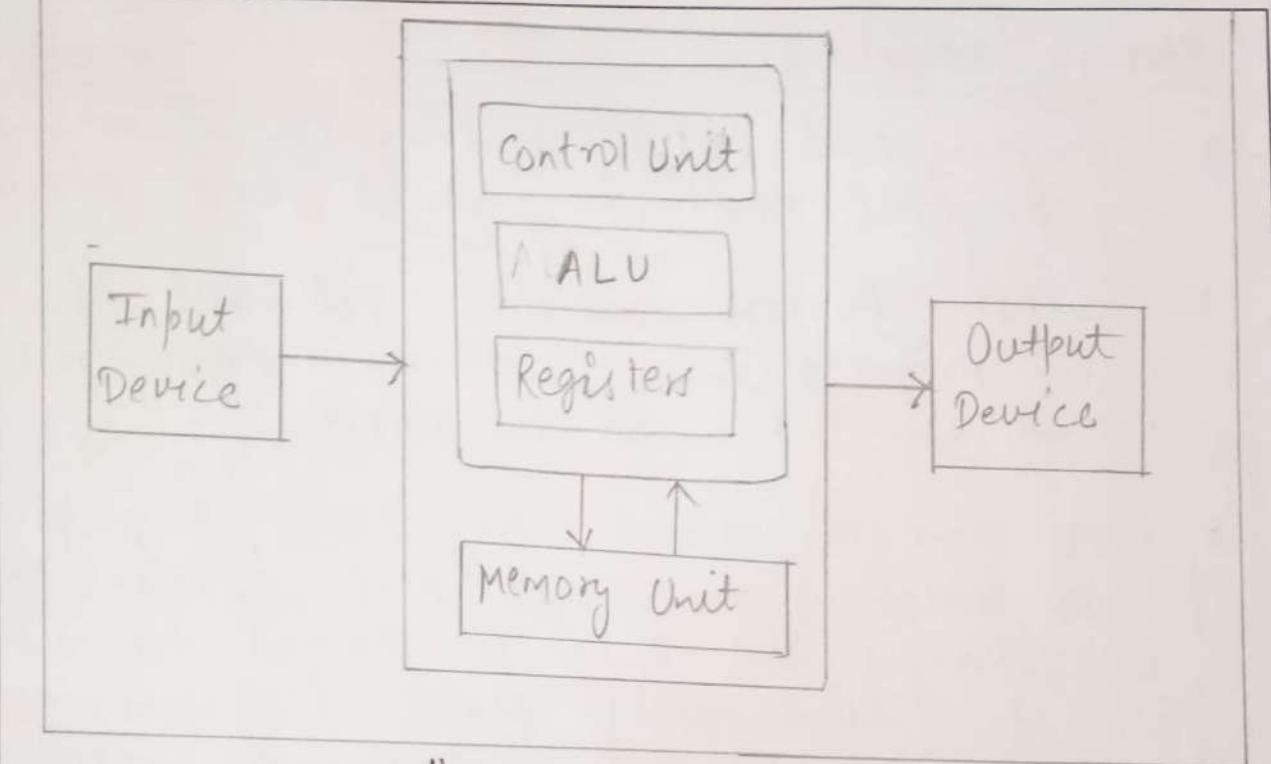


Fig: - "Von-Neumann Architecture (Computer System)"

Interrupts :- The occurrence of an event, from either hardware or software.

- **Hardware Interrupts** :- Triggered by sending a signal to CPU.
- **Software Interrupts** :- Triggered by executing a special operation (System call).



Storage structure:-

- * CPU can load instructions from memory only, so programs must be stored here to run.
- * General purpose computers run most of their programs from rewritable memory (Main Memory). - Implemented in DRAM.
- * ROM can't be changed, only static programs (like bootstrap) are stored here. EEPROM can be changed but can't be changed frequently so it contains mostly static programs. Such as:- Smart phones have EEPROM to store their factory installed programs (Inbuilt).

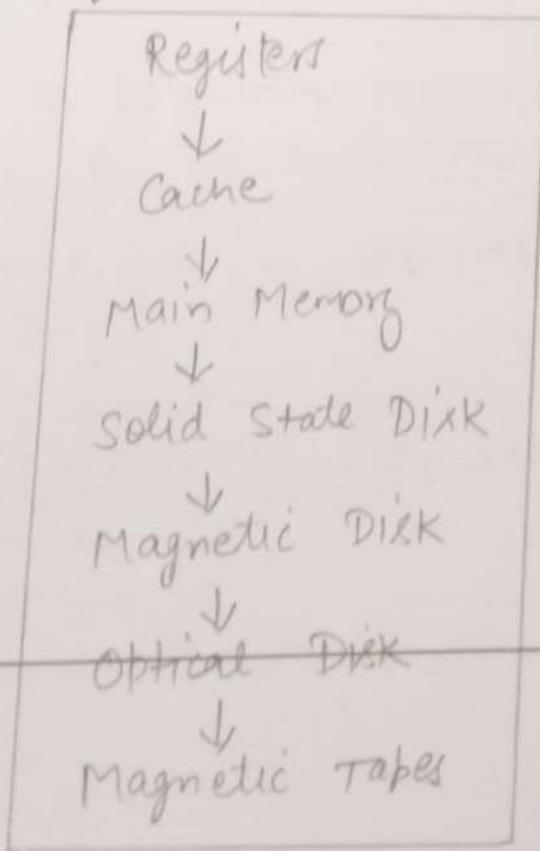


fig : "storage-
Device
Hierarchy"

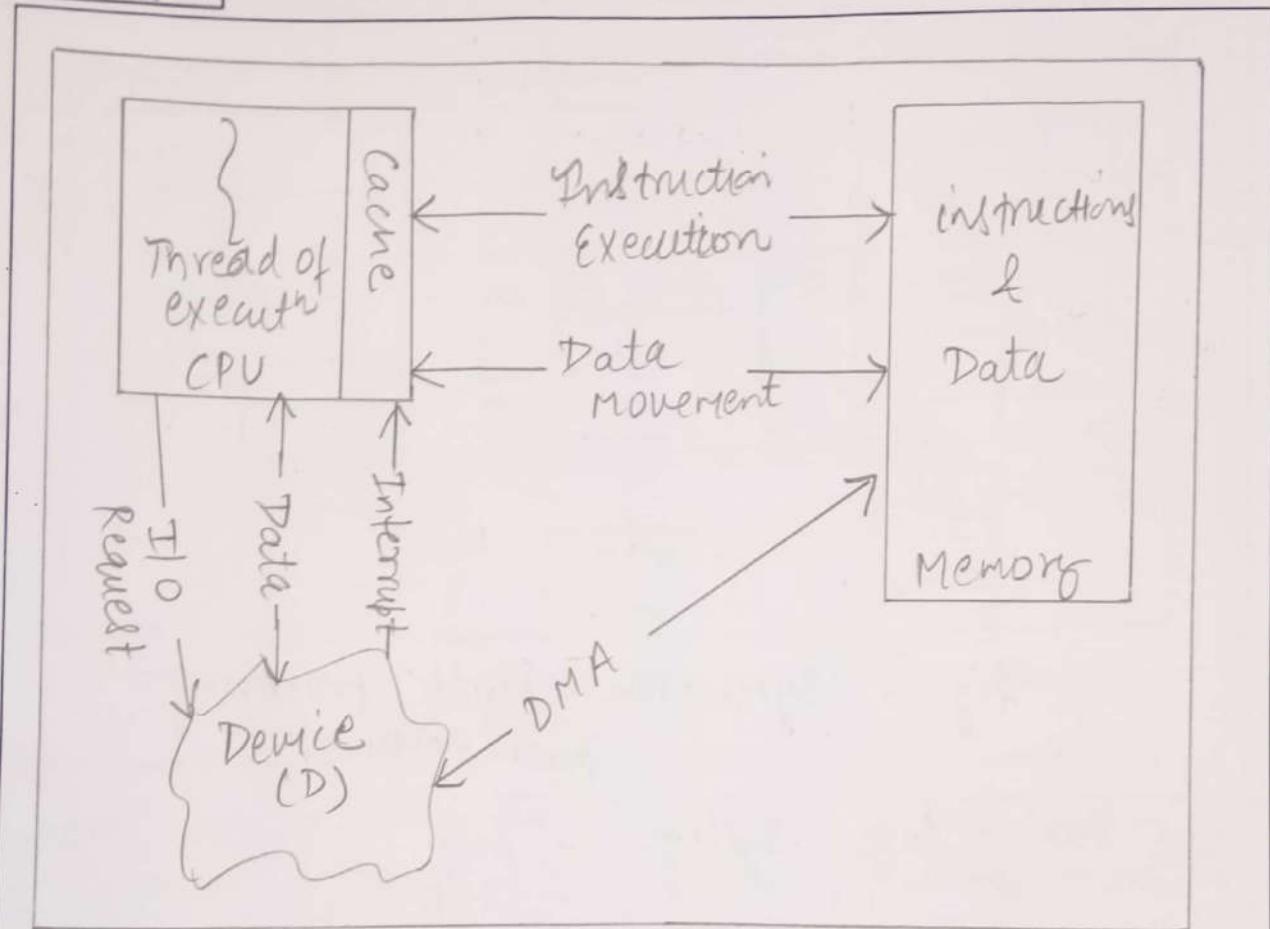


Fig:- "How a Modern Computer System Works".

Some Computer System Architectures:-

- ① single - Processor System :- Only one main CPU is capable of executing a general - purpose instruction set.
- ② Multi - Processor system :- Two or more processors are in close communication, sharing computer Bus, clocks, memory & peripheral devices.

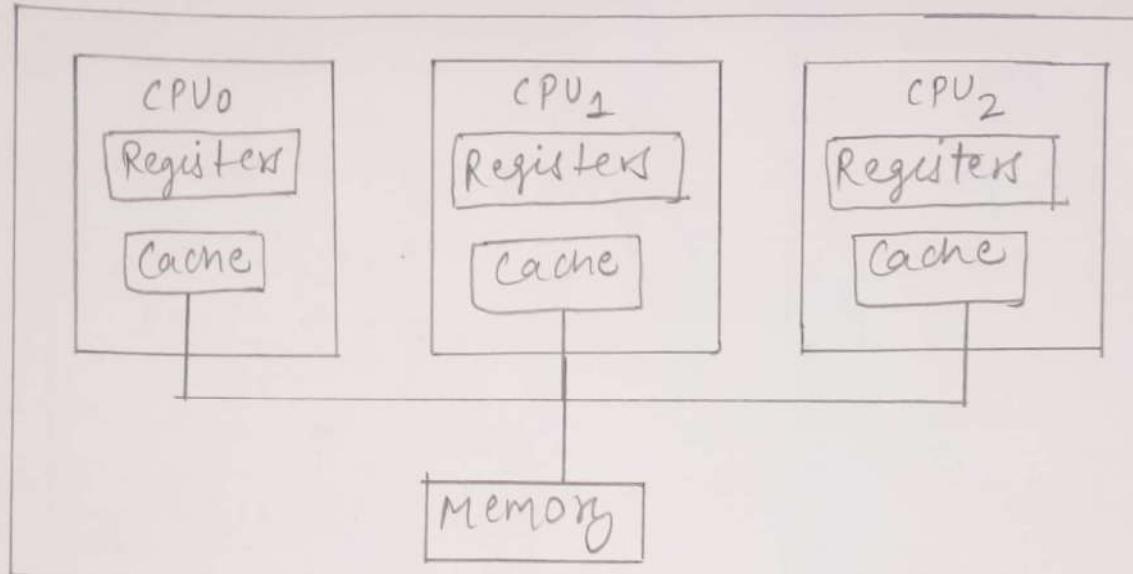


fig :- "Symmetric Multi-processing Architecture"

- ③ Dual - Core system :- Two cores on same chip.
- ④ Clustered System :- Gathers together multiple or more individual CPU's. (Composed of 2 or more individual systems or nodes, joined together).

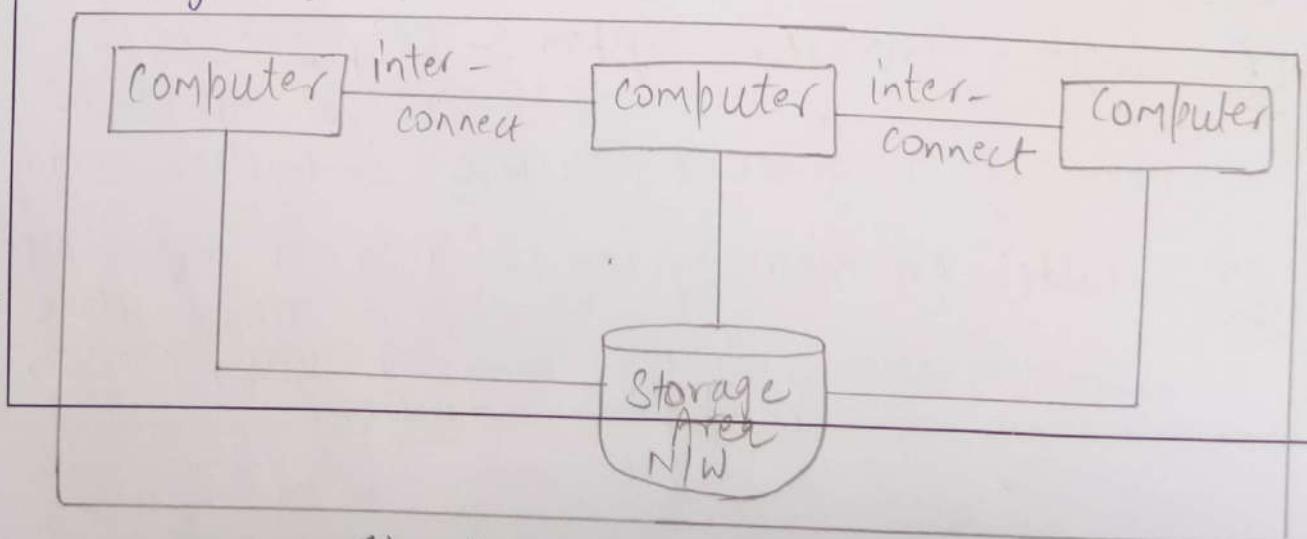


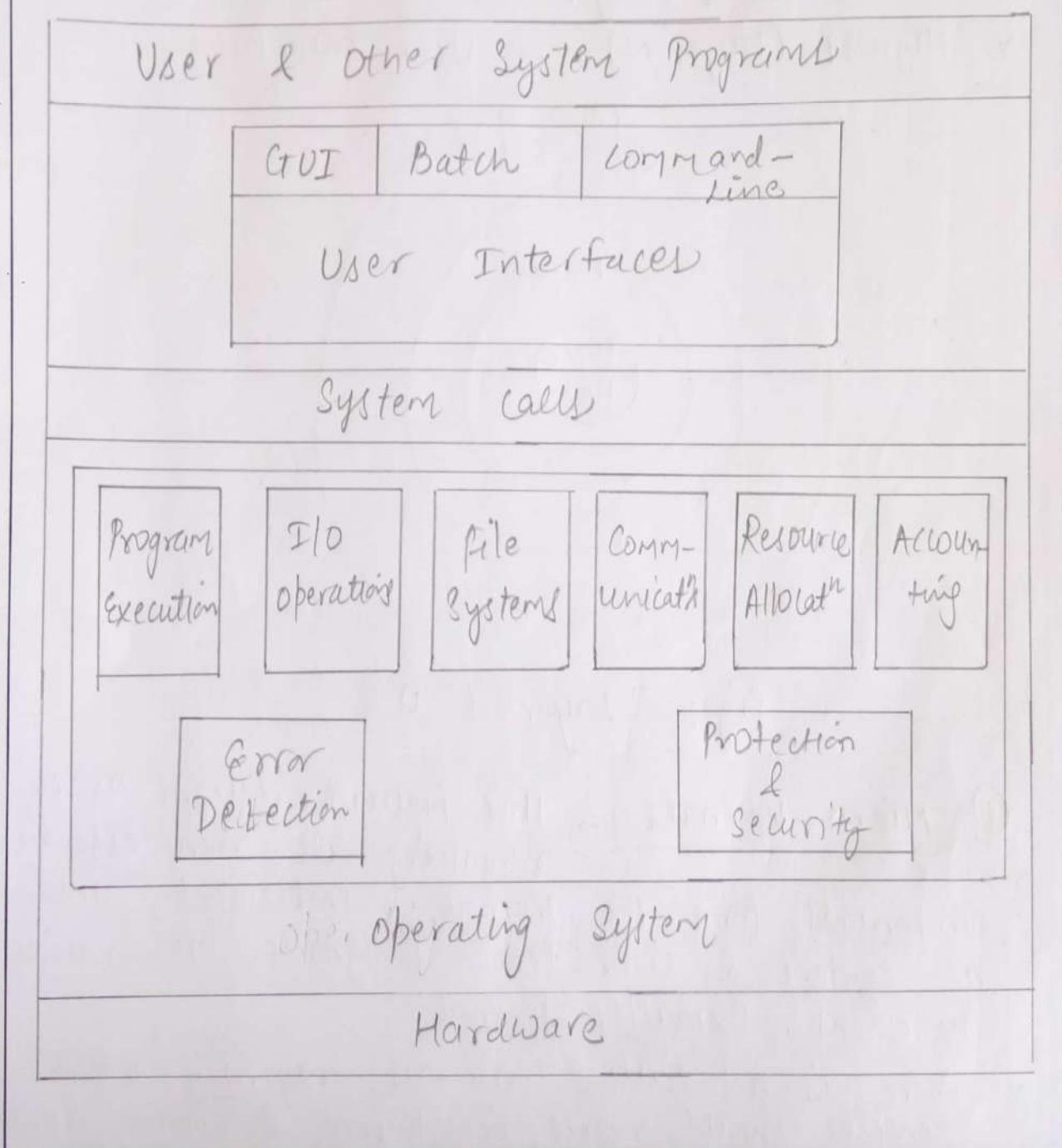
fig: "Structure of clustered system"



Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagar, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Operating System Structure :-

① Simple Structure :-



② Layered Approach :- OS is broken into a no. of layers (levels).

- * Bottom Layer (Layer 0) :- Hardware
- * Highest (Layer N) :- User Interface

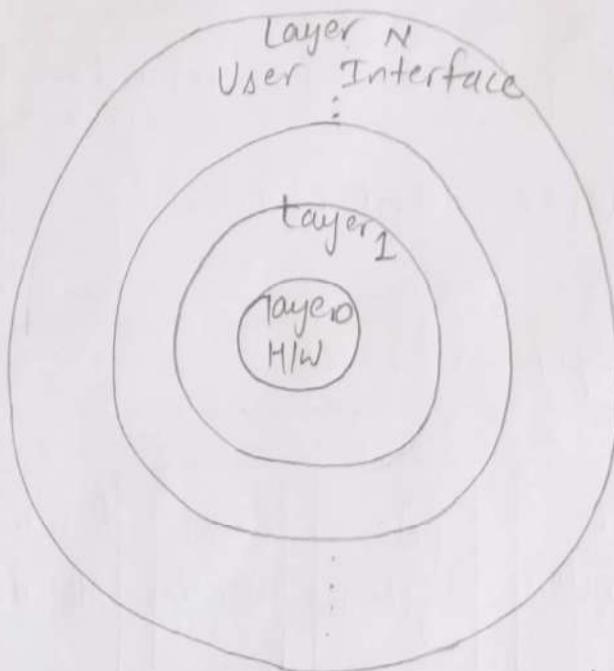
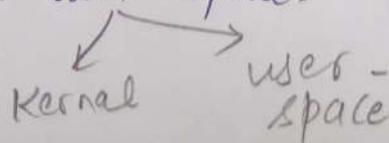


fig: "Layered O.S."

③ Micro-kernels :- This Method designs OS by removing all non-essential components from the Kernel & implementing them as system & user-level programs. The result here is, "smaller Kernel".

- * There is a little consensus regarding which services should reside in Kernel & which should be in user space.



- * Micro-Kernels provide minimal process & memory mgmt, in addition to communication facility.
- * Main function of Microkernel :- provide communication b/w client program & various services that are also running in user space.

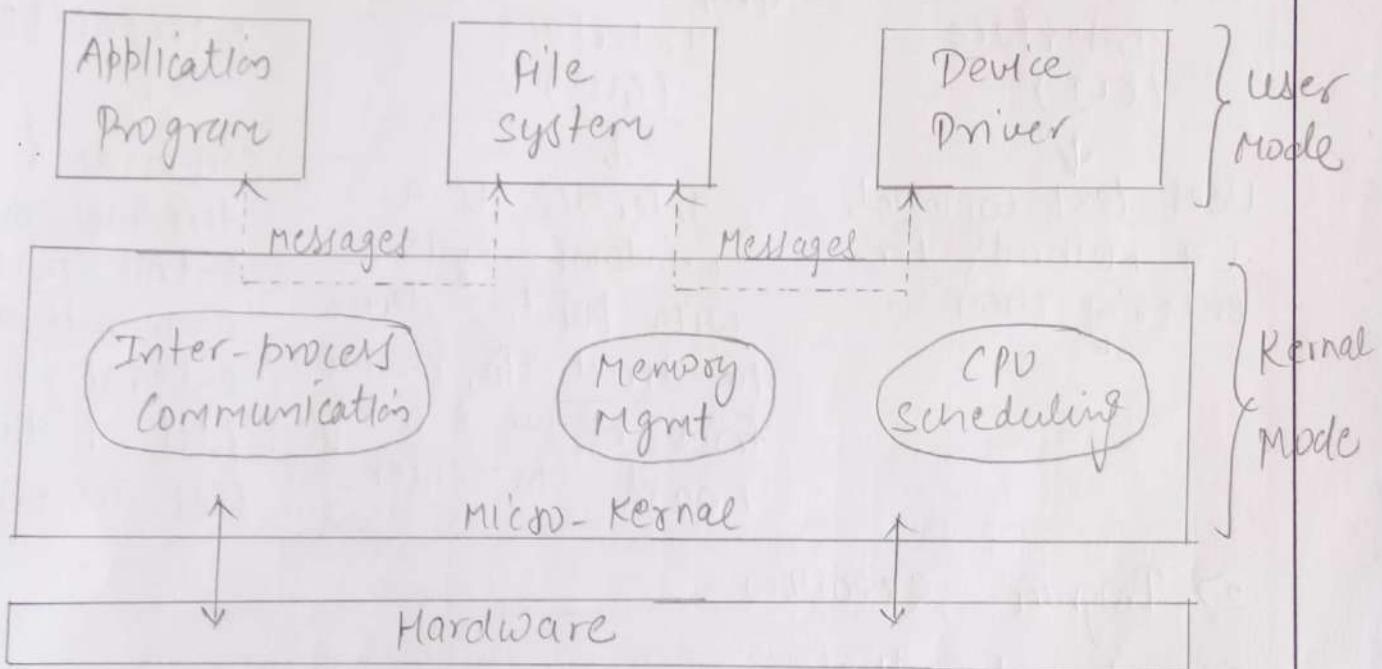


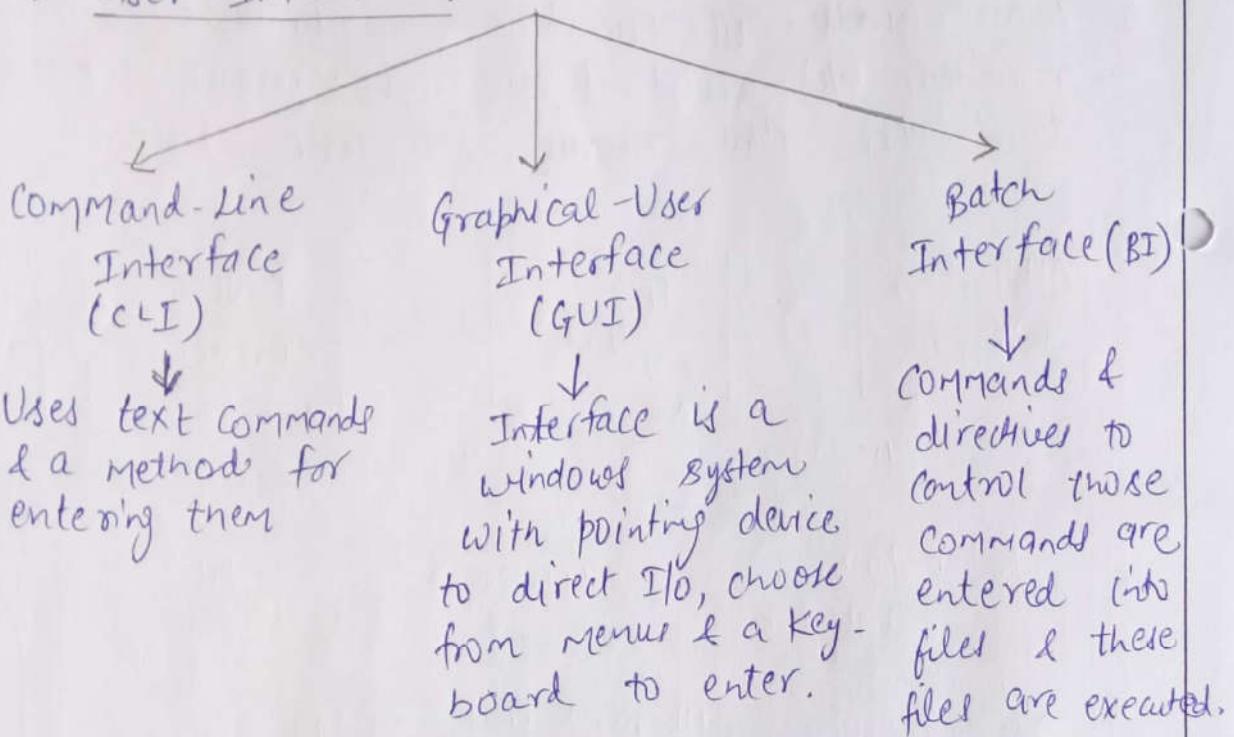
fig:- "Microkernel Architecture"

(ii) Hybrid systems :- Very few OS, adopt strictly defined structure. Others combine different structures, make hybrid systems
 eg:- Apple Mac OS X, iOS & Android
 Linux, Solaris, Window :- Monolithic OS.



Operating System Services:-

1) User Interface:-



2) Program Execution

3) I/O Operations

4) File - System Manipulation

5) Communication

6) Error Detection

7) Resource Allocation :- When multiple users/jobs are there, resource must be allocated to each of them.

8) Protection & Security

9) Accounting :- Track of which user use how much & what kind of computer resources.



System Calls :-

- * Used to access O.S. Services.
- * Provides an interface to the services made available by an O.S.
- * System calls are available as "routines", written in c/c++.

How System call works:-

Example:- Task is to write a program that reads data from one file & copy them to another file.

Source file

→ Destination file

System Call Sequence

Acquire I/P file name

Write prompt to screen

Accept I/P

Acquire O/P file name

Write prompt to screen

Accept I/P

Open I/P file

if file doesn't exist, abort

Create O/P file

if file exists, abort

Loop

Read from i/p file ; Write to o/p file

Until read fails

Close o/p file

Write completion msg to screen

Terminate normally



Types of System calls :- System calls are classified into following 6 major categories :-

① Process Control :-

- a) end, abort b) load, execute
- c) create process, terminate process
- d) get process attributes, set process attributes
- e) Wait for time f) Wait Event, Signal Event
- g) allocate & free memory

Windows	Unix
• CreateProcess()	fork()
• ExitProcess()	exit()
• WaitForSingleObject()	wait()

② File Management :-

- a) Create file, delete file b) open, close
- c) read, write, reposition
- d) get file attributes, set file attributes

Windows	Unix
• CreateFile()	open()
• ReadFile()	read()
• WriteFile()	write()
• CloseHandle()	close()



Swami Keshvanand Institute of Technology, Management & Gramothan,

Ramnagar, Jagatpura, Jaipur-302017, INDIA

Approved by AICTE, Ministry of HRD, Government of India

Recognized by UGC under Section 2(f) of the UGC Act, 1956

Tel. : +91-0141- 5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in

3) Device Management:-

- a) request device, release device
- b) read, write, reposition
- c) get device attributes, set device attributes
- d) logically attach or detach devices

Windows	Unix
<ul style="list-style-type: none">• SetConsoleMode()• ReadConsole()• WriteConsole()	<ul style="list-style-type: none">ioctl()read()write()

4) Information Maintenance:-

- a) get time/date , set time/date
- b) get system data, set system data
- c) get process / file / device attributes , set process/ file / device attributes

Windows	Unix
<ul style="list-style-type: none">• GetCurrentProcessId()• SetTimer()• Sleep()	<ul style="list-style-type: none">getpid()alarm()sleep()



5) Communications :-

- a) Create , delete communication channels/ connection
- b) send , receive messages c) transfer status information
- d) attach / detach remote devices

Windows	Unix
<ul style="list-style-type: none">• CreatePipe()• CreatefileMapping()• MapViewOfFile()	<ul style="list-style-type: none">pipe()shm_open()mmap()

6) Protection :-

Windows	Unix
<ul style="list-style-type: none">• SetFileSecurity()• InitializeSecurityDescriptor()• SetSecurityDescriptorGroup()	<ul style="list-style-type: none">chmod()umask()chown()

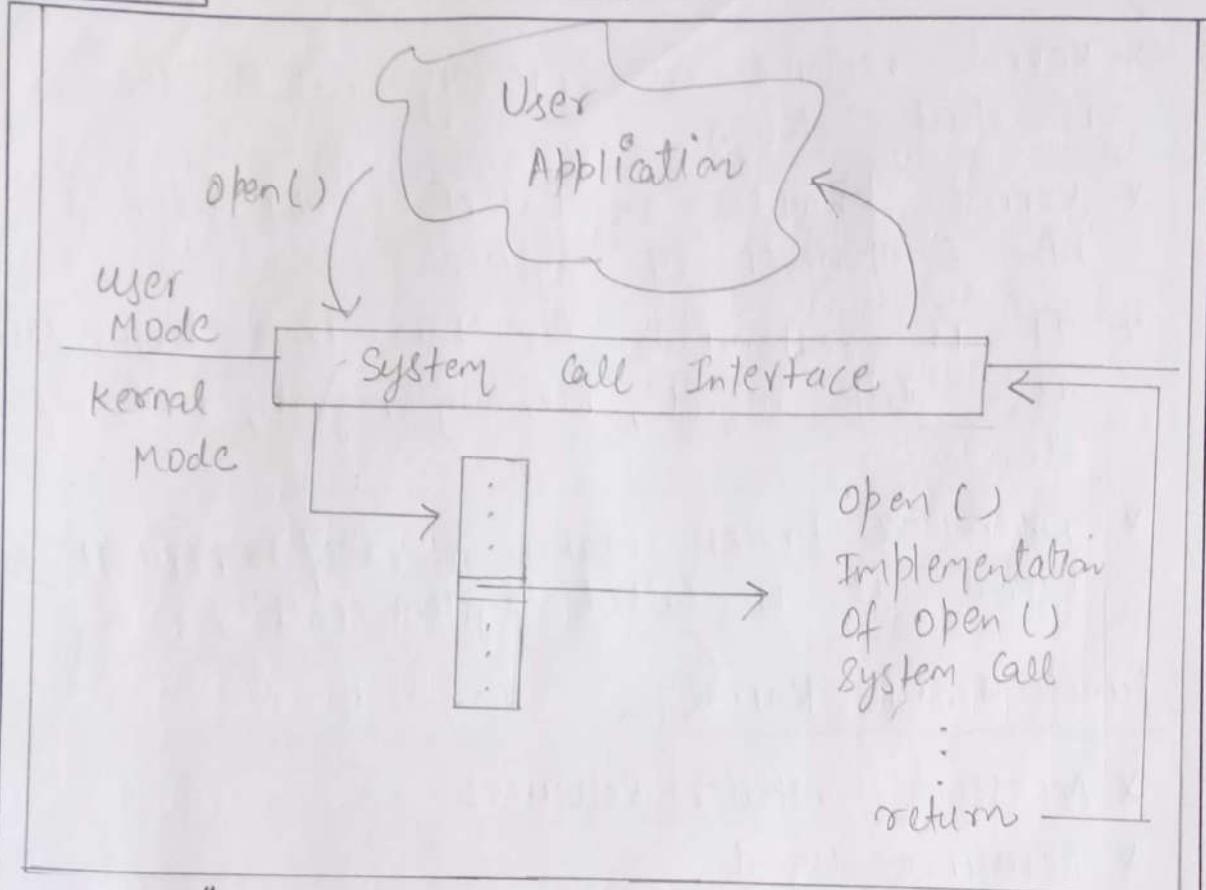


Fig : "Handling of a user Application invoking open() system call()"

Kernel :-

- * Kernel is the computer program at the core of a computer's O.S., with complete control over everything in system.
- * Kernel is the heart/ core of O.S.
- * Since O.S. has control over the system, so it(Kernel) has control over everything in the system.
- * It is the integral/ important part of O.S.
- * When system starts, kernel is the first program to be loaded into memory, after bootloader.



- * Kernel remains in memory until the O.S. is shut-down.
- * Kernel provides an interface b/w user & h/w components of system.
- * It is responsible for low-level tasks such as:- disk mgmt, memory mgmt, task mgmt etc.
- * When a process makes a call/ request to Kernel, it is called "System Call".

Functions of Kernel :-

- * Accessing Computer Resources
- * Resource Mgmt
- * Device Mgmt
- * Memory Mgmt
- * Process Mgmt
- * I/O Communication

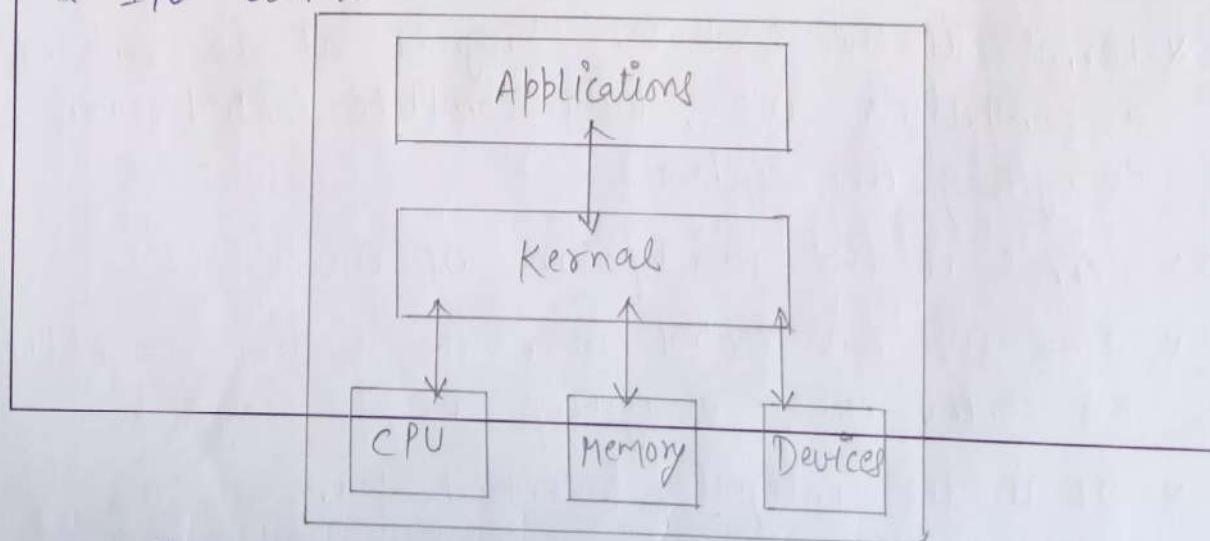


fig: "A Kernel connects Application s/w to h/w of computer"



Kernal Mode & User Mode :-

- * Certain instructions are needed to be executed by Kernel only. So, CPU executes these ~~exe~~ instructions in Kernel mode only.
- * Example:- Memory Mgmt is done in Kernel mode only, while in user mode, CPU executes the processes that are given by the user in user-space.

Types of Kernel :-

(1) Monolithic Kernel :-

- * Where user services & kernel services both are implemented in the same memory space.
- * No separate user space & kernel space.

(2) Microkernel :-

- * User & Kernel services are implemented in diff. memory space (i.e. User space & Kernel space).
- * Reduces size of Kernel, as well as size of O.S.
- * In Monolithic Kernel, as there is only a single space, so kernel size increases, resulting increased size of O.S.

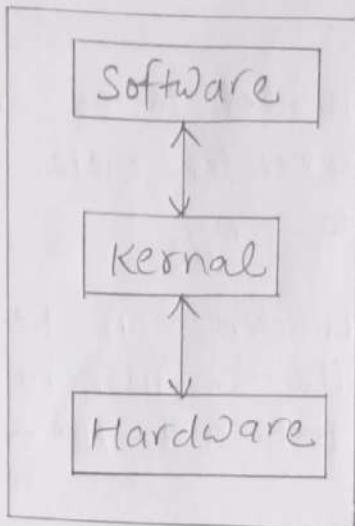


fig: Monolithic Kernel

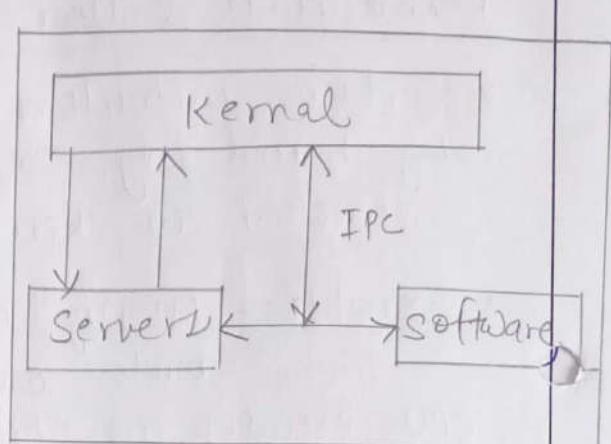


fig : Micro - Kernel

③ Hybrid Kernel :- Combination of Monolithic & Micro-Kernel.

- * User space of Monolithic + Modularity of Micro-Kernel.
- * Network stack or file system run in Kernel space.
Kernel code run in user-space as servers.

④ Nano Kernel :-

- * Kernel code is very small, i.e. code executing in the privileged mode of H/W is very small.

⑤ EXO Kernel :-

- * Resource Protection is separated from the mgmt & results in allowing us to perform application-specific customization.

SHELL :- An interface that allows users to communicate with Kernel. (Or to access O.S. Services).
User → SHELL → Kernel | eg:- Bourne, C, Korn shells etc



असतो मा रामगमय

Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagar, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel.: +91-0141-5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

System Boot :-

- * The process of starting a computer by loading the kernel is known as "Booting".
- * A small piece of code (Bootstrap program/Loader) performs the following tasks:-
 - Locates the Kernel
 - Loads the Kernel into main memory
 - Starts its execution.
- * When the system is powered up or rebooted, the instruction register is loaded with a predefined memory location & execution starts here.
- * Initial Bootstrap program is at that location.
 - & this program is in the form of "Read Only Memory (ROM)", bcz ROM dont need initialization
 - & it can't be easily infected by a computer virus.
- * Bootstrap program performs a variety of tasks:-
 - Run ~~dig~~ diagnostics to determine the state of machine. If it pass, the program can continue with booting steps.
 - Initialize all aspects of system, i.e. from CPU ^{regis} to device controllers & contents of main memory
 - Sooner or later, it starts the OS.



अरातो मा सद्गमय

Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagar, Jagatpura, Jaipur-302017, INDIA

Approved by AICTE, Ministry of HRD, Government of India

Recognized by UGC under Section 2(f) of the UGC Act, 1956

Tel. : +91-0141- 5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in

- * Some systems (such as: - cellular phones, tablets or game consoles) store the entire OS in ROM. Storing OS in ROM is suitable for small OS.
- * Challenge is to change the bootstrap code which requires changing the ROM H/W chips. So, this problem is resolved by EPROM, which is read only, except when an explicit command is given to write.
- * All forms of ROM are "FIRMWARE", b/w their characteristics fall b/w that of H/W & S/W both.
- * Problem with firmware is that code execution in firmware is slower than code execution in RAM.
- * So, avoid above stated problem, some systems store OS in firmware & copy it into RAM for faster execution.
- * For large OS (Windows, Mac OS X, Unix etc), or for systems that change frequently, OS bootstrap is stored in firmware & OS is stored on disk.

Bootstrap Loader:- It reads the hard drive boot sector to continue loading the OS.

* A Program which resides in ROM, EPROM or another non-volatile memory.

* It is automatically executed by the processor, when turning on the computer.

- * When Computer is turned on/restarted, the bootstrap loader first perform a test:-POST (Power on self Test).
- * If POST is successful & no issues are found, bootstrap will locate the kernel, load it into main memory & starts execution.
- * Bootstrap is the first program/code to be executed, when system is started.
- * The entire OS depends on the bootstrap to work correctly. [Bootstrap → Kernel → OS]

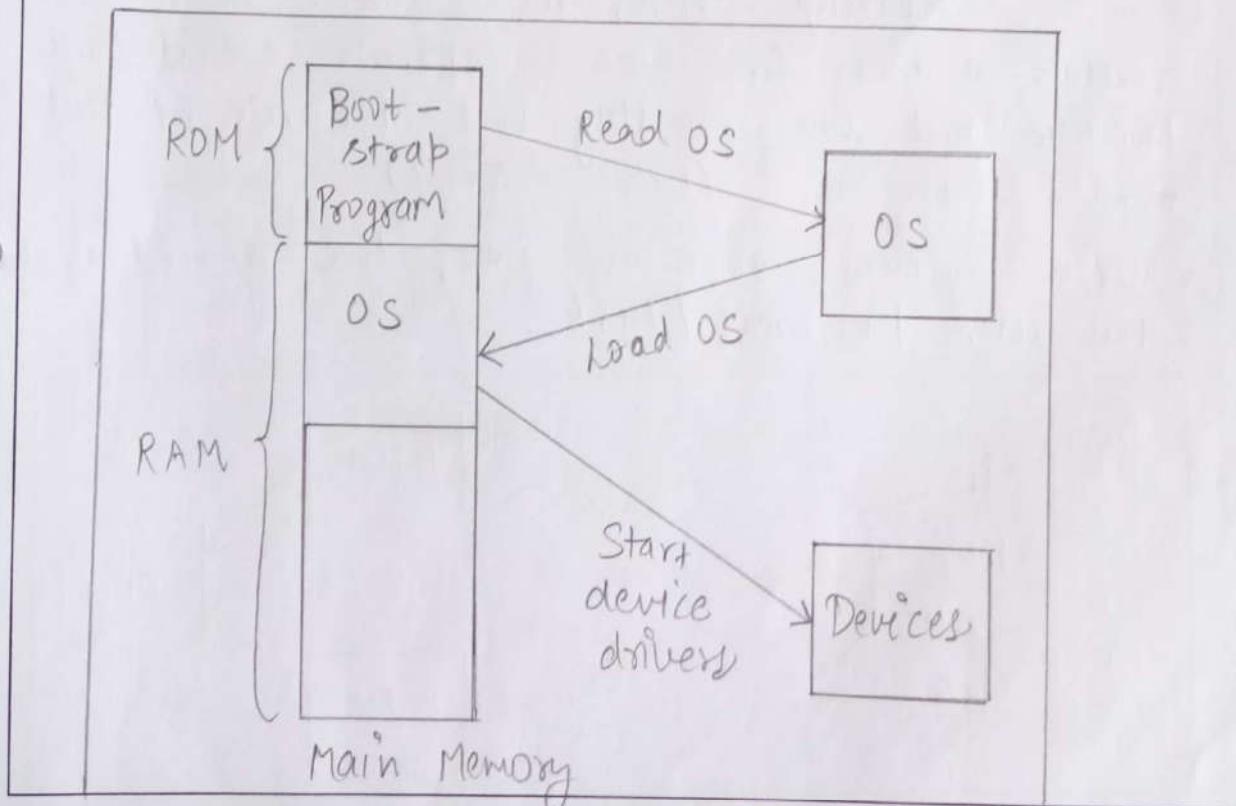


fig: "Bootstrap Program"

"PROCESS MANAGEMENT"

D Batch Processing :- A process by which a computer complete batches of jobs, simultaneously, in a non-stop, sequential order.

* Large jobs are computed in small parts for efficiency.

* It is running of frequently used programs/jobs that are executed without or with minimum end user interaction.

* Example:- "The way credit card companies process billing". Here, the customer does not receive a bill for each separate credit card purchase but one monthly bill for all of that month's purchase. (Batch - Group)

* Batch systems executes jobs, time-shared system has user programs / tasks.

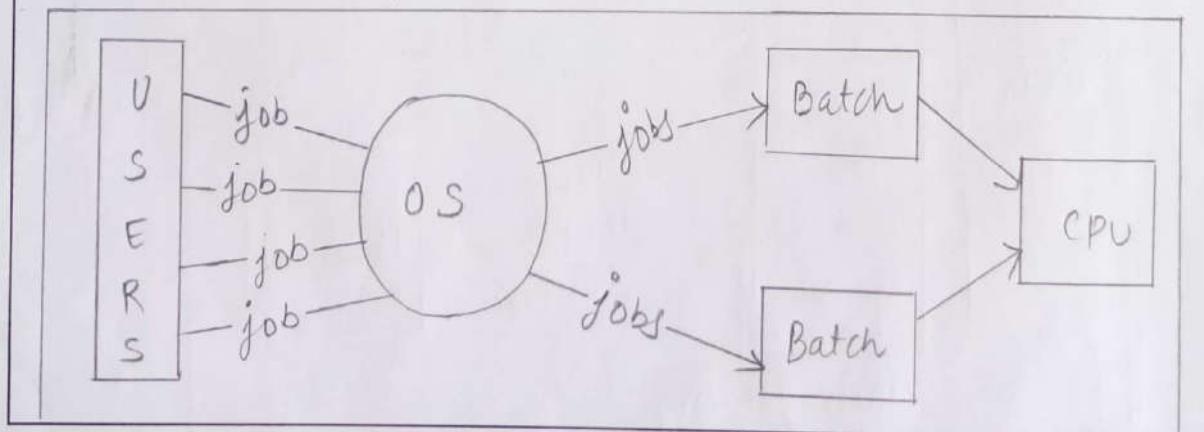


fig: "Batch Processing"

2) Time-Sharing OS. → A technique which enables many people, located at various terminals, to a particular computer system at same time.

- * Time sharing refers to allocation of resources in time slots to several programs (called "Tasks") simultaneously.
- * Here, each task is given some time to execute so that all tasks can work smoothly.

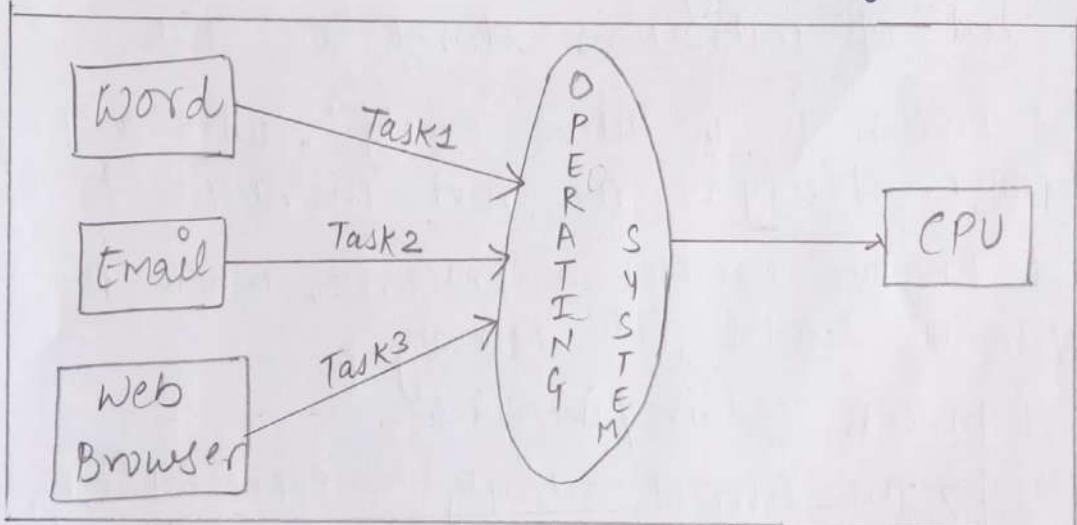


fig: "Time-Sharing System"

3) Distributed O.S. :- Various autonomous computers (homogeneous or heterogeneous) are inter-connected & communicate to each other by using a shared communication channel.

4) Network OS :- Highly coupled system with heterogeneous multi-computer.

Both mostly used in D.S.



Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagar, Jagatpura, Jaipur-302017, INDIA

Approved by AICTE, Ministry of HRD, Government of India

Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagar, Jagatpura, Jaipur-302017, INDIA

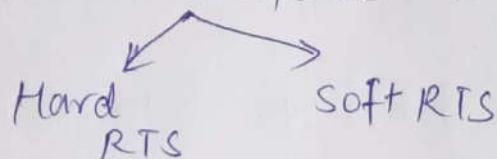
Approved by NCTB, Ministry of HRD, Government of India

Recognized by UGC under Section 2(f) of the UGC Act, 1956

Tel. : +91-0141-5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in

5) Real Time OS : Here, the time required to process & respond to I/P's is very small. (Called "Response Time").



Concept of Process : "A program in execution".

- * A program itself is not a process. Program is a "passive entity", such as:- A file containing a list of instructions stored on disk.
- * A process is an "active entity", with a program counter specifying the next instruction to execute.
- * A program becomes a process, when an exe file is loaded into memory.
- * A process contains / includes :-
 - The current activity :- Represented by the value of PC & contents of processor register.
 - Process stack :- Contains temporary data (like parameters, return add., local variables etc)
 - Data Section :- Contains global variables.
 - Heap :- A dynamically allocated memory during process run time.

max

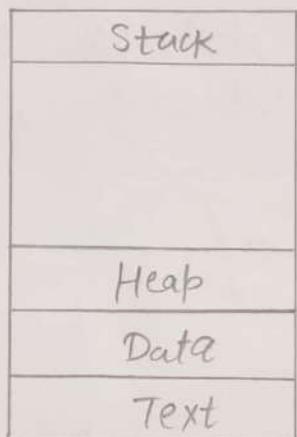


fig: "process in memory"

Process State :- Active / current status of the process.
As process executes, it changes state.

- * New :- When created
- * Running :- When being executed
- * Waiting :- When waiting for an event to occur.
- * Ready :- When ready to assign to processor.
- * Terminate :- When finished execution.

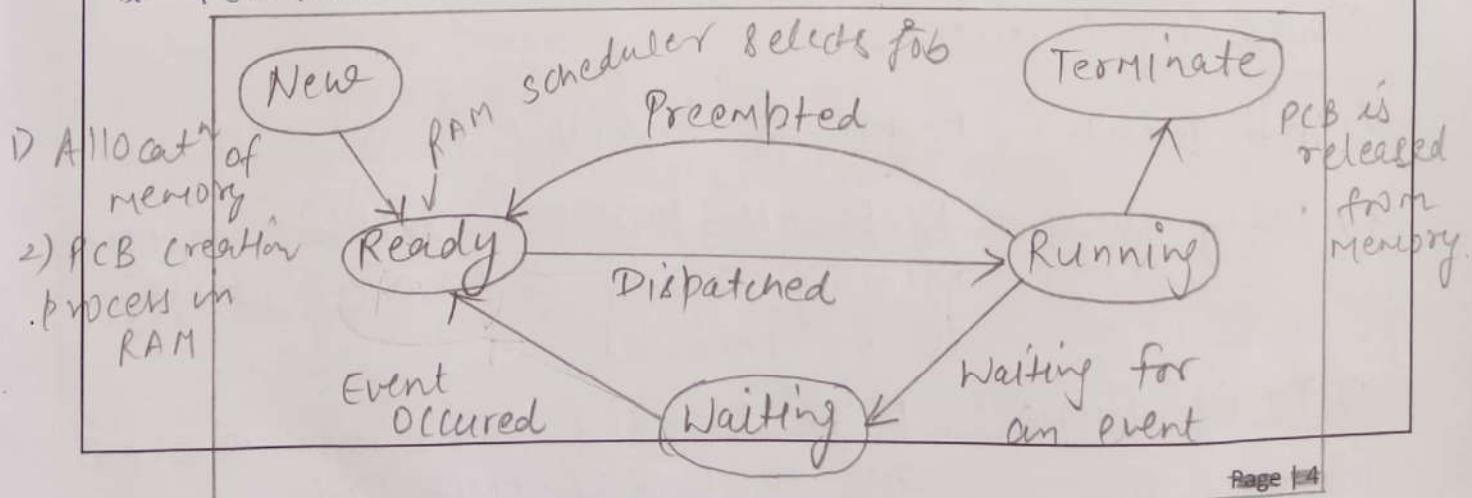
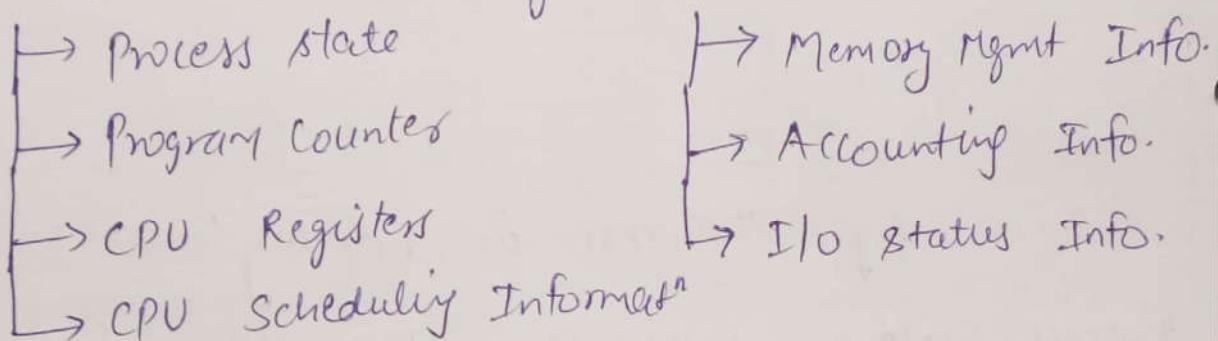


fig: "5-state model"



Process Control Block:- A block that contains various information, associated with a specific process.

* A PCB contains following :-



Process scheduling:- A way to organize processes to work in the system.

* An activity of process manager, that handles the removal of running process from the CPU & Selection of another process on basis of a particular strategy.

Scheduling Queues:-

- a) Job queue :- Keeps all processes in the system.
- b) Ready queue :- Keeps all processes residing in main memory, ready and are waiting to be executed.
- c) Device queue :- Keeps processes which are blocked due to unavailability of an I/O device.

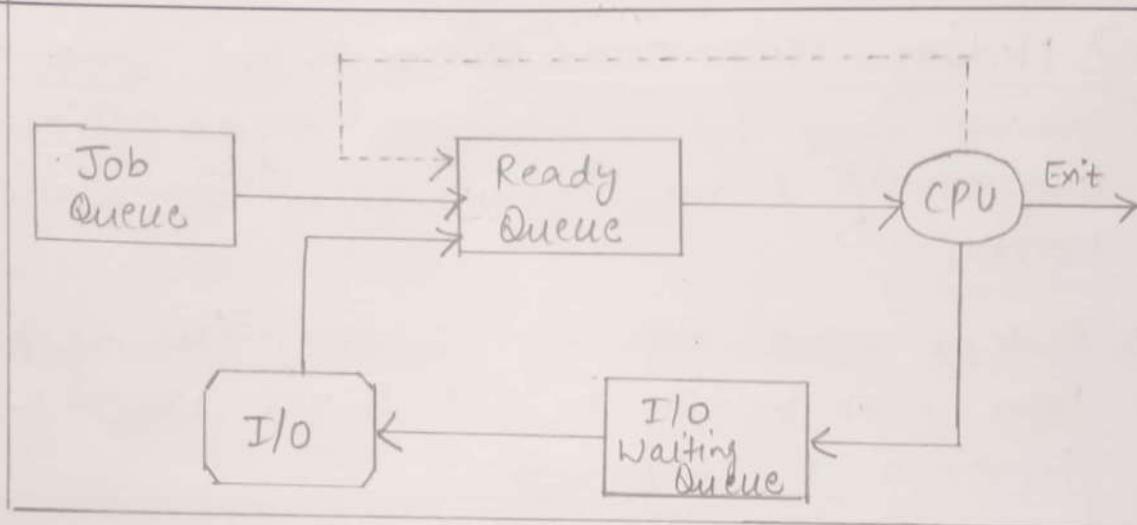
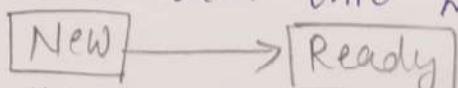


fig: - "Scheduling Queues"

Schedulers :- A medium that migrates a process among various scheduling queues throughout its lifetime.

a) Long-Term Scheduler :- (OR Job scheduler) Selects new processes ("Ready state") from process pool and load them into main memory (Ready state).

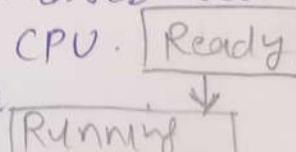


* Controls degree of multi-programming. [no. of processes present in memory/ready state at any point of time]

b) Short-Term / CPU Scheduler :- Responsible for selecting a process from ready state for scheduling it on running state.
 NOTE :- Only selects.

Dispatcher :- The time taken by dispatcher to stop one process & start another

Dispatcher :- Responsible for locating processes selected by CPU scheduler on the CPU. i.e. Ready to Running state



c) Medium - Term Scheduler : \rightarrow Concept is to remove process from memory, reduce degree of multi-programming & then reintroducing it again into memory.

* Perform above task i-e. swapping (Moving processes from main memory to disk & vice versa) :- called Swap-in & Swap-out.

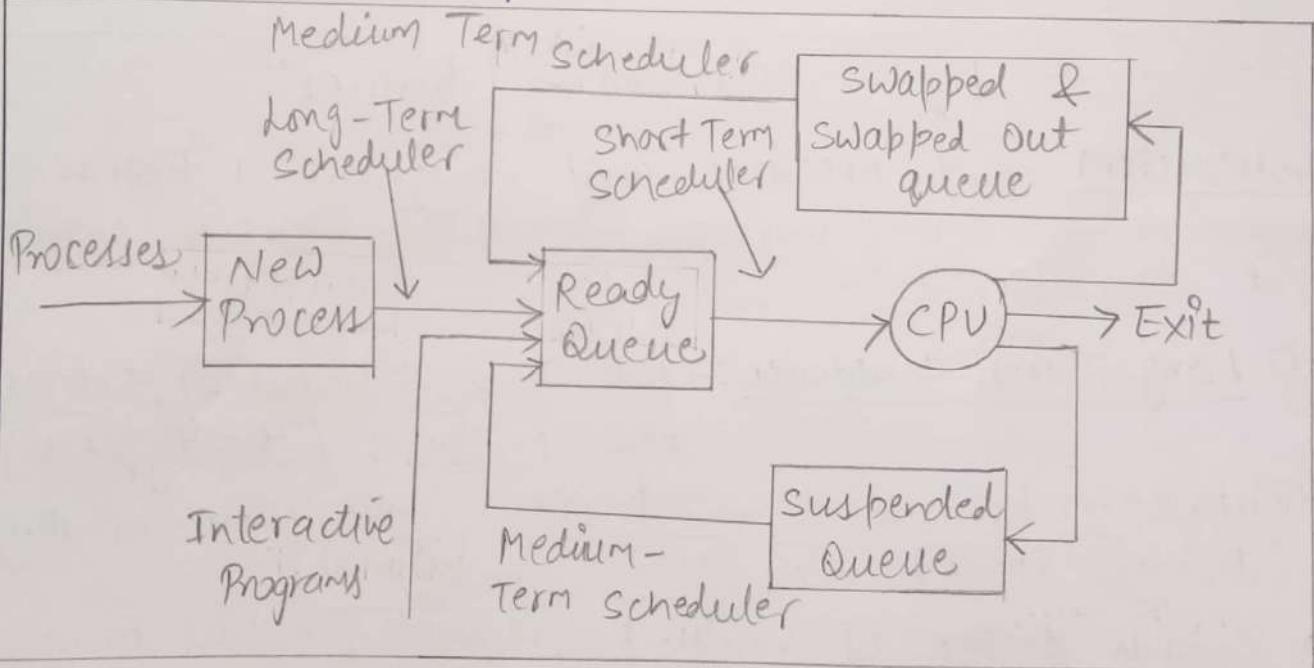


Fig : \Rightarrow "Schedulers"

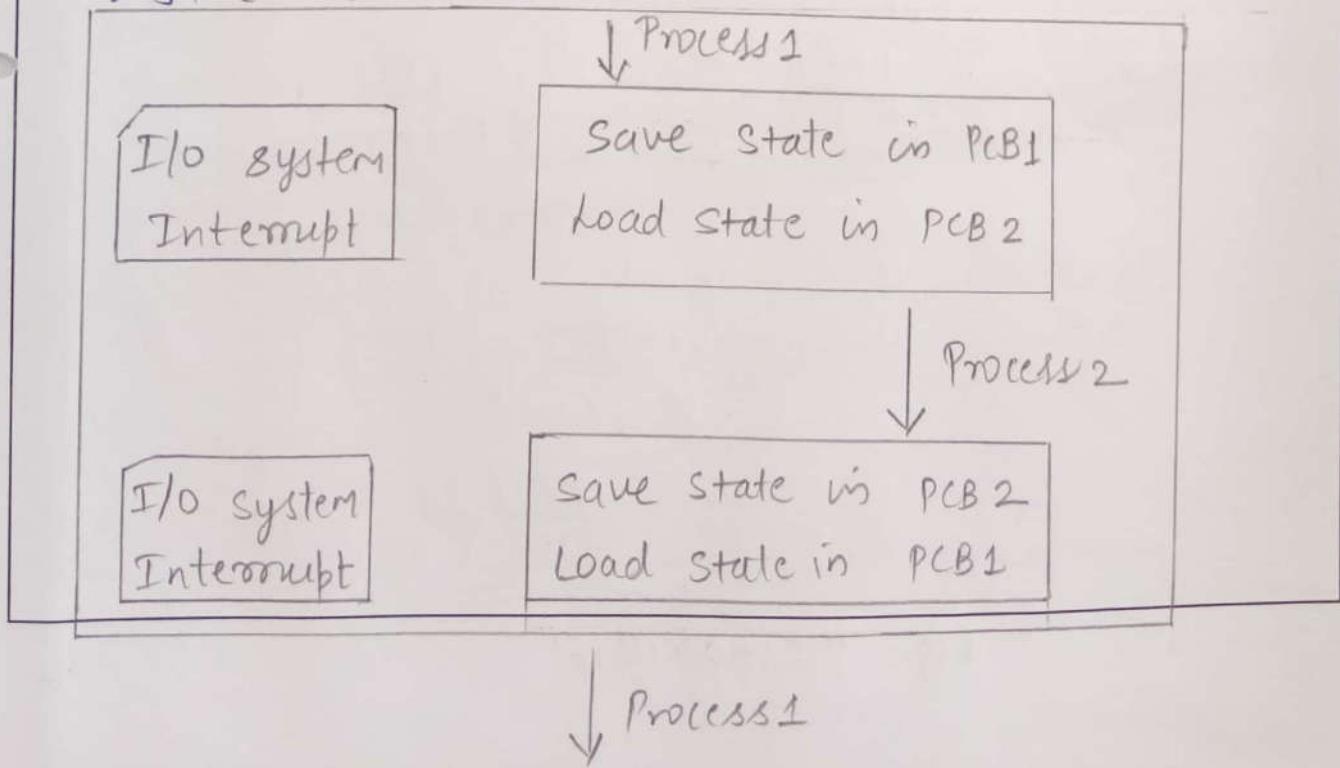
Context Switch : - When Kernel transfers the control of CPU from an executing process to another process that is ready to run.

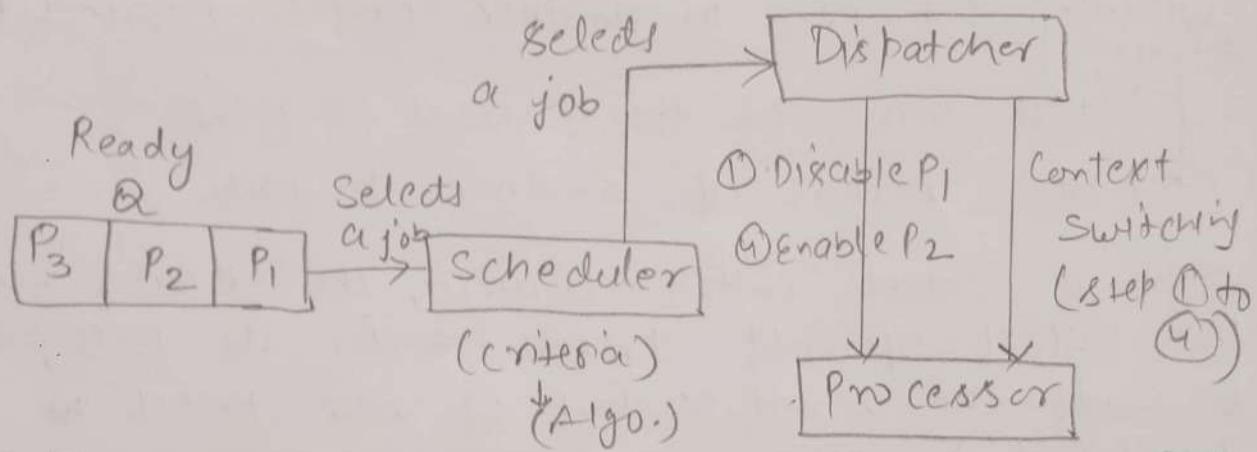
* When the process that was taken off the CPU next runs, it resumes the point at which it was taken off the CPU.

* It



- * Switching the CPU to another process requires performing:
 - State save of the current process
 - State restore of another process
- * When a context switch occurs, the kernel saves the context of old process & in its PCB, & loads the saved context of new process scheduled to run.
- * Context - switch time is pure overhead, bcz the system does no useful work while switching.
- * Switching speed varies from machine to machine depending on memory speed, the no. of registers that must be copied & existence of special instructions.





Dispatcher :-

- ① Analyze PC in PCB
- ② Context switching

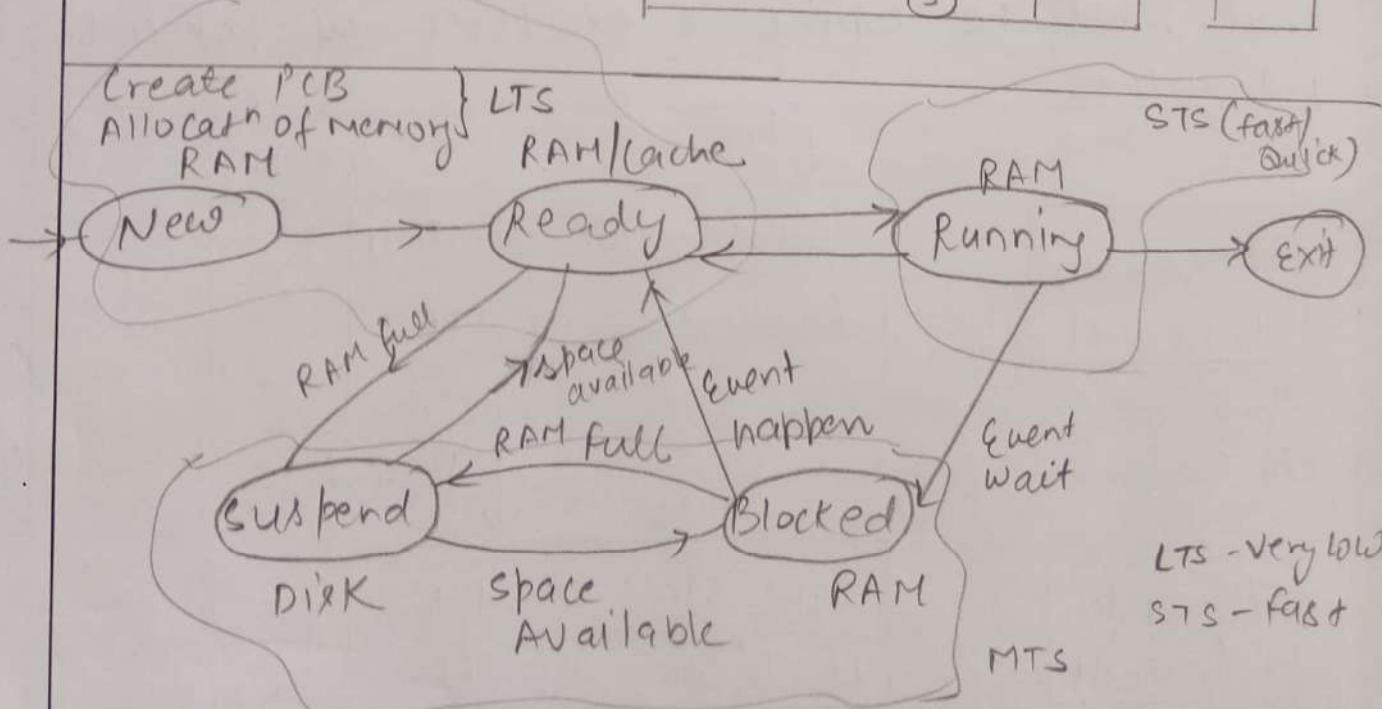
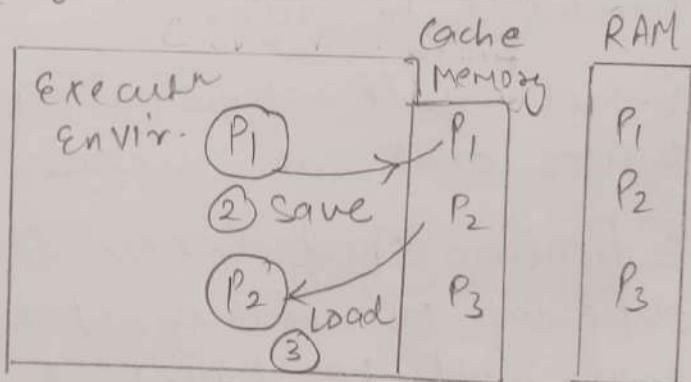


Fig : "Scheduler"

"Process Management"

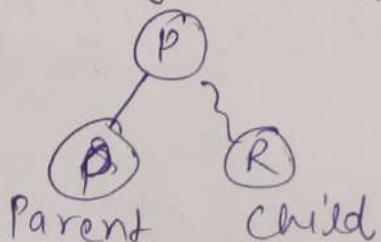
Contents :-

- 1> Process
- 2> Process Initiation
- 3> Process Synchronisation
- 4> Process Control Block
- 5> IPC

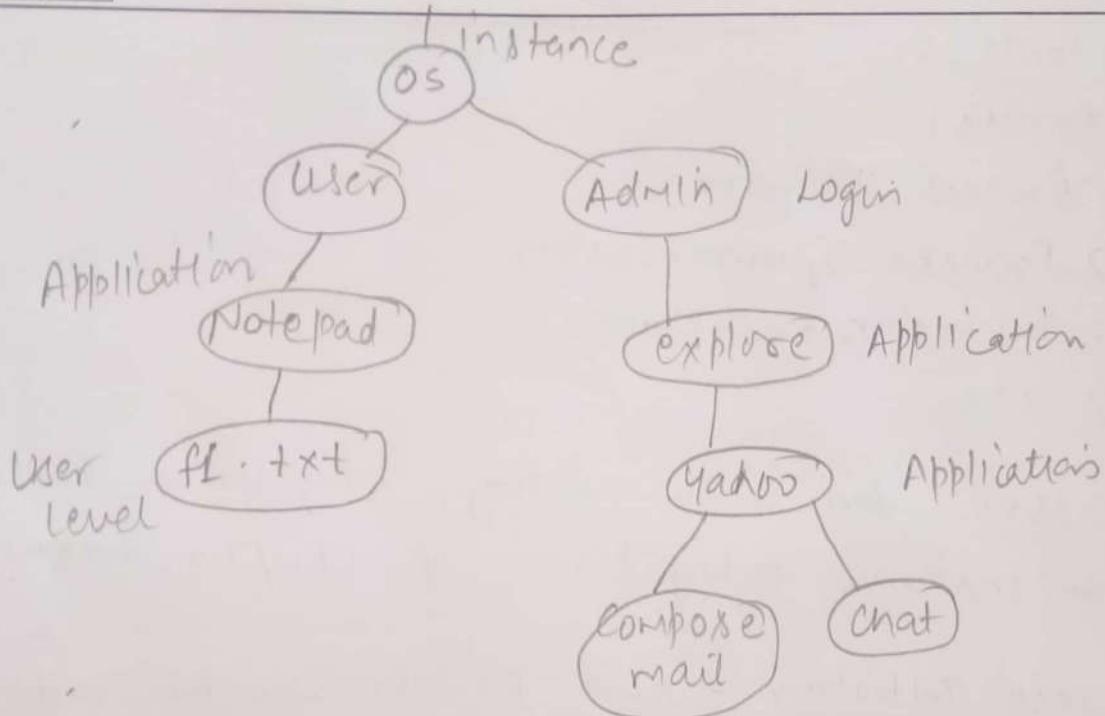
1> Process:- An active entity, program in execution.
x An instance taking care of CPU/IO tasks.

2> Process Initiation:- How process can be entered or initiated :-

- a> User Submission (By clicking on specific program)
- b> System Initiated Jobs (Windows Task manager
eg. to calculate address, to refer specific
memory).
- c> System Calls :- Using fork() in unix.

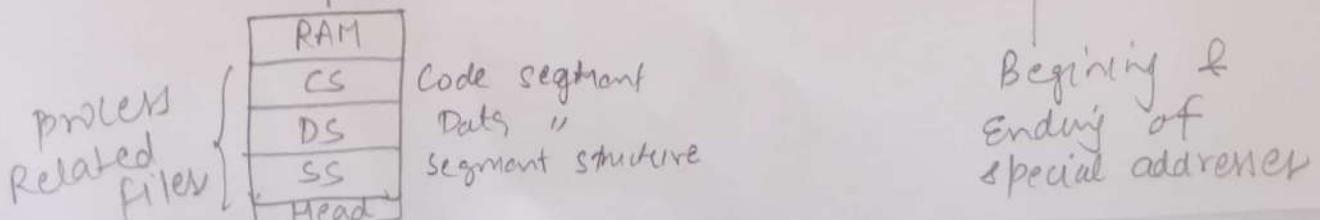


d> Login :-



3) PCB :- Takes many decisions like:- when to initiate process, resume & preempt.

Process ID
Process State
Process Priority
PC
Process Address Space
Links





Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagar, Jagatpura, Jaipur-302017, INDIA

Approved by AICTE, Ministry of HRD, Government of India

Recognized by UGC under Section 2(f) of the UGC Act, 1956

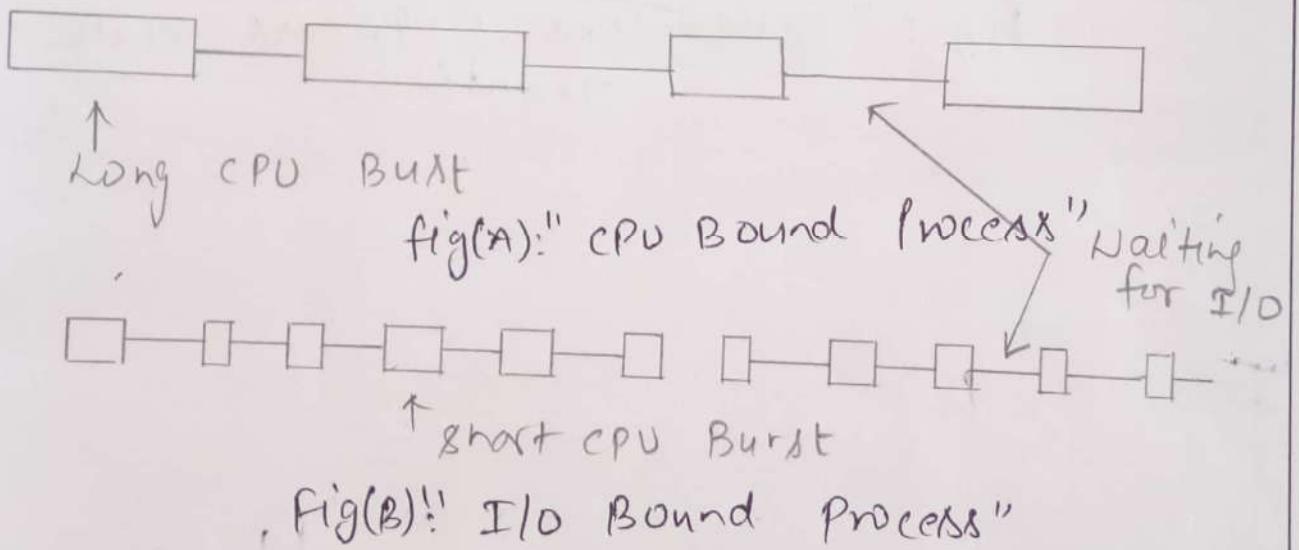
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in

Basic Concepts of CPU / Process Scheduling :-

* Aim of multi-programming :- Keep CPU as much busy as possible

CPU & I/O Bound :-



* In fig (A), processes spend their most of time in computing, so they are called CPU Bound.

* In later fig (B), processes spend their most of time waiting for I/O, called I/O Bound.

* Perfect CPU scheduler is one with:-

- Minimum Latency (Response/ Job completion Time)
- Maximum Throughput (maximum jobs/time)
- Maximum Utilization
- Fairness



Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagar, Jagatpura, Jaipur-302017, INDIA

Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956

Tel. : +91-0141- 5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in

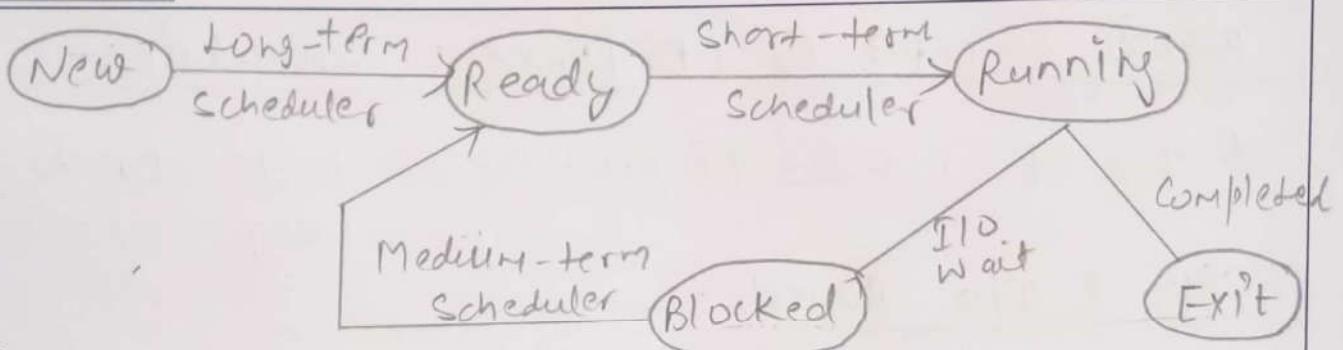
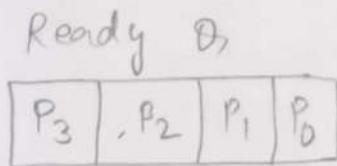


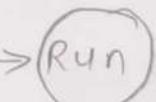
fig : "Schedulers & Process State Transition"

Process Scheduling :-



Selects
a job
(criteria)

CPU Scheduling
(Processor)



- 1) Arrival Time :- Processes are in ready Q.
- 2) Burst Time :- Predicted time to complete a job.
- 3) Waiting Time :- Duration when process is not executing, (waiting in ready Q).
- 4) Turnaround Time :- Amount of time to execute a particular process.
 - ⇒ finished Time - Arrival Time
 - ⇒ Waiting Time + Burst Time
- 5) Response Time :- Amount of time (from submission of a request to when the first response is received), not O/P.
- 6) Throughput :- Performance measuring factor of CPU.
 - ↳ No. of O/P's in an observation time.
- 7) CPU Utilization :- Keep CPU as busy as possible.

In a system, try to maintain :-

- Least waiting time
- Least turn-around time
- High throughput

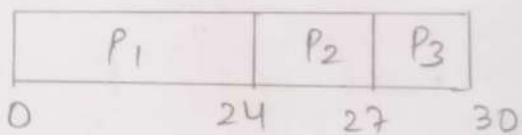


Scheduling Algorithms :-

- ① FCFS :- First come first serve :- Processed arriving first is selected for execution
- * Compute Avg Waiting time (AWT)
 - * Compute Avg Turnaround time (ATT)

Process	Burst Time (ms)
P ₁	24
P ₂	3
P ₃	3

Grantt chart :



$$AWT = \frac{W_1 + W_2 + W_3}{3} = \frac{0 + 24 + 27}{3}$$

$$AWT = 17ms$$

$$ATT = \frac{t_1 + t_2 + t_3}{3} = \frac{(0+24)+(24+3)+(27+3)}{3}$$

[AR, Arrival time is 0]

$$OR = \frac{(24-0) + (27-0) + (30-0)}{3}$$

$$ATT = 27ms$$

Features :-

- * NO preemption
- * Sequential processing
- * Throughput not in control (disadvantage)
- * If job with longer B.T. arrives at first, increases WT & TT of another processes (convey effect)



Preemption :- forceful termination (Abnormal)

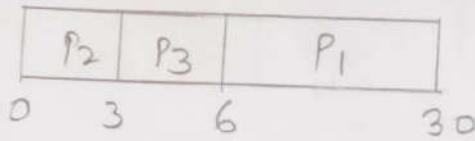
Non preemption :- Normal termination

② SJF :- Shortest Job first :- Process with least BT will be executed first & so on.

a) Non Preemptive

Process	BT
P ₁	24
P ₂	3
P ₃	3

Gantt chart:-



* Compute AWT

* Compute ATT

$$\begin{aligned} W.T.(P_1) &= 6 \\ W.T.(P_2) &= 0 \\ W.T.(P_3) &= 3 \end{aligned} \left\{ \begin{array}{l} A.W.T = (0+3+6)/3 \\ = 3 \text{ ms} \end{array} \right.$$

$$T.T.(P_1) = (6+24) \text{ OR } (30-0) = 30$$

$$T.T.(P_2) = (0+3) \text{ OR } (3-0) = 3$$

$$T.T.(P_3) = (3+3) \text{ OR } (6-0) = 6$$

$$\left. \begin{array}{l} A.T.T = (30+3+6)/3 \\ = 13 \text{ ms} \end{array} \right\}$$

$$\boxed{\begin{aligned} SJF(A.W.T) &< FCFS(A.W.T) \\ SJF(A.T.T) &< FCFS(A.T.T) \end{aligned}}$$

} Better Throughput

* Try following example:-

Advantage :-

* Shortest jobs are favoured.
 * Optimal, bcz gives minimal avg WT for a given set of Procs

Disadvantage :-

* Causes Starvation, if a longer process keeps coming.

* Aging used to overcome

Process	BT
P ₁	6
P ₂	8
P ₃	7
P ₄	3



⇒ Preemptive SJF :- [SRFT] :- shortest Remaining Time First

- * If a process A arrives with less BT, than remaining execution time currently running process B, then B is preempted & A is allowed for the execution. This is called SRFT.

Process	BT	AT
P ₁	1	3
P ₂	4	1
P ₃	2	4
P ₄	6	0
P ₅	3	2

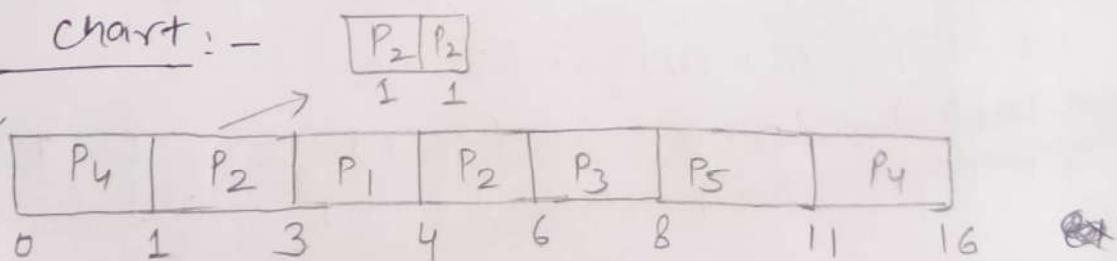
* When Arrival time is given :-

Waiting Time \Rightarrow Actual Waiting OR - Arrival time

WT \Rightarrow Turnaround - Burst

Turn Around Time \Rightarrow Completion Time - Arrival Time

Gantt chart :-



Waiting Time :-

$$P_1 = 3 - 3 = 0$$

$$P_2 = (1 + (4 - 3)) - 1 = 1$$

$$P_3 = 6 - 4 = 2$$

$$P_4 = (11 - 1) - 0 = 10$$

$$P_5 = (8 - 2) = 6$$

Turnaround time :-

$$P_1 = 4 - 3 = 1$$

$$P_2 = 6 - 1 = 5$$

$$P_3 = 8 - 4 = 4$$

$$P_4 = 16 - 0 = 16$$

$$P_5 = 11 - 2 = 9$$

$$ANT = \frac{(0 + 1 + 2 + 10 + 6)}{5}$$

$$ANT = \frac{19}{5} = 3.8 \text{ ms}$$

$$ATT = \frac{(1 + 5 + 4 + 16 + 9)}{5}$$

$$ATT = 7 \text{ ms}$$



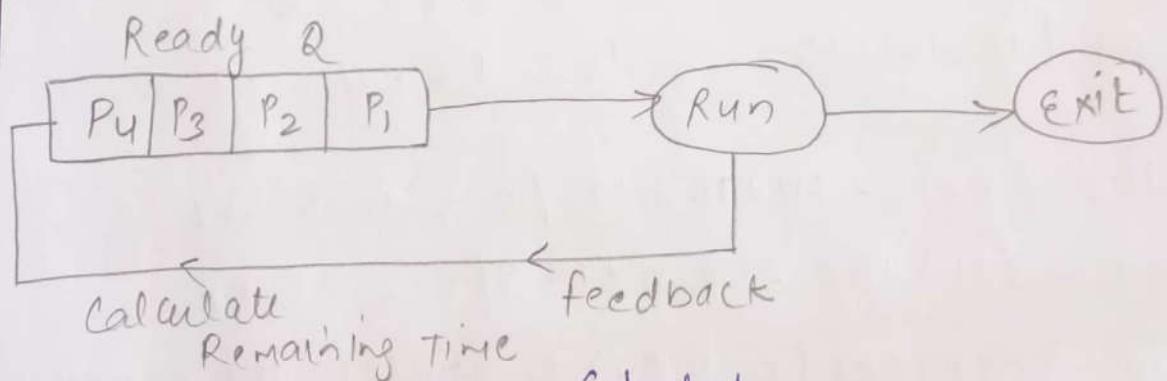
Advantage : - Better throughput

Limitation : - Knowing which incoming process is indeed shorter than another. As, there is no way to know the length of next CPU burst.

(3) Round-Robin scheduling - (Time sharing system).

↳ Preemptive, as we take ~~cut~~ off time - Time quantum (time slice)

↳ All processes are executed.



Example -

Process	BT
P ₁	3
P ₂	5
P ₃	7
P ₄	2
P ₅	4

- ① AWT & ATT
- ② Calculate throughput for 12 Unit observation time
- ③ No. of context switches excluding the last one
- ④ Remaining BT of each process just before 10th context switch



Egant chart :- (Assume, time quantum $q = 2 \text{ ms}$)

P ₁	P ₂	P ₃	P ₄	P ₅	P ₁	P ₂	P ₃	P ₅	P ₂	P ₃
0	2	4	6	8	10	11	13	15	17	18

Waiting Time :-

$$P_1 = 0 + (10 - 2) = 8$$

$$P_2 = 8 + (11 - 4) + (17 - 13) = 13$$

$$P_3 = 4 + (13 - 6) + (18 - 15) = 14$$

$$P_4 = 6$$

$$P_5 = 8 + (15 - 10) = 13$$

Turn Around Time :-

$$P_1 = 8 + 3 = 11$$

$$P_2 = 13 + 5 = 18$$

$$P_3 = 14 + 7 = 21$$

$$P_4 = 6 + 2 = 8$$

$$P_5 = 13 + 4 = 17$$

① AWT = $(8+13+14+16+13)/5 = 10.8 \text{ ms}$

$$ATT = (11+18+21+8+17)/5 = 15 \text{ ms}$$

- ② Two processes (P₁ & P₄) completed their execution within 12 Unit, 80
throughput = $2/12$

- ③ 11 Context Switches

- ④ Before 10th Context switch:-

$$P_1 = 0$$

$$P_2 = 1$$

$$P_3 = 3$$

$$P_4 = 0$$

$$P_5 = 0$$



Limitations :- Setting the length of time quantum.

- Setting it too short, causes too many context switches, & lowers the CPU efficiency.
- Setting it too long, cause poor response time

* Also, it slows down processes, bcz they have to share CPU time with other processes, rather than just finishing up quickly.

A) Priority scheduling:-

with preemption
without preemption ← of no meaning

* A factor i.e. priority (How important the process is) is assigned to each process

0 . . . n
Top priority ~~High priority~~
 Lower

* Possible Response Time according to priority.

$$\text{Response Time} = \text{Starting Time} - \text{Arrival Time}$$

Large Response Time → Poor System

Small " " → Good System

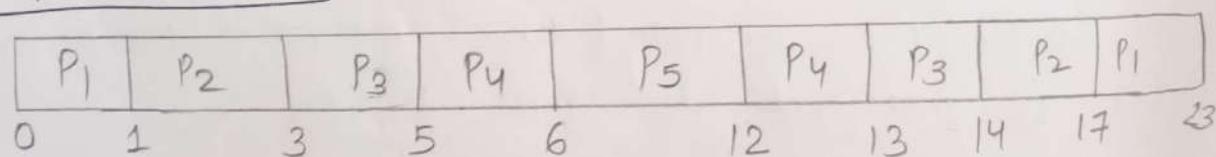


\times priority to a process is assigned based on:-

- 1) Type of user
- 2) Type of resource
- 3) Type of operations
- 4) Status of resources

Process	BT	AT	Priority
P ₁	7	0	5
P ₂	5	1	4
P ₃	3	3	3
P ₄	2	5	2
P ₅	6	6	1

Gantt chart :-



Waiting Time :-

$$P_1: - (0 + (17 - 1)) - 0 = 16$$

$$P_2: - (1 + (17 - 1) + 14 - 3) - 1 = 11$$

$$P_3: - (3 + 13 - 5) - 3 = 8$$

$$P_4: - (5 + 12 - 6) - 5 = 6$$

$$P_5: - (6 - 6) = 0$$

$$AWT = (16 + 11 + 8 + 6 + 0)/5 \\ = 8.2 \text{ ms}$$

Turn Around Time :-

$$P_1: - (23 - 0) = 23$$

$$P_2: - (17 - 1) = 16$$

$$P_3: - (14 - 3) = 11$$

$$P_4: - (13 - 5) = 8$$

$$P_5: - (12 - 6) = 6$$

$$ATT = (23 + 16 + 11 + 8 + 6)/5 \\ = 12.8 \text{ ms}$$

Limitations :- Starvation / Indefinite Blocking

Starvation :- Whenever FCFS is violated.

- Not getting resource for execution for long time
 - Reasons for not getting :-
- | | | |
|---|--------------------------------|--|
| Selection Criteria
(starvation occurs) | Resource Not free (No Problem) | In priority scheduling, higher priority processes can prevent low-priority process from ever getting CPU |
|---|--------------------------------|--|

Aging :- A Routing used for starvation.

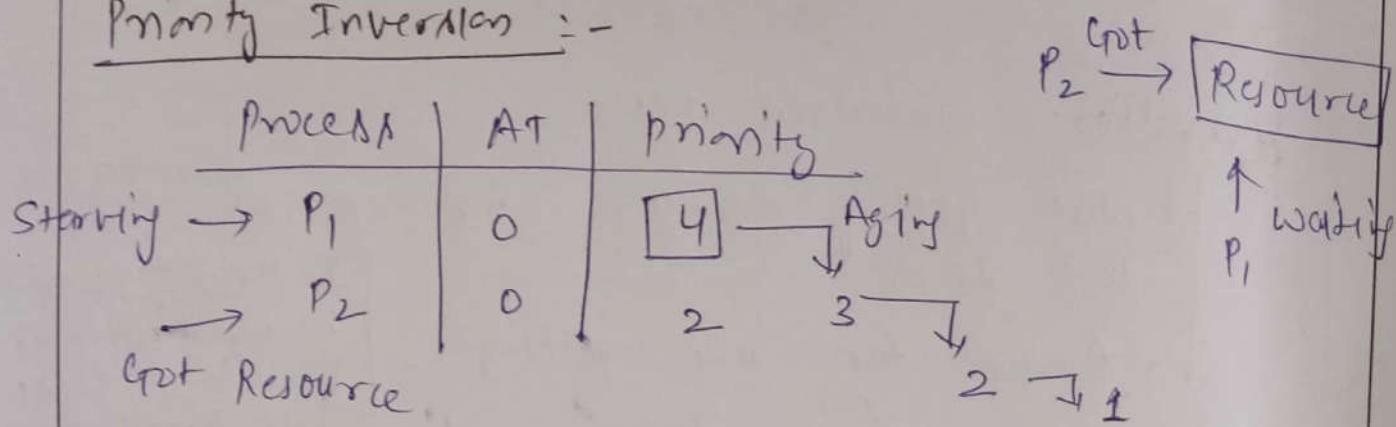
- * To make starving processes overcome starvation, lower priority processes are either redefined to a better priority at regular intervals of time.
- * Aging is a technique of gradually increasing priority of processes that wait for longer time in system

Example :-

Process	AT	Priority
P ₁	0	5 → 4 → 3
P ₂	0	2

Starving ↓
 P₁ preempt P₂ → ① ← It

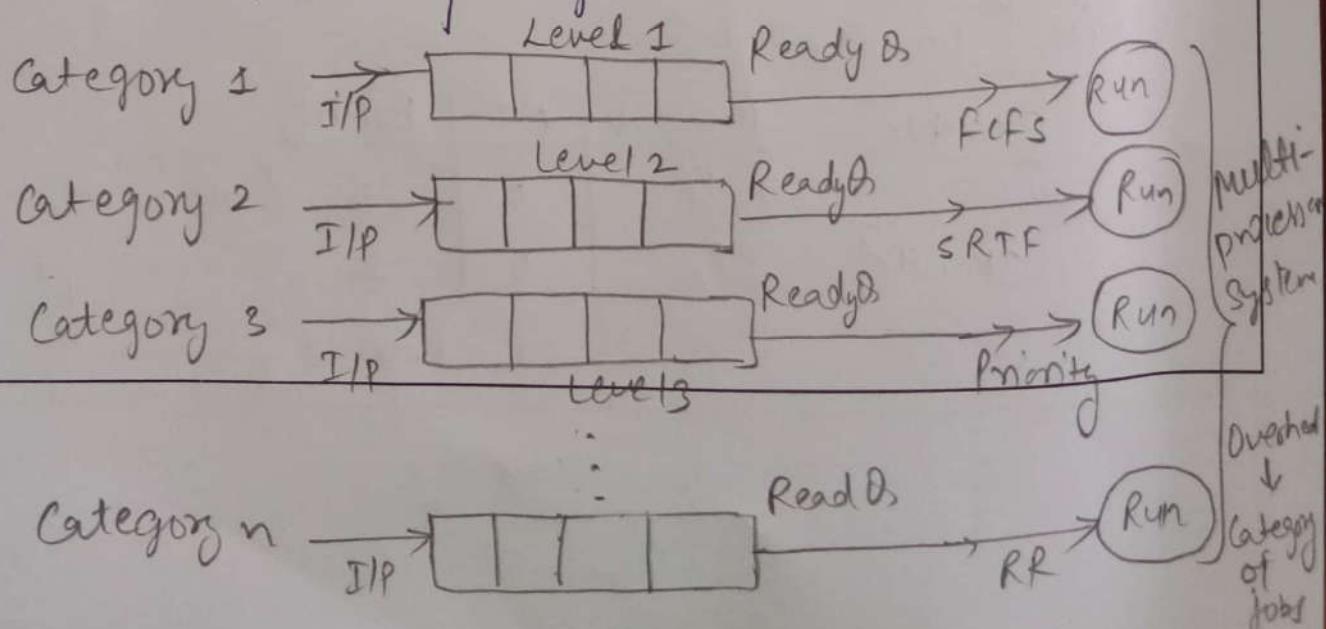
Priority Inversion :-



* When Higher priority job/process is waiting & lower priority process is acquiring the resource, it is called Priority Inversion.
(Problem in priority scheduling)

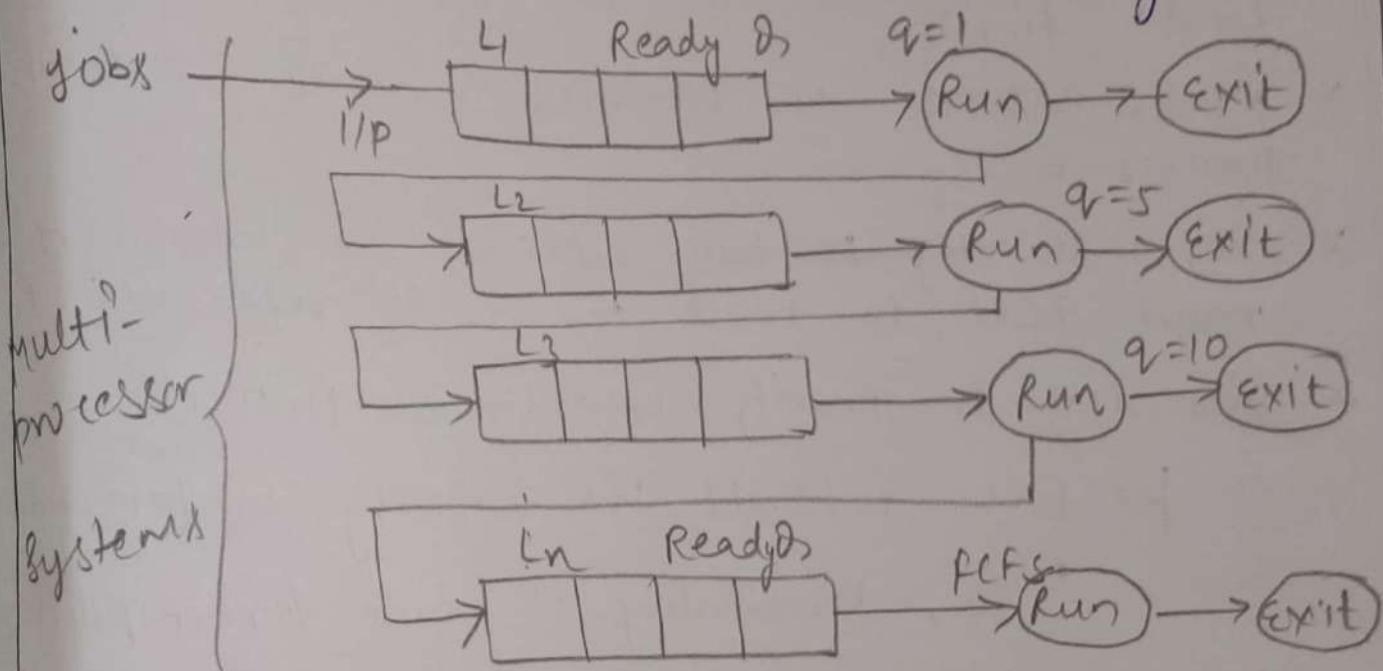
Multi-level Queue Scheduling :-

- ↳ Technique to use maximum advantages of different scheduling algorithms.
- ↳ No single queue, ready queue is partitioned into separate queues, each queue has its own scheduling algorithm.



Multi-level Queue with Scheduling →

↳ Multi-level feedback Queue scheduling



* Priority Inversion Problem

↳ Solution

(Priority - Inheritance Protocol)

Priority Inheritance Protocol:-

* All processes that are accessing resources ~~from~~ needed by a higher priority process inherit their higher priority, until they are finished with the resource.

* When finished → Priorities revert to original values.



Example :- 3 processes $\begin{matrix} \rightarrow L \\ \rightarrow M \\ \rightarrow H \end{matrix}$ } Priority
 $L < M < H$

- * Assume H requires resource R, which is being used by L, so H will have to wait for L to finish.
- * Now, assume M becomes runnable, preempting process L.
 - * So, indirectly M has affected how much H must have to wait for L to relinquish R.
- * According to Priority Inheritance Protocol:-
 - L :- inherits H's priority temporarily
 - ↑ so, preventing M from preempting L
 - When L :- finished
 - ↑ move back to original priority
 - Now R is free
 - ↑ will be Allocated to H now, not to M.



Inter-process Communication :- A procedure that allows co-operating processes to exchange data & information.

* A process can be :-

- Independent Process :- If a process can't be affected by other process. [i.e. don't share data with any process.]
- Cooperating Process :- If a process is affected by another process executing in system. [i.e. shares data with another processes.]

* As mentioned above, cooperating processes require IPC, for data & information sharing.

* 2 models of IPC are :-

Shared Memory Model - Here, a region of memory is established, which is shared by cooperating processes.

* Processes can exchange the information by reading or writing the data from / to this shared region.

* Let consider "Producer - Consumer Problem". Here, the producer produces the information that is consumed by consumer process.

* Such as:- A compiler produces assembly code, which is consumed by the assembler.



Solution:- This solution to the above mentioned problem uses shared memory.

- * It allows producer & consumer to run concurrently, by having a buffer of items, which can be filled by producer & emptied by consumer.
- * This buffer will reside in a region of shared memory b/w producer & consumer. A producer can produce one item x, while the consumer is consuming another item y.
- * Both entities must be synchronised, so that the consumer doesn't try to consume the item that has not yet been produced.

* Two types of buffers can be used for this:-

a) Unbounded Buffer:- Which does not have any limit on size of buffer.

The producer/consumer may have to wait for new items, but the producer can always produce new items.

b) Bounded Buffer:- It has a fixed size buffer.

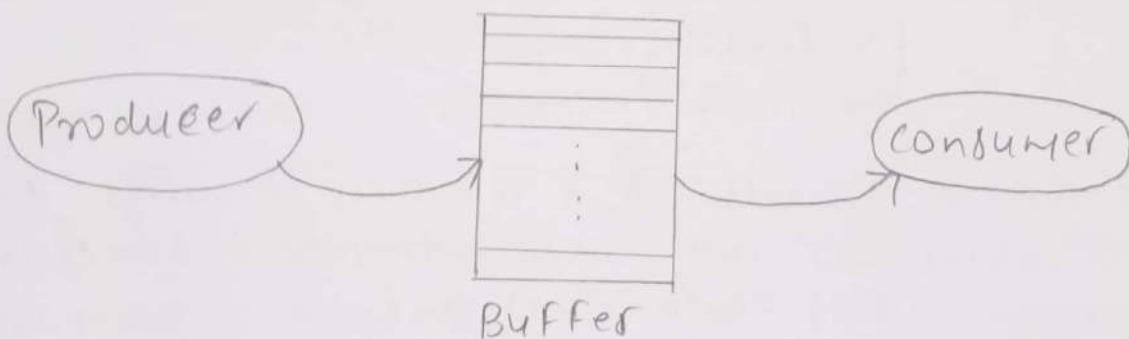
Here, the consumer must wait if buffer is empty & the producer must wait if buffer is full.

```
# define BUFFER_SIZE 10
typedef struct {
    item;
    item buffer [BUFFER_SIZE];
    int in, out = 0;
```



Producer Process using shared Memory :-

```
while (true) {  
    while ((in + 1) % BUFFER_SIZE == out) {  
        do nothing }  
    buffer[in] = next_produced;  
    in = (in+1) % BUFFER_SIZE
```



Consumer Process Using shared memory :-

```
while (true) {  
    while (in == out) {  
        do nothing }  
    next_consumed = buffer[out];  
    out = (out + 1) % BUFFER_SIZE;
```



27 Message - Passing system :- MP provides a mechanism by to allow processes to communicate & synchronize their actions without sharing the same address space.

- * This concept is used in distributed environment where communicating processes may reside on different computers, connected by a n/w.
- * This paradigm provides following 2 operations:
 - send (message)
 - receive (message)
- * If 2 processes P & Q are required to communicate, they need to send & receive msgs, from each other and a "communication link", must exist b/w them which can be implemented using following methods:-
 - Direct / Indirect communication
 - Synchronous / Asynchronous comm,
 - Automatic / Explicit Buffering

Naming:- There must be a way to refer to the processes, that want to communicate.

- * Under direct communication, each process that wants to communicate must explicitly name:
 - Sender
 - Receiver

* These primitives are defined as:-

→ send (P, message) :- Sends a message to process P.

→ receive (B, message) :- Receives a message from process B.

* A communication link is required to be established b/w every pair of processes, properties are:

- Link is established b/w every pair of processes.
- Link is associated with exactly 2 processes.
- Between each pair of processes, there exists one link.

* With Indirect communication, there exists a mailbox. The messages are sent/received to/from these mailboxes/parts.

* A mailbox can be considered as an abstract which into which msgs can be placed & from which msgs can be removed.

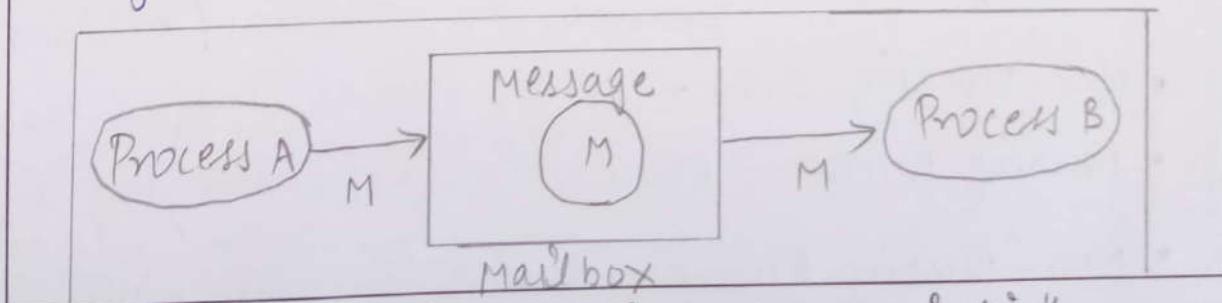


fig: "Indirect communication"



Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagar, Jagatpura, Jaipur-302017, INDIA

Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956

Tel.: +91-0141-5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in

* Each mailbox has a unique identification. Here, send() & receive() primitives are :-

- send (A, message) :- Send message to mailbox A.
- receive (A, message) :- Receive message from mailbox A.

* Communication link requires following properties:-

- Link is established b/w a pair of processes only if both processes share a mailbox.
- Link can be associated with more than 2 processes.
- B/w each pair of processes, a no. of different links may exist, each link correspondingly to one mailbox.

Synchronization :- Message passing may be either Blocking or Non-Blocking, called Synchronous / asynchronous.

- Blocking Send - Sender is blocked, until msg is received by receiver / mailbox.
- Non-Blocking Send - Sender sends msg & resumes.
- Blocking Receive - Receiver blocks until msg is available
- Non-Blocking Receive - Receiver retrieves either a valid message or null.

Buffering:- In direct/Indirect communication, message are exchanged by processes reside in a temporary queue, which can be implemented as:-

a) Zero capacity:- Queue with maximum length of zero. Link can't have message waiting in it. Sender blocks until receiver receives the msg.

b) Bounded Capacity:- Queue with finite length n , If queue is full, the sender blocks until space gets available in it. If queue is not full, sender can send & continue execution without waiting.

c) Un-Bounded Capacity:- Queue with infinite length. Sender never blocks here.

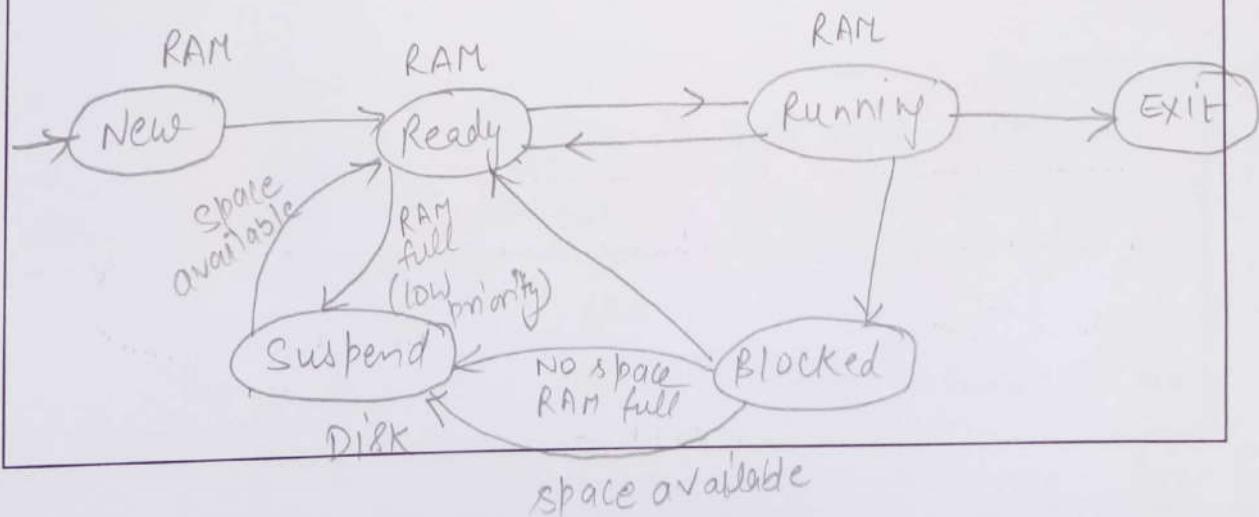


fig: " 6-state Process state model "



Process Synchronisation

- * Process Synchronisation means sharing resources by processes in such a way :-
 - Concurrent Access to shared data is handled thereby minimizing the chance of inconsistent data.
 - Maintaining Data consistency requires synchronisation among processes.
- * Processors can share variables, buffer (memory), codes, resources like :- CPU, Printer, Scanner etc.
- * Example :- "Printer" :- a shared resource, but must be always accessed in non-shareable (Mutual exclusive manner). i.e. only one process can access the resource at a time.
- * Another example :- "Train" :- Shared resource (Non-shareable Manner).

Why Process Synchronisation is Required? →

- * Consider following example:-
 - 2 processes P₁ & P₂ have access to the shared variable 'a'.
 - The order of execution of 2 processes is shown in figure:-



code:- let Initially, $a = 10$

```
P ()  
{  
    R(a)  
    a = a + 10  
    W(a)  
}
```

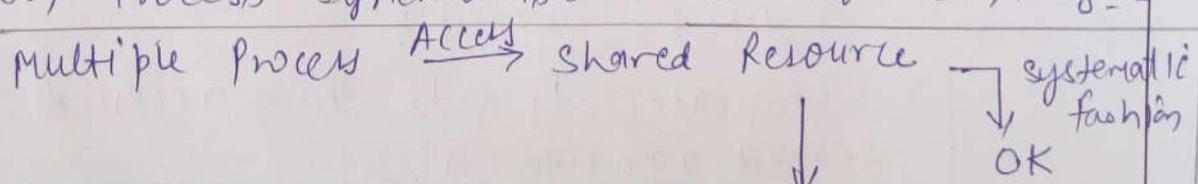
- * If P_1 & P_2 execute in a systematic manner i.e. $(P_1 \rightarrow P_2)$ or $(P_2 \rightarrow P_1)$: -
order $(P_1 \rightarrow P_2)$: - $P_1 = 11$
 $P_2 = 12$

- * But if P_1 & P_2 execute in a distributed manner (i.e. ~~they~~ Both P_1 & P_2 try to access a at same time) : -

At same time

$P_1 : R(a)$	\leftarrow preempted & P_2 is allowed to access
P_1 (After $R(a)$)	P_2 (After $w(a)$)
\downarrow	\downarrow
10	11

- * But still (after 11) P_1 read value 10;
* It causes inconsistent values as result, It is called "Race Condition", where order of execution of processes can change result.
* So, Process Synchronisation is required, B4g:-



Inconsistent Result
(Race condition) \leftarrow Distributed fashion
(i.e. at same time)



Race Condition :- When several processes access & manipulate same data concurrently and outcome of execution depends on the order in which access takes place.

Critical Section :- Consider a system with n processes.

* Each process has a segment of code (called { $P_0 \dots P_n$) critical section) in which the process may be changing common variables, updating a table, writing a file etc.

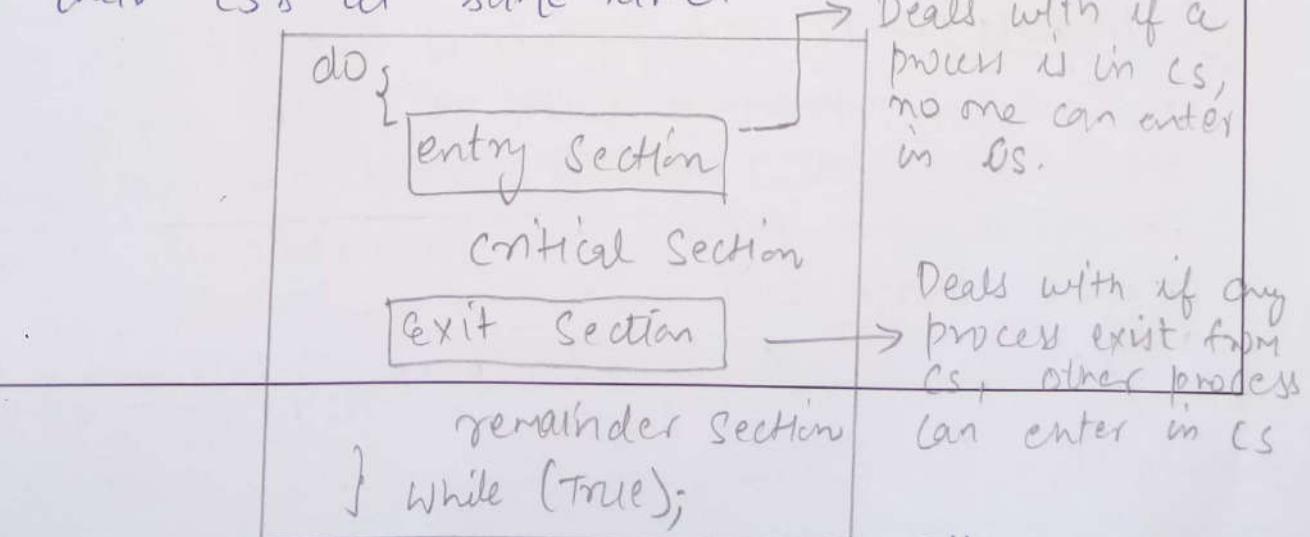
* critical section can be :-

→ A processor, file, register, printer, data item etc.

feature of cs :-

Execution of a process in CS is mutual exclusive in time

* When one process is executing in its CS, no other process is allowed to enter in its CS. i.e. no 2 processes can execute in their CS's at same time.



"Structure of a process P"



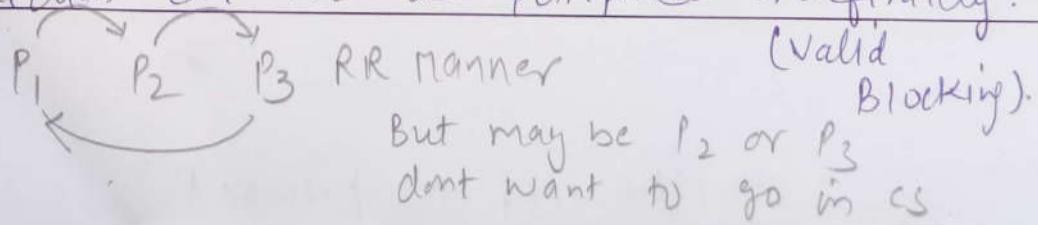
Critical Section Problem :- The critical section problem is to design a protocol that the processes can use to co-operate.

- * Each process must request permission to enter its CS. The section of code implementing this request is called "Entry Section".
- * CS is followed by an "Exit section". The remaining code is the "Remainder section".
- * A solution to CS problems must satisfy the following 3 requirements:-

(a) Mutual exclusion :- If a process P_i is executing in its CS, no other process can be allowed to enter in its CS.

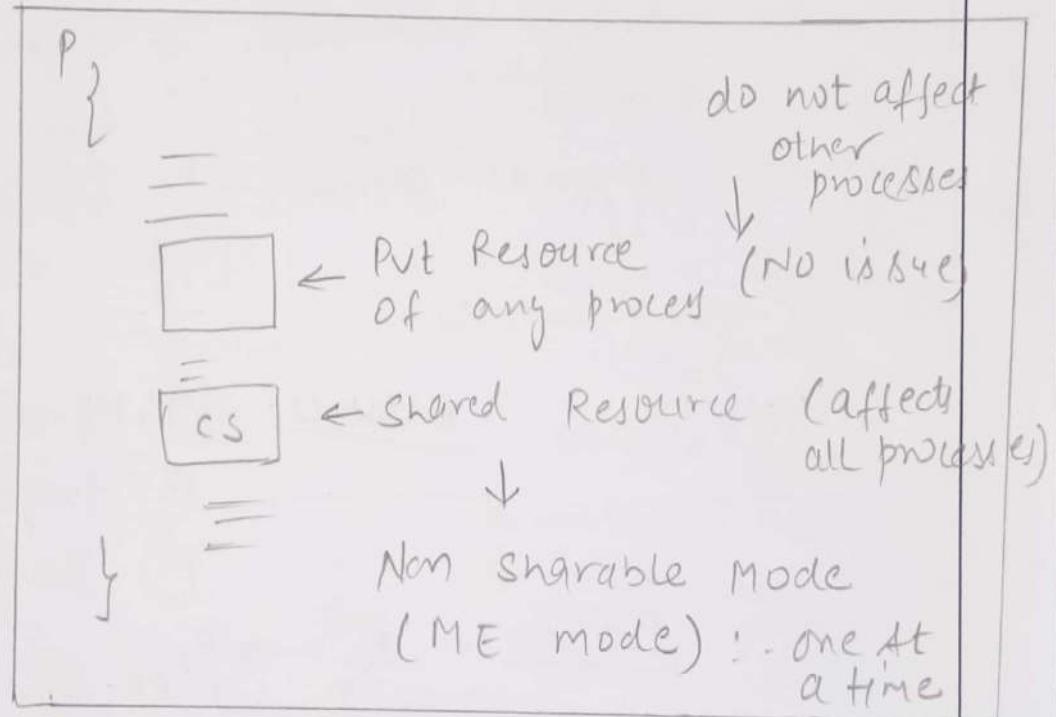
* It is a property due to which a process agrees to wait until the resource becomes free.

(b) Progress :- If no process is executing in CS, and some processes wish to enter in CS, then only those processes, which are not executing in remainder section can participate in deciding which process will enter in CS next, & this selection can not be postponed indefinitely.

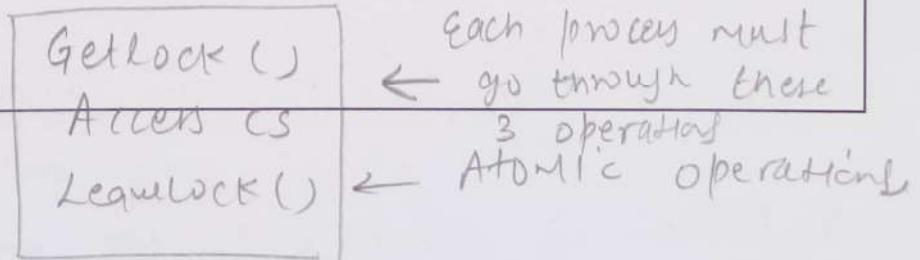


(C) Bounded Waiting:- There exists a bound/limit on the no. of times other processes are allowed to enter their CS after a request has made by another process to enter its CS & before that request is granted.
 [i.e. after a ^{max} no. of turns, process will enter in CS].

* Out of above mentioned requirements, (a) & (b) are mandatory requirements & (C) is non-mandatory.



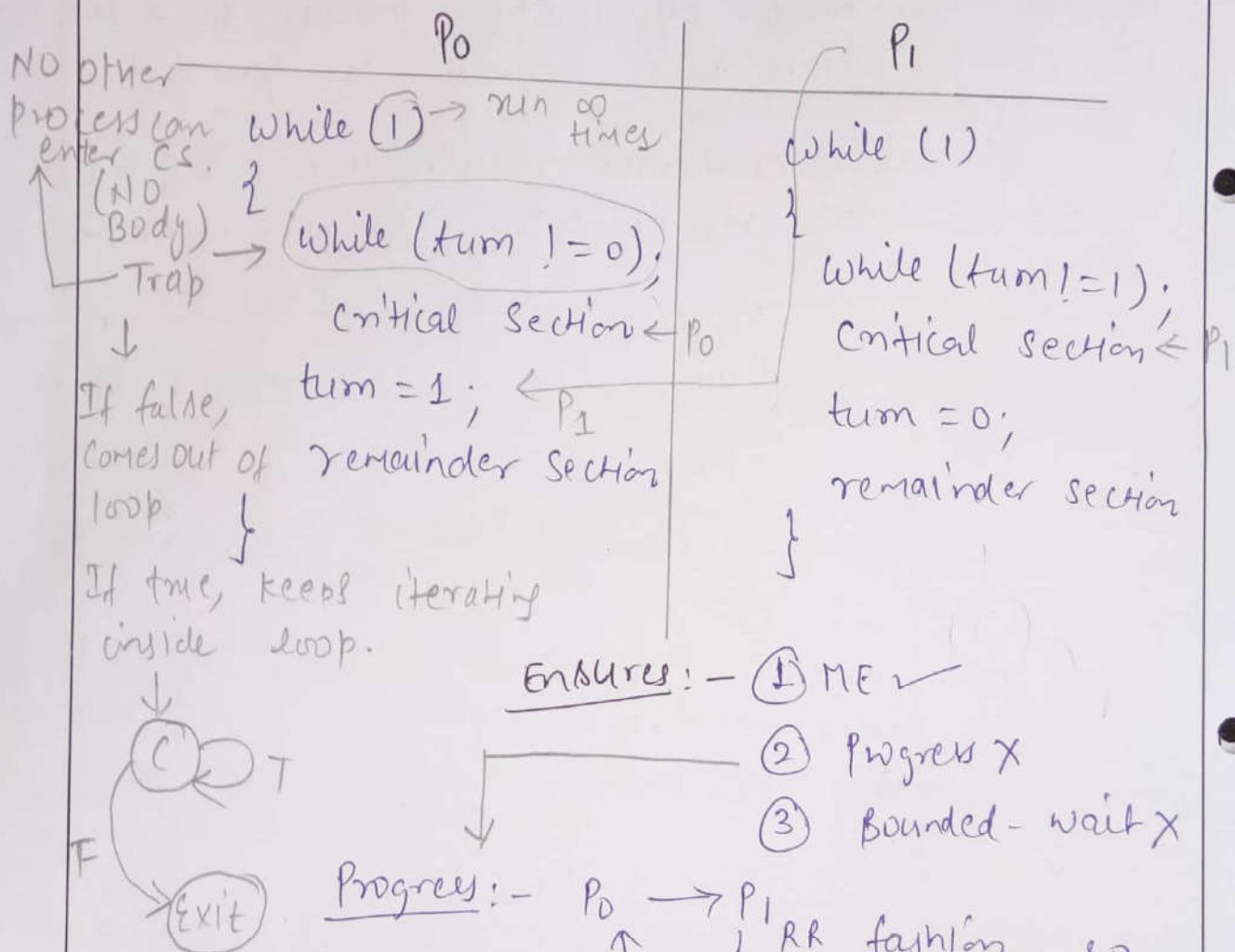
Synchronisation:- A process/Method of resolving conflicts among con-current processes.





Two- Process Solutions:-

sol² :- Initially $tum = 0$ \leftarrow Independent Boolean Variable.



- * Both processes can be context switched in CS also, but it can't enter in CS.
- * Both P_0 & P_1 , in above code, can enter in CS only once.
- * Problem in above sol²:- We never asked process if it wants to enter CS or not.



**Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagar, Jagatpura, Jaipur-302017, INDIA**

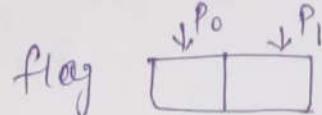
Approved by AICTE, Ministry of HRD, Government of India

Recognized by UGC under Section 2(f) of the UGC Act, 1956

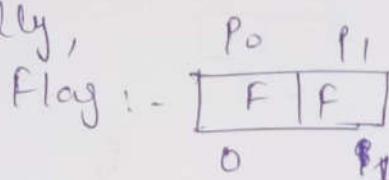
Tel. : +91-0141-5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in

Q1ⁿ 2 :- Another Variable flag.



Initially,



If Process wants to
enter CS :- ~~Flag=true(0)~~
else :- ~~Flag=false(1)~~

P ₀	P ₁
while (1)	while (1)
{	{
flag[0] = T ;	flag[1] = T ;
while (flag[0]);	while (flag[1]);
critical section	critical section
flag[0] = F ;	flag[1] = F ;
}	}

Ensures :-

(1) ME ✓

(2) Progress :- If P₁ dont want to enter in CS, then
X P₀ can take any no. of rounds in CS

But, flag : $\begin{array}{ c c } \hline T & T \\ \hline \end{array}$	$\begin{array}{ c c } \hline \text{preempt } P_0 & \text{preempt } P_1 \\ \hline \end{array}$	$\begin{array}{ c c } \hline \text{flag[1] = T} & \text{flag[1] = T} \\ \hline \text{preempt } P_1 & \text{preempt } P_0 \\ \hline \end{array}$	only P ₀ is competing for CS.
System in deadlock. bcz, CS available, but no process allowed to enter	flag $\begin{array}{ c c } \hline T & T \\ \hline \end{array}$	So, $\begin{array}{ c c } \hline \text{No Progress} \\ \hline \end{array}$	



Sol 3 :- Peterson's Solution :- used both

Initially

turn = 0/1

turn $\swarrow \rightarrow$ flag

flag =

F	F
0	1

P₀

P₁

while (1)

{

flag [0] = T;

turn = 1;

while (turn == 1 &

flag [1] == T);

critical section

flag [0] = F;

}

while (1)

{

flag [1] = T;

turn = 0;

while (turn == 0 &

flag [0] == T);

critical section

flag [1] = F;

}

① ME ✓

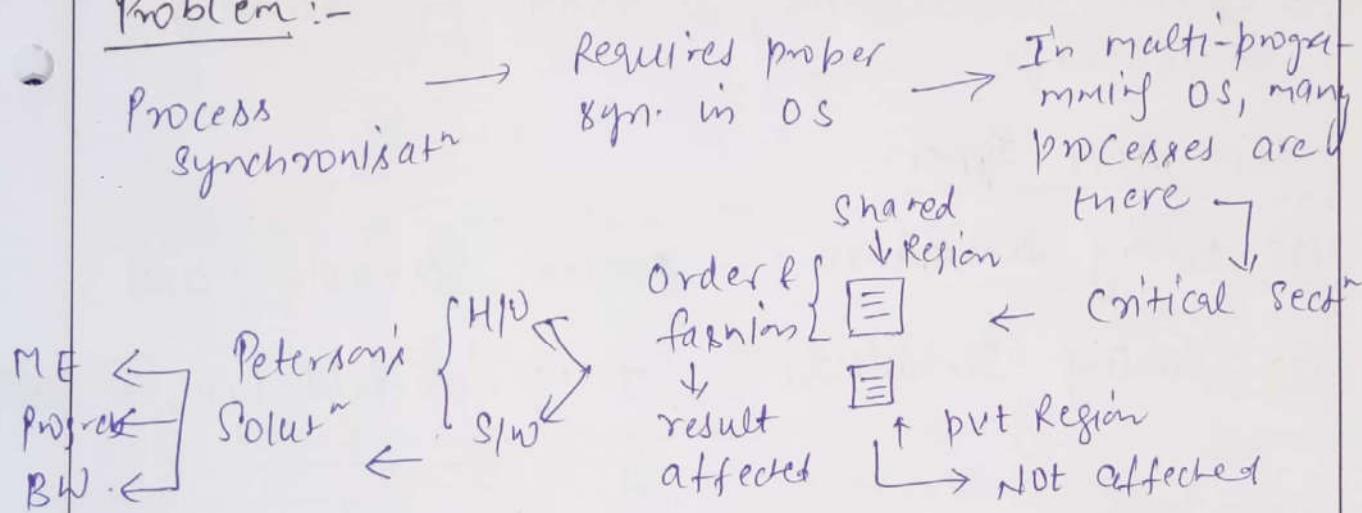
② Progress ✓ :- No progress, if P₁ dont want to enter in CS, P₀ can again enter in CS.

③ Bounded Wait :- P₀ can't enter 2nd time in CS, if already requested by P₁.



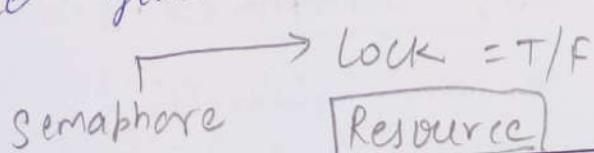
Semaphores :- Solution of CS problem are not easy to generalize to more complex problems. To overcome this difficulty, we can use a synchronisation tool called "Semaphores".

Problem :-



* But Peterson's solution is only a process solution. In OS, in an n-process system, synchronisation is required among all processes. So, Semaphores are used, which gives solution for n-process system.

* Semaphore is a special variable, if variable value gives the lock status information.



$$L = F/T$$

Printer



Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagar, Jagatpura, Jaipur-302017, INDIA

Approved by AICTE, Ministry of HRD, Government of India

Recognized by UGC under Section 2(f) of the UGC Act, 1956

Tel.: +91-0141-5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in

* Semaphore is an integer variable(s) that apart from initialization, can be accessed only through 2 standard operations:-

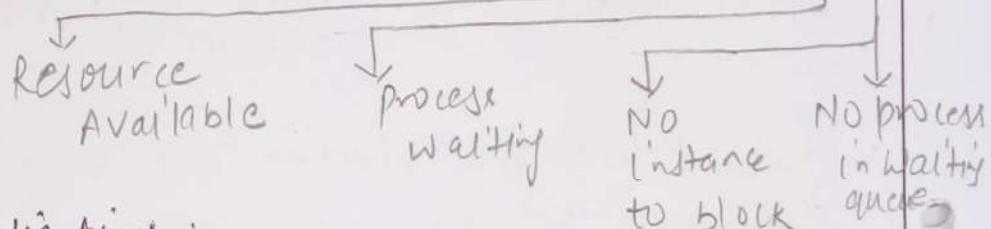
→ Wait() & → signal()

* After initialization, s can't be directly accessed.

Semaphore Types:-

(1) Binary Semaphore :- Boolean Variable (0,1)

(2) Counting Semaphore :- Integer Variable (+ve, -ve, 0)



Semaphore Applications:-

(1) Solution of critical section problem

(2) Deciding order of execution among processes

(3) Resource Right

Solving CS Problem using Semaphores:-

* For solving CS problem, initialize s with 1.

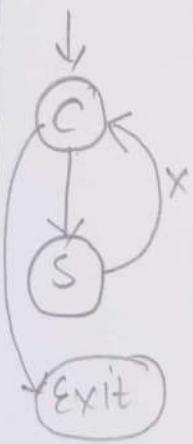
$$\boxed{\text{int } s = 1}$$



* 2 operations of semaphores are :-

- wait(s) :- Atomic operation, reduces s by 1, i.e. $(s--)$
- signal(s) :- Atomic operation, increases s by 1, i.e. $(s++)$

* Above 2 operations have following code:-



Wait (s)	Signal (s)
{ while($s \leq 0$); $s = s - 1$; }	{ $s = s + 1$; }

wait(s) → getLock()
cs → Access Resource
signal(s) → Leave lock()

Wait() :-

- Always placed before cs.
- A slow logic, must be executed.
- Acquires lock for current process.
- Operation unsuccessful - Block current process.
- Decrements semaphore by 1.
- Also called P(semaphore)



Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagar, Jagatpura, Jaipur-302017, INDIA

Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956

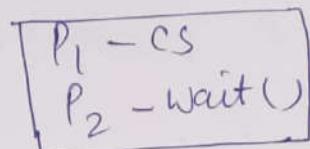
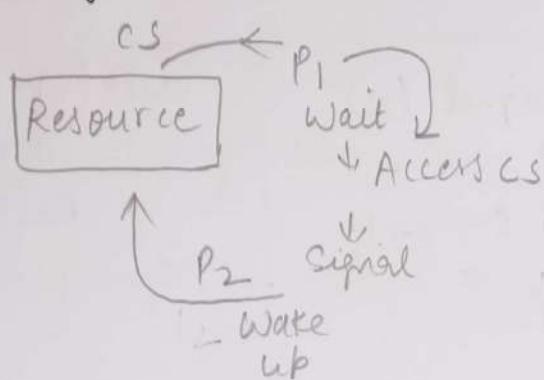
Tel.: +91-0141-5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in

Signal (s) :-

- used after CS Access.
- Releases acquired lock.
- A SW logic (step of process).
- increments value by 1.
- Also called \vee (Semaphore).

Binary Semaphore :-



Semaphore Code :-

Initially, $S = 1$

Above code:

- ① Ensures ME
- ② Progress achieved automatically, bcz wait() & signal() are executed by process.
- ③ No Bounded wait

```
do {  
    wait (s);  
    // Critical Section  
    signal (s);  
    // Remainder Section  
} while(T)
```

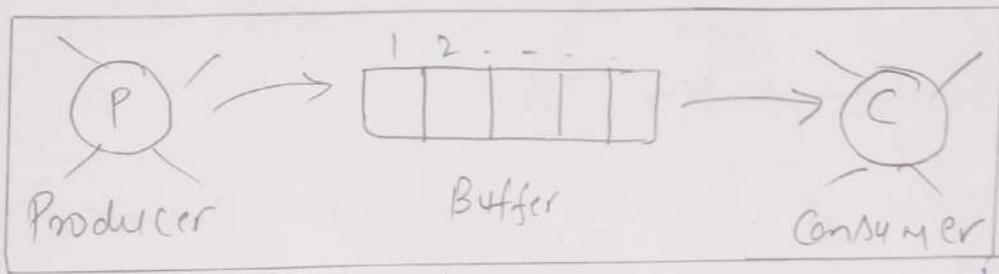
Ensured :- $P_1 \geq S = X \otimes X \otimes X \dots$



Classical Problems of Synchronisation :-

① Producer - Consumer Problem / Bounded Buffer Problem :-

* 2 Processes → Producer
 ↓
 Consumer

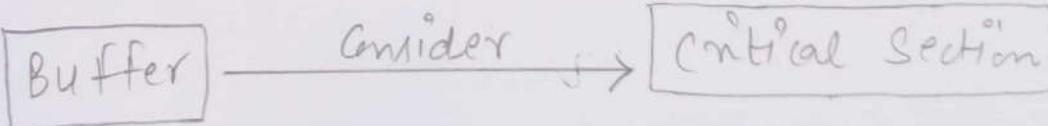


* Between above 2 processes, there must be coordination like that:-

→ Producer must check for empty slots in buffer before proceeding (Overflow).

→ Producer & Consumer should not perform their actions at same time, otherwise they may clash.

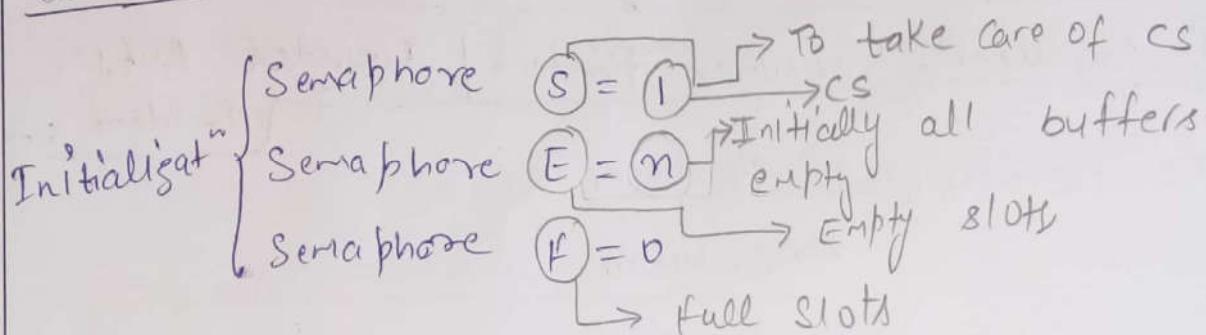
→ Consumer must check for underflow.



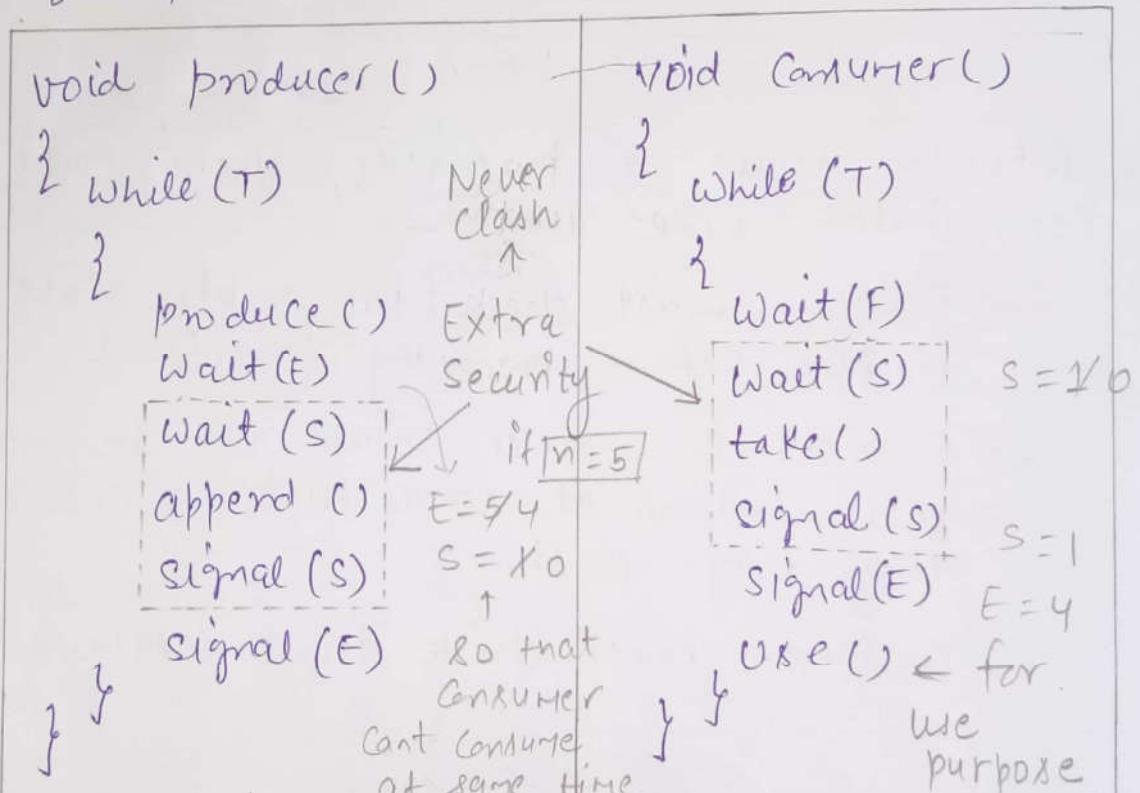
↑ only 1 process must work at a time
(ME fashion)



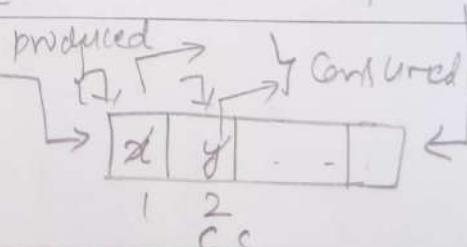
Solution: - Use 3 Semaphores: -



* Size of Buffer: - n



Producing & Putting 1 element in buffer



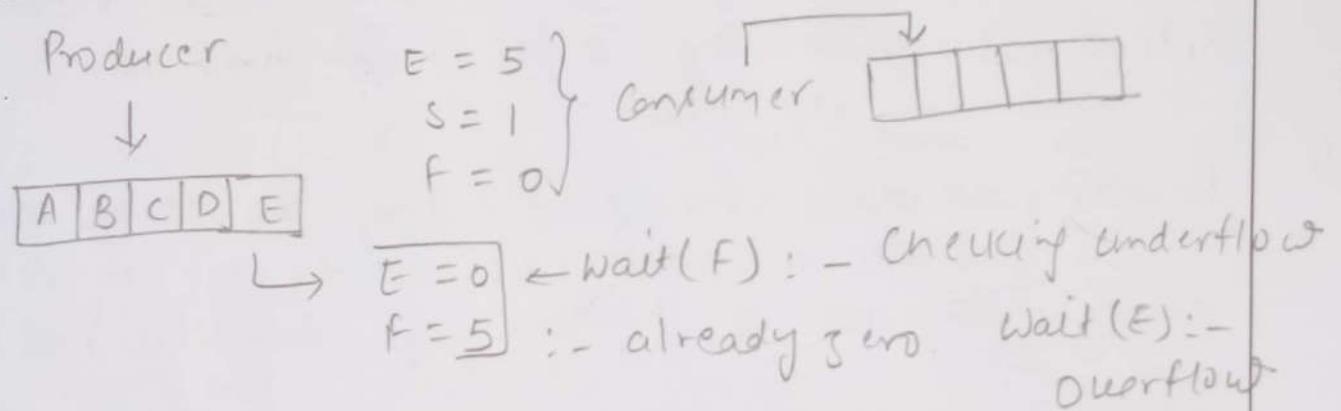
- Empty slots: - decreases by 1.
- full slots: - increases by 1.

$$\begin{aligned}
 E &= 843 \\
 F &= 0 \neq 2 \\
 S &= 0 \times 01
 \end{aligned}$$



- * If buffer would have single cell, we would require strict alteration (i.e. Round Robin fashion)
 - * semaphore S is acting as a security for CS.

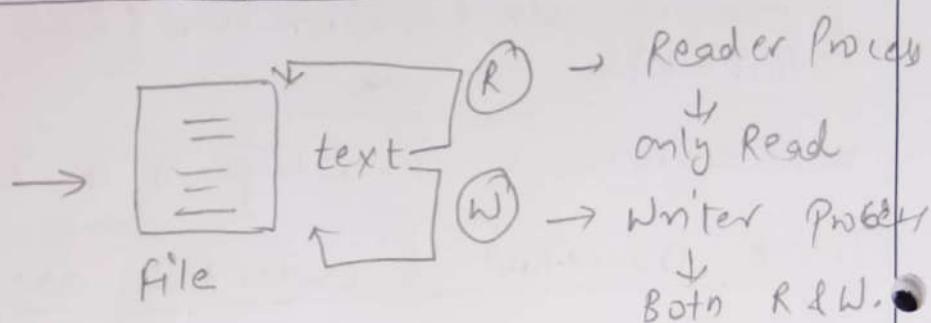
Check Overflow & Underflow as:-





② Reader - Writer Problem :-

Considered
as CS



* Read operations by 2 or more than 2 processes :- No clash:

Reader :- No CS for Reader bcoz no clashes can occur in read operations, no inconsistency. Any no. of processes can perform read.
→ R+R :- No clash

Writer :- No reader/writer can execute if writer process is in CS.

- R+W :- clash
- W+W :- clash

Solve Ax :-

- for Reader :- CS can be such that more than 1 Reader can enter & but no writer can enter.
- for Writer :- CS must be like:- only 1 writer can be there, no Reader.

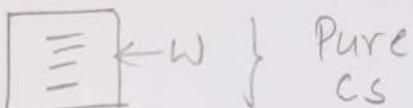
Note :- → If a Reader is inside CS, don't allow any writer.
→ If a writer is inside CS, don't allow any process.



**Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagar, Jagatpura, Jaipur-302017, INDIA**
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Solution :- \rightarrow 2 Semaphores $\left\{ \begin{array}{l} \rightarrow \text{Wrt} \\ \rightarrow \text{mutex (To agm.)} \end{array} \right.$
 Input : \rightarrow 1 Variable - readcount
 (rc) b/w & multiple users

for Winter :-



Initialize wrt = $\neq \phi 1$ - ..

wait (wrt)
write operation
signal (wrt)

No other writer will be able to go in CS, bcz it will be in trap at $wrt = 0$

for Reader :-

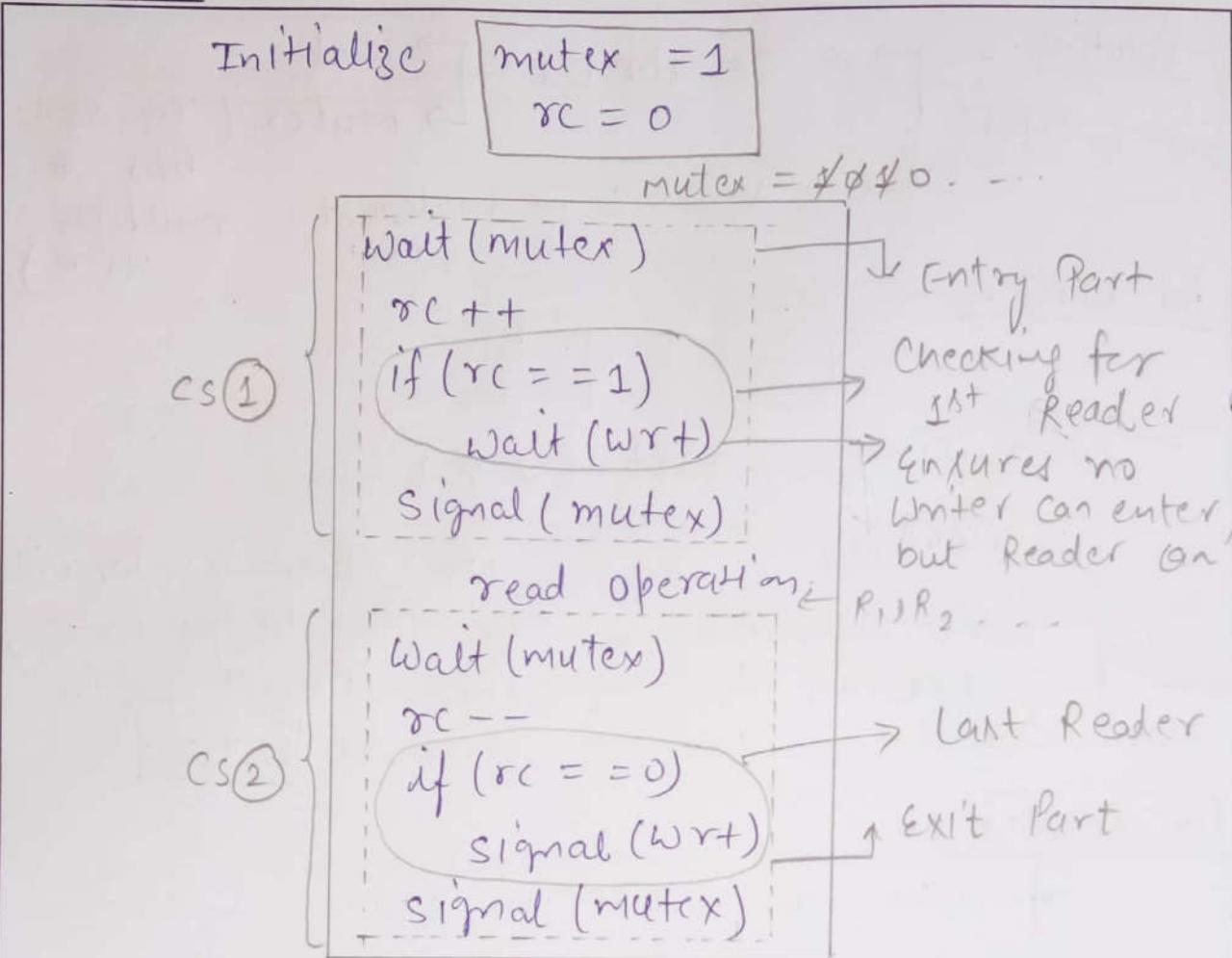
- When no writer is inside
 - Check if some wants to enter
 - Reader → allow
 - Writer → may cause clash

* Another Semaphore mutex : - Synchronise b/w different readers.

* **Readcount (rc)** :- will take care of how many readers are in CS.

↓
Shared Variable

→ No more than 1 process is allowed to update it, else race conditions can occur.



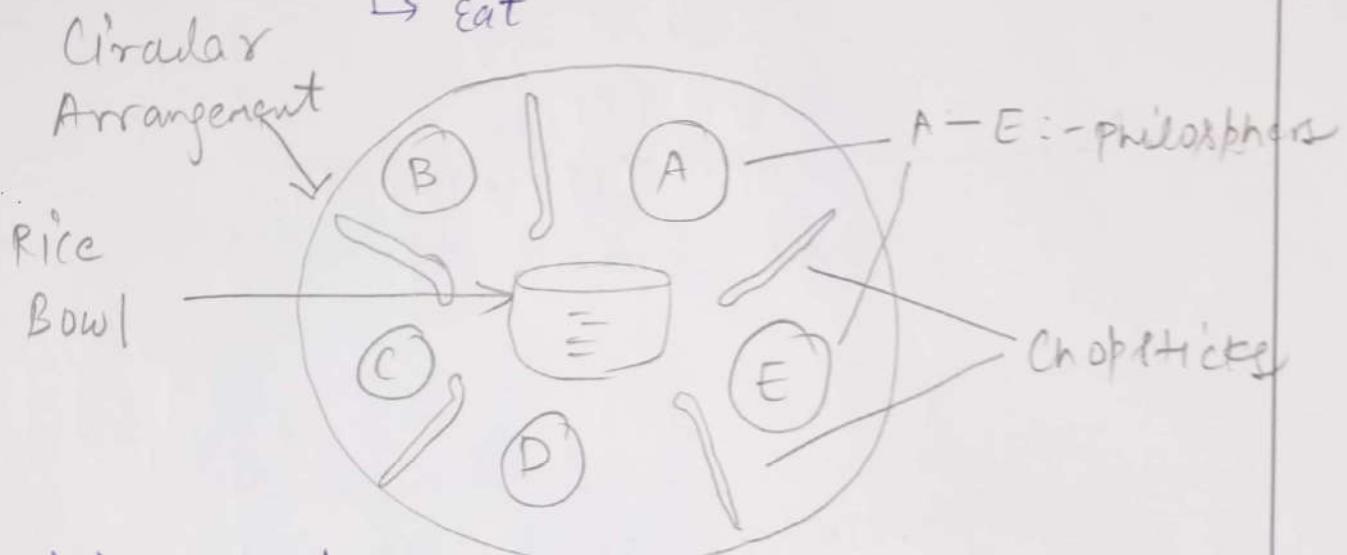
- * In entry, first Reader must take care of no writer is allowed to enter CS.
- * Similarly, In exit, last Reader must also do signal(wrt), so that another writer can enter.

③ Dining Philosopher Problem :-

Problem :- circular Arrangement (Assume 5 Person)

* only performs 2 tasks:-

- Think
- Eat



Critical Section Possibilities:-

→ Bowl X

→ Chopstick : 5 (5 CS)

* everyone needs 2 chopsticks:-

→ One chopstick is required to be used by single person only at a time

→ When a person releases the chopstick, it is now free to use for others.

→ Left & right chopsticks are used by a person.



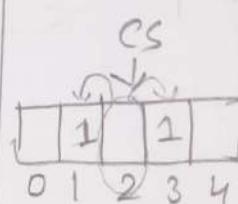
Swami Keshvanand Institute of Technology, Management & Gramothan,
Rammagaria, Jagatpura, Jaipur-302017, INDIA

Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956

Tel. : +91-0141- 5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in

```
Semaphore      chopstick [5];
do                  ↳ c[5]
{
    wait (c[i]);
    wait (c[(i+1)%5]);
    /* EAT */
    signal (c[i]);
    signal (c[(i+1)%5]);
    /* THINK */
    —
}
white (T);
```



"Structure of Philosopher i."

* Above solution guarantees:-

→ No 2 neighbors are eating simultaneously, bcz it creates deadlock.

* Some possible soln of deadlock problem :-

→ Allow at most 4 philosophers to be seated.

→ Allow a philosopher to pick up chopsticks only if both chopsticks are available.

UNIT - II



**Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagar, Jagatpura, Jaipur-302017, INDIA**
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Memory Management

- * Computer, as a wider perspective:-

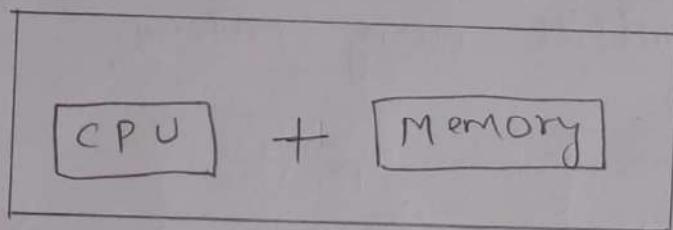


fig: "computer"

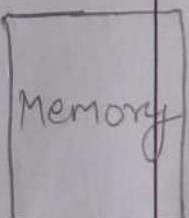
- * What a computer ~~does~~ can or can't do is completely depends on following 2 factors:-

→ How strong is CPU, - logical / influencing engine
→ How large is memory.

- * This time will be discussing on memory.

Ideal criterias for Memory :-

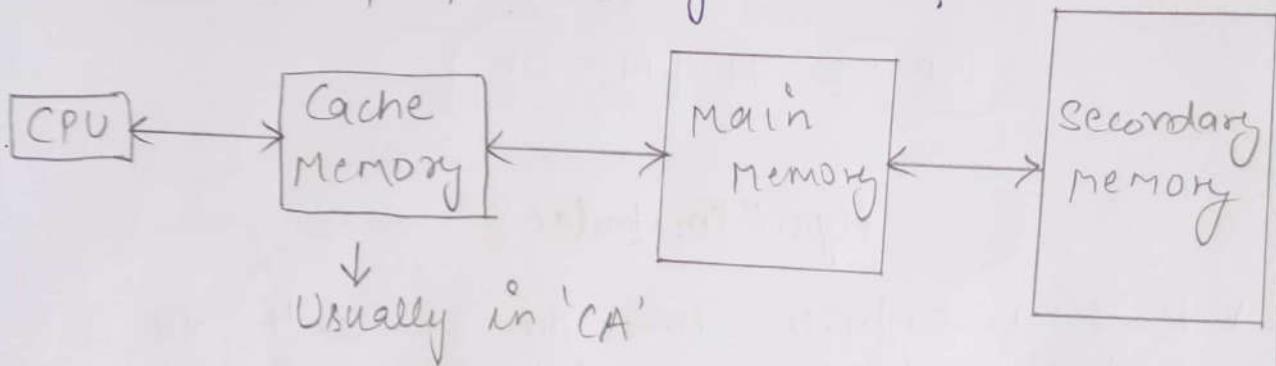
- * Size (must be larger)
- * Access Time (must be less/small)
- * Per Unit Unit (must be less)



- * But all above criterias together are not practically possible, because these are contradictory.



- * So, we don't have single memory, we are having a hierarchy of memories.
- * This is having different memories, bcz we want to fulfill every criteria.



Locality of Reference :- A tendency of processor to access same set of memory locations repetitively over a shorter period of time.
(Also called "Principal of Locality").

- * A program executes sequentially

Default \rightarrow Sequential Nature, but can be violated bcz conditional, jump, go to statements can be there.

Program

I_1

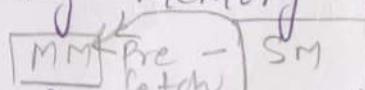
I_{200}

I_{1000}

due to sequential nature, if I_{200} is getting executed, then are maximum probability that next instruction to be executed will be I_{201} .

- * Most of the times, data will be found in Main Memory \rightarrow Locality of Reference



- * Like: - We don't watch movie randomly.
- * So, we can prefetch those instructions in some small memory.
- * Advantage of Secondary Memory :-
 - Very Big size
 - Less per Unit cost
- * Problem with Secondary Memory
 - Larger Access Time
- * So, we can associate one more memory, comparatively smaller than Secondary memory (called Main Memory).
 - 
- * The idea is to pre-fetch the instructions/info main memory from secondary memory which are currently required by the CPU. (or important data).
- * Example: - In laptops today:-
 - Main memory (RAM) :- Smaller (4GB).
 - Secondary :- 1TB & so on.
- * Advantage of this scheme will be:
 - Less Access Time of main memory
 - ↑ u will try to access data, it will be available (max time) here, As Locality of Reference.



**Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagar, Jagatpura, Jaipur-302017, INDIA**

Approved by AICTE, Ministry of HRD, Government of India

Recognized by UGC under Section 2(f) of the UGC Act, 1956

Tel. : +91-0141- 5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in

* Hit Ratio :- 90 out of 100 times, we will find data in main memory.

Example:- In market, u can have shop.

↳ And for more items / collection, u can have a godown.

↳ If item is not available in shop, it can be taken from godown.

* So, will be having:-

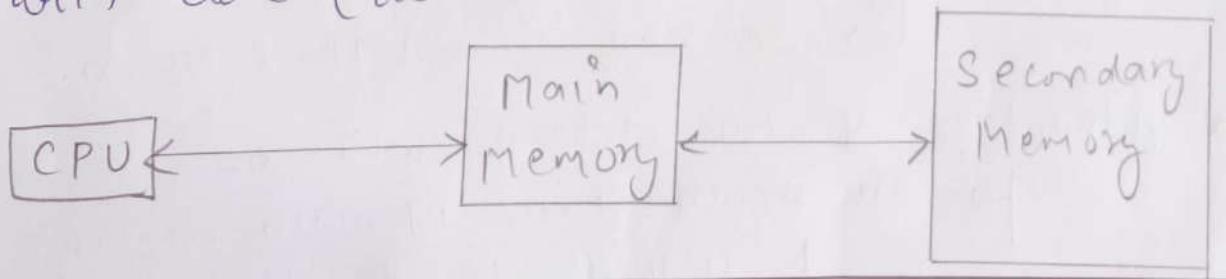
↳ Less Access time

↳ Less cost

↳ Larger size

* Similarly, this logic can also be extended further i.e. currently executing data of main memory can be brought into cache memory (which is also multi-level).

* In OS, we deal with main memory only, not with cache (dealt in CA). So:-





Example :-

* Secondary Memory Access Time = 100 ms

* Main " " " " = 10 ms

* Hit Ratio = 90% (i.e. 90% times data will be found in main memory only).

Average
Memory Access Time = $0.9(10) + 0.1(10+100)$
 $= 9.0 + 0.1(110) = \boxed{20 \text{ ms}}$

* Both criteria are fulfilled now:-

- small memory
- fast access
- In backup - Secondary Memory

2 Important Questions →

(1) How process comes from Secondary to Main memory, i.e. [Memory Allocation : Contiguous & Non-Contiguous Memory Allocation].

(2) Address Translation.

→ CPU generates - Logical Address
↓
Works on Secondary Memory

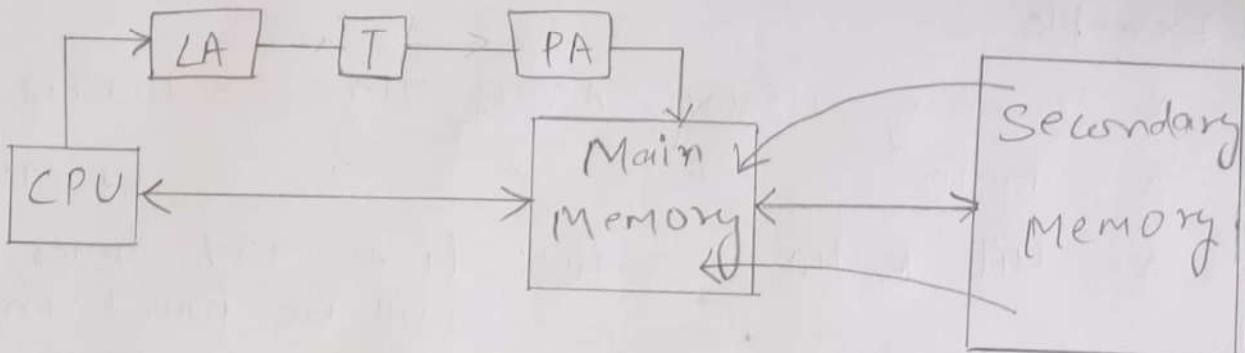
→ Main Memory - Physical Address

Second Duty of O.S.

Logical Add. → Physical Translatⁿ Add.



**Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagar, Jagatpura, Jaipur-302017, INDIA**
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in





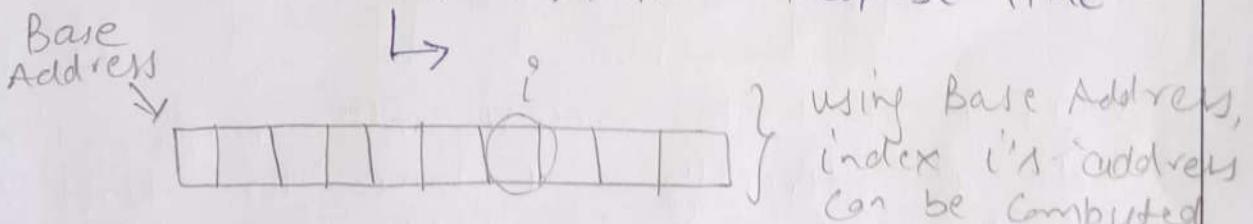
Memory Allocation Policies :-

① Contiguous Allocation:-

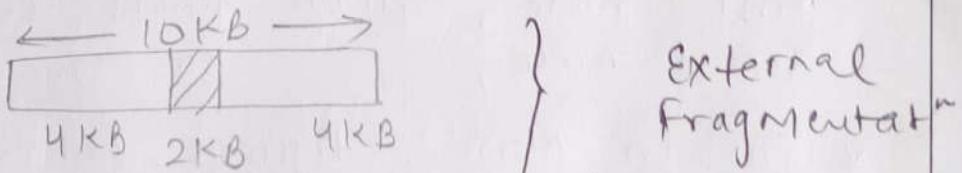
- Data that will be pre-fetched from secondary memory to main memory will be stored together / continuous / contiguous fashion.
- * In starting (1970-80's), this scheme was used, but due to external fragmentation, Non-contiguous fashion is used mostly.

Example:- Consider an array (Space is allocated continuously)

↳ Advantages: → faster Response Time



Example:-



$P_1 \rightarrow 5\text{ KB}$ (Request) Can't be Allocated, bcz

We don't have 5KB space available continuously, Although we have 8KB available.

* So, P_1 's request can't be granted & it

leads to "External Fragmentation"

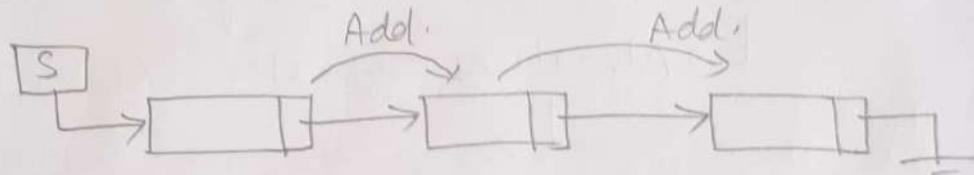
* Though space available, but in discrete, not in continuous fashion.

Advantage :- Address Translation is easy, because only base address is required, & access will be very easy.

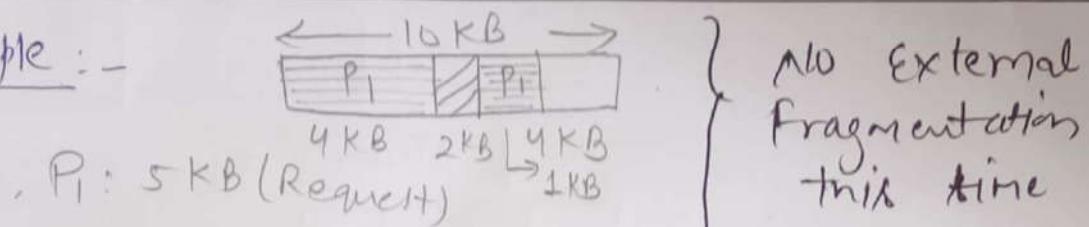
Disadvantage :- External fragmentation

② Non-Contiguous Allocation : - Processes can be divided into multiple parts, and then these parts/pieces can be arranged separately in main memory.

Example : - Consider a Linked List



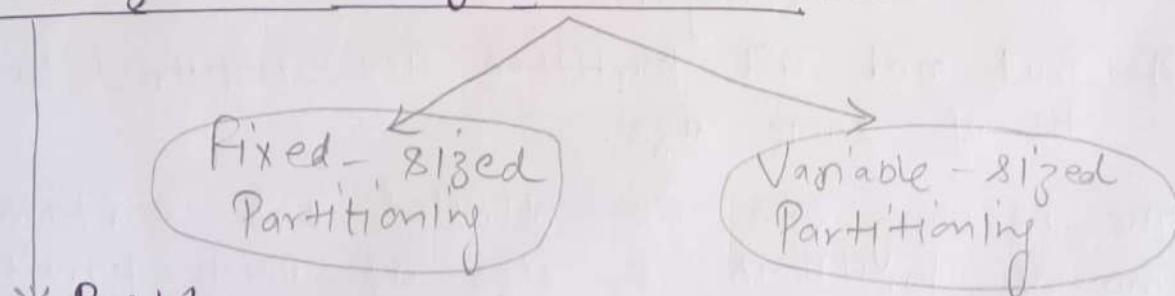
Example :-



Advantage :- No external fragmentation

Disadvantage :- Access will be very slow. (If I want to access last part of process, entire process needs to be accessed). [One by one in a linear fashion].

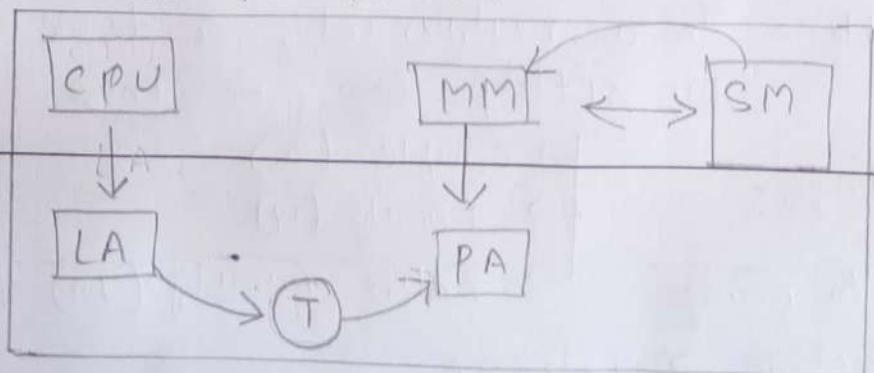
Contiguous Memory Allocation :-

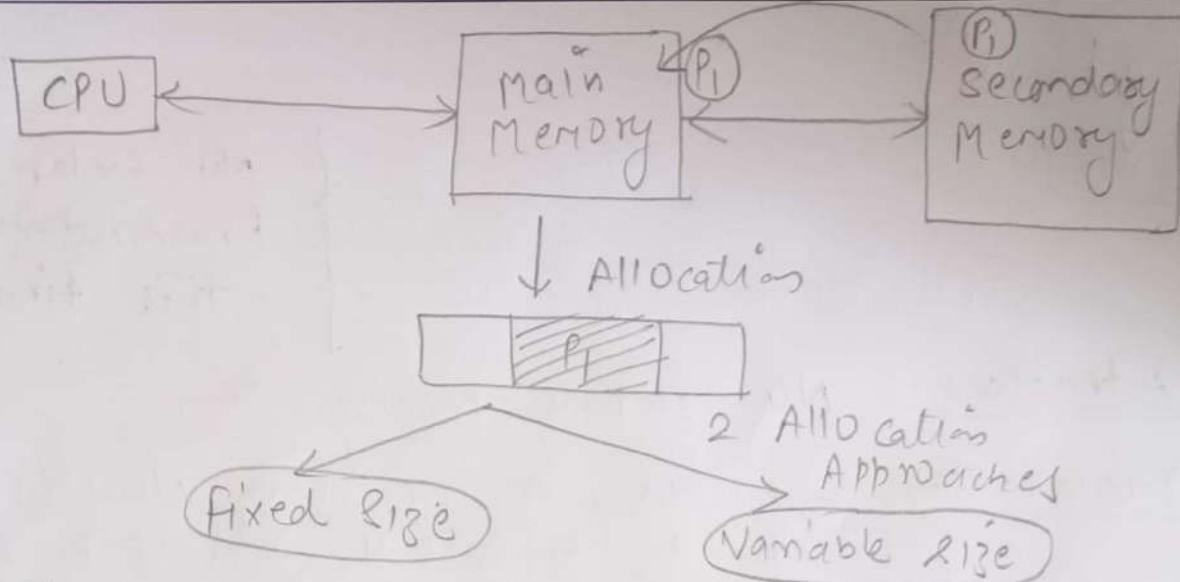


↓ Problems

- ① → space allocation → How a process will be stored in contiguous memory i.e. what are the Algo for this policy

- ② → Address Translation





Q) Fixed- Sized Partitioning -

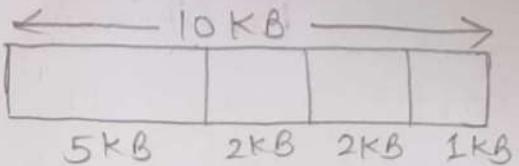
- i) → Size of partitions will be constant, i.e. once defined size of any partition, it can't be changed.
- ii) → But not all partitions are required to be of same size.
- * The memory can be divided into certain no. of partitions & size of each partition is fixed & can't be changed.
- * Both above (i) & (ii) factors depends on system i.e. may differ from system to system. (on Administrator / System Designer).

example:- In a restaurant, a table with following seating can be these :-

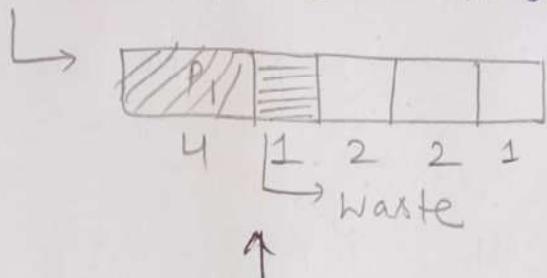
<u>Advantage</u>	<ul style="list-style-type: none"> → Couple (2) → family (4) → Entire family (10) 	But these can't be changed
<u>Easy to manage</u>		

Disadvantage → Internal

Example -



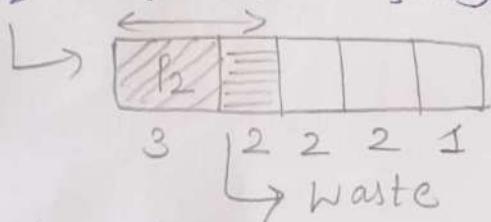
- If P_1 requests - 4 KB



i.e. entire 5 KB will be allocated, it can't be partitioned, so 1 KB chunk will be wasted.

"Internal fragmentation"

- If P_2 requests - 3 KB



Problems with fixed sized Partitioning:-

↳ Partition can not be re-used (i.e. only one process)

↳ Internal fragmentation

↳ Example → [Notebook] → Internal fragmentation

Advantage: Easy to Manage



Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagar, Jagatpura, Jaipur-302017, INDIA

Approved by AICTE, Ministry of HRD, Government of India

Recognized by UGC under Section 2(f) of the UGC Act, 1956

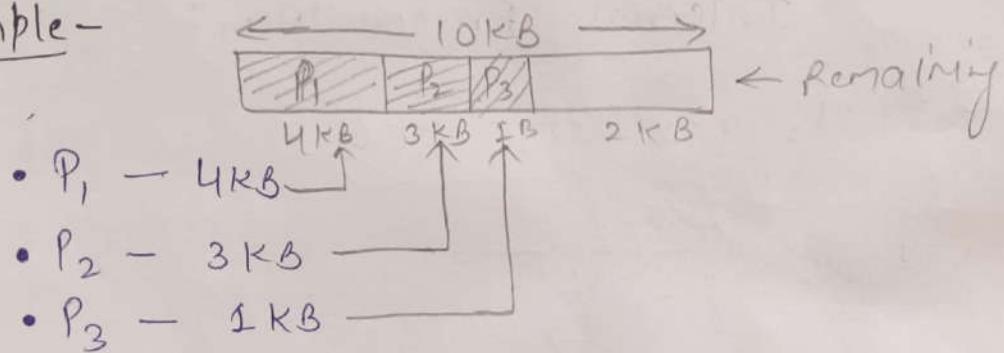
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in

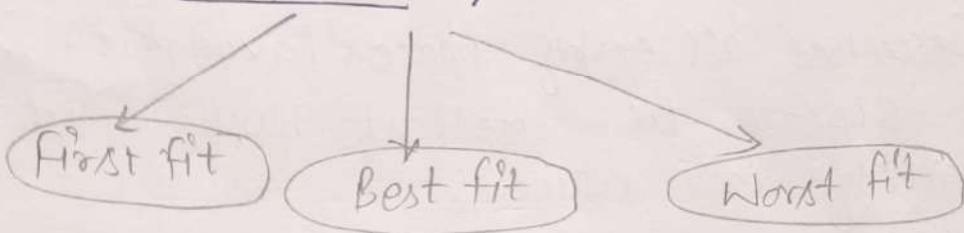
b) Variable - sized Partitioning -

- No partition at all
- As process comes, space is allocated to it according the requirement.
- Entire we don't have predefined partitions, entire memory is treated as a single chunk of memory, & can be reused.
- No chance of Internal fragmentation.

Example-



Space Allocation Policies:-



- * All above 3 space allocation policies work for both fixed size & variable size partitioning.

Variable Sized Partitioning:- Things to remember:-

- Occupy the only space which is required.
- Remaining space can be reused.

@ First fit :-

- * Start searching space from base address / initial address ~~and~~
- * Use first partition/ block which is capable enough for the process. [which can accommodate the process].

Example -

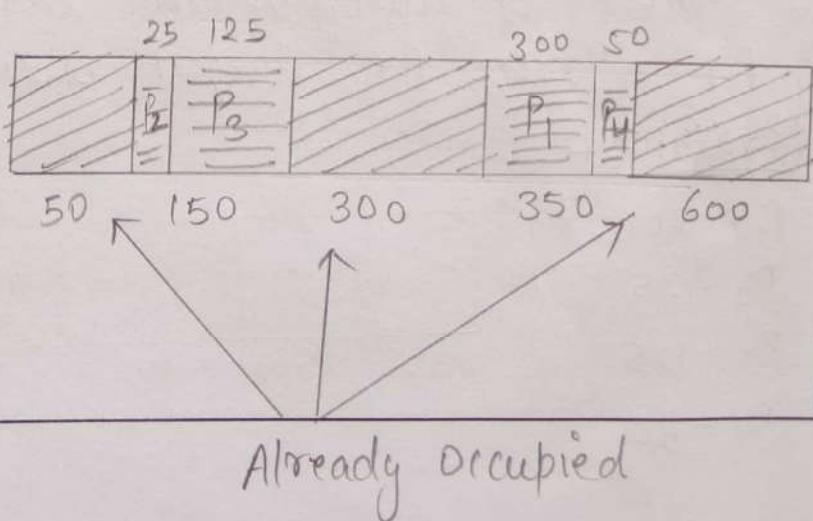
Requests :-

P₁ - 300

P₂ - 25

P₃ - 125

P₄ - 50



(b) Best fit:-

- * Searches all empty spaces (available) in memory & allocate the smallest block which can satisfy the request.

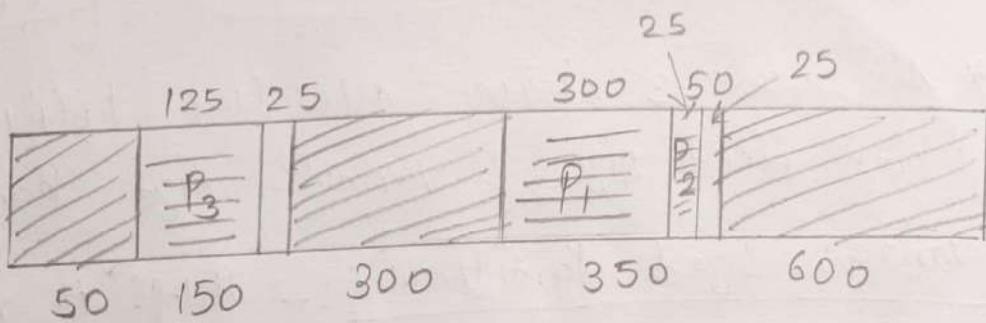
Example -

$$P_1 - 300$$

$$P_2 - 25$$

$$P_3 - 125$$

$$P_4 - 50$$



- * "External Fragmentation" exists bcz:-

$$\text{Total space Available} = 50$$

$$\text{Need for } P_4 = 50$$

But space available in non-contiguous memory block, so P_4 's request can't be satisfied.

(c) Worst fit:-

- * Instead of allocating the smallest block, Worst fit allocates the largest block of memory available

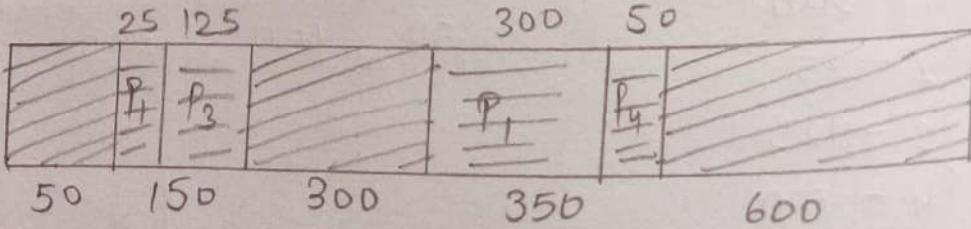
Example -

$$P_1 - 300$$

$$P_2 - 25$$

$$P_3 - 125$$

$$P_4 - 50$$





- * In variable size partitioning :- No matter which block u take, remaining space can be reused again.
- * In Variable size partitioning - Worst fit performs better.
Explanation:- When "Worst fit" policy is used, the space/block to be allocated is the largest block possible, so the remaining space will be reused of large size & there is large probability that this large size block/space can be reused again.
- * But in "Best fit" :- we use smallest block which is in the position to satisfy the request. The remaining space/block will be of small size this time, so very least probability is there of reusing that block/space.
- * So, the conclusion is:-

Variable size Partitioning	Best fit	Worst fit
	Performs Poor (leftover space reusability probability is very low)	Performs Well (leftover space-large, so reusable)

fixed Size Partitioning:- Things to remember:-

- size of partitions is fixed, can't be changed
- Partition can't be broken
- If size of process $<$ size of partition
 :- Remaining space can not be used so it is wasted.

① first fit:- Use the first partition capable enough.

All Available



Example:-

	357	43	210	390	468	32	
P ₁ - 357		P ₁	X	P ₂	X	P ₃	X
P ₂ - 210	200	400		600	7	500	300
P ₃ - 468							250
P ₄ - 491							

IE = (43 + 390 + 32)

↑
size fixed,
no reuse
possible

↓
Wasted (can't be reused)
(Due to fixed size
partitioning)

* Total space Available
 $= 200 + 300 + 250$

* Still P₄ - 491 X

* Big space available
 in non-contiguous
 fashion

So, $EF = 491$

* EF - External fragmentation
 * IF - Internal "

↳ Due to Contiguous Allocation

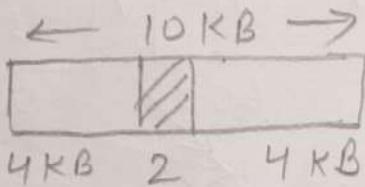


* External fragmentation depends on the problem vector / process.

Example :- Available Memory : $\rightarrow 200 + 300 + 250$

- If $P_5 - 290$ arrives : - Can be satisfied

Example :-



$$\begin{aligned} \text{Total Available} &\Rightarrow 4+4 \\ &= 8 \text{ KB} \end{aligned}$$

$$\text{Used} \Rightarrow 2 \text{ KB}$$

- If $P_1 - 9$ arrives : - No external fragmentation

Blog:-

If Total space Available	<	Required space
--------------------------	---	----------------

\hookrightarrow No external fragmentation.

- \hookrightarrow Not enough space in memory
- \hookrightarrow Not our mistake

- If $P_2 - 7$ arrives : - external fragmentation exists

Blog: Space Available (8), but not in continuous fashion.

(b) Best fit:- Allocate the smallest block possible

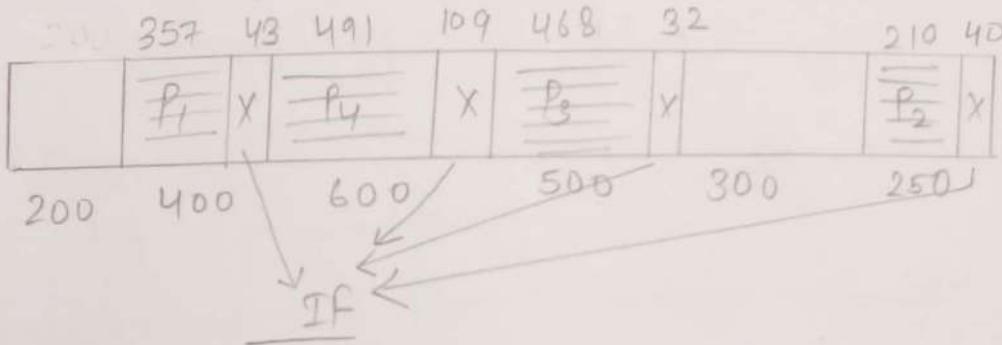
Example -

$$P_1 = 357$$

$$P_2 = 210$$

$$P_3 = 468$$

$$P_4 = 491$$



- IF = [43 + 109 + 32 + 40] — wasted
- EF = NIL as all requests are satisfied. (No process remaining)

(c) Worst fit:- Allocate the largest block possible.

↳ Remaining wasted, so practically it performs worst.

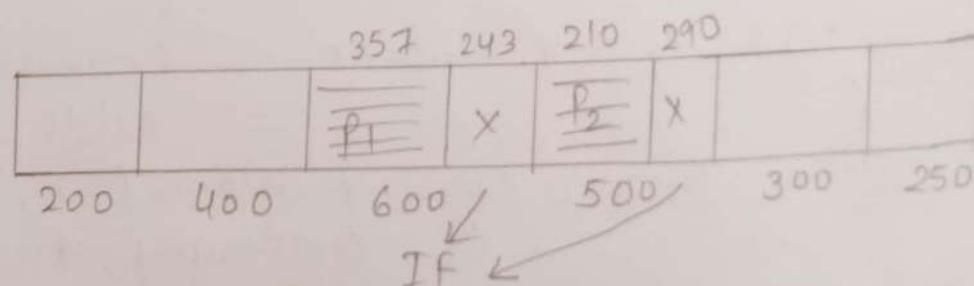
Example -

$$P_1 = 357$$

$$P_2 = 210$$

$$P_3 = 468$$

$$P_4 = 491$$



- IF = [243 + 290] - wasted
- EF = 468 + 491

* Performs worst bcz:-

→ Already initially wasted the largest partition by allocating to small processes.



* "In fixed size partitioning - Best fit performs best!"

Explanation:-

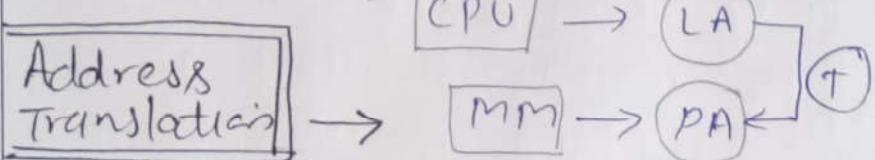
- x whatever partition we use, the remaining space is not going to be reused, so best fit performs well by choosing smallest block enough & then satisfying remaining requests by ~~re~~ using other partition.
- x Worst fit :- At start, allocate largest blocks, so very much space gets wasted.

fixed sized Partitioning	Best fit	worst fit
	Performs well (small size blocks are used first)	Perform poor (max. space wasted)



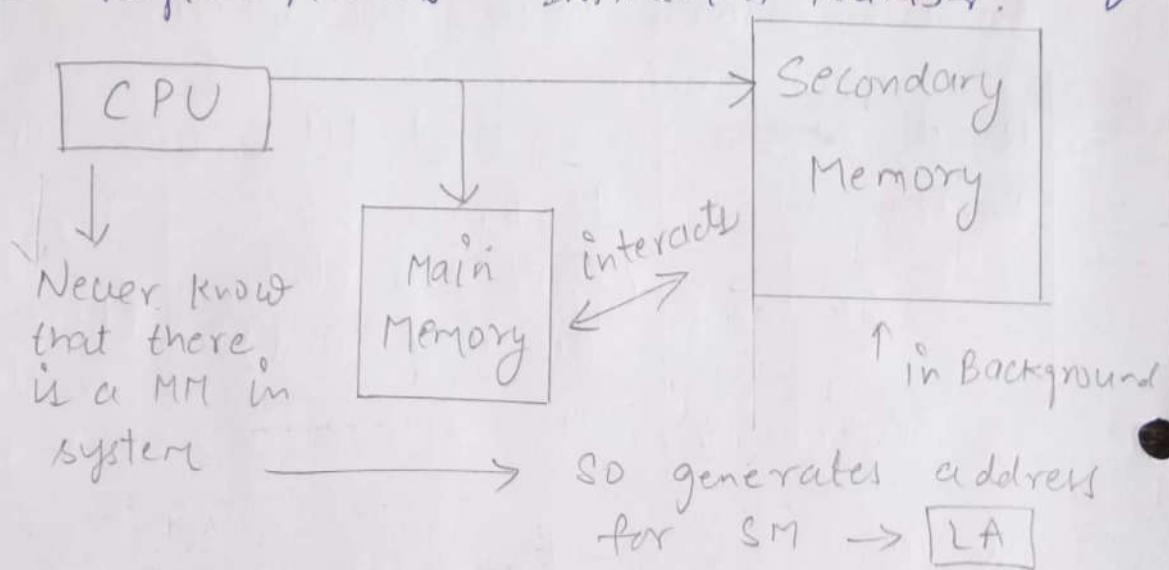
2 Major tasks of O.S. →

- Memory Allocation (Contiguous & non-Cont.)
- Address Translation



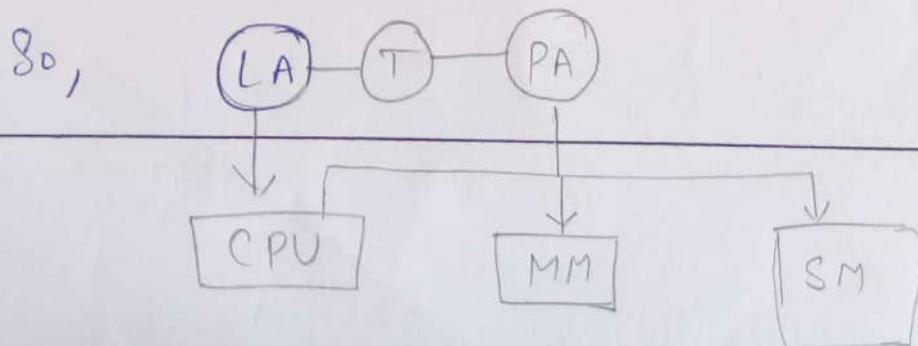
Basic Idea:-

- * CPU generates address for Secondary Memory
Called "Logical Address" — Instruction Number.



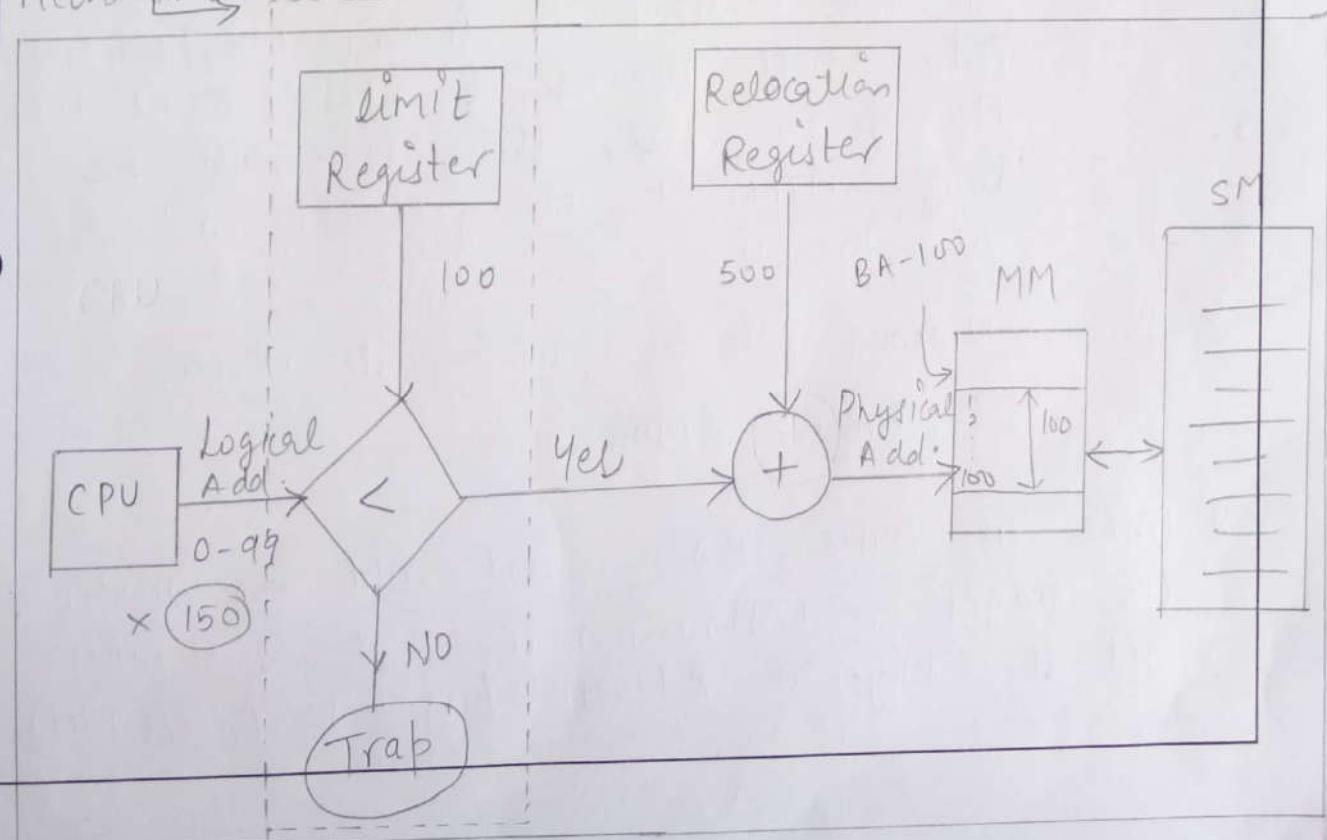
- * CPU Access SM:-

- Large Access Time
- Large size



Some Points to Remember :-

- * LA \rightarrow PA : MM Access (Locality of Reference)
 - * Secondary memory: Important Data - main memory
 - * CPU don't know MM, already
 - * CPU Always generates address for SM, which is "Logical Address".
 - * But anyhow we need to understand which important data is placed where in MM, using this information (LOR), we will then perform conversion i.e. (LA) \rightarrow (PA) : used to access MM.
- ↓
Used to access SM.



"Address Translation"

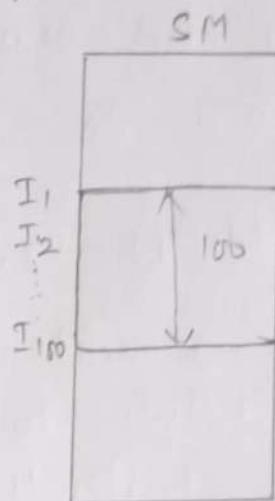
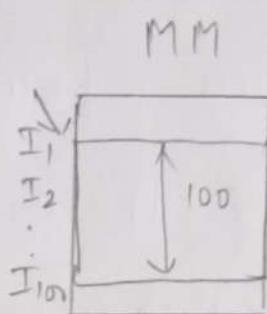


Relocation Address - A special purpose register in CPU, used to hold the ↳
Base Address of process in MM.

- * In Contiguous Memory Allocation, the address translation is easy task bcz we pick the entire process from SM. & push it into MM, we only need to know the base address of process and now we can know the address of any instruction of process.
- * But in Non-Contiguous Allocation, it will be a difficult task to do because:
 - ↳ If the process is fragmented/partitioned into small pieces & stored at diff. places in MM, then it will be very difficult to get the address of process.
- * In Contiguous Allocation, we just require
 - ↳ BASE ADDRESS
- * Once we have BA :- just add the whatever the pages or instruction u want, & u will get the "Physical Address". (used to Access MM).

* Now, Let's take an example :-

$$\text{BA} = \\ 100$$



* In SM, we have instruction $I_1 - I_{100}$. If CPU wants to access I_{52} , so this entire process needs to bring in MM. (in contiguous fashion).

* If u know :

- ↳ Base Address &
- ↳ Which element/line want to access
- ↳ size

* then we can calculate the address too, of any instruction.

* So, if $\text{BA} = 100$, I_{52} will be somewhere at 152 ($100 + 52$).

* Whatever Logical Address \rightarrow (Instruction No. / Logical Unit) comes from CPU, value of relocation Register is added & this gives the Physical Address.



* Consider Another example: -

If $BA = 500$ & u want to access I_{48} ,
then $PA = 500 + 48 = \boxed{548}$ \uparrow
 (LA)

* So, Address Translation is done as: -

- ↳ CPU generates the instruction No. (LA), it wants to access.
- ↳ This instruction no. is added to the value of Relocation Register, which gives the base address of where process stays in MM.
- ↳ Sum of above 2 factors gives "physical Address (PA)": $PA = LA + \text{Relocation Register}$

* But there exists a "Security Problem", i.e. consider the size of process - 100 units.

↳ Now, system will add 500 (BA) into whatever address u generate, to get actual address.

* Now, if some process (either intentionally or non-intentionally) generates logical address 120.

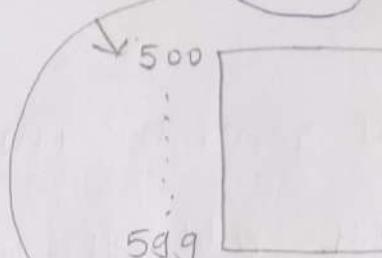
So, 120 is illegal address, bcoz process size is 100 units, then how can it request to access 120?

- * A process can do this intentionally or if it wants to access data of some another process. So,

If BA / Relocation Register = 500

$$LA = 120$$

$$PA = 500 + 120 = 680$$



Not an Authentic Request → Data of some another process } Problem

- * So, we need to maintain authenticity by implanting some security mechanism.
- * So, we need a "Limit Register" here, which stores the size of process.
- * If CPU wants to access some address which is less than this limit register value, then it can proceed (i.e. relocation Rx value will be added to it & it will give PA).
- * If it is greater than limit register value, then it will go into trap, it won't be allowed to proceed further.



Swami Keshvanand Institute of Technology, Management & Gramothan,

Ramnagar, Jagatpura, Jaipur-302017, INDIA

Approved by AICTE, Ministry of HRD, Government of India

Recognized by UGC under Section 2(f) of the UGC Act, 1956

Tel. : +91-0141- 5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in

i.e.

e.g. Limit Rx = 100 :- u can generate address (0-99)

* If CPU generate 50,

$50 < 100$

↓ NO problem

Add 50 &

Physical Address. ← Relocatⁿ Register value

* If CPU generate 150, $150 \neq 100$ → Trap.

↳ this address will not be allowed to get converted into PA, bcz it actually wants to access data of some other process.

Advantage of security mechanism:-

→ If process want to access illegal data or the data for which we don't have authority (belongs to another process)
↳ Trap

→ else, ↳ Continue with translation,
i.e. add value of
Relocatⁿ Register →
to it

"Physical
Address"



Alphanumeric Example:

→ Hint Register - Size of process
Relocation Register - Base Address

Process	LR	RR	Request
P ₀	500	1200	400
P ₁	275	550	300
P ₂	212	880	210
P ₃	420	1400	450
P ₄	118	200	80

↑ ↓
must be
less than]

* P₀ - 400 ; 400 < 500 ✓ ; PA = 1200 + 400
= 1600

* P₂ - 300 ; 300 < 275 X ; Illegal Request

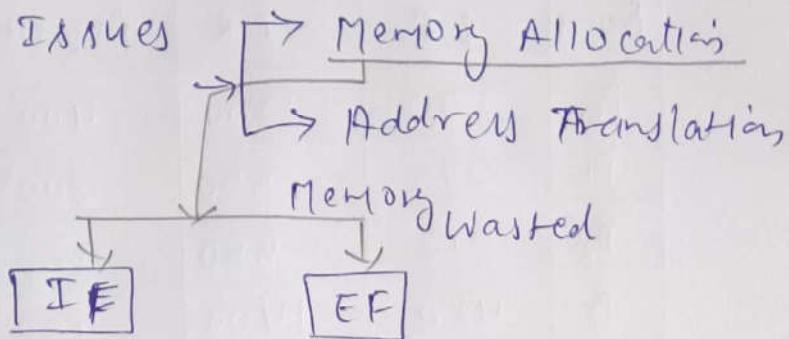
* P₃ - 210 ; 210 < 212 ✓ ; PA = 880 + 210
= 1090

* P₄ - 450 ; 450 > 420 X ; Illegal Request

* P₅ - 80 ; 80 < 118 ✓ ; PA = 200 + 80
= 280

Summary of Contiguous Allocation Policy :-

- * 2 Major Issues



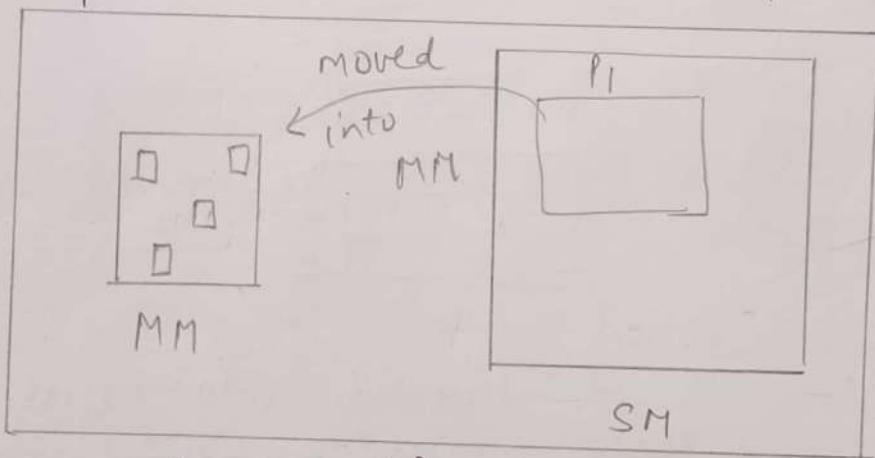
- * Internal to memory
- * Not major issue compared to EF.

- * Major Issue
- * Occurs in large No.
- * To get rid of EF, solve this problem by following 2 approaches.

- ① → Process space — Defragment
- ② → Allocate space to process in Non-contiguous fashion
 - Paging
 - Segmentation

Non-Contiguous Allocation :- Paging → \rightarrow P NO. EF
IF exists

- * Switched to this allocation due to external fragmentation in contiguous approach which was a major issue.
- * Here, we want to store the process in a non-contiguous fashion i.e. now the process can be partitioned & stored separately in MM.



"Non-Contiguous Allocation"

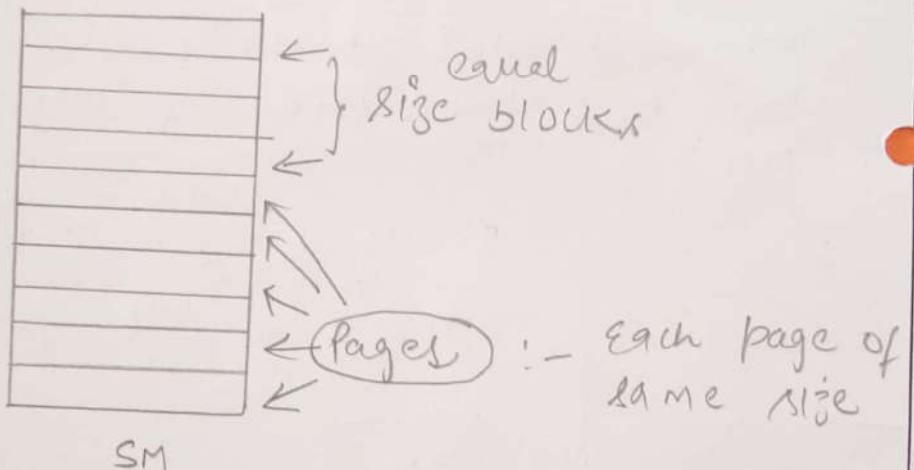
- * We partition the SM into specific no. of partitions, similarly main memory (MM) will also be partitioned into equal no. of partitions:-

→ Size of each partition/block is fixed.
Fixed size partitioning - Internal fragmentation - not as much major issue as EF.

→ If size of each block is different it will be difficult to work with & manage them.

* So, SM will be divided into equal size partitions to support non-contiguous allocation. But size of each partition will be the same, so mgmt of each partition will be easy. These partitions are called "Pages".

Divided into equal size partitions



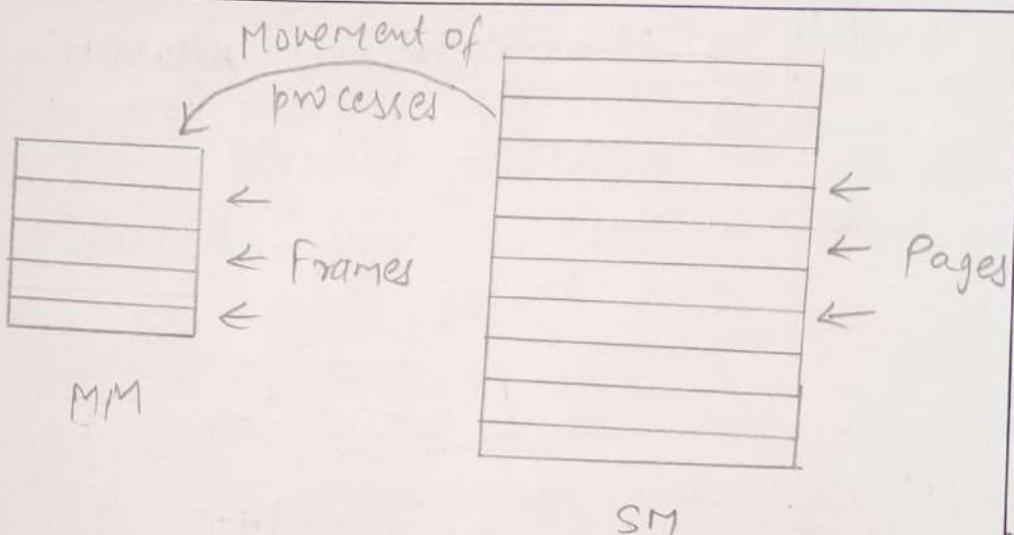
Pages :- SM is divided into equal size blocks & these blocks/partitions are called Pages. Size of each page is same. (A book).

* The same size of pages (as in SM) is used in MM too.

* MM is also divided into equal size partitions. The size of partitions here, will be same as that of SM. These partitions are called "Frames".

Frames :- MM is divided into equal size partitions, as that of SM. These partitions are called frames.

Divided into
equal
size →
partition
(AS in
SM)

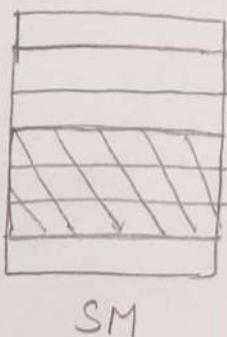


MM	SM
Partitions Frames ↗	Partitions Pages ↗
frame size = Page size	

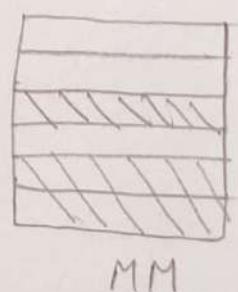
* If a process P, requires 3 pages in SM, it will require same no. of frames in MM too.

i.e.

P
↓
Requires 3
Pages

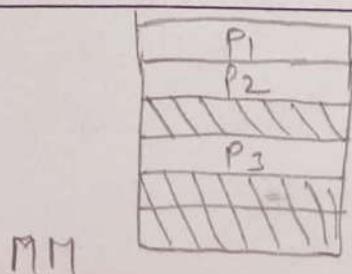


But, If



i.e.
Total
available
frames → 3

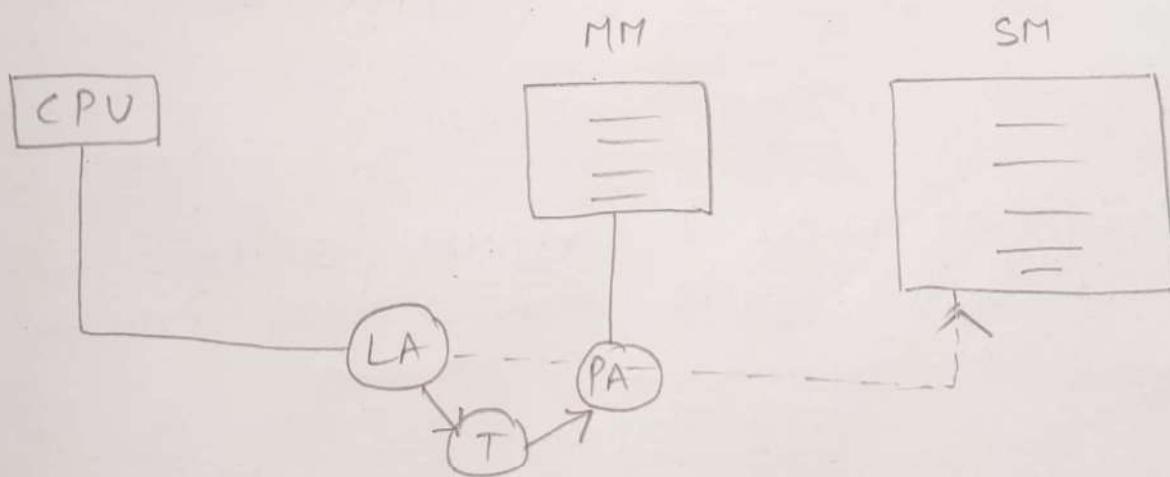
* Then process P can be divided into 3 parts (P_1, P_2, P_3) & stored as:-



← Allocation (Beauty of
Non-Contiguous
Allocation)

Beauty of Non-Contiguous Allocation:-

- * If a process — divided into Pages (in SM).
 - ↳ Whenever empty frames are there in MM, process will be adjusted in these frames.
- * So, there will be complete independance from "External fragmentation".
- * In Non-contiguous Allocation too, Logical Address is required to convert into Physical address.

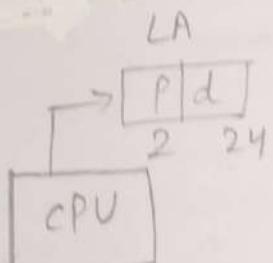


Question :- How to do the conversion i.e.

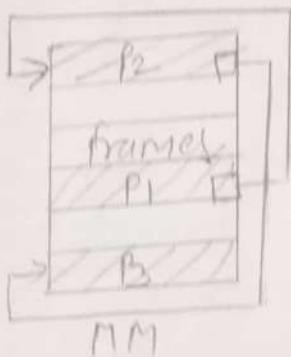
$$\boxed{LA \rightarrow PA \Rightarrow ?}$$

Answer:-

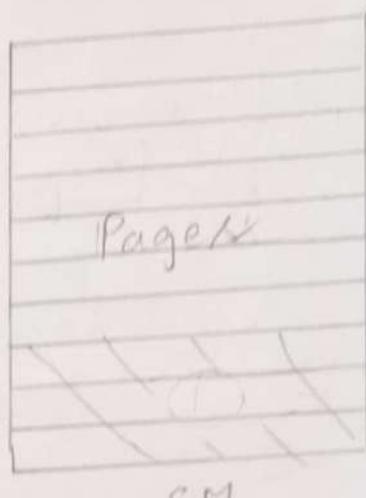
- In Contiguous Allocation, Address Translation was very , bcoz we were having "Base Address".
- But, in Non-Contiguous Allocation, no ordering of pages are there (i.e. a page can be anywhere in SM), then how we will find out address of pages in memory ??



Process



P₁
P₂
P₃
Pages



Page 1

P | d
↓
Page No.

Instruction
Offset

Divided
into
Pages

1	100 int
2	.
3	.
4	.
5	.
:	:
300	

lineNo.
325
Instructn No. 25

Contiguous
Story
of
300 pages
Page No - 4

- * P₁, P₂, P₃ are Pages in SM, which are then randomly arranged into MM.
- * If, CPU generates LA :- [2 | 24]
 - ↳ Page No. 2 & instruction offset 24
- * Now, the task is to find out address of Pages (P₁, P₂ & P₃) in MM ?? (for every process)
 - ↳ Solution :- Base Address of Page(frame) (pointer will tell the address of next page & so on).
- * for [2 | 24] :- U should know address of Page No. 1 which will give address of Page No. 2



* It's an easy approach. Not the every page address is required to remember,
But **Problem** is :-

↳ Access will be very slow
(i.e. if process X wants to access 200 (1-1000), then access will be very slow & access time will be higher, system slow too).

* Solution - "Page Table" (Index Table)

Page Table:- A kind of data structure which is actually not stored in PCB, bcoz it's very big in size.

* PTBR will tell you where the Page Table is.

• **PTBR** - Special Purpose Register
↳ holds ^{base} address of Page Table

* Some Points to remember for Page Table:-

→ A data structure, not a H/w.

→ Contains Base Address of every Page.

→ Can contain a no. of entries equal to the no. of ~~processes~~, a process have in MM.

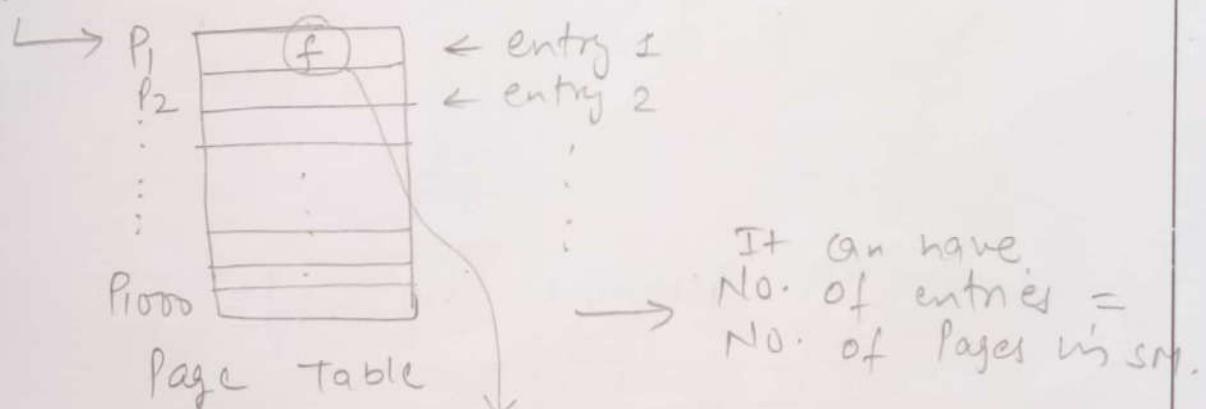
→ Specific for each process.

→ Page Table is nothing, But the index table.

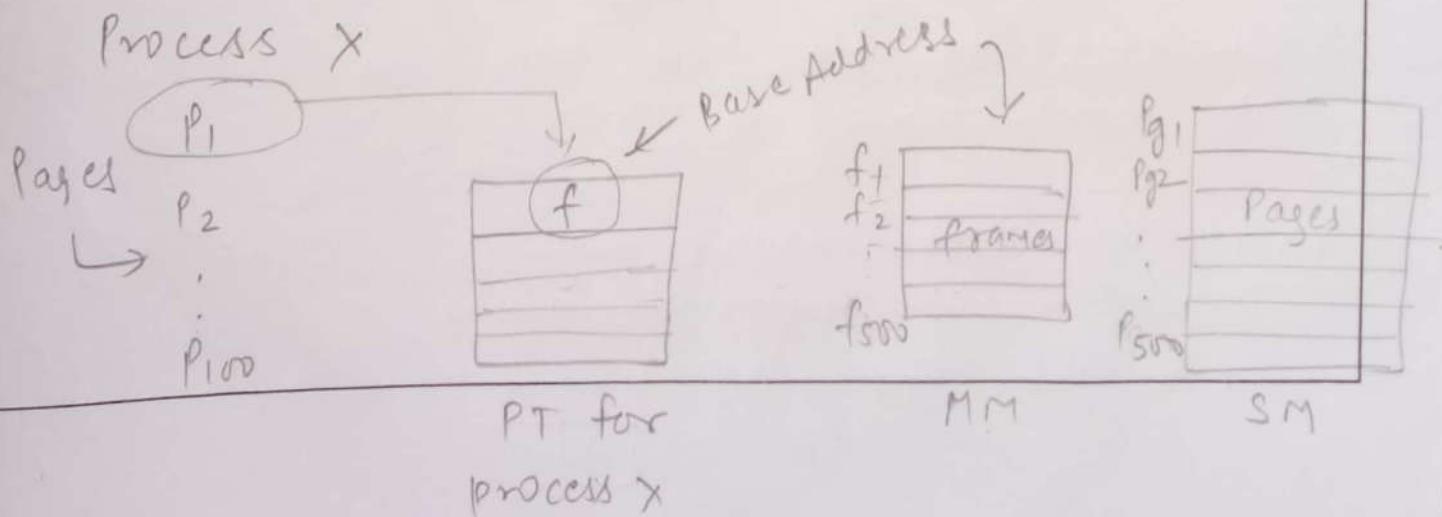
→ It will have frame number

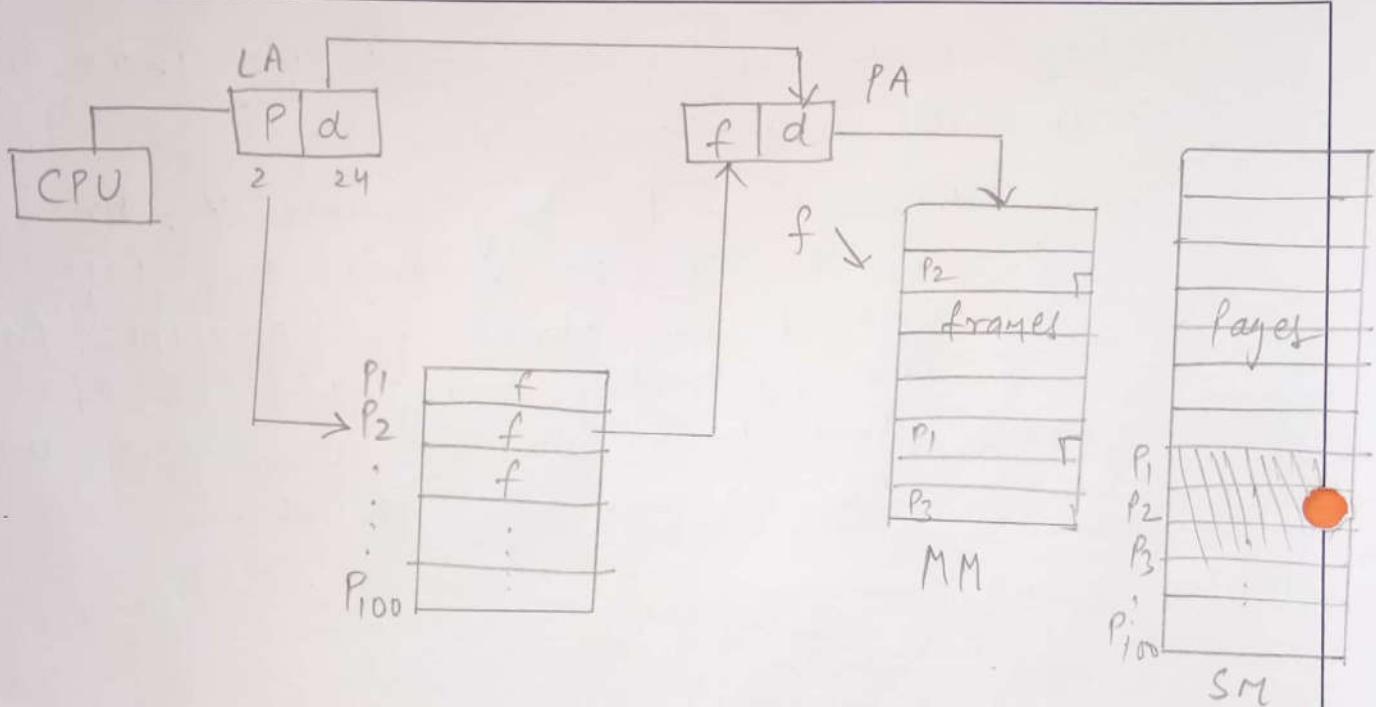
↳ means: - for any process X , to which the PT belongs to, page No. 1 is stored in specific frame No. [Actually Base address of that frame No. is MM where that Page is currently stored.]

Page No.

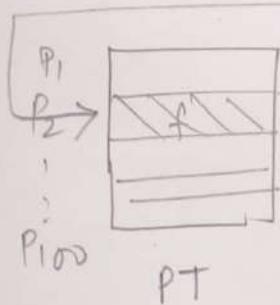


Base Address of frame No. in MM, where that Page is currently stored





* So, if CPU generates LA :- 2|25 P|d



BA ← Not the frames But the base address of frame, which will tell, where the page No. 2 is in MM.

* Wherever we place the page, the instruction No. won't change So, f|d [i.e. line No. 25 will be the same]

* So, we don't require any translation of instruction offset. (d will remain same).
 ↑
 No. matter of which page

i.e. PA :- f|d

Advantages :-

- ↳ Access fast & Non-Contiguous

Disadvantage :-

- ↳ Use of Independent Data Structure
(i.e. Page Table)
- ↳ Page is required bcz we must know the Base Address of every page.
- ↳ Page Table — Meta Data
- ↳ specific / Independent for each process
- ↳ system going to slow & times.

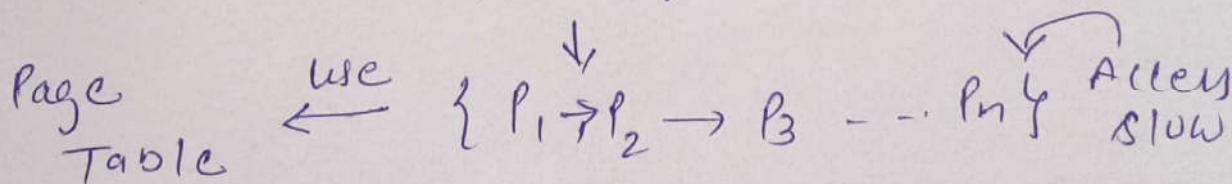
Summary of Non-Contiguous Allocation :-

* Problem with Contiguous Allocation

- ↳ External Fragmentation

↓
SOL

Non-Contiguous Approach.



↓
To remember
Base Address

of each Page.



Paging - fixed size partitioning

↳ can buffer from IF.

↳ No EF.

* Because of Paging, system is going to be slow 2 times, due to:-

→ When we access data, we have to access Page Table to convert LA \rightarrow PA.
Page Table is also stored in MM.

→ So, we will access MM & find frame no. & combine frame no. with instruction no. offset & then get PA, which is again stored in MM.

* So, Instruction's Access Time would be double, which is second major disadvantage.



Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagar, Jagatpura, Jaipur-302017, INDIA

Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956

Tel.: +91-0141-5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in

Now, the following questions arise here:-

↳ How to find size of memory if u know the size of address?

↳ How to find size of address if u know the size of memory? (i.e. vice-versa)

① Translation :- Space \longrightarrow Address

* In Computer system, all tasks are done in machine Language (in binary system - 0 & 1).

* So, now the question is "How many bits can generate how many combinations?".

* We know:-

1 bit - 2 combinations - 0 & 1

2 bits - 4 " - 00, 01, 10, 11

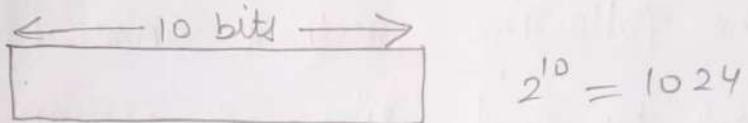
3 bits - 8 " - 000, 001, ... 111

!

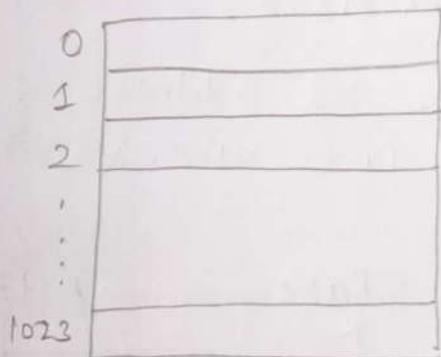
n bits - 2^n "



eg :-



$$2^{10} = 1024$$



eg :- If u have an address of 10 bits, then what is the total size of memory, which will be there, if the system is byte addressable (i.e. 1 byte).

Ans:- Total size of memory depends on 2 things :-

- No. of locations it has [2^n]
- size of locations.

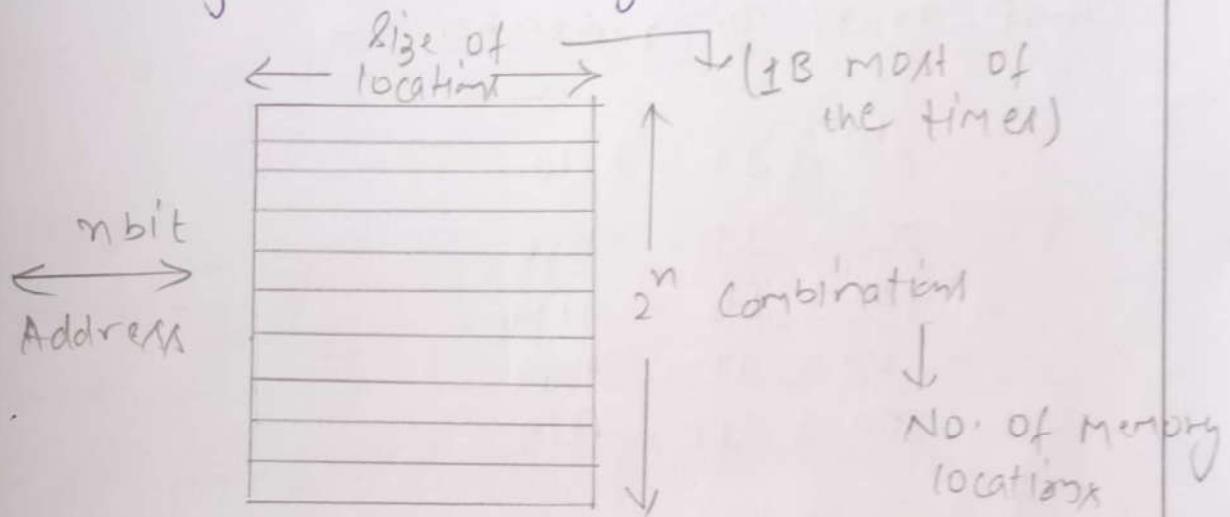
Accordingly, :- No. of locations \times size of each location

$$\Rightarrow 2^{10} \times 1B \Rightarrow 1024 \times 1B$$

$$\Rightarrow 1024B = \boxed{1KB}$$



* Now, if we have 10 bit & byte addressable system, then 10 bit address can be used for addressing 1 KB memory. ~~size of location~~.



* If u have n-bit address, :-

$$\text{n-bit Address} \Rightarrow 2^n \times \text{size of location}$$

$$\Rightarrow 2^n \times 1B \quad \begin{matrix} \uparrow \\ \text{size of memory} \end{matrix}$$

Example:-

(1) $\xrightarrow{\text{14 bit Addr.}} \boxed{(1B)} \Rightarrow 2^{14} \times 1B = 2^4 \cdot 2^{10} \times 1B$
 $\Rightarrow \boxed{16 KB}$

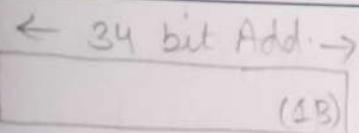
(2) $\xrightarrow{\text{22 bit Addr.}} \boxed{(2B)} \Rightarrow 2^{22} \times 2B = 2^2 \cdot 2^{20} \times 2B$
 $\Rightarrow \boxed{8 MB}$



असतो मा रामगमय

Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagar, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel.: +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

③



$$\Rightarrow 2^{34} \times 1B = 2^4 \times 2^{30} \times 1B$$
$$\Rightarrow 16 GB$$

Short Tricks to remember :-

$$2^{10} = 1K = \text{Kilo}$$

$$2^{20} = 1M = \text{Mega}$$

$$2^{30} = 1G = \text{Giga}$$

$$2^{40} = 1T = \text{Tera}$$

$$2^{50} = 1P = \text{Peta}$$

②

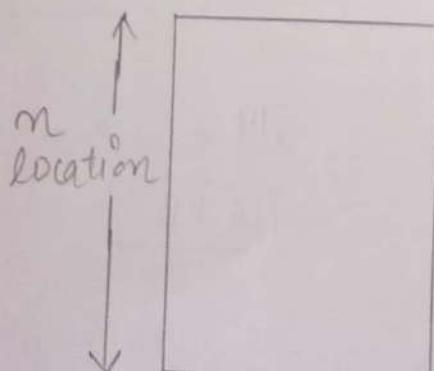
Translation : Address \longrightarrow Space

eg:-

Memory size - 64 KB

(System - Byte Addressable i.e. 1 Byte)

* How many bits in memory?



Memory size =
Total No. of locations
i.e. size of each location

$$MS = n \times 1B$$

$$n = \frac{MS}{1B} = \frac{64 KB}{1B}$$

$$n = 64 K$$

$$\hookrightarrow 64,000 \text{ locat.}$$



* With n bits 2^n Addresses are generated.

So $n \rightarrow \log_2 n \rightarrow$ No. of bits required.

$$64K \rightarrow 2^6 \cdot 2^{10} \Rightarrow 2^{16} = \boxed{16 \text{ bits}}$$

\downarrow
 $\log_2 16$

Examples :-

①

$$\begin{array}{c} \xrightarrow{\text{Size of Memory}} \\ \boxed{32 \text{ KB}} \quad (1B) \end{array}$$

$$\Rightarrow \frac{32 \text{ KB}}{B} = 32 \text{ K}$$

$$\Rightarrow 32 \text{ K} \rightarrow 2^5 \cdot 2^{10} = 2^{15}$$

②

$$\boxed{256 \text{ MB}} \quad (1B)$$

$$\boxed{15 \text{ bits}}$$

$$\Rightarrow \frac{256 \text{ MB}}{1B} = 256 \text{ M} \Rightarrow 2^8 \cdot 2^{20} = 2^{28}$$

\downarrow

$$= \boxed{28 \text{ bits}}$$

③

$$\boxed{16 \text{ GB}} \quad (4B)$$

$$\Rightarrow \frac{16 \text{ GB}}{4B} = 44 = 2^2 \cdot 2^{30}$$

\downarrow

$$= 2^{32} - \boxed{32 \text{ bits}}$$



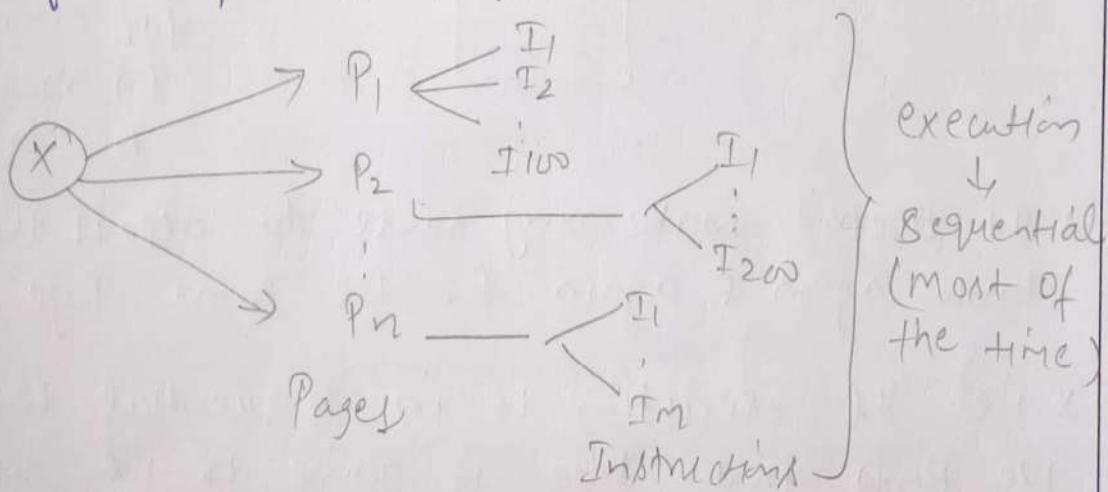
Translation Lookaside Buffer :-

- * When we use Page Table :- It is independent for each process.
- * Which results in 2 time access of MM
 - ① → MM Access - for accessing PT stored in MM
 - ② → MM Access - for storing the PA in MM.
- * So, due to paging the access is 2 MM , i.e. which slow down the system 2 times.
- * In a system in computer science, there is a tradeoff b/w
 - system
 - time
- * Paging
 - used to remove External fragmentation
 - Causes the penalty :- speed will be half.
- * Now, we need to remove this effect.



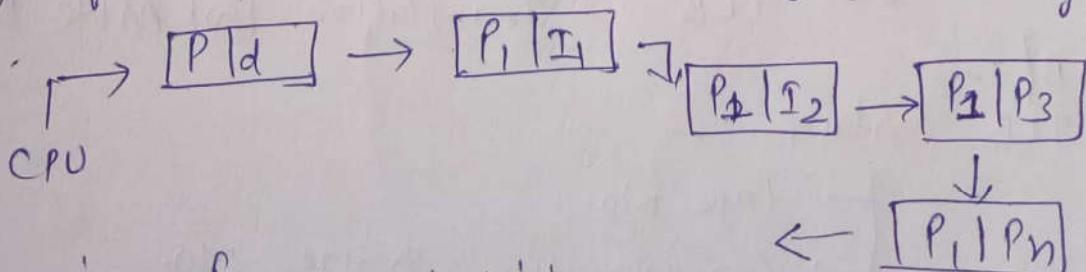
असतो मा सद्गमय

- * We know that processes execute sequentially & locality of reference (LOR) says : "Most the time, Data will find in MM itself".
- * So, if a process X :-



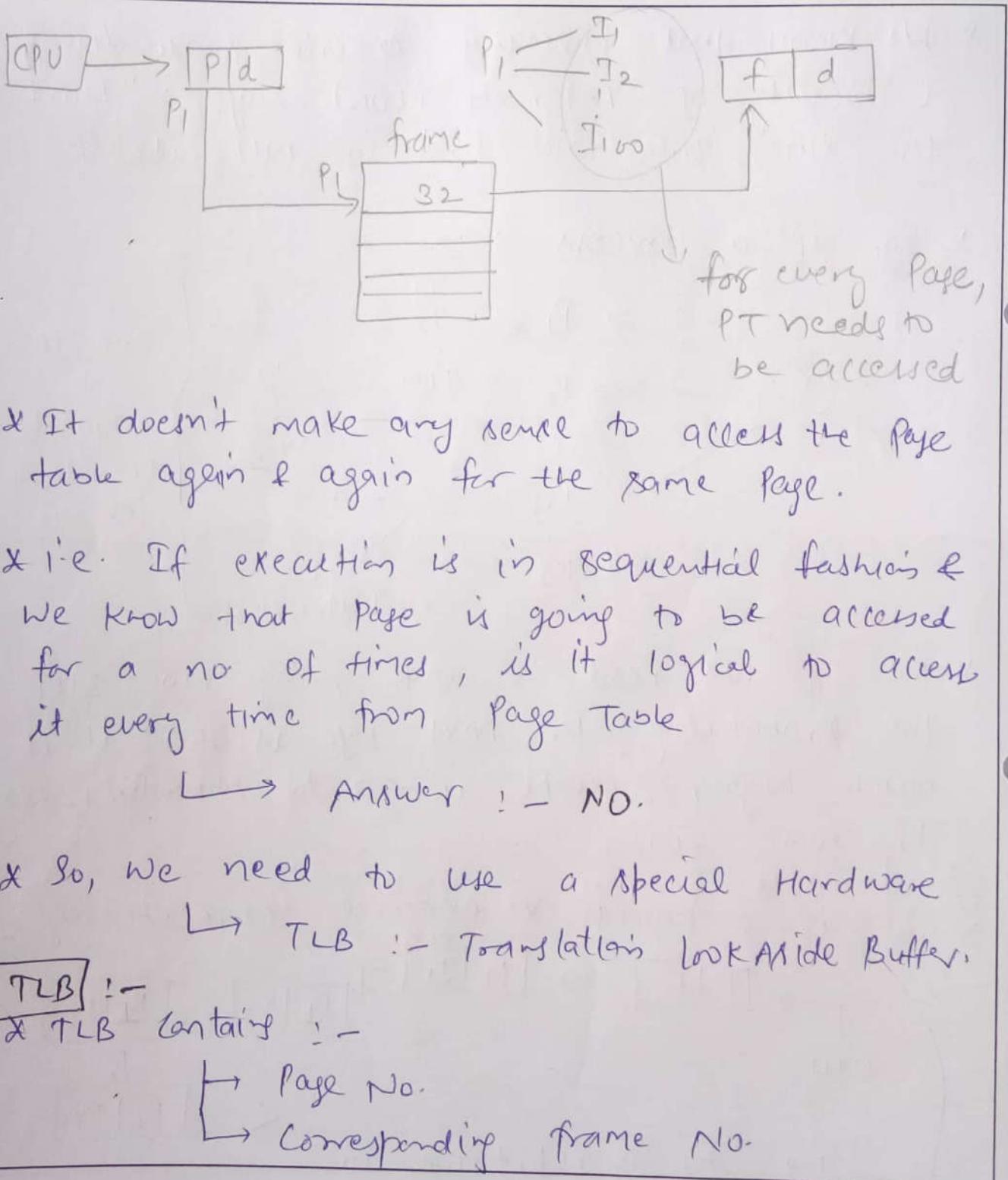
- * i.e. If we read a book, Page No. 349, the probability for next page is 350, it is much higher, except u go to Appendix/index for some time.

- * So, if a process X executes sequentially :-



i.e. for a repetitive no.

of times, u have to access the Page Table which will tell u every time that Page 1 refers to frame 32. 45



- Belong to a system & its a H/W, so can't say that every process has a different TLB.



- Initially TLB :- Empty
- When a process starts, it will first go to ~~PT~~ & will check for the frame No. of Page in PT.
[eg:- Page 9 frame 32] → stored in TLB for future
- So, next time instead of checking the PT again, u will go to TLB & will check if there exists an entry for a specific page?
- If yes, u can directly access the frame No.(f) here, combine it with instruction offset(o) & generate Physical Address (PA) :- $f|d$
- If no, put that entry in TLB.

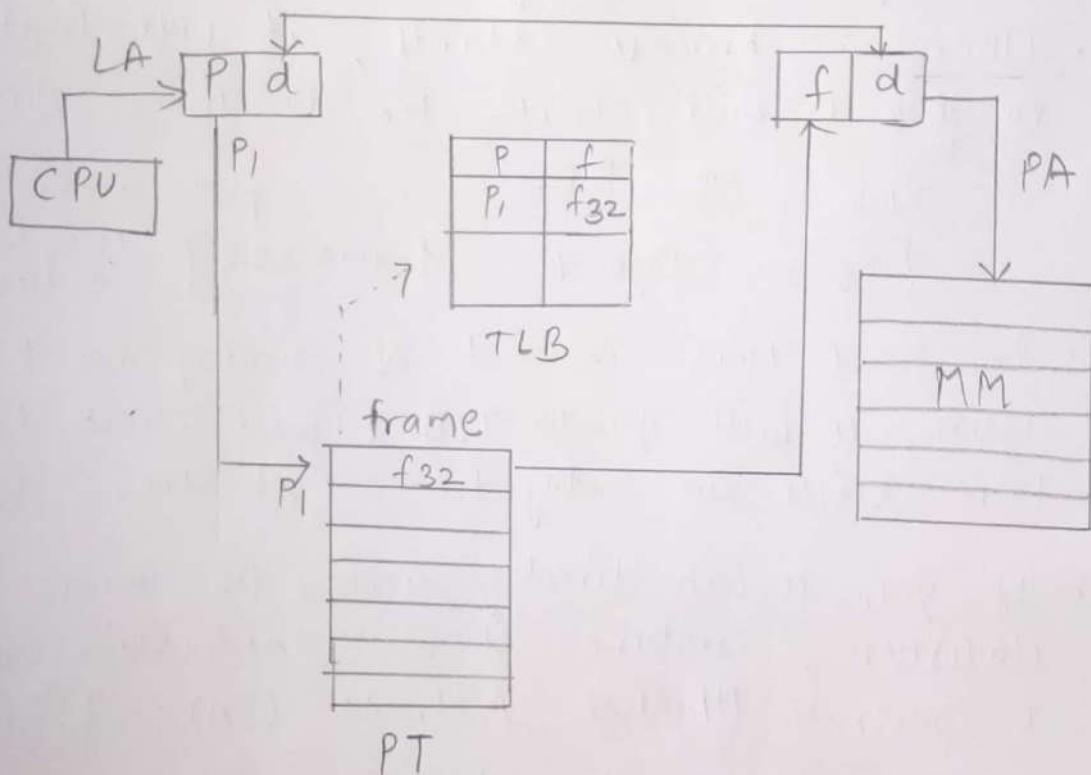
Page	Frame
P ₁	f ₃₂
P ₂	f ₅₉
:	:

TLB

- When u access a page for a repetitive no. of times, first time, it need to be accessed from PT.

↳ After it, it is saved into TLB.

So, now we have:-



* As TLB is a H/W, so **Access Time** will be very less this time.

* TLB generally have. $\begin{matrix} \rightarrow 16 \\ \rightarrow 32 \\ \rightarrow 64 \end{matrix} \} \text{Memory Locations}$

* Hit Ratio - 90% [of TLB]

* Page Table - large Access Time & stored in Mm.

• But TLB - Access Time is less & H/W.

* This time MM needs to be accessed only once.



Advantages: - Less Access Time

Disadvantages: -

- ↳ When there is a context switcher, TLB is required to be deleted / clear bcz when new process comes, pages of this process can be stored in different frame No.
- ↳ When there are multiple context switches. TLB needs to flushed out again & again.

Conclusion: -

[CPU] → [LA] → [P | d] : - Needs Accessing MM & times → Degrading Performance
↳ So, use TLB ↗ Page No. ↘ frame No.

* for each Page, first time → go to PT ↗
store in ← find frame ↘
TLB

* Now for next Access to same Page → Retrieve frame No. from TLB
(Not from PF).

* If TLB Hit :- fine

* If TLB Miss :- store in TLB, for future use.



Example :- Consider following :-

$$\text{MM Access Time} = 400 \mu\text{s}$$

$$\text{TLB Access Time} = 50 \mu\text{s}$$

$$\text{hit Ratio}_{(\text{TLB})} = 90\%$$

CASE I :- If we don't use TLB :-

$$\begin{aligned}\text{Avg. Memory Access Time} &\Rightarrow 2 \times \text{MM} \\ &= 2 \times 400 = 800 \mu\text{s}\end{aligned}$$

CASE II :- If we use TLB :-

$$\begin{aligned}\text{Avg memory Access Time} &= 0.9(50+400) + 0.1(50+400) \\ &= 0.9(450) + 0.1(850) \\ &= 405 + 85 = 490 \mu\text{s}\end{aligned}$$

* As compared to TLB, Avg. memory Access Time very large when we use MM 2 times.

Formula :- \rightarrow Effective Access Time (EAT)

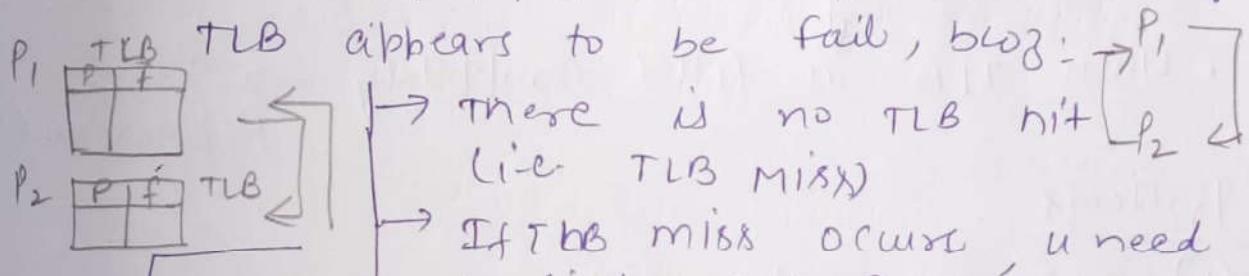
$$\text{Avg Memory/Instruction Access Time} = h(\text{TLB} + \text{MM}) + (1-h)(\frac{\text{TLB}}{2} + 2\text{MM})$$

$$\begin{aligned}\frac{0.9(50+400)}{\text{TLB Hit}} + \frac{0.1(50+\frac{400}{2}+400)}{\text{TLB Miss}}\end{aligned}$$



Disadvantage (of TLB): -

- ↳ Can store Data of only process at a time
- ↳ When multiple context switches occurs.



- There is no TLB hit (i.e. TLB miss)
- If TLB miss occurs u need to first access Page Table (PT) & then MM.
- Contents of TLB needs to be flushed every time.

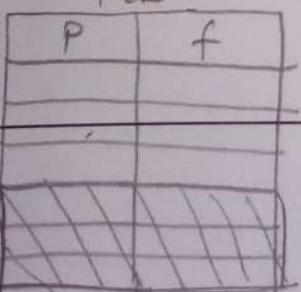
Solutions

(I) ↓

- If u have multiple copies of TLB, then the problem wont occur.
- Contents of process P₁ will be saved if P₂ arrives meanwhile. P₁'s contents need not to be flushed then.
- This leads to cost expensive solution.

(II) ↓

Solution 2

- Some part of TLB can be put reserved for OS, called (wired down entries).

TLB
- No process can access this part.
- When Context Switch occurs it does not flush content of currently running process in TLB, it just uses this reserved part.



- When OS given charge back to previously running process, now it does not need to start from scratch as it has the data stored already.
- When TLB is filled completely → Page Replacement

Problems:-

① TLB hit Ratio = 90%. TLB consumes 10ns & RAM takes 50ns for access. Compute EAT (Effective Access Time).

$$EAT = 0.9(10 + 50) + 0.1(10 + 50 + 50)$$

$$\boxed{EAT = 65 \text{ ns}}$$

② Consider a system with TLB & paging. It takes 200ns for a memory reference during TLB hit, while it takes 350ns during TLB miss.

Calculate hit Ratio if EAT = 240ns.

$$240 = x(200) + (1-x)(350)$$

$$\text{So, } \boxed{x = 60\%}$$



**Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagar, Jagatpura, Jaipur-302017, INDIA**

Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956

Tel. : +91-0141- 5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in

Example :-

Given 64 MB RAM free for allocation. Let following processes request memory allocation & further 64 MB is divided into 8 fixed & equal size partitions.

Compute memory fragmentation, if exists.

P₁ - 4 MB

P₂ - 7 MB

P₃ - 2 MB

P₄ - 8 MB

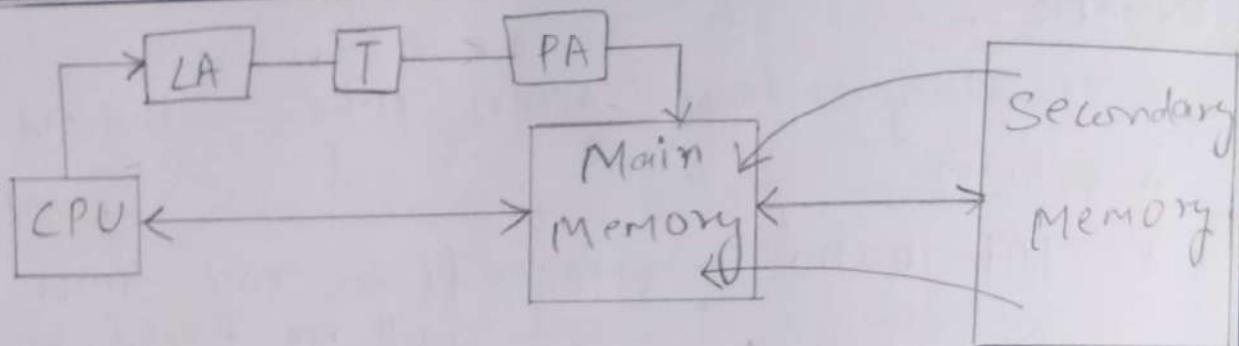
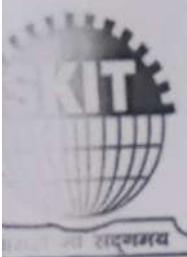
P₅ - 1 MB

P₆ - 9 MB

P₇ - 2 MB

P₈ - 3 MB

P ₁	4	8 MB
P ₂	1	8 MB
P ₃	6	8 MB
P ₄		8 MB
P ₅	7	8 MB
P ₆	6	8 MB
P ₇	5	8 MB
Free		8 MB



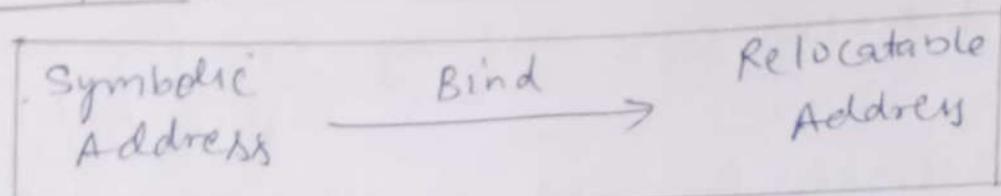
Address Binding :-

- * Usually, a program resides on disk in the binary executable form, for execution it must be brought into MM.
- * The processes waiting to be brought into MM form a I/P queue. Then any one process from this I/P queue is selected & loaded into memory.
- * Then this process access data/instructions from memory, works on this & when gets terminated, the space is declared available.
- * A program requires several steps before execution.
- * Addresses in source program are - symbolic.
- * Compiler :

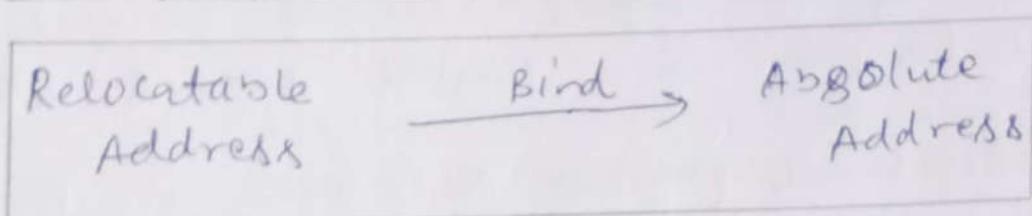
Symbolic Address	<u>Binds</u>	Relocatable Address
------------------	--------------	---------------------



I) Compiler:-



II) Linkage editor / Loader:-



* Binding is mapping from one address space to another. It can be viewed in following ways too:-

a) Compile Time:-

↳ If, at compile time, it is known where the process will reside in M.R., then "absolute code" is generated.

b) Load Time:-

↳ If not known at compile time, then "relocatable code" is generated.

c) Execution Time:-

↳ During execution, if process can be moved from one memory segment to another, binding must be delayed until run time.



Swami Keshvanand Institute of Technology, Management & Gramothan,
Rammagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400 Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Logical & Physical Address Space :-

- * Address generated by CPU (for SM):-
 ↳ Logical Address (LA).
- * Address for MM → Physical Address (PA).
- * Set of all logical Addresses generated by a program → "Logical Address Space".
- * Set of all physical Addresses corresponding to these logical Addresses → "Physical Address Space".



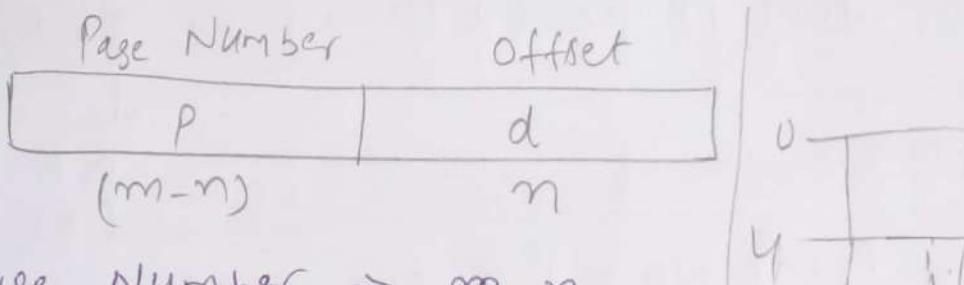
Page Table :- $(LA \xrightarrow{\text{map}} PA) \Rightarrow ?$

* Size of a page :- power of 2. (512 - 1GB per page)

* If

↳ Size of Logical Address Space = 2^m

↳ Size of Page (Page size) = 2^n



* Page Number $\Rightarrow m-n$

* Page Offset $\Rightarrow n$

Example :-

Page	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Page 0	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
Page 1																
Page 2																
Page 3																

Page No.	Frame No.
0	5
1	6
2	1
3	2

Page Table

Logical Memory
(SM)

Physical Memory (MM)
a
b
c
d
e
f
g
h



In Logical Address, $n = 2$
 $m = 4$ } Given
Physical Memory \Rightarrow 32 bytes.

* Size of logical address space $= 2^m = 2^4 = 16$

* Page size $= 2^n = 2^2 = 4$

So, MM (Physical) Memory \Rightarrow 32 bytes

formula :-

LA = frame No. \times Page size + offset

↓
8 pages
(Size - 4)

→ (0-3)

① LA [0] - (Page 0, offset 0)

↳ In Page table : Page 0 - frame 5

So, LA $= (5 \times 4) + 0 = 20$ (PA)

② LA [3] - (Page 0, offset 3)

So, LA $= (5 \times 4) + 3 = 23$ (PA)

③ LA [4] - (Page 1, offset 0) Page 1
frame 6

So, LA $= (6 \times 4) + 0 = 24$ (PA)

④ LA [13] - (Page 3, offset 1) : - Page 3 -
frame 2
LA $= (2 \times 4) + 1 = 9$ (PA)

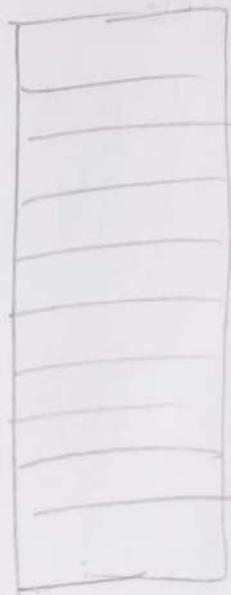


Example :-

Page Table frame	
Page	frame
0	2
1	3
2	1
3	0

Page size \Rightarrow 4 pages

PA \Rightarrow 32 bytes (8 Pages)



PA

- * Map all logical Addresses to Physical Addresses.
- * One additional bit attached to each entry in PT. \rightarrow valid/invalid bit.

Valid :- If associated Page is in process's Logical Address space & this is a valid page (Both).

Invalid :- If Page is not in process's Logical space.

frame	valid / invalid bit
2	v
6	v
5	v
9	i
8	i
3	v
4	v

Page Table



Example:-

Logical Address Space : 00000 - 160000

Page size : 2KB

frames

00000	Page 0
2000	Page 1
4000	2
6000	3
8000	4
10000	5
12000	

Logical Memory

Page	frame	valid/
0	2	V
1	3	V
2	4	V
3	7	V
4	8	V
5	9	V
6	0	1'
7	0	1'

Page Table

0
1
2
3
4
5
6
7
8
9

Page 0
Page 1
Page 2

Page 3
Page 4
Page 5

Page 6
Page 7
Page 8
Page 9

Physical Memory

- * Addresses → Beyond 12000

↓
Invalid



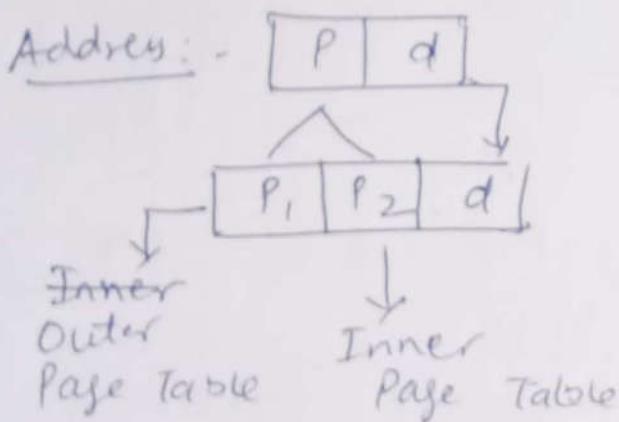
Page Table Structures :-

I Multi-Level Paging :-

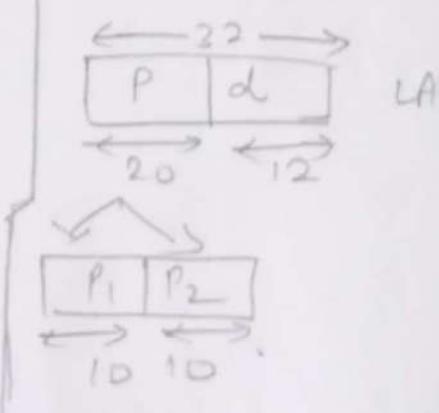
↳ Difficult to store large Page Table (Approx 100 Pages) at same place.

↳ So levels of Page Tables are used.

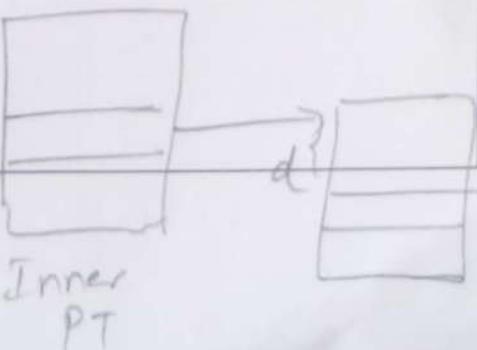
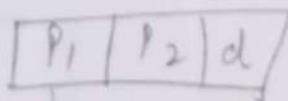
Address :-

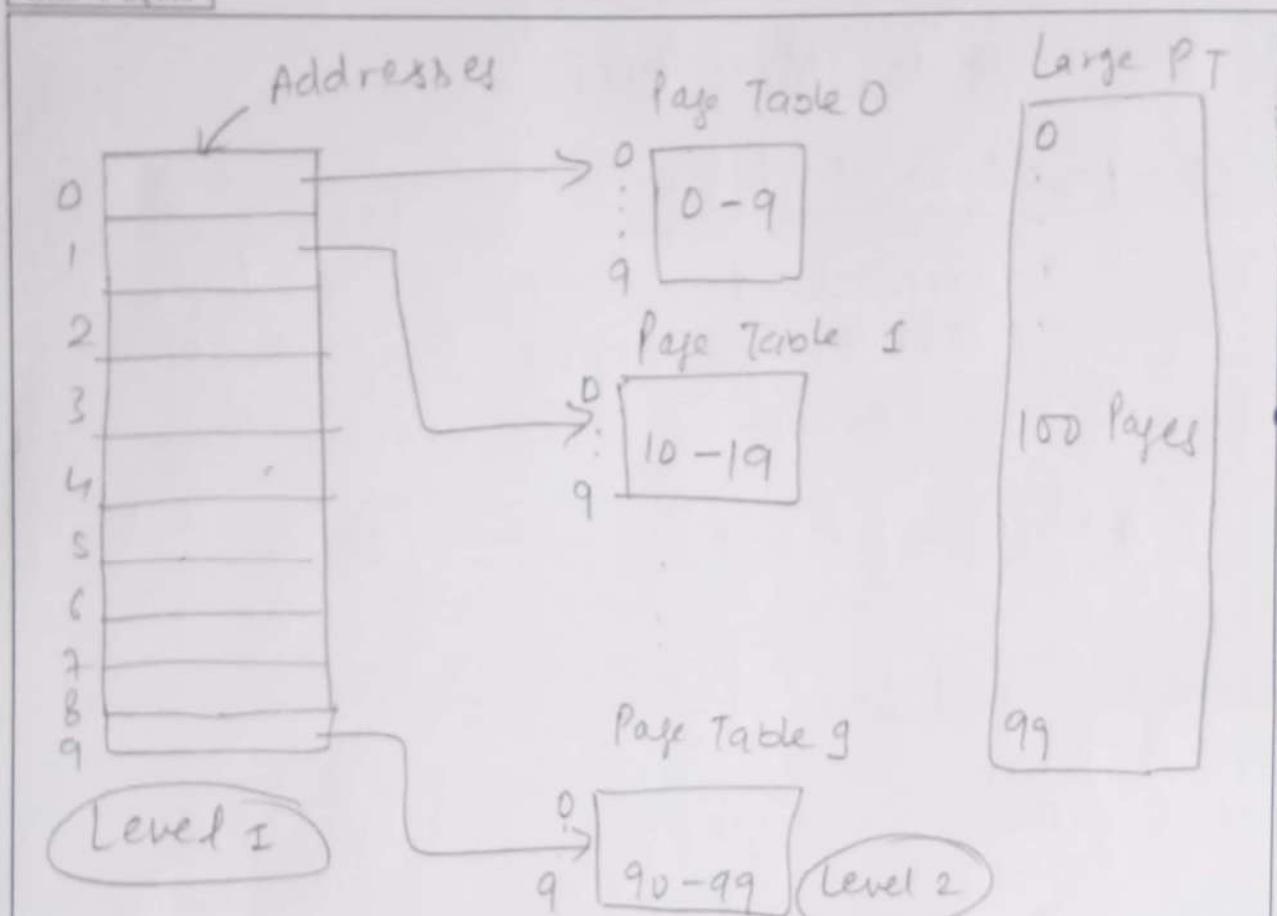


MM = 64 MB
LA = 32 bit
Page Size = 4 KB



LA

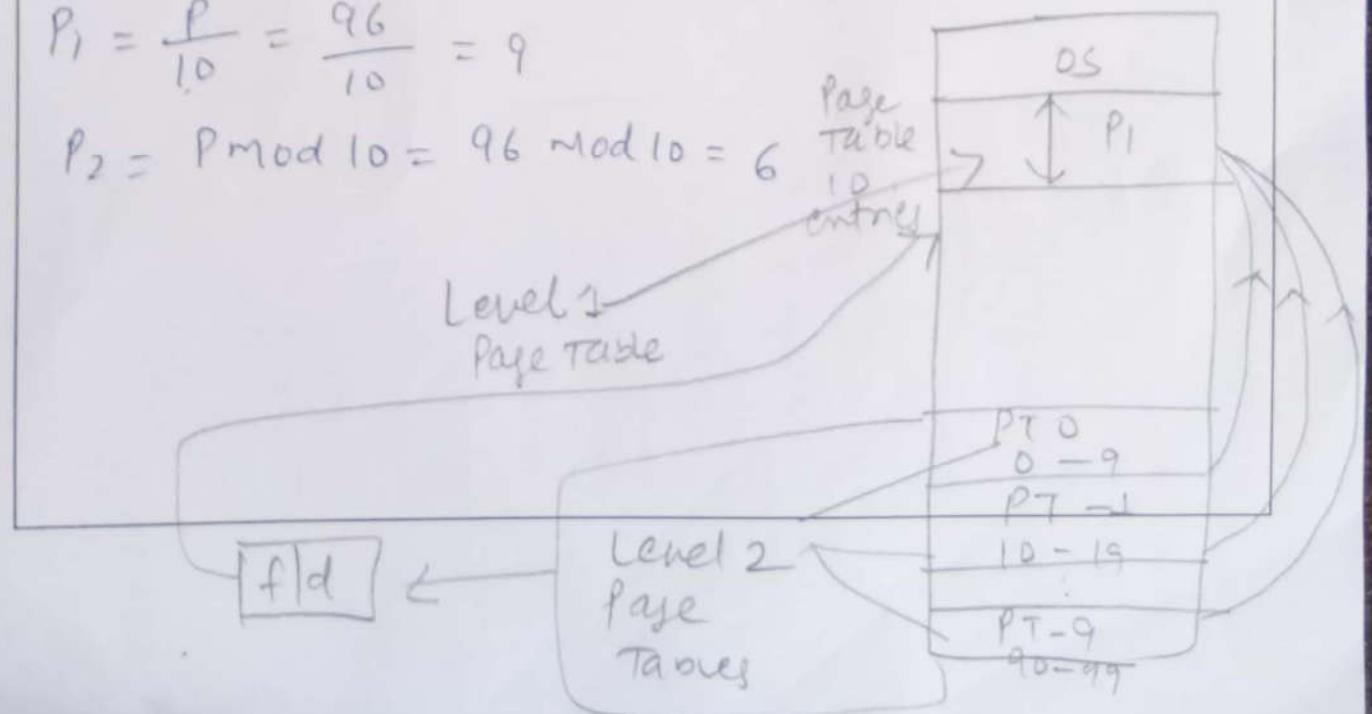




* If CPU generates Page - 96 - P

$$P_1 = \frac{P}{10} = \frac{96}{10} = 9$$

$$P_2 = P \bmod 10 = 96 \bmod 10 = 6$$

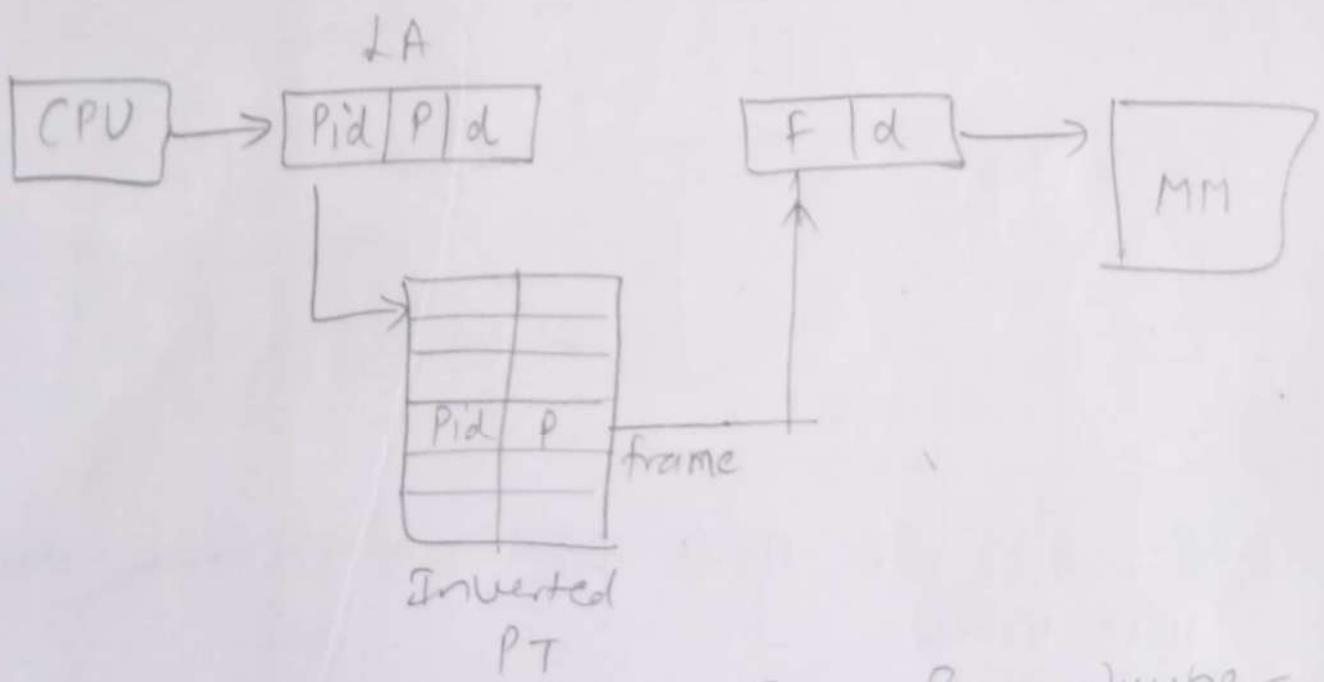


(ii)

Inverted Page Table :- Degree of multiprogramming increased (Multiple processes)

↳ Multi-level Paging - Large Access time & wastage of space in PT.

- * So, for saving space - use Inverted PT.
- * Many times PT contains more entries for invalid frames & less entries for valid frames.



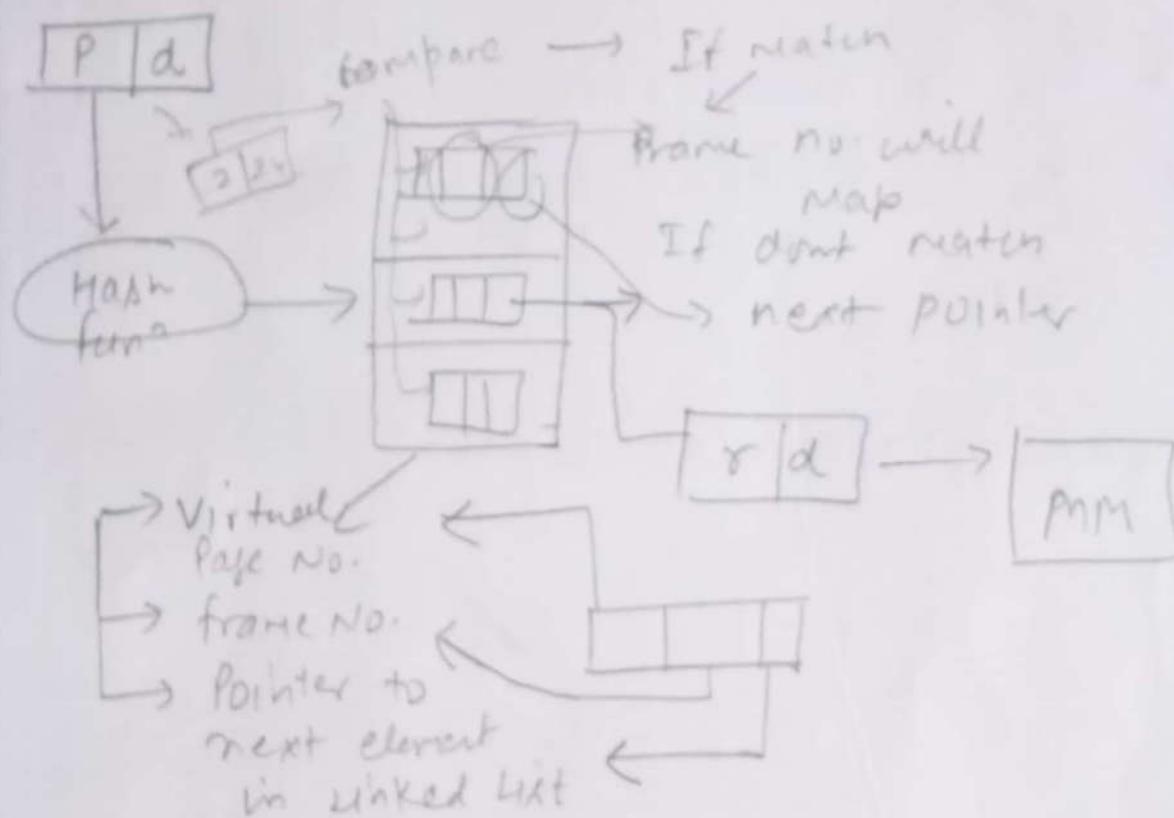
Pid: - Process id , P: - Page Number

- * It will work on pages of multiple process

(III)

Hashed Page Table :-

↳ used if Logical Address space $>$ 32 bits



- * 3 entries are there for each element in Hash table.
- * If CPU generates address, it is compared with first entry of Hash table, if Page no. = Virtual Page No., frame no. (2nd entry) used to map physical address.
- * If no match - go to next pointer location (3rd entry)



Swami Keshvanand Institute of Technology, Management & Gramothan,

Ramnagar, Jagatpura, Jaipur-302017, INDIA

Approved by AICTE, Ministry of HRD, Government of India

Recognized by UGC under Section 2(f) of the UGC Act, 1956

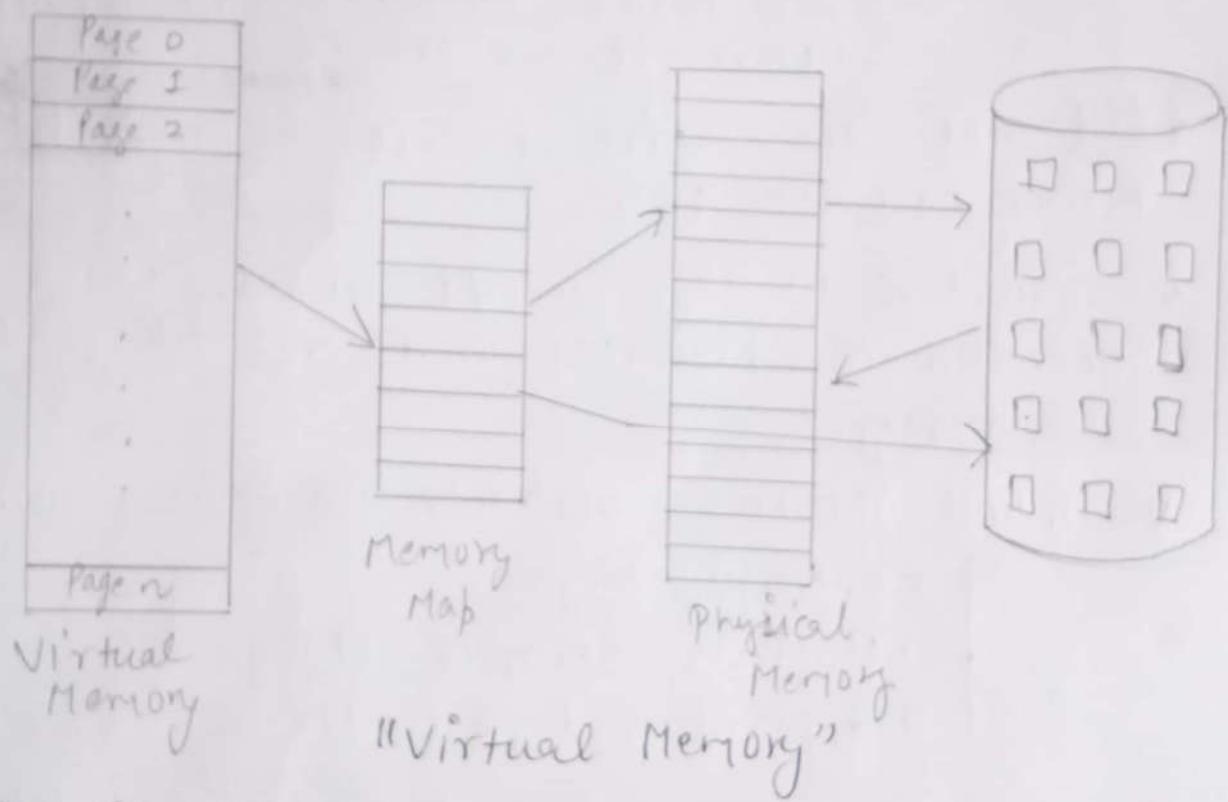
Tel.: +91-0141-5160400 Fax: +91-0141-2759855

E-mail: info@skit.ac.in Web: www.skit.ac.in

Virtual Memory :-

- * All memory mgmt strategies discussed upto so far require:-
 - All Entire process be in memory before it can execute.
- * But not the entire process required in memory all the time.
- * So, virtual memory concept arises, which allows execution of processes that are not completely in memory.
 - Virtual Memory abstracts the main memory (MM)
 - extremely large
 - Uniform, array of storage
 - separating Logical Memory
- * Even when the entire program is needed, it may not all be needed at the same time. So, the ability to execute a program that is only partially in memory is acceptable.
- * Virtual Memory is larger in size as compared to MM, it may confer to following benefits:-
 - Program would no longer be constrained by the amount of physical memory available
 - More programs can run at same time
 - Less I/O would be needed to load/swap.

* Virtual Address Space — Refers to logical (virtual) view of how a process is stored in memory.

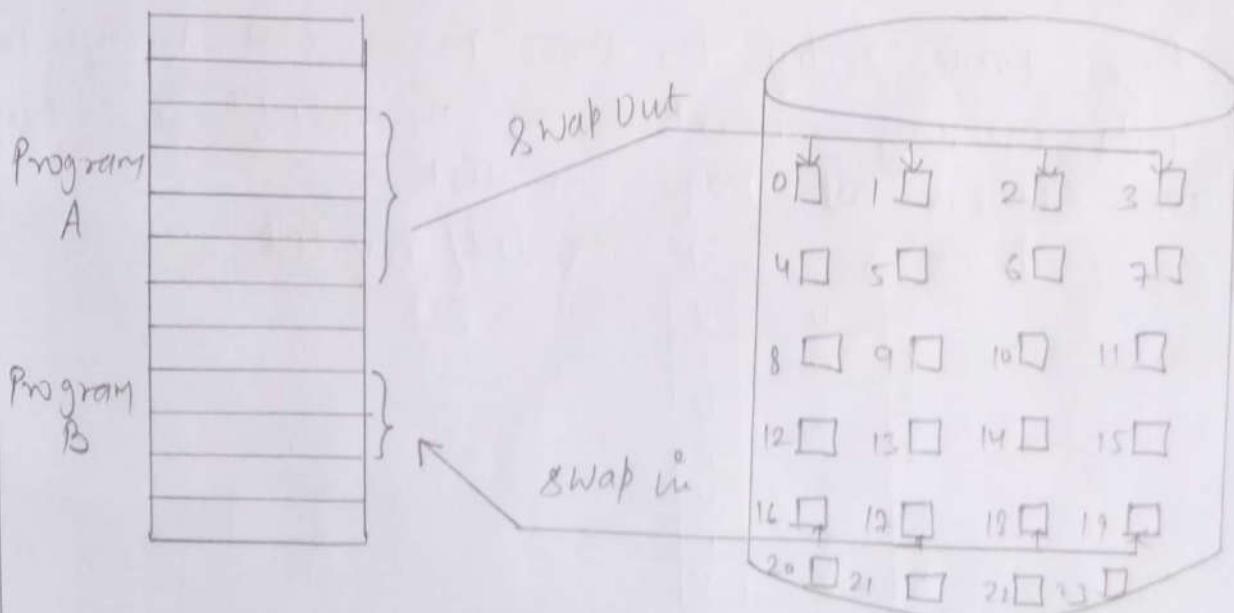


Benefits of VM :-

- Separation of Logical Memory from Physical
- System Libraries can be shared by several processes.
- Processes can share memory.
- Pages can be shared.

Demand Paging → In traditional execution environment the entire program needs to be loaded into memory.

- * But we may not need the entire program in memory at same time.
- * Instead of loading the entire program, an alternative strategy is to :-
 - ↳ load pages only when they are needed (i.e. on demand).
- * With demand-paged virtual memory, pages are loaded only when they are demanded during program execution.





Swami Keshvanand Institute of Technology, Management & Gramothan,

Ramnagar, Jagatpura, Jaipur-302017, INDIA

Approved by AICTE, Ministry of HRD, Government of India

Recognized by UGC under Section 2(f) of the UGC Act, 1956

Tel.: +91-0141-5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in

- * When we want to execute the process, we swap it into memory, rather than swapping the entire process.
- * This is done by "pager".
- * Pager brings only those pages which are required. This time Page Table's Valid/invalid bit play vital role.
 - ↳ If bit - valid : \rightarrow Associated Page is legal & in memory.
 - ↳ If bit-Invalid: \rightarrow Either the page is not legal or it is on disk.
- * Page Table entry for the page - brought into memory - marked as "valid".
- * Page Table entry for the page - Not brought into memory - marked as "invalid" or contains the address of page on disk.

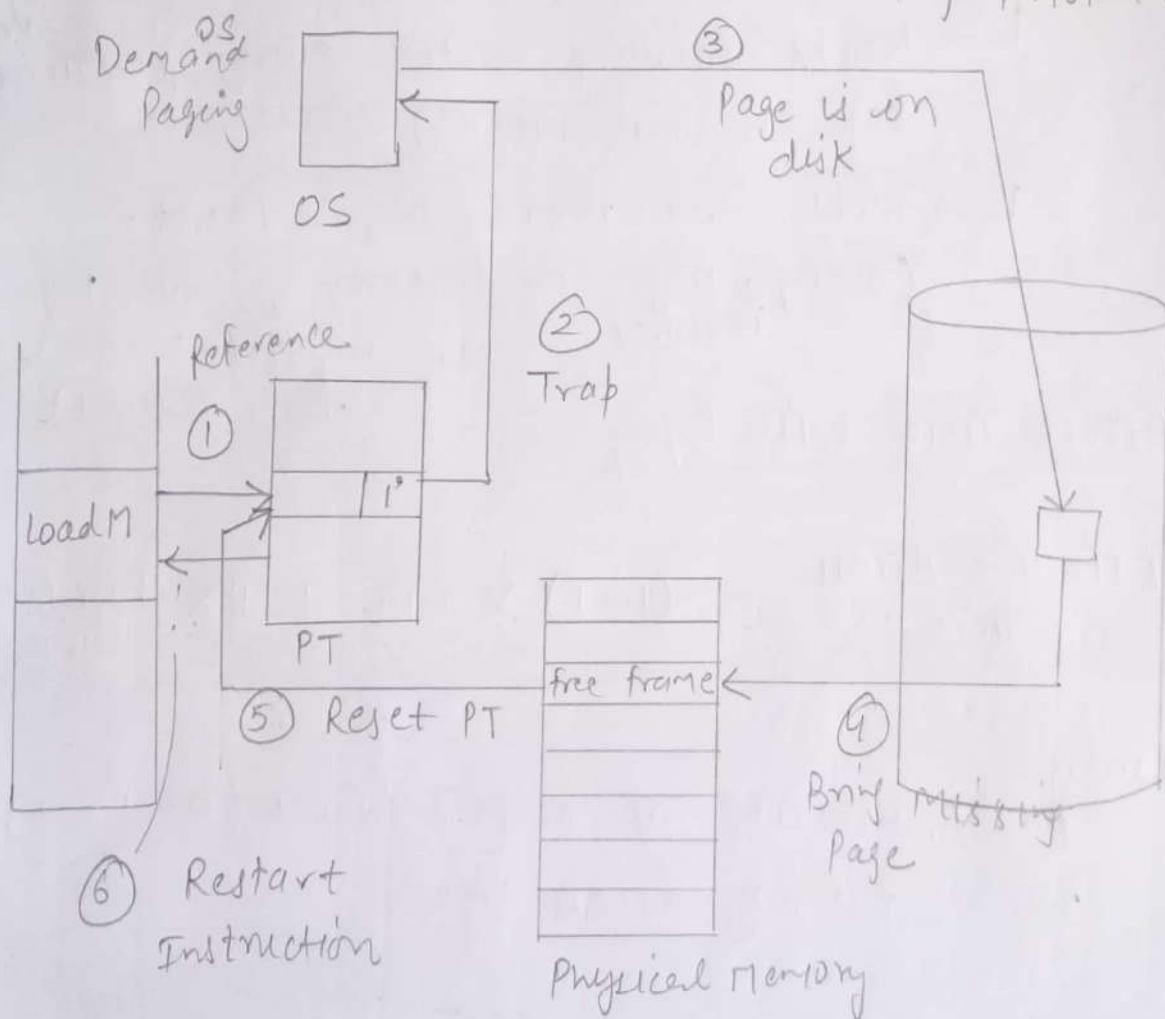
frame		Valid/ Invalid
0	4	V
1	9	I
2	8	V
3	6	I
:	:	:
8	2	V

Page Table

* What if the required page is not brought into memory?

↳ "Access to an Invalid Page causes"

↓
 Page fault → occurs when we try to access the page, not in memory



"Steps to Handle Page fault"



Pure Demand Paging:-

- ↳ Start execution with no pages in memory.
- ↳ When first instruction is accessed
 - ↳ Process immediately fault for page.
 - ↓
 - then, that page — brought into memory & is now executed
 - ↳ Keeps faulting until every required page is in memory.
 - ↳ Now, no more page fault.
(every page in memory, no page required is brought which is not required).

Access Time with Paging:-

$$\text{Effective Access Time} = (1-p) \times ma + p \times \text{Page fault Time}$$

Where,

p — Probability of a page fault ($0 \leq p \leq 1$)

ma — memory access time



Problem :- With an average ~~fix~~ page - fault service time of ~~8ms~~ & memory access time of ~~8~~ 200 ns. Calculate EAF.

$$EAF = (1-P) \times (200) + P(8ms)$$

$$EAF = (1-P) \times 200 + P(8,00,000)$$

$$EAF = 200 + 7999800 \times P$$

i.e. $\Rightarrow EAF \propto P$

- * A little bit blocking of process is present, which can create slightly slow system.
- * Assume there are no free frames in RAM, so presented page in MM are of 2 types,-
 - ↳ modified (change in RAM not in disk)
 - ↳ Non-modified (same copy available on disk & RAM)
- * Dirty bit is used for identifying modified / non-modified.



Page Replacement :-

- * Need → When a user wants to access a page, not brought in memory — Page fault occurs.
- * Now, OS determines where the page is residing on disk, but finds there are no free frames, i.e. all frames in use.
- * So, it requires Page Replacement, i.e. replace the page (which is not getting used) residing in memory by the page required to be loaded into memory.

(1) FIFO:- Replace the page on "FIFO" basis.

* Reference String:- Specific string of memory references.

(2) Optimal Page Replacement :-

↳ Replace the page with page which will be used very later.

↳ Go in ~~opt~~ forward direction.

(3) LRU Replacement :-

↳ Least Recently Used.

↳ Replace the page with page used least recently.

↳ Go in Backward Direction.



FIFO

Reference String	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
	7	7	7	2		2	2	4	4	4	0		0	0			7	7	7	
	0	0	0		3	3	3	2	2	2		1	1		1	1	0	0		
	1	1		1	0	0	0	3	3	3		3	2		3	2	2	2	1	

Total Page faults = 15

Optimal

Ref. String	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
	7	7	7	2		2	2		2	2		2		2		2		7		
	0	0	0		0	0	0	4		0		0		0		0		0		
	1	1		1	3	3	3		3	3		3		3		3		1		

Total Page faults = 9

LRU

Ref. String	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
	7	7	7	2		2		4	4	4	0		1		1		1			
	0	0	0		0	0		0	0	0		3	3		3	0	0			
	1	1		1	3	3		3	2	2		2		2		2	2	7		

Total Page faults = 12

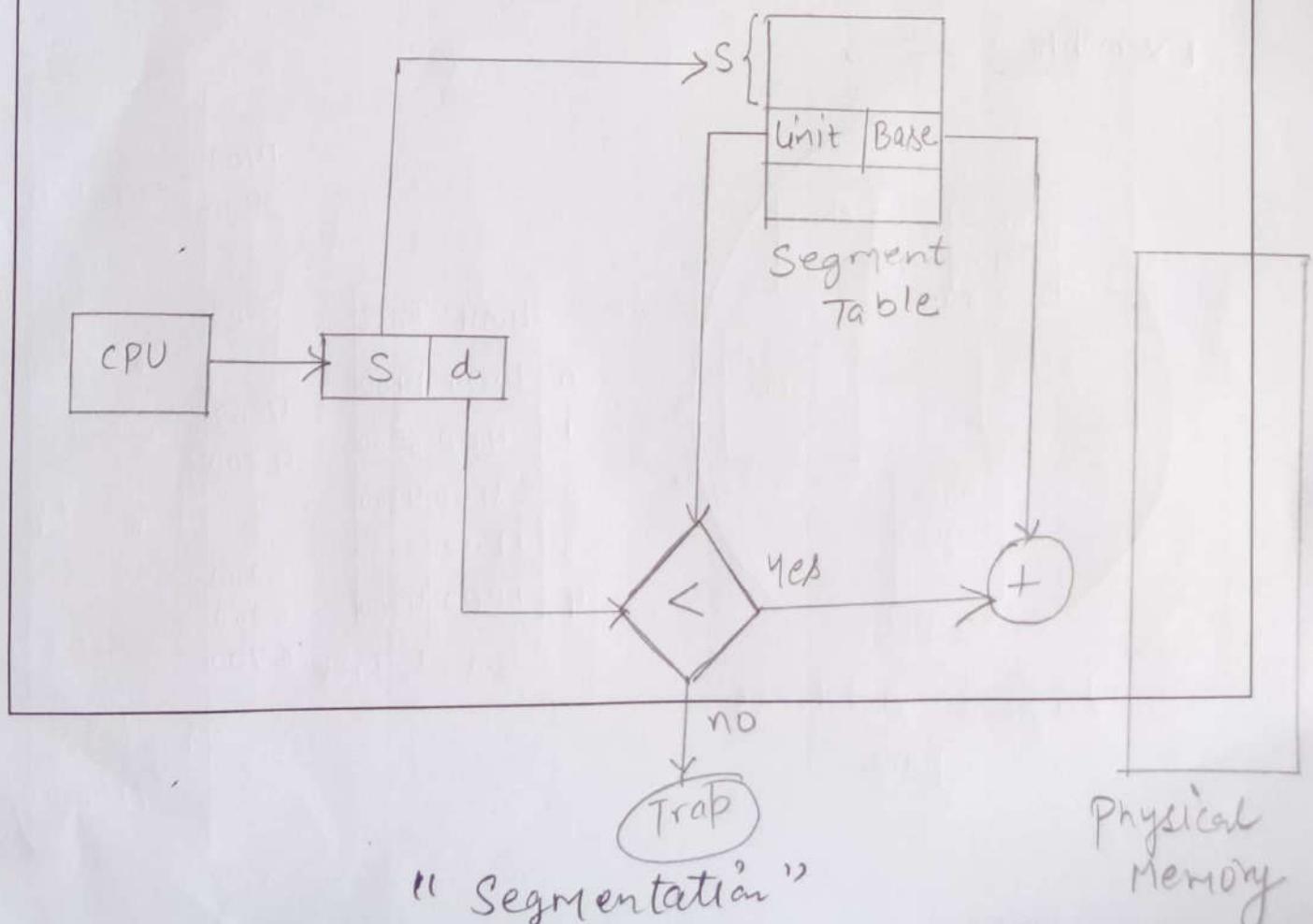
Optimal

Ref. String	7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
	7	7	7	2		2		2		2		2		2		2		2	
	0	0	0		0	0		0		0		3	3		3	0	0		
	1	1		1	3	3		3	2	2		2		2		2	2	7	

Segmentation :-

- * A Memory mgmt scheme in which
 - ↳ Logical Address space - A collection of Segments
 - ↳ Each segment has:-
 - ↳ Segment name (segment number)
 - ↳ Offset

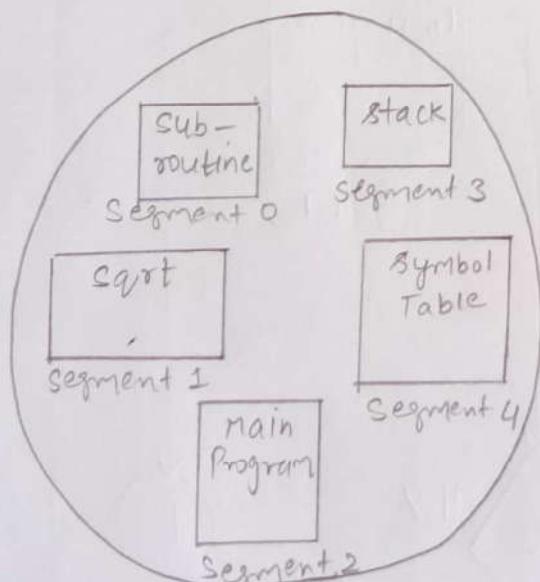
Logical Address : <segment Number, offset>





- * Each entry in segment Table has:-
 - Segment Base :- Starting Physical Address (Base Add) Where segment resides in memory
 - Segment limit :- Length of segment
- * Offset d must be b/w 0 & segment limit. If it is less than segment limit, then add it to segment base → "PA".
- * If not, trap will be there.

Example:-



Logical Address Space

	Limit	Base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

Segment Table

1400	Segment 0
2400	
3200	Segment 3
4300	
4700	Segment 2
5700	
6300	Segment 4
6700	Segment 1

Physical Memory



* Find out Mapping of following logical addresses:

① LA(2, 53) :- Segment Number - 2.
offset - 53.

$$\boxed{PA = \text{Base Address} \text{ of segment} + \text{offset}}$$

* first check ($53 < 400$): - True

$$PA = 4300 + 53 = 4353.$$

② LA(3, 852) :- $852 < 1100$ ✓

$$PA = 3200 + 852 = 4052.$$

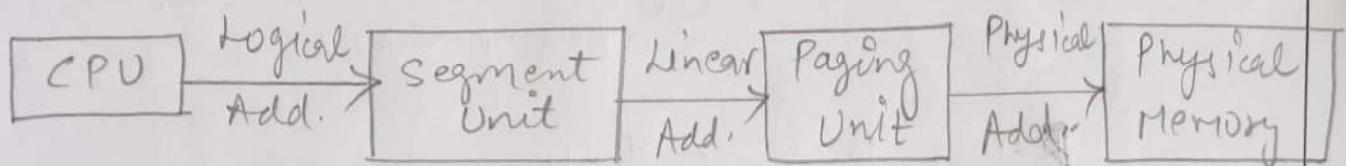
③ LA(0, 1222) :- $1222 > 1000$ ✗

↪ Can't be mapped to PA.

* Paging :- Non Contiguous Allocation Policy
based on "fixed size Partitioning".

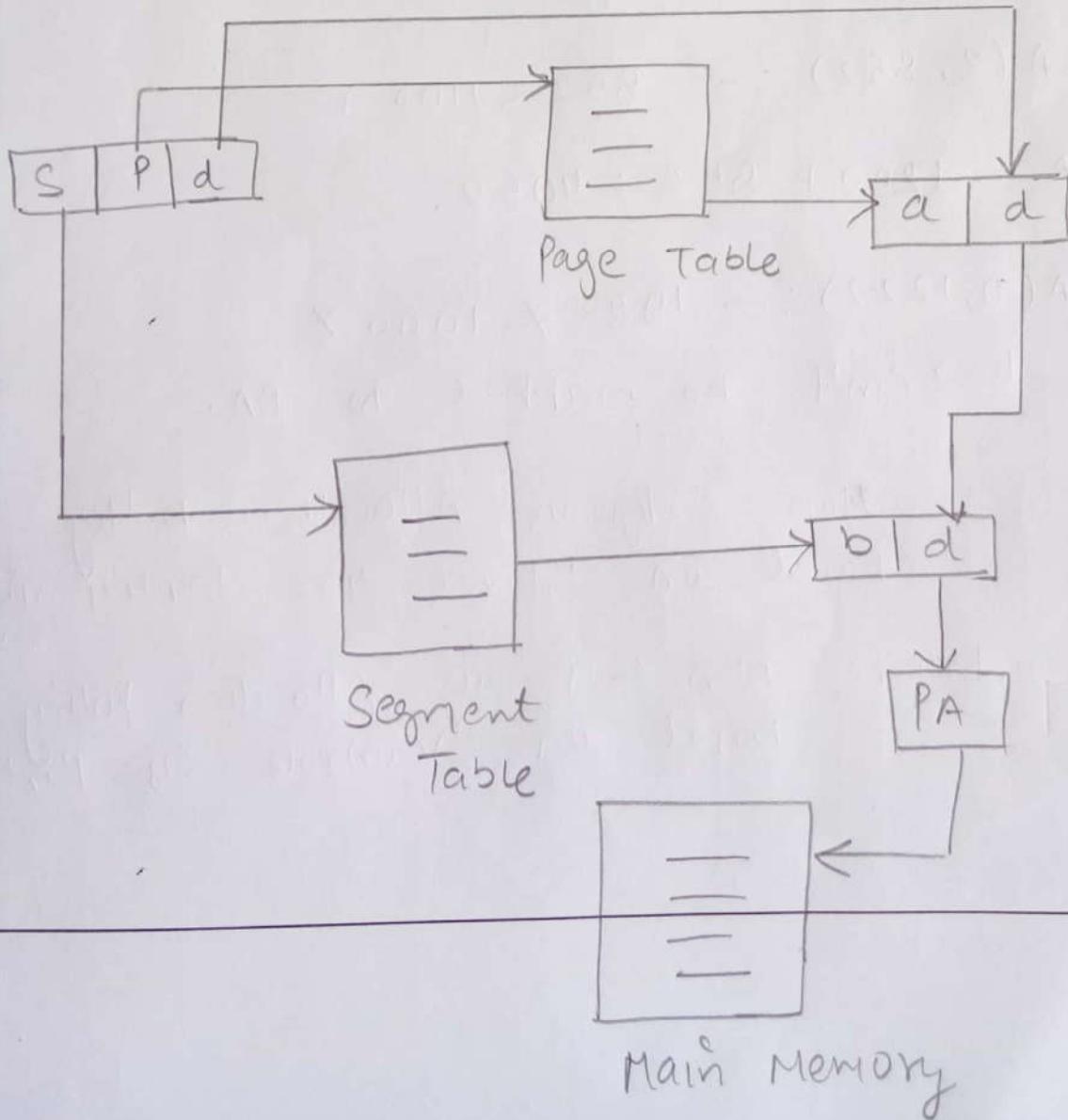
* Segmentation :- Non Contiguous Allocation policy
based on "Variable Size Parti-"

Segmentation with Paging:-



Logical Address \rightarrow Physical Address

* Intel Pentium used this Concept.





FIFO

String	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
	7	7	2		2	2	4	4	4	0		0	0	0	0	0	0	7	7	7
	0	0	0		3	3	3	2	2	2		1	1	1	1	1	1	0	0	0
	1	1	1		0	0	0	3	3	3		3	2	2	2	2	2	2	2	1

Total Page faults = 15

Optimal

Ref. string	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
	7	7	2		2	2		2		2		2		2		2		7		
	0	0	0		0	0		4		0		0		0		0		0		
	1	1	1		3	3		3		3		3		1		1		1		

Total Page faults = 9

LRU

Ref. string	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
	7	7	2		2			4	4	4	0		1	1	1	1	1	1		
	0	0	0		0			0	0	3	3		3	0	0	0	0	0		
	1	1	1		3			3	2	2	2		2	2	2	2	2	2	7	

Total Page faults = 12

Reference

Ref. string

LRU

Optimal

FIFO

RR

SC

FCFS

LRU

Optimal



Deadlock:- In a multi-programming system, a number of process compete for limited number of resources, and if a resource is not available at that instance, then process enters into waiting state.

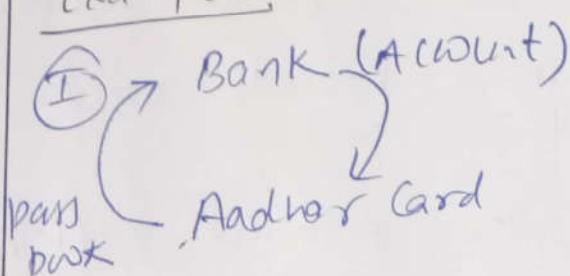
- * If a process unable to change its waiting state indefinitely (bcoz the resource requested by it, are held by another waiting process, then the system is said to be in "Deadlock")

System Model :-

I Every process will

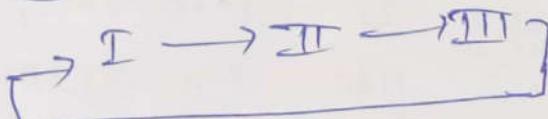
examples:-

(i)



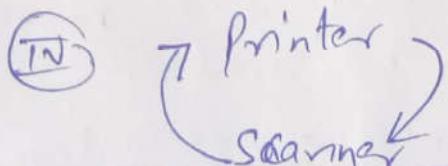
(ii)

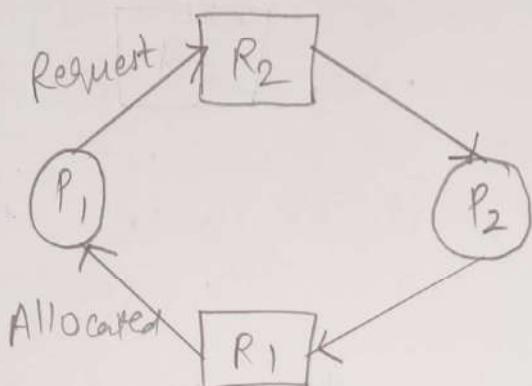
govt. offices.



(iii) A person having plate only & another having spoon only to eat food.

(iv)





"Resource Allocation Graph"

* Process $\xrightarrow{\text{Request}}$ Resource

* Resource $\xrightarrow{\text{Allocat'n}}$ Process

System Model :-

- Ⅰ Every process will request for the resource.
- Ⅱ If interested, only then process will use the resource.
- Ⅲ Process must release the resource after use.

Necessary Conditions of Deadlock:-

- ① Mutual exclusion - At least one resource type in system, which can be used in non-shareable mode i.e. mutual exclusion (one at a time).



② Hold & Wait:- A process is currently holding at least one resource and requesting another resource, which is being held by another process.

③ Non-Preemption:- A resource can not be preempted from a process by any other process. It gets released only voluntarily by the process holding it.

④ Circular Wait:- Every process must be waiting for a resource which is being held by another process.

$P_1 \rightarrow P_2 \rightarrow P_3$:- No Deadlock

$\rightarrow P_1 \rightarrow P_2 \rightarrow P_3$] :- Deadlock

Circular Wait

Deadlock Handling Methods:-

① Prevention:- Design such a system which violates at least one of 4 necessary conditions of deadlock & ensure independence from deadlock.

② Avoidance:- System maintains a set of data using which it takes a decision whether to maintain a new request or not to be in new state. (Banker's Algo).

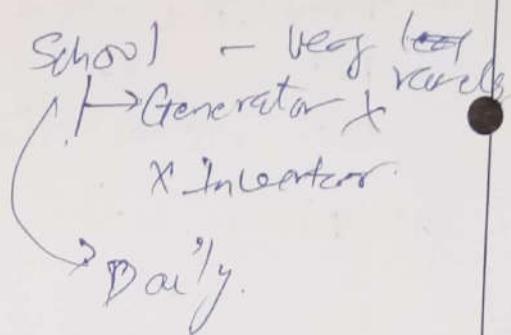


(3) Detection & Recovery :- Wait until deadlock occurs & once we detect it, we recover from it.

(4) Ignore :- We ignore the problem as if it does not exist.

Example:- Hospital

→ Generator



Polio - Severe & frequency

↳ Prevention - vaccinations

↳ more cost

When resources are fixed

↳ Prioritize them accordingly.

India = Polio free Nation

~~Prevention~~

Deadlock Prevention :- (Violate any one condition)

↳ creates possibility of no occurrence of deadlock.

↳ very much cost

↳ will be used only where consequences of deadlock are severe.

e.g. (Aircraft)

↳ No risk can be taken

→ (Hospital)

↳ No risk — ".

4 Conditions :-

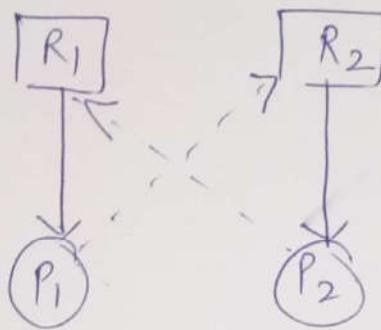
- ① ME
- ② HW
- ③ No-LP
- ④ CW

* Violate any one condition. (Remove), then we can guarantee, no deadlock can occur.

~~①~~ ME :- Non-shareable manner.

X

eg:-



-
- * Can't be violated,
 - * Whether a resource can be shareable among more than 1 process, depends on H/W properties of resource.

* but we are working on single printer only.

eg: - Printer

* Conclusion :- Mutual Exclusion can't be violated.

~~②~~ Hold & Wait :-

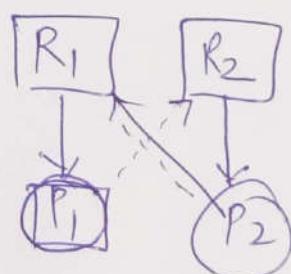
② Conservative approach :- Process is allowed to start execution if & only if it has ~~acq~~ acquired all resources.

↳ Less efficient

↳ Not implementable

✗ Easy

↳ Deadlock Independence



↳ In starting it is not known ~~whether~~ how many resources will be required.

② DO hold :- Process will acquire only desired resources, but before making any fresh request if it must release all resources that it currently held

total $\frac{7}{10}$

- ↳ Efficient
- ↳ Implementable

hold: - R, + Q₄

③ Wait timeout :- We place a maximum time upto which a process can wait after which process & release all the holding resources, & exit.

↳ time for wait (after it process will have to release all resources).

III NO-Preemption :-

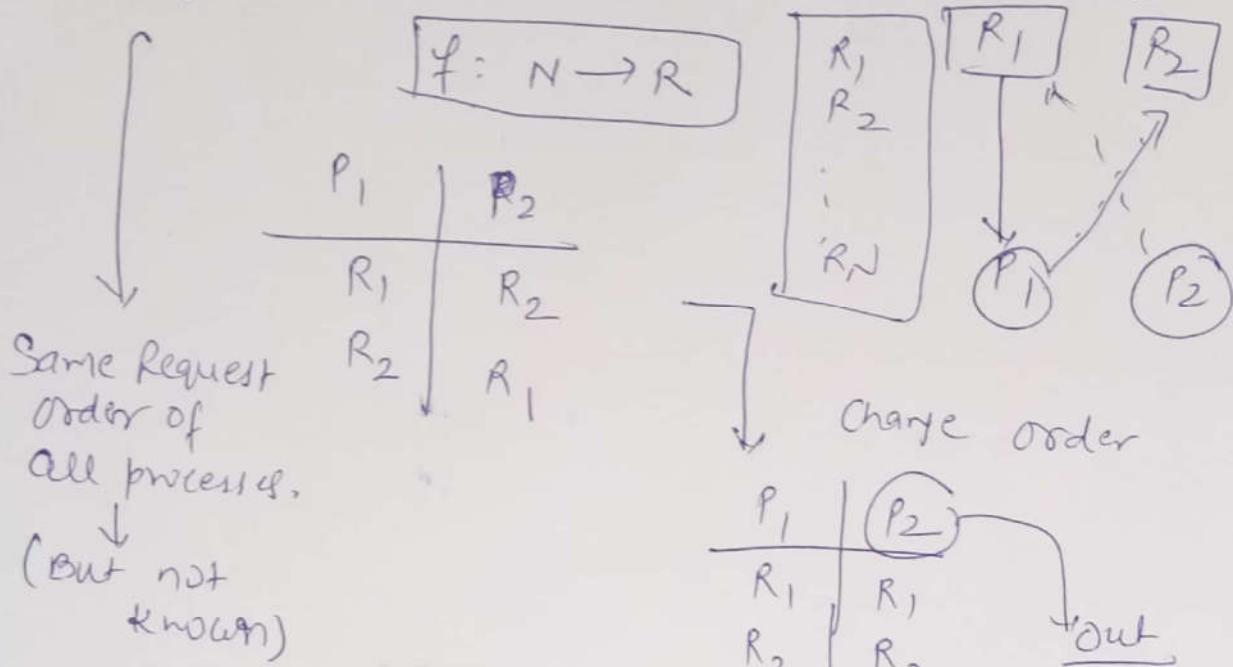
* Forcefull Preemption :- allow a process to forcefully preempt the resource hold by another process.

- ↳ Used by high priority processes or System processes.
- ↳ The processes which are in waiting state must be selected as a victim instead of process in running state.
→ Police vehicle (no issue)
- Traffic Jam
→ Spoon → When not eating.
↓ ↑
 preempt

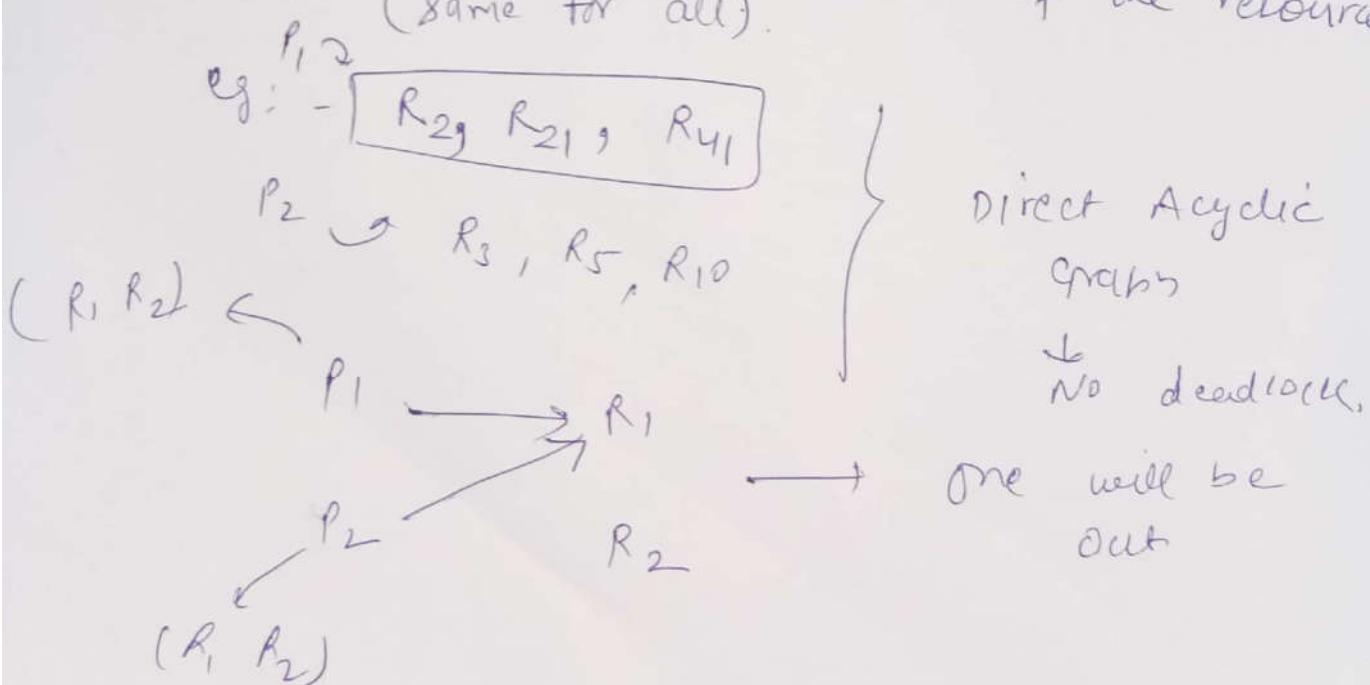
(iv) Circular Wait:

$$P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow \dots$$

- * Circular Wait can be eliminated by first going a natural no. to every resource



- * Allow every process to increase or decrease number. (same for all).
 make a request either only in the order of the resource



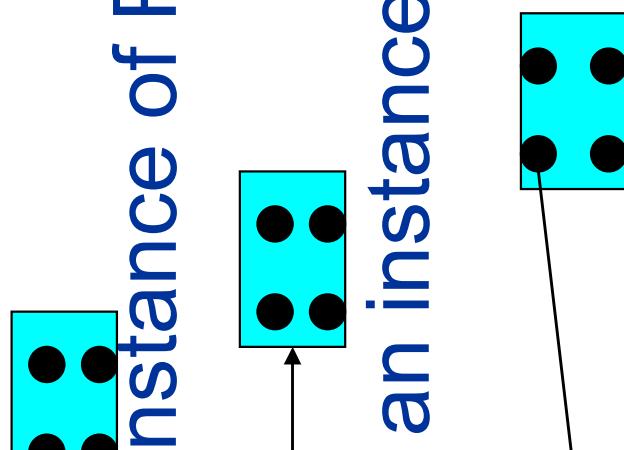
- * If process require a lesser no. (in case of increasing order), then it must first release all resources larger than required No. $R_{11}, R_{21}, R_{31}, R_{41}$
- ↳ More Practical Approach,

Resource Allocation Graph

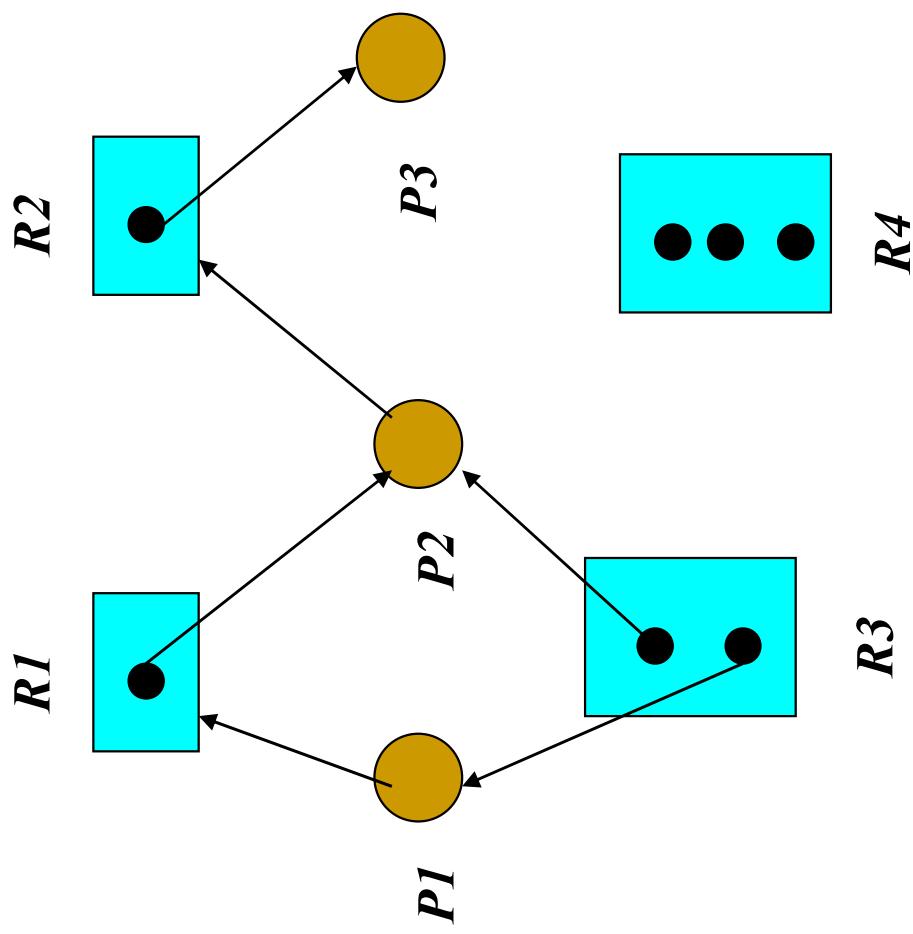
- Deadlocks can be described in terms of a directed graph called a **system resource-allocation graph**.
- A set of vertices V and a set of edges E
- V is partitioned into 2 types
 - $P = \{P_1, P_2, \dots, P_n\}$ - the set of processes in the system
 - $R = \{R_1, R_2, \dots, R_n\}$ - the set of resource types in the system
- Two kinds of edges
 - Request edge - Directed edge $P_i \rightarrow R_j$
 - Assignment edge - Directed edge $R_j \rightarrow P_i$

Resource Allocation Graph

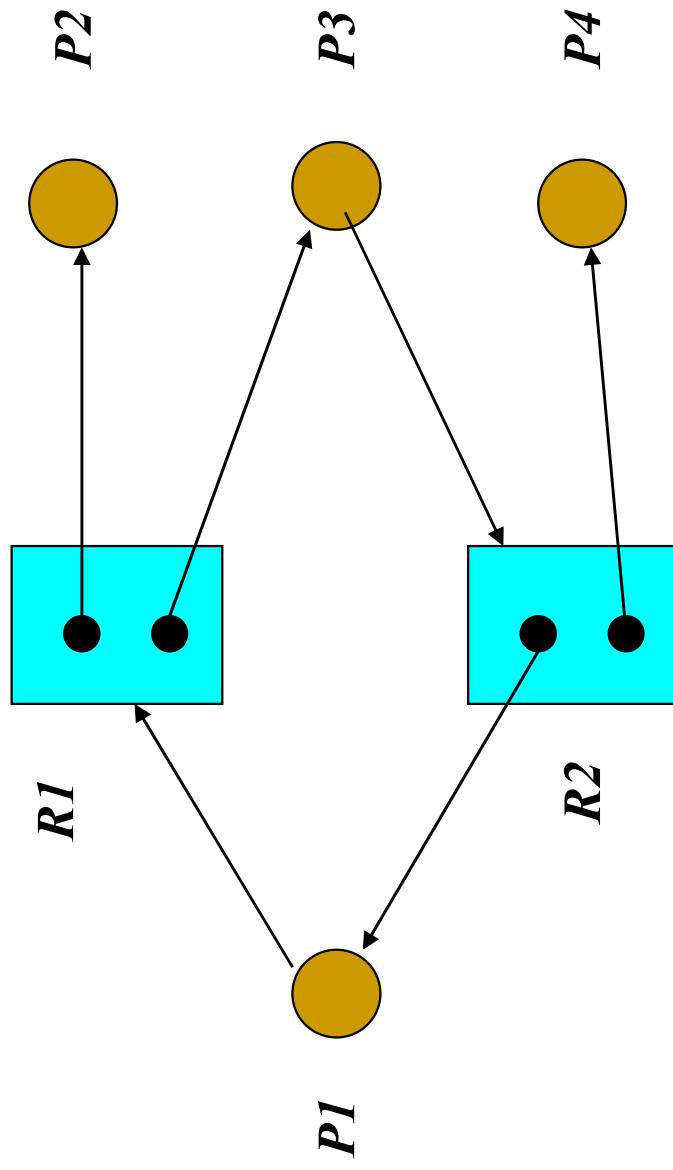
- Process
- Resource type with 4 instances
- P_i requests instance of R_j
- P_i is holding an instance of R_j



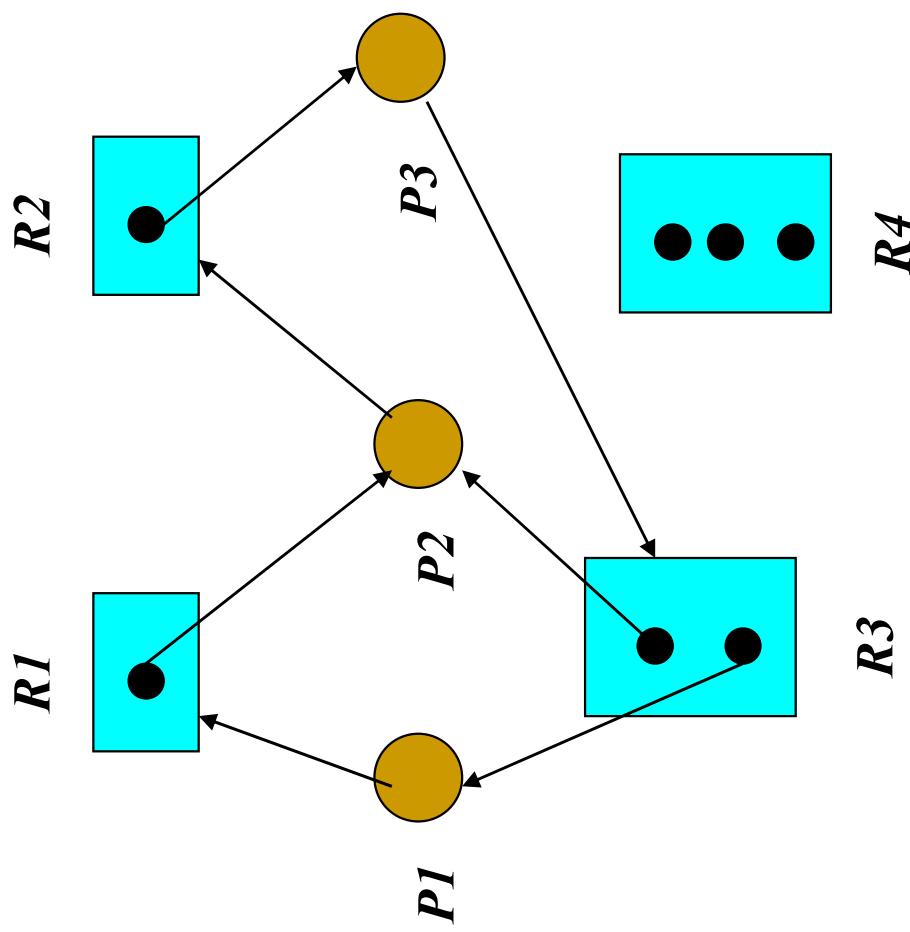
Graph with no cycles



Graph with cycles



Graph with cycles and deadlock



Basic facts

- If graph contains no cycles
 - NO DEADLOCK
- If graph contains a cycle
 - if only one instance per resource type, then deadlock
 - if several instances per resource type, possibility of deadlock.

Methods for handling deadlocks

- There are three ways to handle the deadlock
 - 1. Use protocols to **prevent** or **avoid** deadlocks, ensuring that the system will never enter a deadlock state.
 - 2. Allow the system to potentially enter a deadlock state, **detect** it and then **recover**.
 - 3. Ignore the problem and imagine that deadlocks **never occur in the system**;
 - Used by many operating systems, e.g. UNIX and Windows

Deadlock Management

- **Prevention**
 - Design the system in such a way that deadlocks can never occur. i.e ensure that at least one of necessary conditions can't hold.
- **Avoidance**
 - Impose less stringent conditions than for prevention, OS be given in advance additional information concerning which resource a process will request and use during its lifetime. allowing the possibility of deadlock but bypassing it as it occurs.
- **Detection**
 - Allow possibility of deadlock, determine if deadlock has occurred and which processes and resources are involved.
- **Recovery**
 - After detection, clear the problem, allow processes to complete and resources to be reused. May involve destroying and restarting processes.

Deadlock Prevention

- If any one of the conditions for deadlock (with reusable resources) is denied, deadlock is impossible.
- **Restrain ways in which requests can be made**
 - **Mutual Exclusion**
 - non-issue for sharable resources. e.g reading a file
 - It must hold for non-sharable resources. e.g Printer
 - **Hold and Wait - guarantee that when a process requests a resource, it does not hold other resources.**
 - Force each process to acquire all the required resources at once. Process cannot proceed until all resources have been acquired.
 - Low resource utilization, starvation possible

Deadlock Prevention (cont.)

- **No Preemption**

- If a process that is holding some resources requests another resource that cannot be immediately allocated to it, the process releases the resources currently being held.
- Preempted resources are added to the list of resources for which the process is waiting.
- Process will be restarted only when it can regain its old resources as well as the new ones that it is requesting.

- **Circular Wait**

- Impose a total ordering of all resource types.
- Require that processes request resources in increasing order of enumeration; if a resource of type N is held, process can only request resources of types $> N$.

Deadlock Avoidance

- Requires that the system has some additional apriori information available.
 - Simplest and most useful model requires that each process declare the maximum number of resources of each type that it 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 is defined by the number of available and allocated resources, and the maximum needs of the processes.

Safe state

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a **safe state**.
- System is in safe state if there exists a safe sequence of all processes.
 - A Sequence of processes $\langle P_1, P_2, \dots, P_n \rangle$ is safe, if for each P_i , the resources request that P_i can be satisfied by currently available resources + resources held by P_j with $j < i$.
 - If P_i resource needs are not available, P_i can wait until all P_j have finished.
 - When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate.
 - When P_i terminates, P_{i+1} can obtain its needed resources...

Safe State-Example

- Total Resources=12

Process	Maximum Demand	Allocated	Need
P1	10	5	5
P2	4	2	2
P3	9	2	7

- At time t_0 system in safe state and safe sequence is
 $<P2,P1,P3>$
- At time t_1 process P3 need one more resource and it is allocated to P3, then system in unsafe state--deadlocked

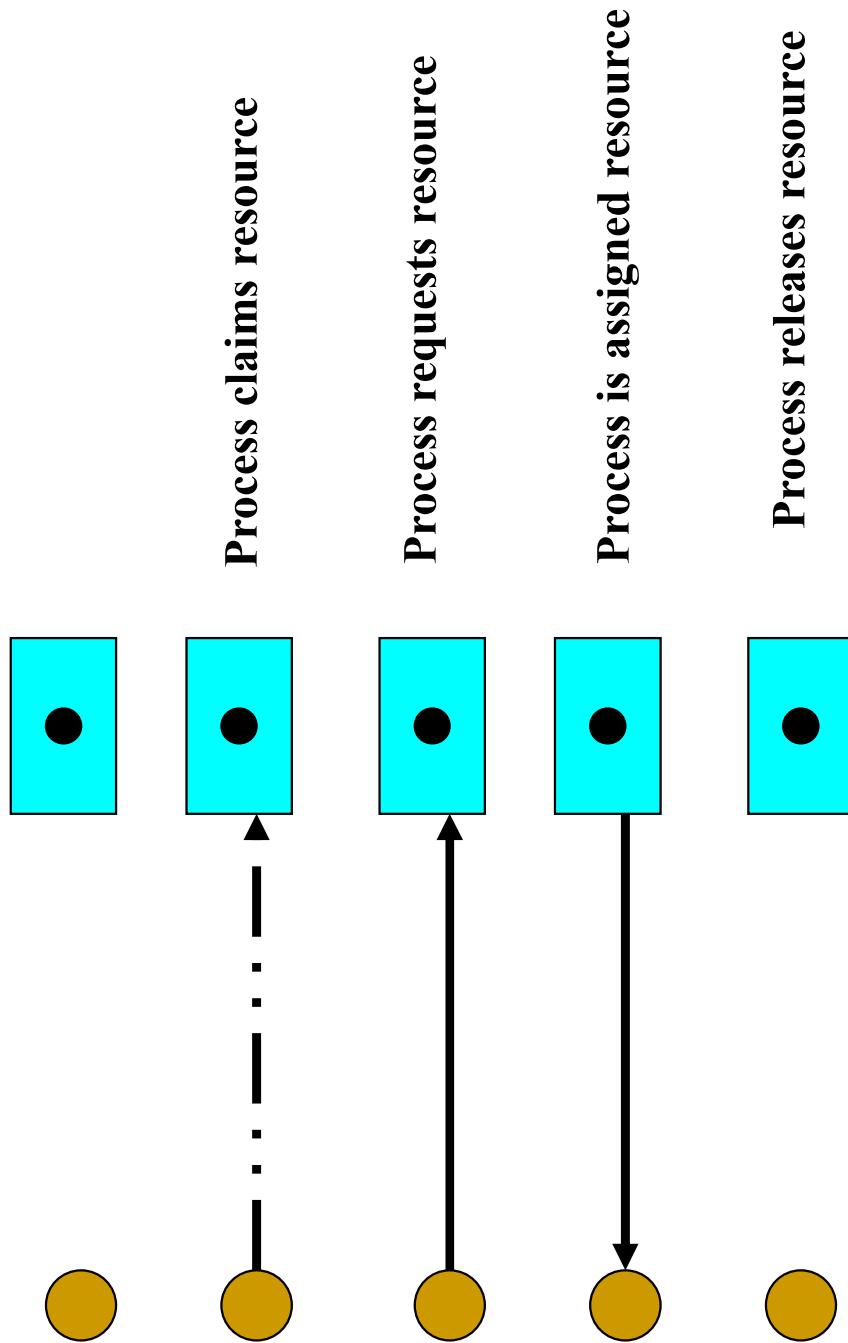
Basic Facts

- If a system is in a safe state □ no deadlocks.
- If a system is in unsafe state □ possibility of deadlock.
 - Avoidance □ ensure that a system will never reach an unsafe state.
 - Deadlock Avoidance algorithms
 - Resource Allocation Algorithms:
 - Use for single instance of resource
 - Banker's Algorithms
 - Use for Multiple instance of resource

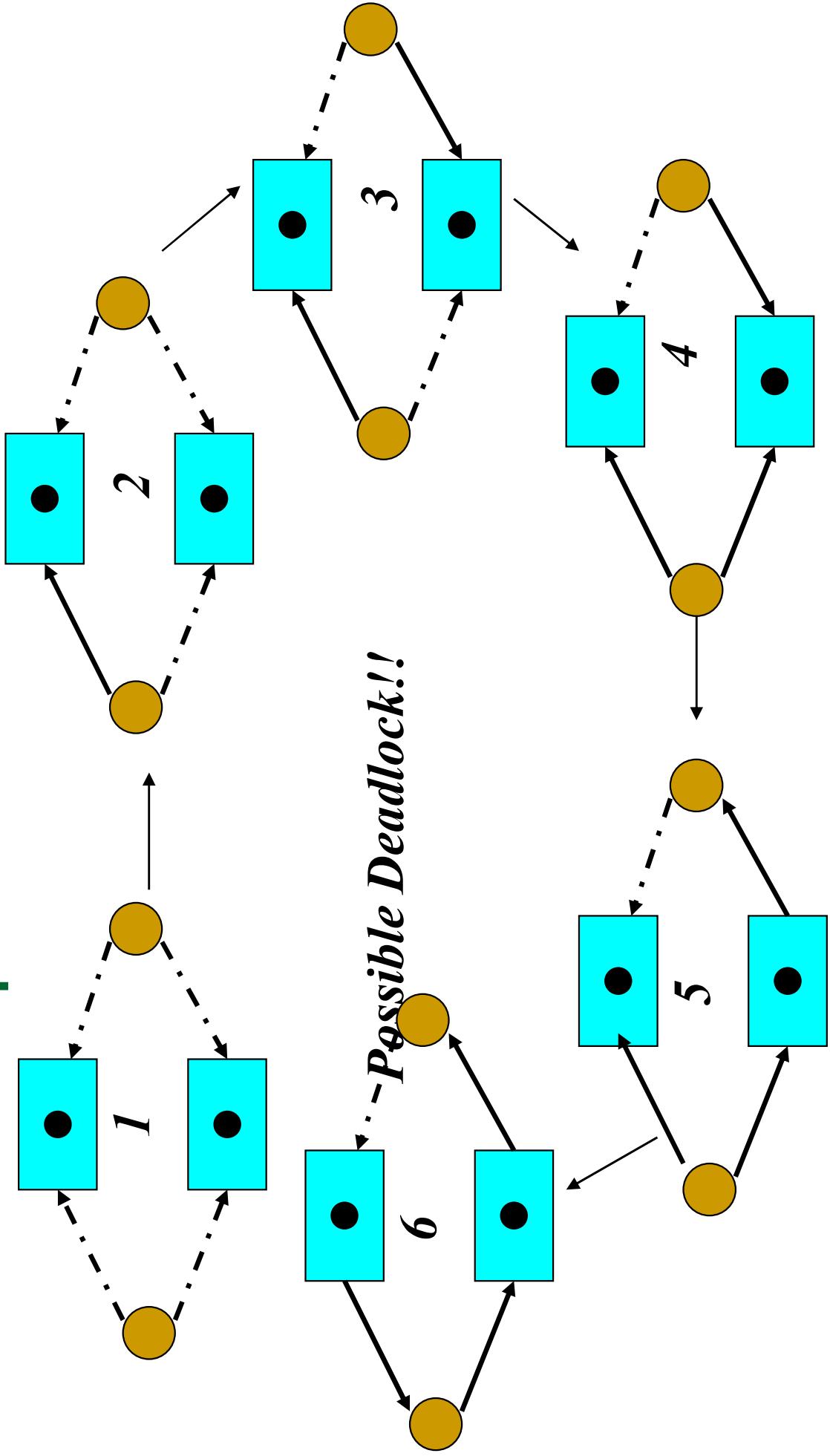
Resource Allocation Graph Algorithm

- Used for deadlock avoidance when there is only one instance of each resource type.
 - Claim edge: $P_i \square R_j$ indicates that process P_i may request resource R_j ; represented by a dashed line.
 - Claim edge converts to request edge when a process requests a resource.
 - Resources must be claimed a priori in the system.
- If request assignment does not result in the formation of a cycle in the resource allocation graph - safe state, else unsafe state.

Claim Graph



Claim Graph



Banker's Algorithm

- Used for multiple instances of each resource type.
- Name was chosen because this algorithm could be used in banking system:
 - To ensure that the bank never allocated its available cash in such a way that it could no longer satisfy the needs of all its customers.
- Each process must a priori claim maximum use of each resource type.
- When a process requests a resource then the system must determine whether the allocation of these resources will leave the system in safe state:
 - If yes then resources are allocated to requested process.
 - Else it must wait until some other process release enough resources.
- When a process gets all its resources it must return them in a finite amount of time.

Data Structures for the Banker's Algorithm

- Let $n = \text{number of processes}$ and $m = \text{number of resource types}.$
 - Available:** A vector of length m indicates the no. of available resources of each type. If $\text{Available}[j] = k$, there are k instances of resource type R_j available.
 - Max:** An $n \times m$ matrix define maximum demand of each process. If $\text{Max}[i,j] = k$, then process P_i may request at most k instances of resource type R_j .
 - Allocation:** An $n \times m$ matrix define the no. of resources of each type currently allocated to each process. If $\text{Allocation}[i,j] = k$, then process P_i is currently allocated k instances of resource type R_j .
 - Need:** An $n \times m$ matrix indicate the remaining resource need of each process. If $\text{Need}[i,j] = k$, then process P_i may need k more instances of resource type R_j to complete its task.
$$\text{Need}[i,j] = \text{Max}[i,j] - \text{Allocation}[i,j]$$

Safety Algorithm

- Let $Work$ and $Finish$ be vectors of length m and n , respectively. Initialize
 - $Work := Available$
 - $Finish[i] := false$ for $i = 1, 2, \dots, n$.
- Find an i (i.e. process P_i) such that both:
 - $Finish[i] == false$
 - $Need_i <= Work$
 - If no such i exists, go to step 4.
- $Work := Work + Allocation_i$
 - $Finish[i] := true$
 - go to step 2
- If $Finish[i] = true$ for all i , then the system is in a safe state.

Resource-Request Algorithm for Process P_i

- Determines whether the request is safely granted or not
 - Request_j = request vector for process P_i . If Request_{j[j]} = k, then process P_i wants k instances of resource type R_j .
- **STEP 1:** If Request(i) \mid Need(i), go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.
- **STEP 2:** If Request(i) \mid Available, go to step 3. Otherwise, P_i must wait since resources are not available.
- **STEP 3:** Pretend to allocate requested resources to P_i by modifying the state as follows:

```
Available := Available - Request (i);  
Allocation (i) := Allocation (i) + Request (i);  
Need (i) := Need (i) - Request (i);
```
- If safe \square resources are allocated to P_i .
- If unsafe \square P_i must wait and the old resource-allocation state is restored.

Example of Banker's Algorithm

- 5 processes
 - P0 - P4;
- 3 resource types
 - A(10 instances), B (5 instances),
and C (7 instances)
- Snapshot at time T0
- The content of the matrix *Need* is defined to be *Max - Allocation*.

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2
P1	2	0	0	3	2	2			
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
P4	0	0	2	4	3	3			

	Need		
	A	B	C
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1

- The system is in a safe state since the sequence
 $\langle P1, P3, P4, P2, P0 \rangle$ satisfies safety criteria

Example: P1 requests (1,0,2)

- Check to see that Request | Need
 - $((1,0,2) \mid (1,2,2)) \square$ true.
- Check to see that Request | Available
 - $((1,0,2) \mid (3,3,2)) \square$ true.
- Now change Need, Allocation and Available
- Executing the safety algorithm shows that sequence <P1, P3, P4, P0, P2> satisfies safety requirement.
- Can request for (3,3,0) by P4 be granted?
- Can request for (0,2,0) by P0 be granted?

	Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	4	3	2	3	0
P1	3	0	2	0	2	0			
P2	3	0	2	6	0	0			
P3	2	1	1	0	1	1			
P4	0	0	2	4	3	1			

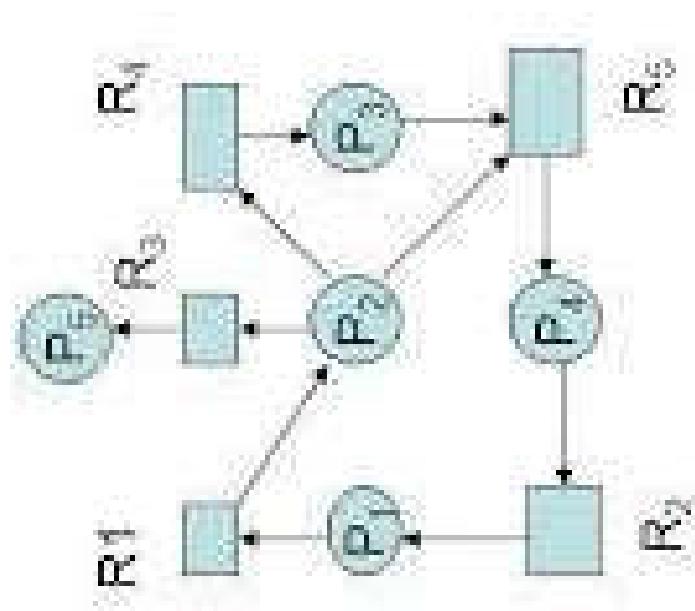
Deadlock Detection

- Allow system to enter deadlock state
 - An algorithm that examines the state of system to determine whether a deadlock has occurred (Deadlock Detection algorithms)
 - An algorithm to recover from deadlock (Recovery Scheme)
- Detection Algorithms
 - Single Instance of each resource type
 - Wait-for graph
 - Several instances of a resource type
 - Banker's Algorithm

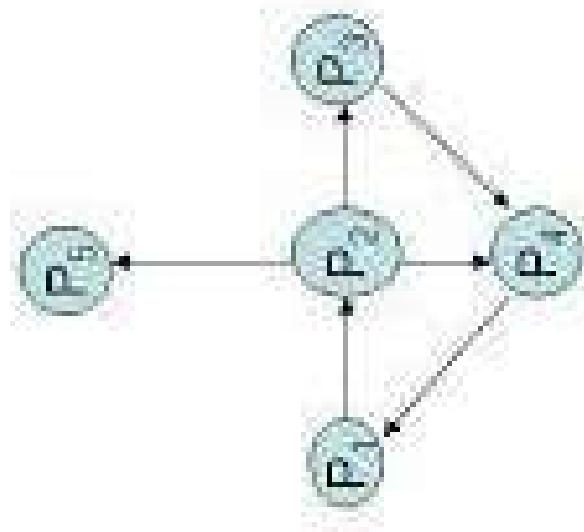
Single Instance of each resource type

- Maintain wait-for graph: if graph contain cycle then deadlock else not.
 - Nodes are processes
 - $P_i \sqcap P_j$ if P_i is waiting for P_j .
- Periodically invoke an algorithm that searches for a cycle in the graph.
- 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.

Wait-for-Graph



Resource allocation graph



Corresponding wait-for Graph

Several instances of a resource type

- Banker's algorithms
- Data Structures used

- *Available*: Vector of length m . If $Available[j] = k$, there are k instances of resource type R_j available.
- *Allocation*: $n \times m$ matrix. If $Allocation[i,j] = k$, then process P_i is currently allocated k instances of resource type R_j .
- *Request* : An $n \times m$ 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 .

Deadlock Detection Algorithm

- **Step 1:** Let $Work$ and $Finish$ be vectors of length m and n , respectively.
Initialize
 - $Work := Available$
 - For $i = 1, 2, \dots, n$, if $Allocation(i) \sqsubseteq 0$, then $Finish[i] := false$, otherwise $Finish[i] := true$.
- **Step 2:** Find an index i such that both:
 - $Finish[i] = false$
 - $Request(i) \sqsubseteq Work$
 - If no such i exists, go to step 4.
- **Step 3:** $Work := Work + Allocation(i)$
 - $Finish[i] := true$
 - go to step 2
- **Step 4:** If $Finish[i] = false$ for some $i, 1 \leq i \leq n$, then the system is in a deadlock state. Moreover, if $Finish[i] = false$, then P_i is deadlocked.

Algorithm requires an order of $m \cdot (n^2)$ operations to detect whether the system is in a deadlocked state.

Example of Detection Algorithm

- 5 processes - $P_0 - P_4$; 3 resource types - $A(7 \text{ instances}), B(2 \text{ instances}), C(6 \text{ instances})$
- Snapshot at time T_0 : $\langle P_0, P_2, P_3, P_1, P_4 \rangle$ will result in $Finish[i] = \text{true}$ for all i .
- P_2 requests an additional instance of type C.
- State of system
 - Can reclaim resources held by process P_0 , but insufficient resources to fulfill other processes' requests.
 - Deadlock exists, consisting of P_1, P_2, P_3 and P_4 .

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

Detection-Algorithm Use

- 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
- How often --
 - Every time a request for allocation cannot be granted immediately
- Allows us to detect set of deadlocked processes and process that “caused” deadlock. Extra overhead.
- Every hour or whenever CPU utilization drops.
 - With arbitrary invocation there may be many cycles in the resource graph and 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 the 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 from that state.
- Starvation
 - same process may always be picked as victim; include number of rollback in cost factor.

Combined approach to deadlock handling

- Combine the three basic approaches
 - Prevention
 - Avoidance
 - Detection
- allowing the use of the optimal approach for each class of resources in the system.
- Partition resources into hierarchically ordered classes.
 - Use most appropriate technique for handling deadlocks within each class.

Operating System

UNIT - IV

Unit - IV
File Management

Priyanka Sharma

Assistant Professor

Department of Computer Science



What is File?



File Management

Examples of files :

- Text File
- Source File
- Object File



File Attributes

Parameters used to keep track of files in OS.

- Name
- Identifier
- Type
- Location
- Size



File Attributes (Cont...)

- Protection
- Time & Date
- User Id



File Operations



- **Creating a file**
 - ABC.txt
- **Writing into a file**
 - write(a, “Text which is being written”)
- **Reading a file**
 - read(a)

File Operations (Cont...)

- Repositioning within a file
- Deleting a file
- Truncating a file



File Operations (Cont...)

- **Append**
- **Rename**
- Some other operations can be done on a file by combining the primitive operations. Such as –
 - Copying a file
 - Get & set various attributes of a file
 - Determine the status of a file (file length, file attributes etc).



File Operations (Cont...)

- Most of above mentioned operations requires :
 - Searching the directory for the named file
- Searching makes processing slow.
- To avoid searching every time ,most system requires :
 - **open()** System call
- OS maintains a table : **open-file table**



Open() System Call

- Open() operation takes :
 - File name
 - Searches the directory, copying the directory entry into open-file table.
- Available modes : create, read only, read write, append etc.
- Returns a pointer to the entry in open-file table.



Open-File Table

- When a file operation is required to be executed, the file specified via an index into open-file table.
 - no searching is required
- When file is no longer used : respective entry is removed from open-file table by OS.
- Uses the information **open count**.



Open-File Table (Cont...)

- open count :**

- Number of opened files
 - Decreases with each close() operation
 - When turns 0 , then file is no longer in use. (Entry gets removed from table)
- Several other informations for a open file
 - File Pointer
 - Disk location of file
 - Access rights



File Locking

- Enables access for one process and prevents others.
 - **Shared Lock** : Reader lock (several processes can acquire)
 - **Exclusive Lock** : Writer lock (only one processes can acquire)



File Types

Type of File	File Extension	Description
Executable	exe, com, bin	Machine Language Program (Ready to run)
Object	obj , o	Machine language (compiled), no linking done
Source Code	c, app, java , perl	Set of programs in various programming languages
Batch	bat, sh	Command interpreter commands
Markup	xml, html, tex	Documents, textual data



File Types (Cont...)

Type of File	File Extension	Description
Library	lib, a , so, dll	Routine Libraries
Word Processor	xml	Word processor formats
Archive	rtf, docx	Group of related files (compressed/storage)
Print/view	gif, pdf, jpeg	Binary/ASCII files in printable or viewable format.
Multimedia	mpeg, mov, mp3, mp4, avi	Binary file containing audio or A/V information



File Structure

- Disk space always allocated in blocks.
- Well defined block size (size of sector)
- Disk I/O – performed in units of one block (Physical record)
- All blocks are of same size.
- Structure – various logical records are packed in physical blocks.



File Structure (Cont...)

- No. of logical records (in each block) determined by :

- Logical Record Size

- Physical Block Ssize

- Packing Technique



File Structure (Cont...)

- Some amount of **Internal Fragmentation** exists.

Example –

- Block Size : 1024 bytes (1KB)
- A file of 3000 bytes would be allocated to 3 blocks (3,072 bytes)
- Resulting in wastage of 72 bytes (Internal Fragmentation)
- Larger the block size, higher the internal fragmentation



File Access Methods

- Sequential Access
- Direct Access
- Other Access Methods



Sequential Access

- Information Processing – Sequential
- Mostly used by editors and compilers
- Performs well on sequential access devices and random access devices
- Based on the tape model of file

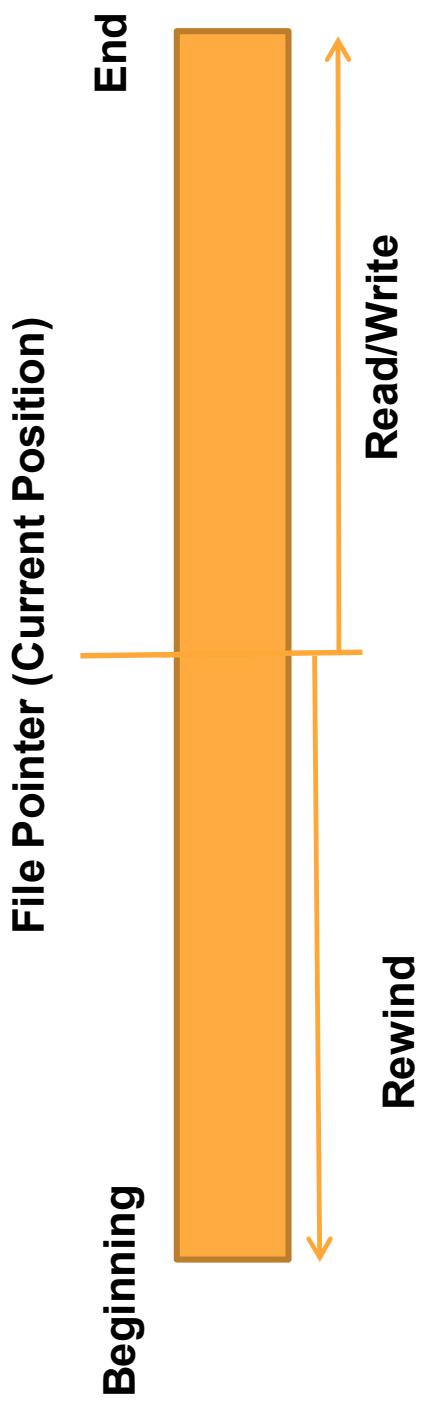


Sequential Access (Cont...)

- OS read the file word by word.
- A pointer is maintained which initially points to the base address of the file.
- If the user wants to read first word of the file then the pointer provides that word to the user and increases its value by 1 word.
- This process continues till the end of the file.



Sequential Access (Cont...)



Sequential Access (Cont...)

- **File operations :**

- Read
- Write



Sequential Access (Cont...)

- **Write Operation : write_next ()**
 - Appends to the end of the file
 - Advance the file pointer to the end of newly written material (i.e. new end of file)



Sequential Access (Cont...)

- **Reset**
 - Resets to the beginning

- **Skip**
 - Skip n records backward/forward



Sequential Access (Cont...)

- Applications
 - Payroll of Employees
 - Student Data Processing



Direct Access

- Relative Access Method
- File - Fixed Length Logical Records
- Based on the disk model of file
- File – A numbered sequence of blocks and records
- Allow read/ write operations to be performed rapidly (i.e. no specific order)
- Eg- first read block 50, then read block 24, read then block 87 and so on.



Direct Access (Cont...)

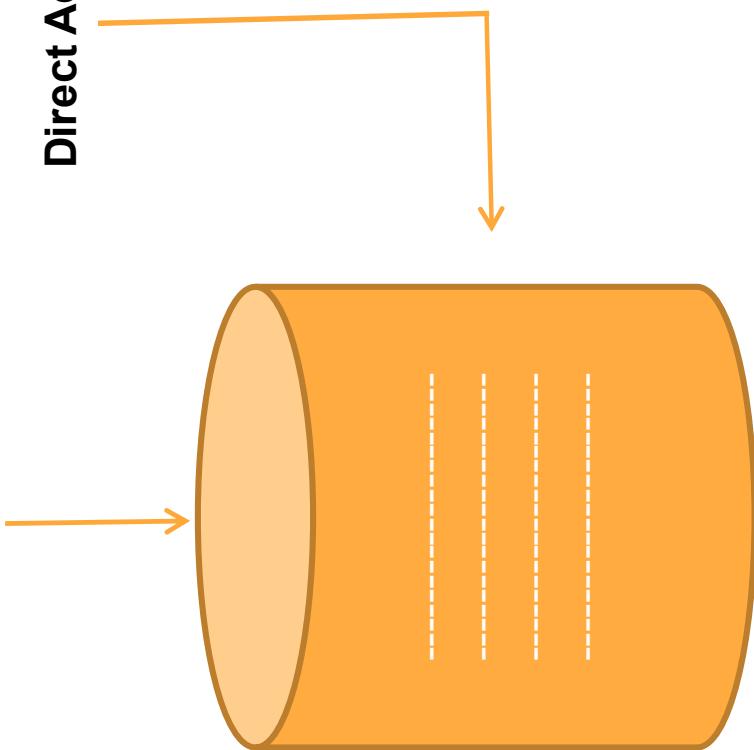
- Used when we need filtered information from the database (Sequential access can be very slow and inefficient).
- No need of determining the desired block number.
- Generally implemented in database applications.
- Large amount of information – immediate access



Direct Access (Cont...)

Sequential Access

Direct Access



Direct Access (Cont....)

- **Example – Airline Reservation System**
- Stores all information about any specific flight in block number identified by flight number
- Number of seats booked for flight number 100 is stored in block 100 of reservation file.



Direct Access (Cont....)

- **File operations :**

- Read
- Write



Direct Access (Cont...)

- **Read Opeartion : read (n)**
 - n : Block Number
 - Reads block number n of file



Direct Access (Cont...)

- Write Operation : write (n)
 - n : Block Number
 - Writes to block number n of file



Direct Access (Cont...)

- Retain **read_next()** and **write_next()**
 - Sequential Access
 - Add **position_file(n)** ; n - Block Number
 - Read (n) : **position_file(n) + read_next()**



Direct Access (Cont...)

- Address of an arbitrary record N

$$N = L * (N - 1)$$

Where L : length of Logical Record



Direct Access (Cont...)

- Simulation of Sequential Access

Sequential Access	Implementation for Direct Access
reset	cp = 0;
read_next()	read cp; cp = cp + 1;
write_next()	write cp; cp = cp + 1;



Direct Access (Cont...)

- **Applications**
 - Customer account processing in bank
 - Airline Reservation System



Other Access Methods

- Built on top of Direct Access Method
- Requires an “index” for file
- Index – contains pointer to various blocks



Other Access Methods (Cont...)

- **Searching a record/entry in the file**
 - Search the index
 - Use pointer to access the file directly for desired record



Other Access Methods (Cont...)

- Primary/ Secondary Index
 - For larger files – index file also required to be too large in memory
 - Solution – Create “index” for index file
 - Primary Index – Points to Secondary Index
 - Secondary Index – Points to data



File Allocation

- **Basic Idea**
 - Many files stored on same disk.
 - How to allocate space to these file

- **Aim**
 - Effective utilization of disk space
 - Quick access to files



File Allocation Methods

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation



Contiguous Allocation

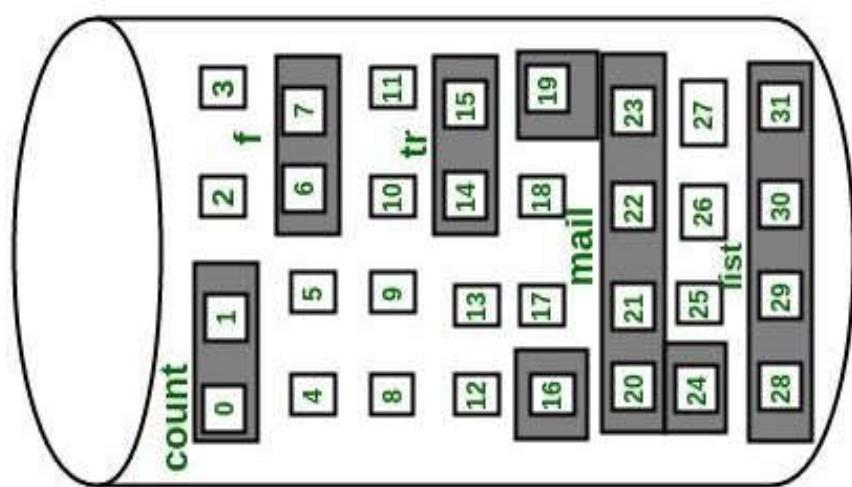
- Each file occupy a set of contiguous blocks on disk.
- Disk addresses define a linear ordering on the disk.
- Assume only single job is accessing the disk.
- First access block b , then accessing block $b+1$ does not require any head movement.



Contiguous Allocation (Cont...)

Directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2



Contiguous Allocation (Cont....)

- **Advantages**

- Easy to implement
- Fast Access

- **Disadvantages**

- Internal Fragmentation
- External Fragmentation

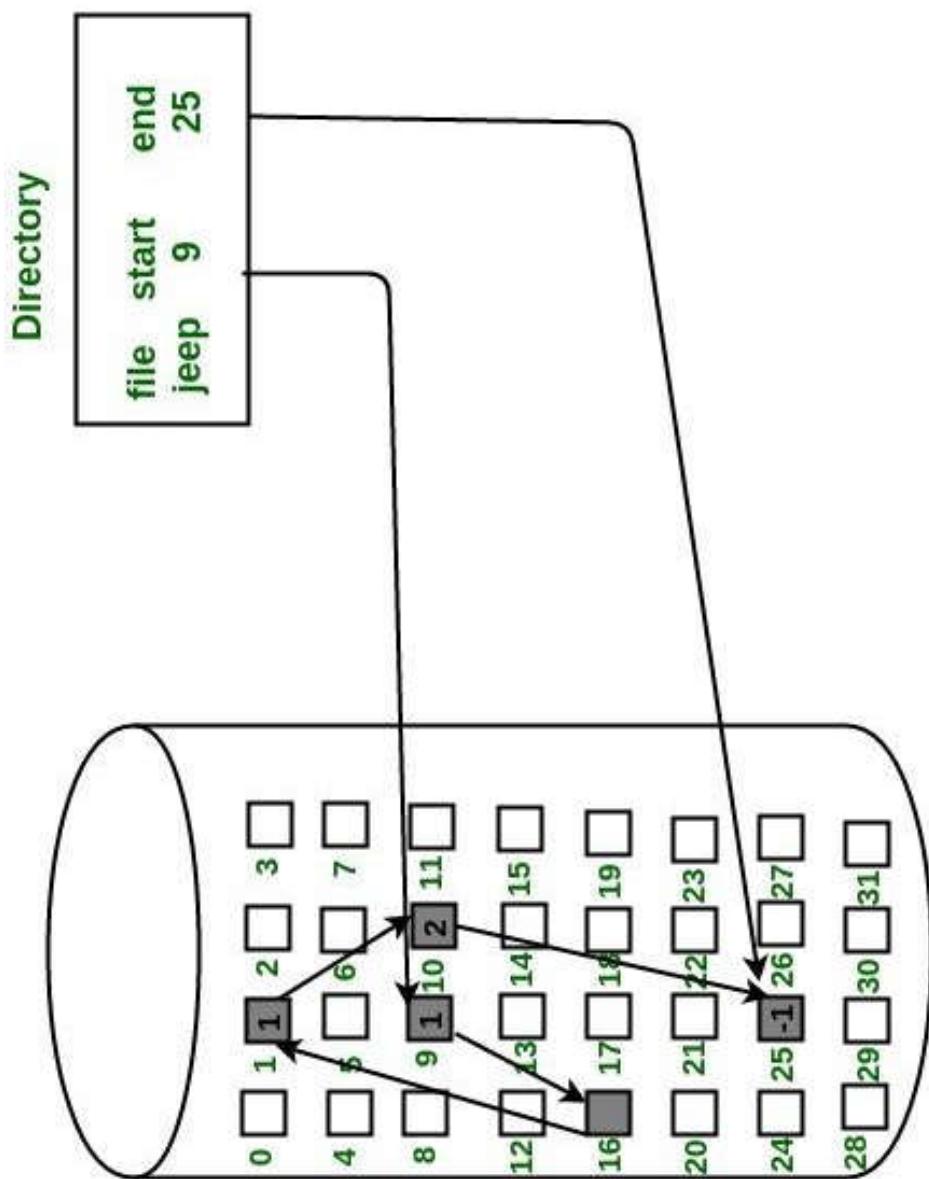


Linked Allocation

- Resolves all issues of contiguous allocation.
- File – Linked list of disk blocks.
- Disk blocks may be separated anywhere on the disk.
- Directory contains :
 - Pointer to first block of file
 - Pointer to last block of file



Linked Allocation (Cont...)



Linked Allocation (Cont...)

- **Advantages**
 - No External Fragmentation

- **Disadvantages**
 - Slow Access



Indexed Allocation

- Resolves all issues of contiguous allocation & Linked allocation.
- Brings all pointers together into one location – “**Index Block**”.
- Each file has its own index block.
- Block – An array of disk block addresses (DBA)
- **Index of Addresses**

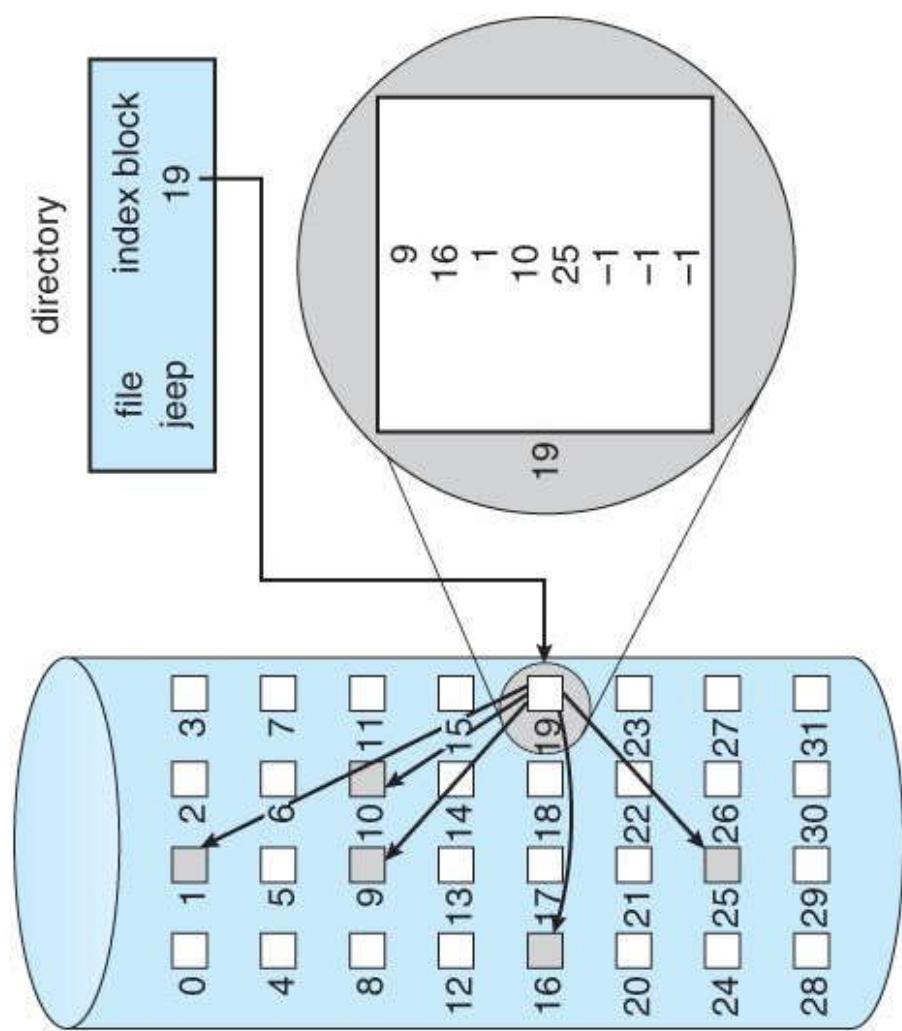


Indexed Allocation (Cont...)

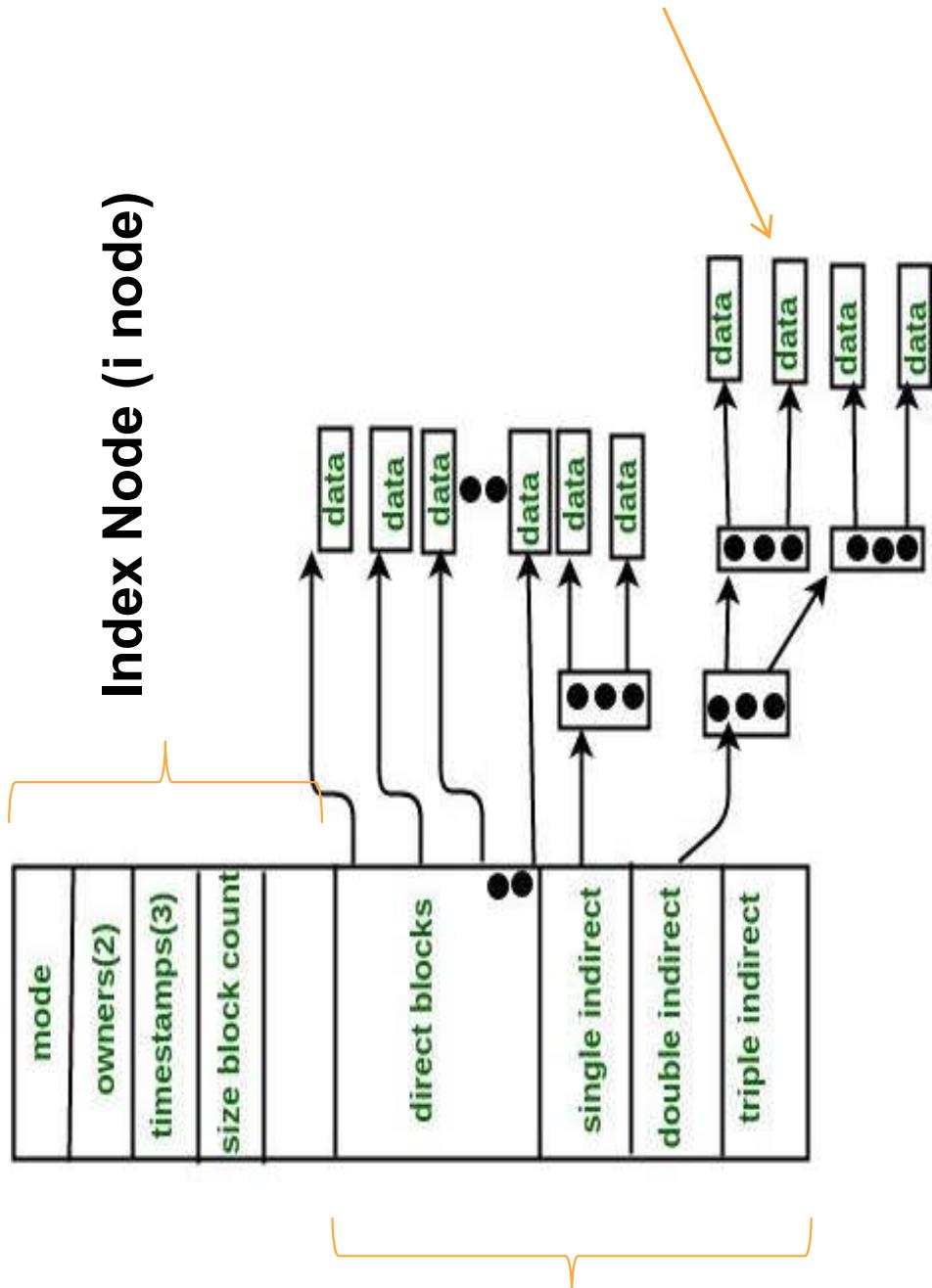
- i^{th} entry in index block points to i^{th} block of file.
- Directory contains :
 - Address of index block
- To find i^{th} block –
 - Use pointer in the i^{th} block entry



Indexed Allocation (Cont...)



Indexed Allocation (Cont...)



Indexed Allocation (Cont....)

- **Advantages**

- Problem of limited file size is resolved
- No space wastage (direct pointer)
- No External Fragmentation
- Time is relatively very low

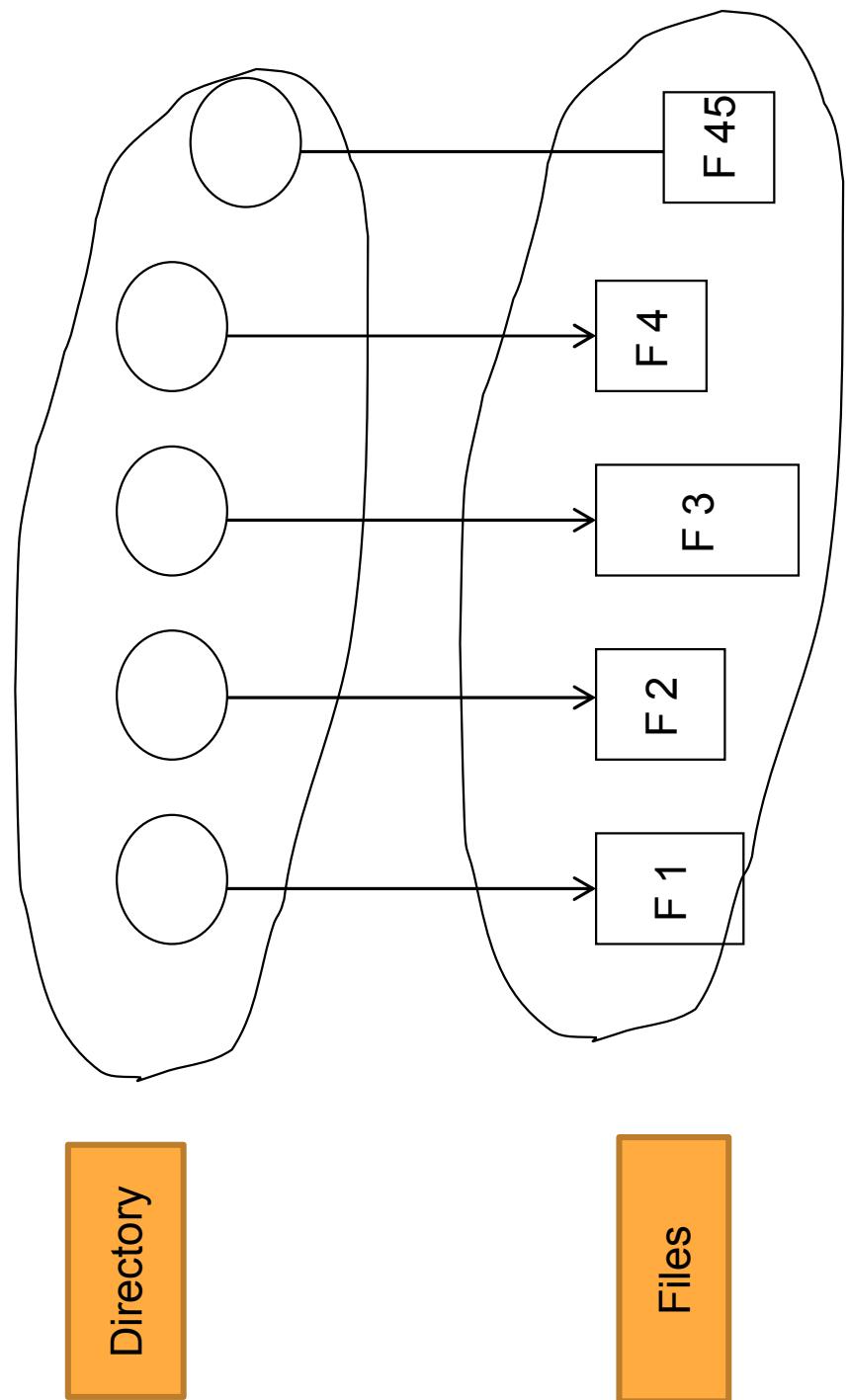


Directory

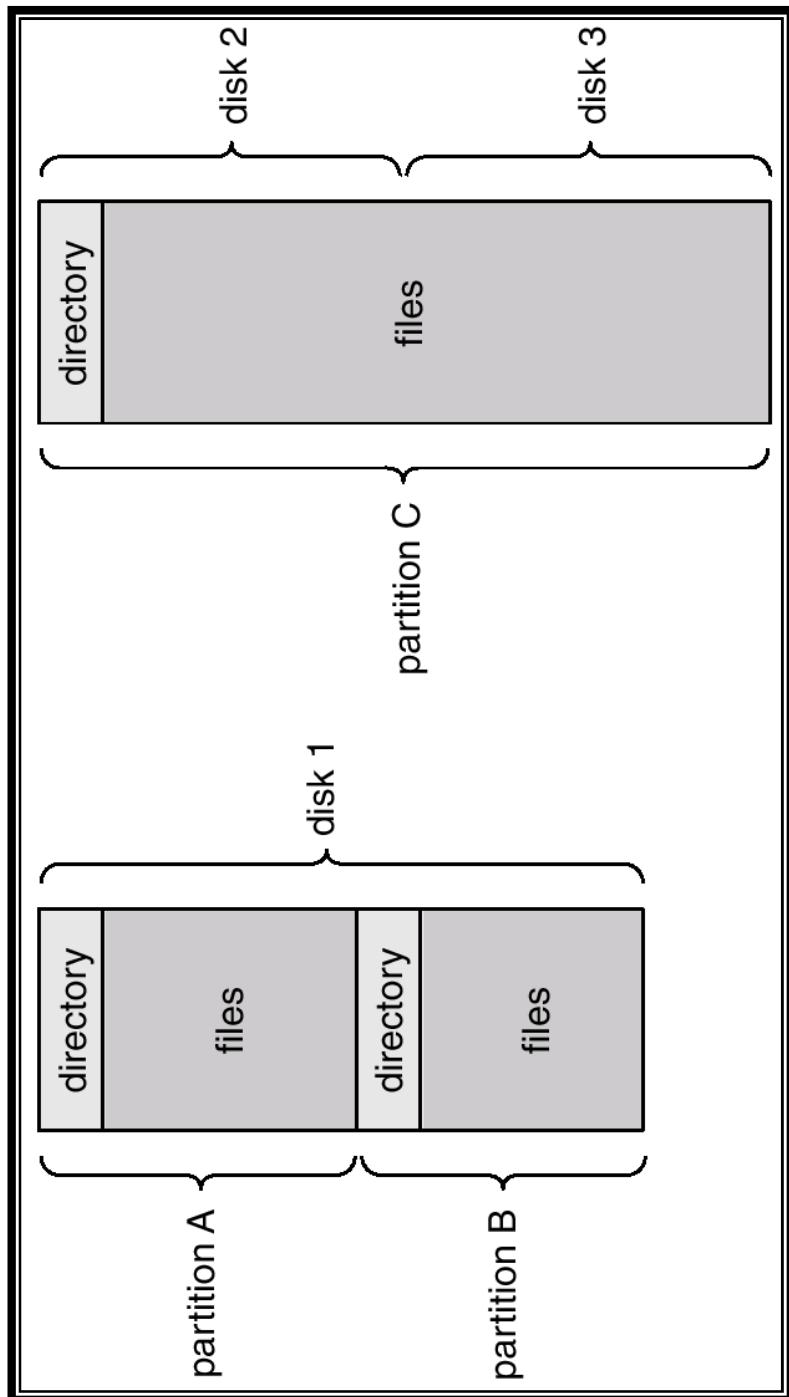
- Physical Disk can be broken up into multiple partitions (multi-disks)
- A collection of nodes containing information about all files.
- Both the directory structure and the files reside on disk.
- Backups of these two structures are kept on tapes.



Directory (Cont....)



Directory (Cont....)



A Typical File System Organization



Directory (Cont....)

- **Information in a Device Directory**

- Name
- Type
- Address
- Current length
- Maximum length
- Date last accessed



Directory (Cont....)

- **Information in a Device Directory**

- Date last updated
- Owner ID
- Protection information



Directory (Cont....)

- **Operation Performed on a Directory**

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system



Directory (Cont....)

- Recommended ways to organize a Directory

- Efficiency – locating a file quickly.
- Naming – convenient to users.
 - Two users can have same name for different files.
 - The same file can have several different names.
- Grouping – logical grouping of files by properties, (e.g., all Java programs, all games, ...)



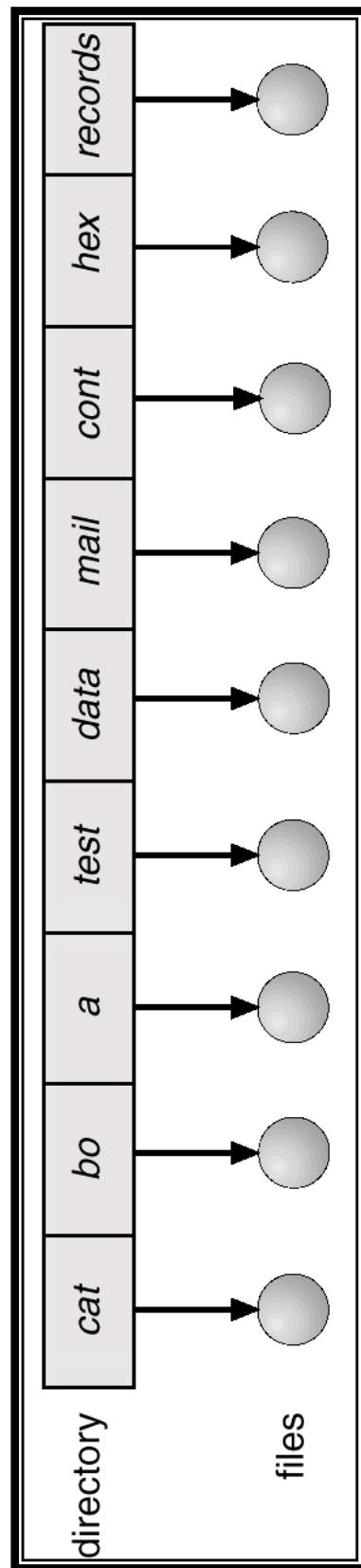
Types of Directory

- Single-Level Directory
- Two-Level Directory
- Tree Structured Directory
- Acyclic Graph Directory
- General Graph Directory



Single Level Directory

- A single directory for all users.
- All files are contained in same directory.



Single Level Directory (Cont....)



- **Advantages**
 - Easy Implementation
 - With smaller file size, searching will become fast
 - Creation, searching, deletion, updation are very easy

- **Limitations**
 - Increased Number of file in system
 - When system has more than one user
 - Naming Problem

Two Level Directory

- Separate directory for each users.
- Each user has its own User File Directory (UFD).
- All UFD's have similar structure.
- When user logs in, Master File Directory (MFD) is searched.

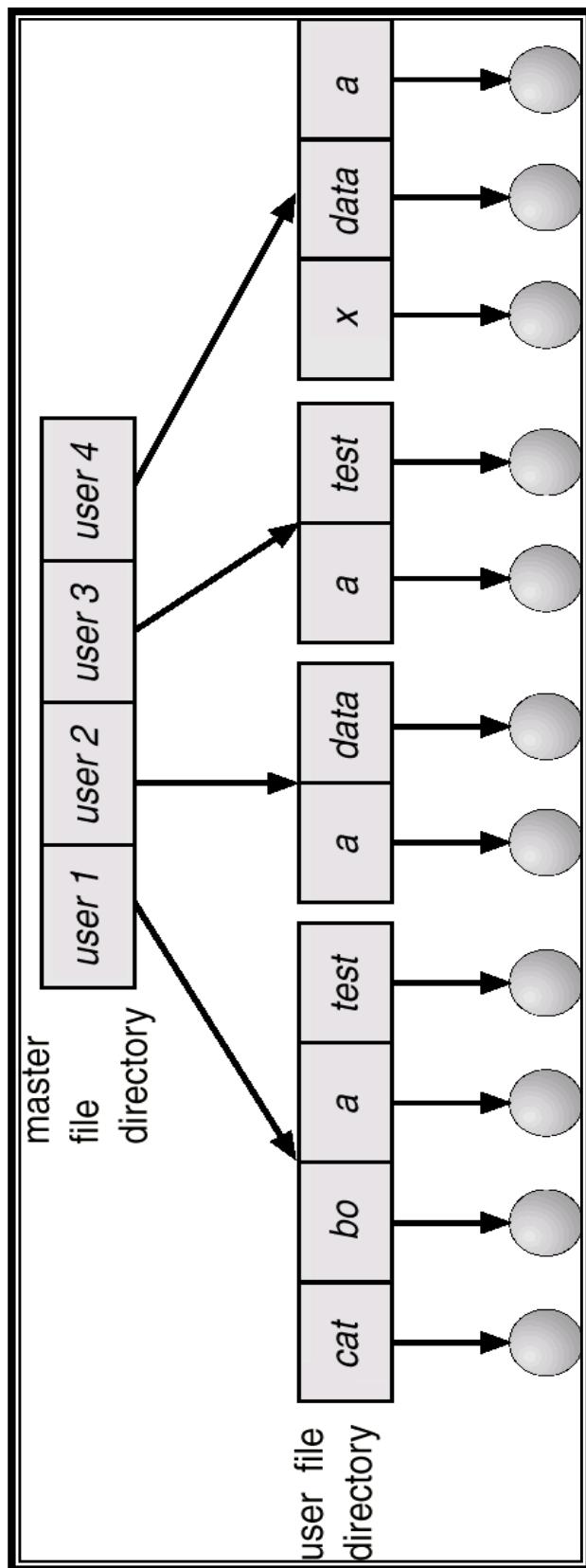


Two Level Directory (Cont...)

- MFD : indexed by user name/account number .
- Each entry points to UFD for that user.
- When user refers to a specific file, only his own UFD is searched.



Two Level Directory (Cont...)





Two Level Directory (Cont...)

- **Advantages**

- Full path like /User-name/directory-name/.
- Different users can have same directory as well as file name.
- Searching of files become more easy due to path name and user-grouping.

- **Disadvantages**

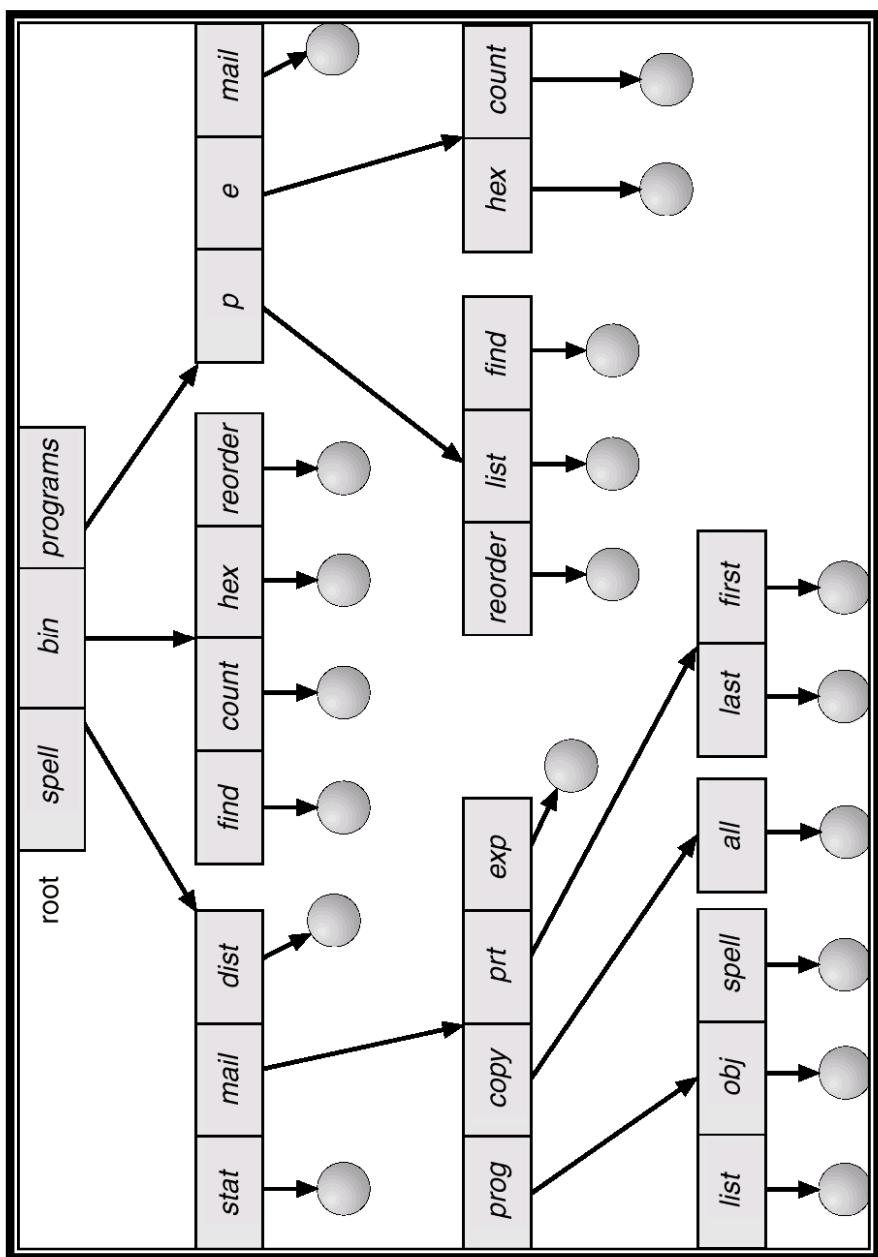
- A user is not allowed to share files with other users.
- Still it not very scalable, two files of the same type cannot be grouped together in the same user.

Tree - Structured Directory

- A root directory and every file has a unique path
- A directory contains :
 - Set of files or sub-directories.
- All Directories have same internal format.
- **Current Directory (CD)**
 - Contains most of the files that are of current interest to the process.
- When a file reference is made, the current directory is searched.



Tree - Structured Directory (Cont....)



Tree - Structured Directory (Cont....)

- **Advantages**

- Full path name can be given.
- Probability of name collision is less.
- Searching becomes very easy, (absolute path as well as relative)

- **Disadvantages**

- Every file does not fit into the hierarchical model, files may be saved into multiple directories.
- We can not share files.
- It is inefficient, because accessing a file may go under multiple directories.



Tree - Structured Directory (Cont...)

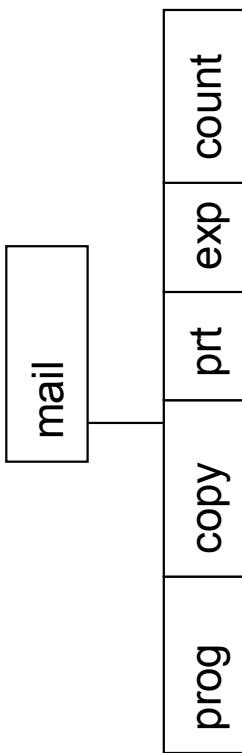
- Creating a new file is done in current directory.
- Delete a file
`rm <file-name>`
- Creating a new subdirectory is done in current directory.

`mkdir <dir-name>`

Example: if in current directory `/mail`

`mkdir count`

Note - Deleting “mail” will delete the entire sub-tree rooted by “mail”.

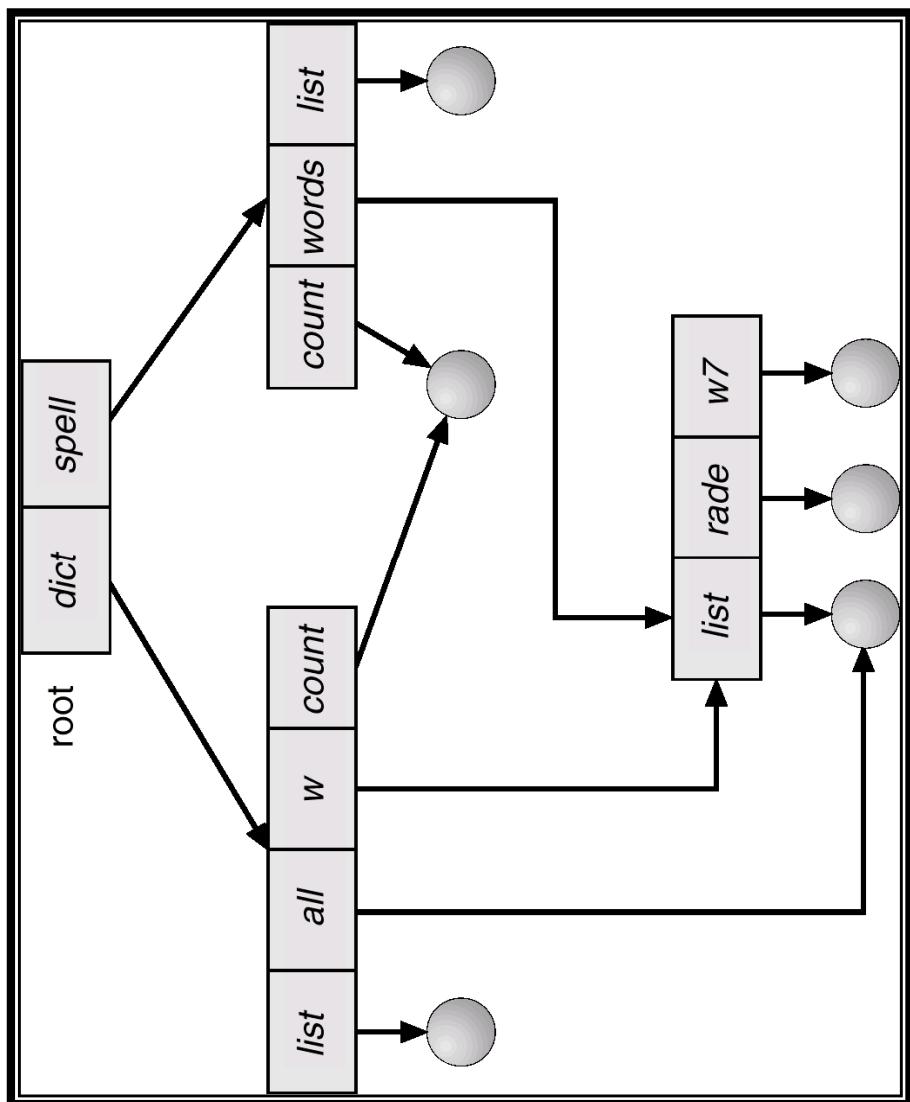


Acyclic Graph Directory

- A Graph with no cycle.
- Have shared sub-directories and files.
- Allows directories to share sub-directories and files.
- i.e. Same file or sub-directory may be in two different locations.



Acyclic Graph Directory (Cont....)



Acyclic Graph Directory (Cont....)

- **Advantages**

- We can share files.
- Searching is easy due to different-different paths.

- **Disadvantages**

- We share the files via linking, in case of deleting it may create the problem,

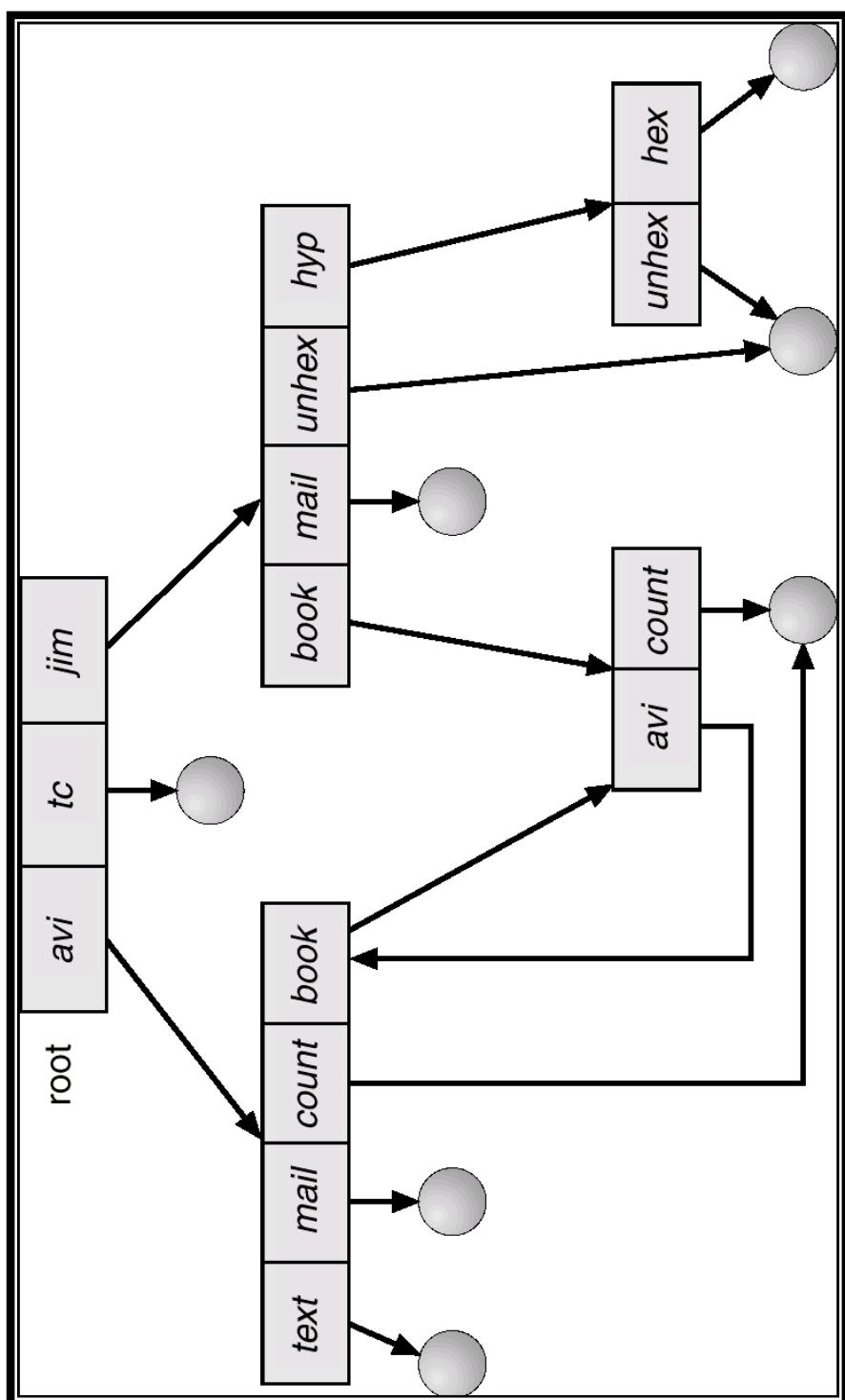


General Graph Directory

- Cycles are allowed within a directory structure
- where multiple directories can be derived from more than one parent directory.
- Problem - to calculate total size or space



General Graph Directory



General Graph Directory (Cont....)

- **Advantages**

- Allows cycles
- More flexible than other directories structure

- **Disadvantages**

- Costly
- Needs garbage collection



Free Space Management

- **Problem :** Limited Disk Space
- **Solution :** for new files, reuse space from deleted files.
- **Write once optical disks :** Allow only once to write, so no reuse is possible.
- **Free Space List :** Keeps track of all free disk spaces. (not those allocated to some file or directory)



Free Space Management (Cont...)

- **To create a file :**
 - Search free space list for required amount of space
 - Allocate that space to the new file
- Space is then removed from free space list.
- When a file is deleted, its disk space is added to free space list.



Implementing Free Space List

- Implementation of Free Space List :
 - bit map/bit vector
- Each block is represented by 1 bit.
- If block is free, bit is 1, if block is allocated, bit is 0.



Implementing Free Space List

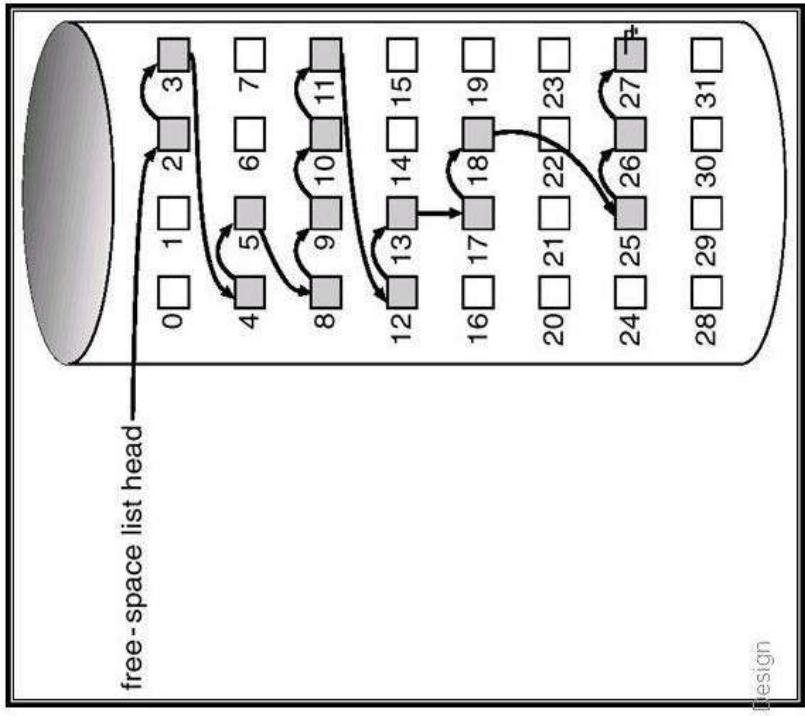
- Example : Consider a disk block, where : Block Size - 0 to 31
 - Blocks 2,3,4,5,8,9,10,11,12,13,17,18,25,26,27 are free
 - and rest of the blocks are allocated.
- Free Space bit map : 001110011111000.....



Implementing Free Space List



- Advantages :
 - Simplicity
 - Efficiency
- Another approach :
 - Link together all free disk blocks.



Design

Mounting

- File System must be mounted just as a file needs to be opened before it is used.
- Then it will be available to processes on system.
- Mount Point :
 - Location within the file structure where the file system is to be attached.
 - An Empty Directory



Mounting (Cont...)

- Example : In Unix File System, a file system containing a user's home directories might be mounted as /home.
- To access directory structure within this file system , precede directory name with /home as /home/abc.



Mounting (Cont....)

- **Triangle** : Subtree of directories.

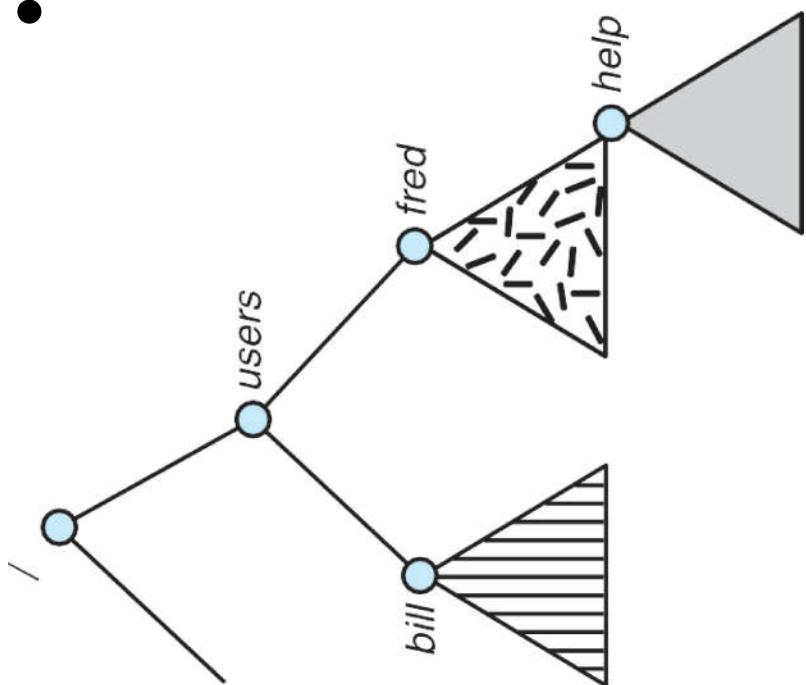


Fig 1: Existing File System



Mounting (Cont...)

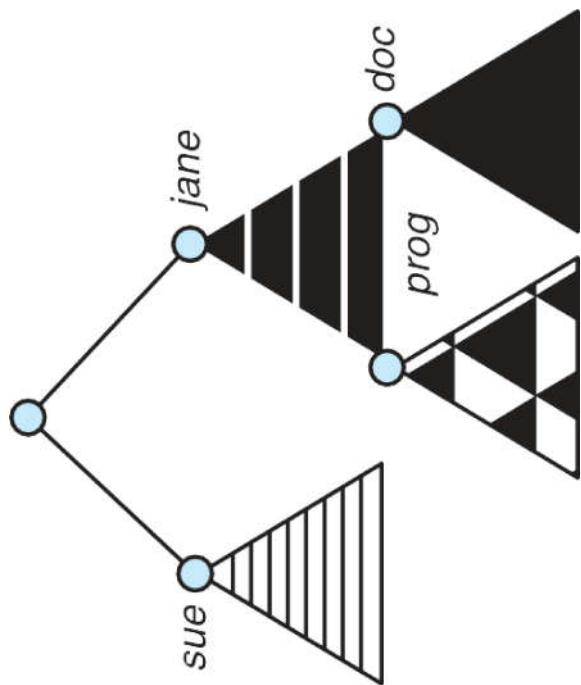


Fig 2: Unmounted File System



Mounting (Cont...)

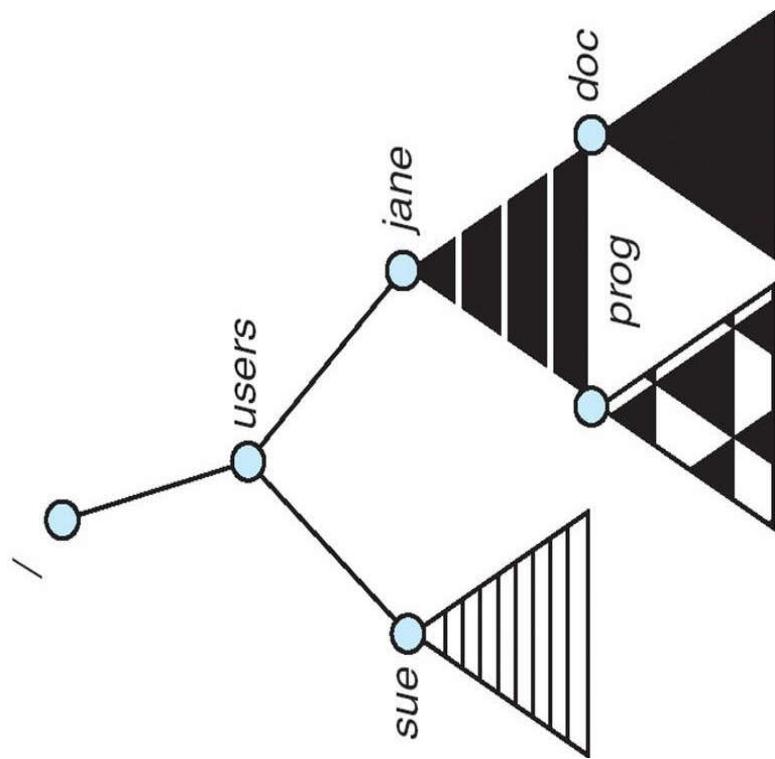


Fig 3: Mount Point



File Sharing

- Sharing of files on multi-user systems is desirable.
- Sharing may be done through a **protection** scheme.
- On distributed systems, files may be shared across a network.
- Network File System (NFS) is a common distributed file-sharing method.



File Sharing – Remote File Systems

- Uses networking to allow file system access between systems
 - Manually via programs like FTP
 - Automatically, seamlessly using **distributed file systems**
 - Semi automatically via the **world wide web**
- **Client-server** model allows clients to mount remote file systems from servers
 - Server can serve multiple clients
 - Client and user-on-client identification is insecure or complicated
 - **NFS** is standard UNIX client-server file sharing protocol
 - **CIFS** is standard Windows protocol
 - Standard operating system file calls are translated into remote calls



File Sharing – Failure Models

- All file systems have failure modes
 - For example corruption of directory structures or other non-user data, called **metadata**
- Remote file systems add new failure modes, due to network failure, server failure
- Recovery from failure can involve **state information** about status of each remote request
- **Stateless** protocols such as NFS v3 include all information in each request, allowing easy recovery but less security



File Protection

- File owner/creator should be able to control:
 - what can be done
 - by whom
- Types of access
 - Read
 - Write
 - Execute
 - Append
 - Delete
 - List

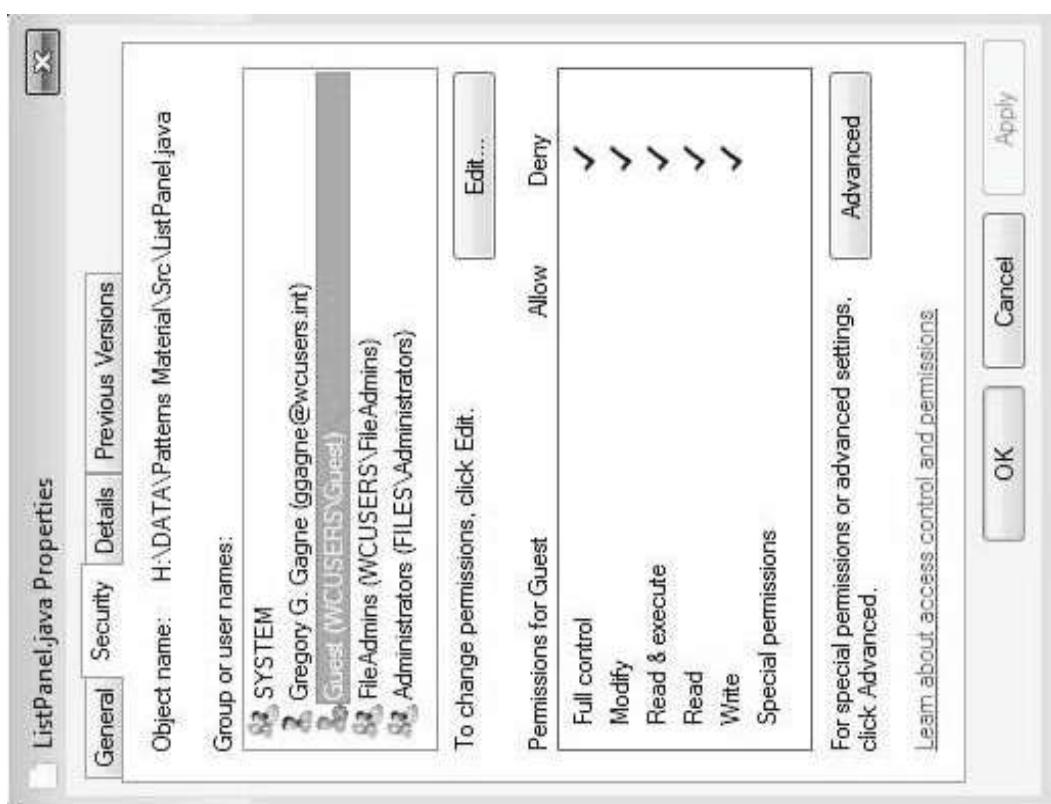


Access Control List

- Specify user names and types of access allowed for each user.
- When a user requests a specific file, OS checks ACL associated with that file.
 - If the user is listed for the requested access, the access is allowed, Otherwise denied.
- Mode of Access : Read, Write, Execute



Window 7 Access Control List Management



REFERENCES BOOKS

- A. Silberschatz and Peter B Galvin: Operating System Principles, Wiley India Pvt. Ltd.
- Tanenbaum: Modern Operating System, Prentice Hall.



Thank You . . .



UNIT - V

CASE STUDY: UNIX

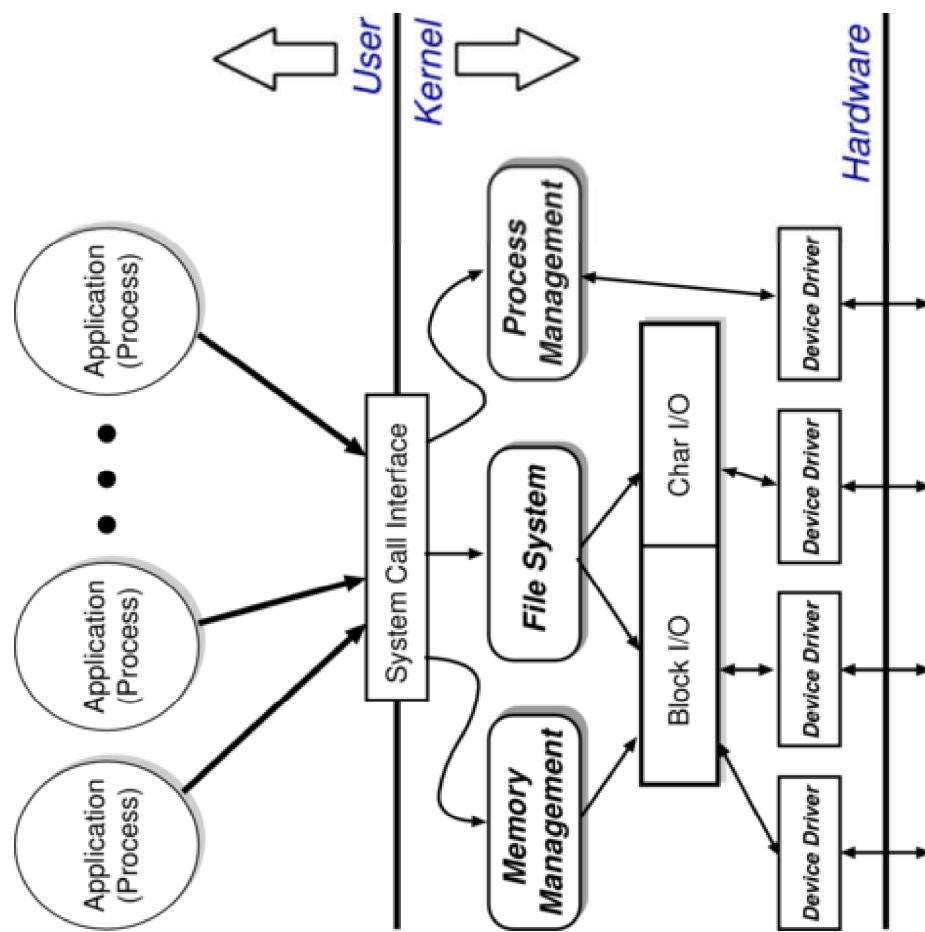
OUTLIN

E

- Introduction
- Design Principles
 - Structural, Files, Directory Hierarchy
- Filesystem
 - Files, Directories, Links, On-Disk Structures
 - Mounting Filesystems, In-Memory Tables, Consistency
- IO
 - Implementation, The Buffer Cache
- Processes
 - Unix Process Dynamics, Start of Day, Scheduling and States
- The Shell
 - Examples, Standard IO
- Summary

STRUCTURAL OVERVIEW

- Clear separation between user and kernel portions was the big difference between Unix and contemporary systems—only the **essential features** *inside* OS, not the editors, command interpreters, compilers, etc.
- Processes are unit of scheduling and protection: the command interpreter ("shell") just a process
- No concurrency within kernel
- All IO looks like operations on files: in Unix, everything is a file



FILE ABSTRACTION

File as an unstructured sequence of bytes which was relatively unusual at the time:
most systems lent towards files being composed of records

- Cons: don't get nice type information; programmer must worry about format of things inside file
- Pros: less stuff to worry about in the kernel; and programmer has flexibility to choose format within file!

Represented in user-space by a file descriptor (`fd`) this is just an opaque identifier
—a good technique for ensuring protection between user and kernel

FILE

OPERATIONS

Operations on files are:

- `fd = open (pathname, mode)`
- `fd = creat (pathname, mode)`
- `bytes = read (fd, buffer, nbytes)`
- `count = write (fd, buffer, nbytes)`
- `reply = seek (fd, offset, whence)`
- `reply = close (fd)`

The kernel keeps track of the current position within the file

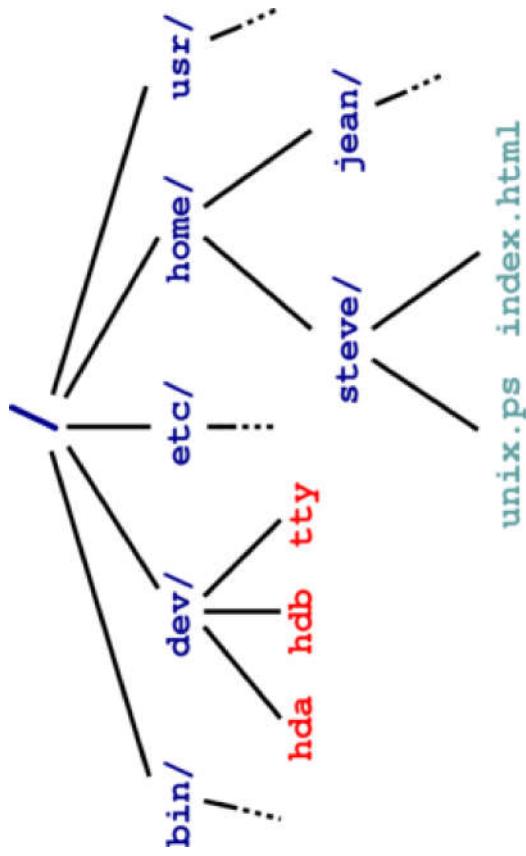
Devices are represented by special files:

- Support above operations, although perhaps with bizarre semantics
- Also have `ioctl` for access to device-specific functionality

DIRECTORY HIERARCHY

Directories map names to files (and directories) starting from distinguished root directory called /

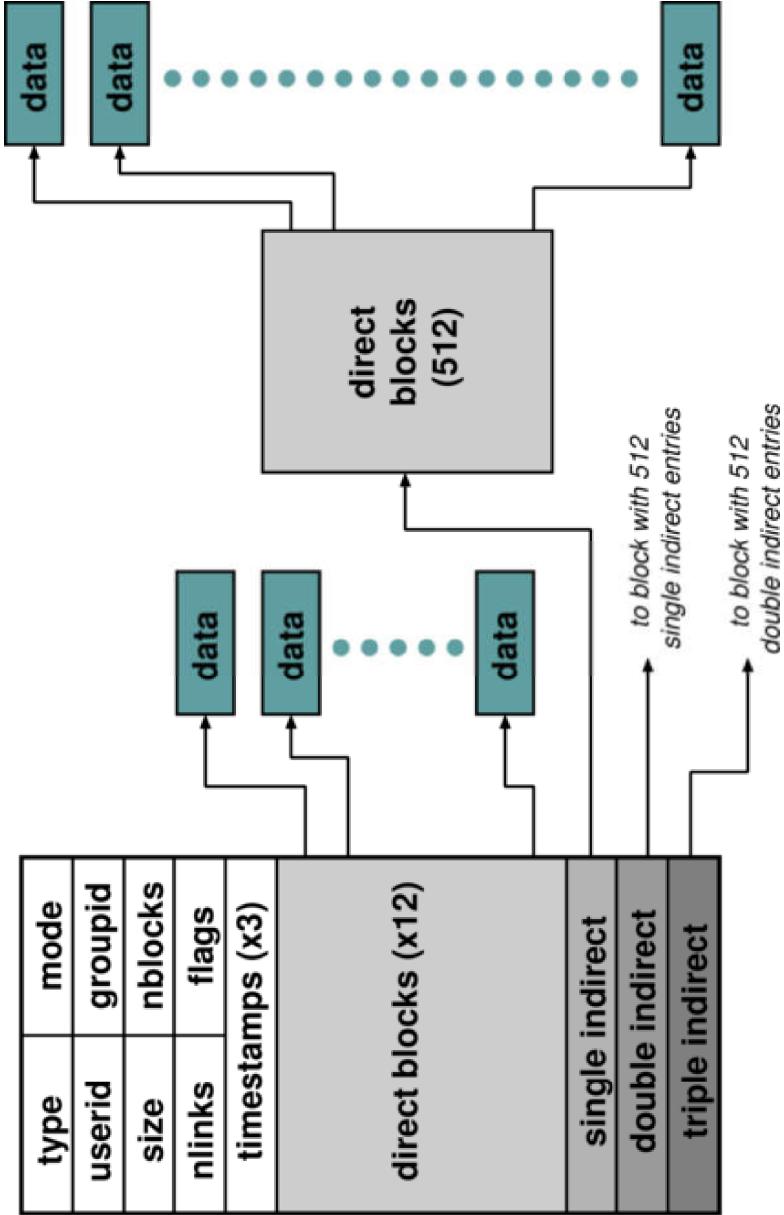
Fully qualified pathnames mean performing traversal from root



Every directory has . and .. entries: refer to self and parent respectively. Also have shortcut of **current working directory** (cwd) which allows relative path names; and the shell provides access to home directory as ~username (e.g. ~mort/). Note that kernel knows about former but not latter

Structure is a tree in general though this is slightly relaxed

FILE SYSTEM IMPLEMENTATION



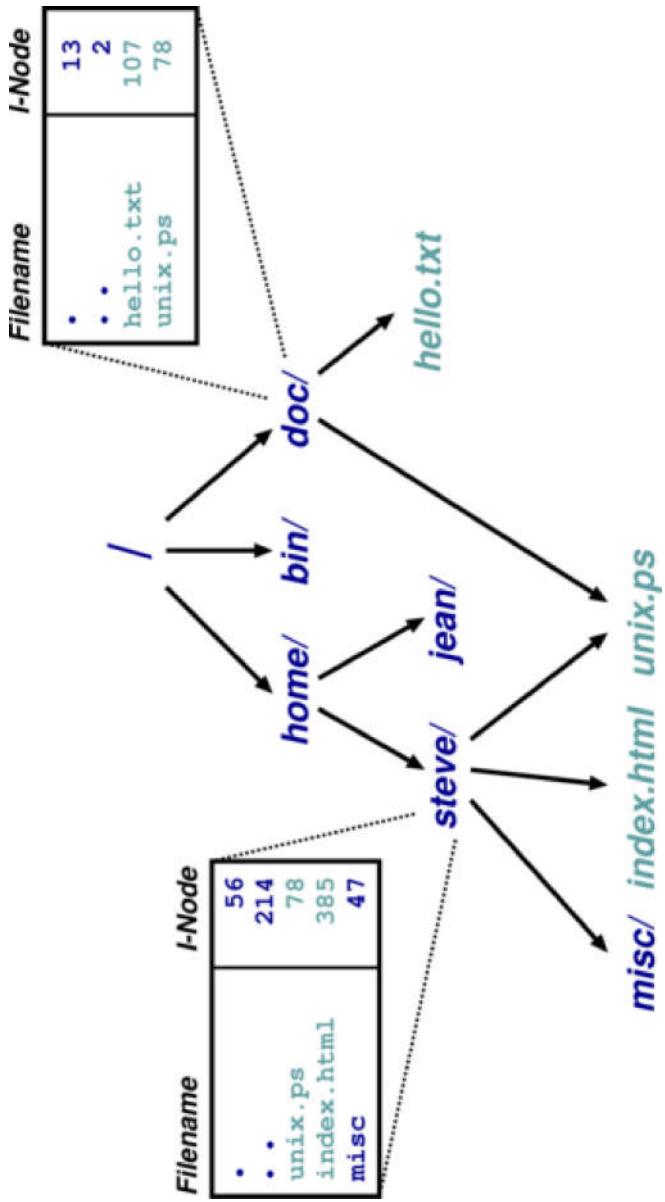
Inside the kernel, a file is represented by a data structure called an **index-node** or **i-node** which hold file meta-data: owner, permissions, reference count, etc. and location on disk of actual data (file contents)

DIRECTORIES AND LINKS

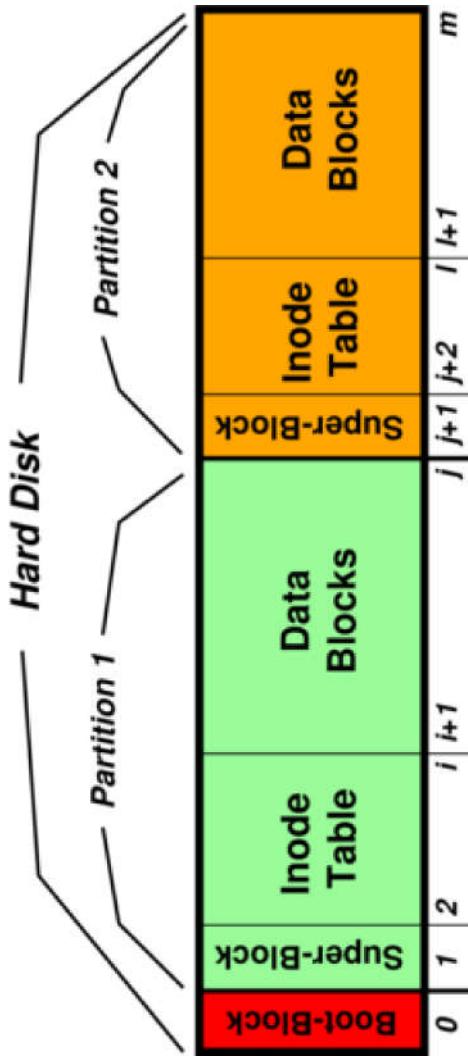
Directory is (just) a file which maps filenames to i-nodes — that is, it has its own i-node pointing to its contents

An instance of a file in a directory is a (hard) link hence the reference count in the i-node. Directories can have at most 1 (real) link. Why?

Also get soft- or symbolic-links: a 'normal' file which contains a filename



ON-DISK STRUCTURES



A disk consists of a boot block followed by one or more partitions. Very old disks would have just a single partition. Nowadays have a **boot block** containing a partition table allowing OS to determine where the filesystems are

Figure shows two completely independent filesystems; this is not replication for redundancy. Also note $| \text{inode table} | > | \text{superblock} |$; $| \text{data blocks} | > | \text{inode table} |$

ON-DISK STRUCTURES

A partition is just a contiguous range of N fixed-size blocks of size k for some N and k , and a Unix filesystem resides within a partition

Common block sizes: 512B, 1kB, 2kB, 4kB, 8kB

Superblock contains info such as:

- Number of blocks and free blocks in filesystem
- Start of the free-block and free-inode list
- Various bookkeeping information

Free blocks and inodes intermingle with allocated ones

On-disk have a chain of tables (with head in superblock) for each of these.
Unfortunately this leaves superblock and inode-table vulnerable to headcrashes so we must replicate in practice. In fact, now a wide range of Unix filesystems that are completely different; e.g., log-structure

MOUNTING FILE SYSTEMS

Entire filesystems can be mounted on an existing directory in an already mounted filesystem

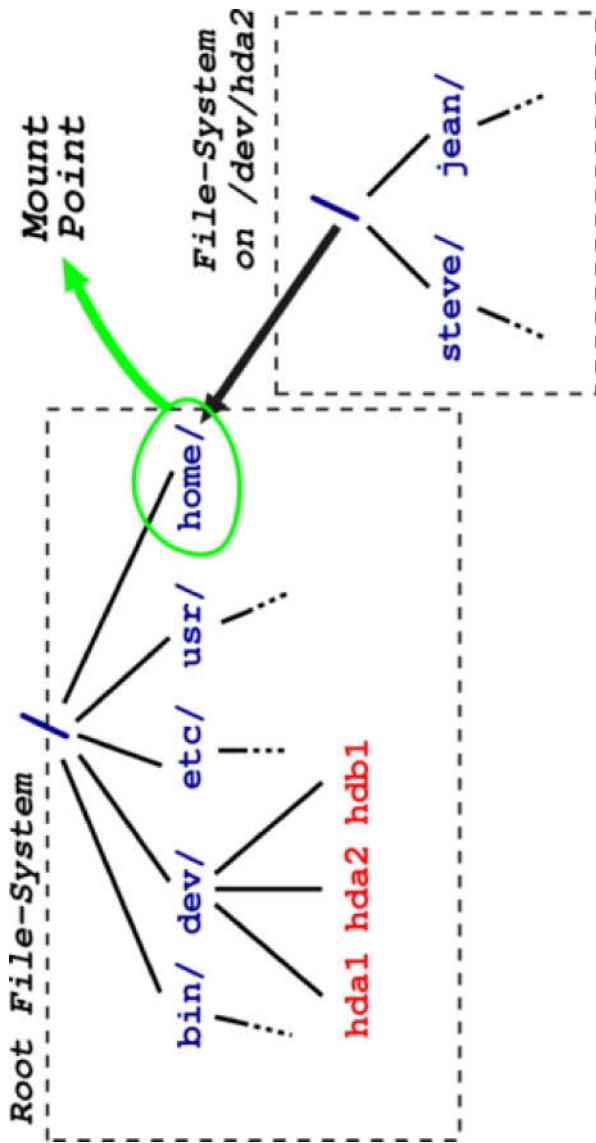
At very start, only / exists so must mount a root filesystem

Subsequently can mount other filesystems, e.g.

```
mount (" /dev/hda2" ,  
" /home" , options)
```

Provides a unified name-space: e.g. access /home/mort/ directly (contrast with Windows9x or NT)

Cannot have hard links across mount points: why? What about softlinks?



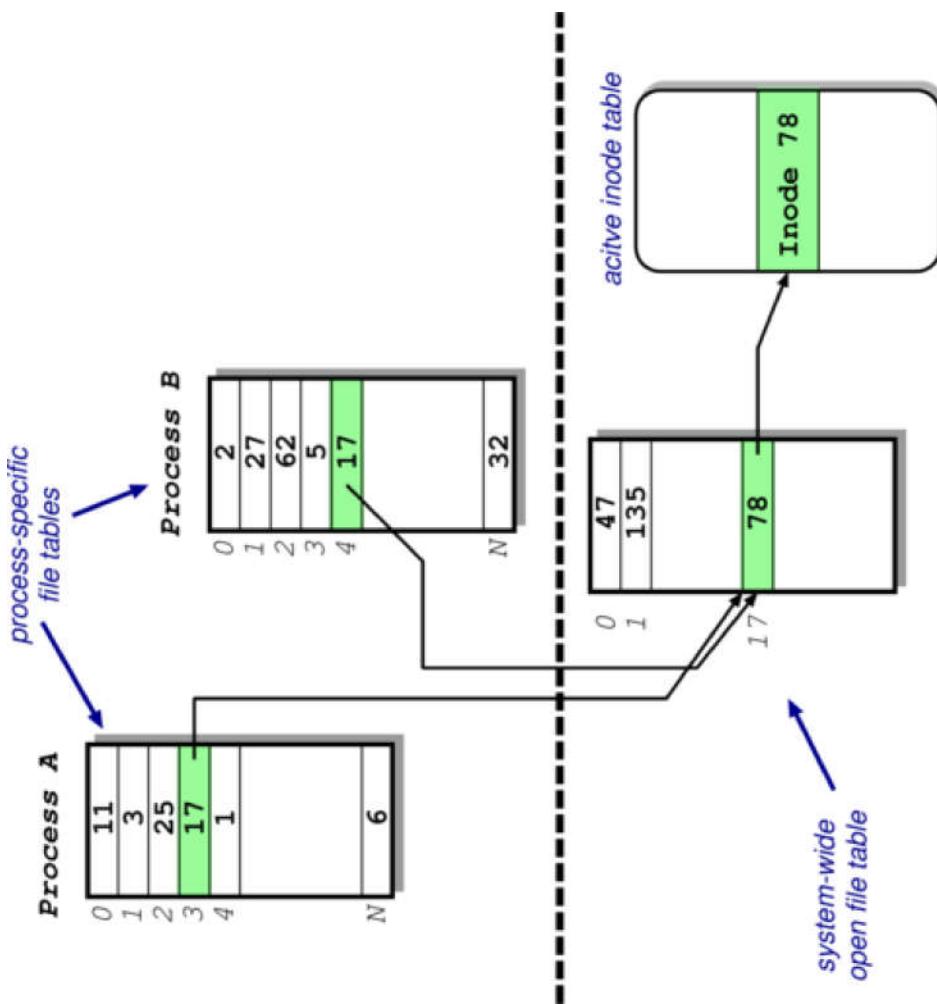
IN-MEMORY TABLES

Recall process sees files as file descriptors

In implementation these are just indices into process-specific open file table

Entries point to system-wide openfile table. Why?

These in turn point to (in memory) inode table



ACCESS CONTROL

Owner			Group			World		
R	W	E	R	W	E	R	W	E
Green	Green	Red	Red	Red	Red	Green	Green	Red

= 0640

= 0755

Access control information held in each inode

- Three bits for each of owner, group and world: read, write and execute
- What do these mean for directories? Read entry, write entry, traversedirectory

In addition have setuid and setgid bits:

- Normally processes inherit permissions of invoking user
- Setuid/setgid allow user to "become" someone else when running a given program
- E.g. prof owns both executable test (0711 and setuid), and score file (0600)

CONSISTENCY

ISSUES

To delete a file, use the unlink system call —from the shell, this is `rm <filename>`

Procedure is:

- Check if user has sufficient permissions on the file (must have write access)
- Check if user has sufficient permissions on the directory (must have write access)
- If ok, remove entry from directory
 - Decrement reference count on inode
 - If now zero: free data blocks and free inode

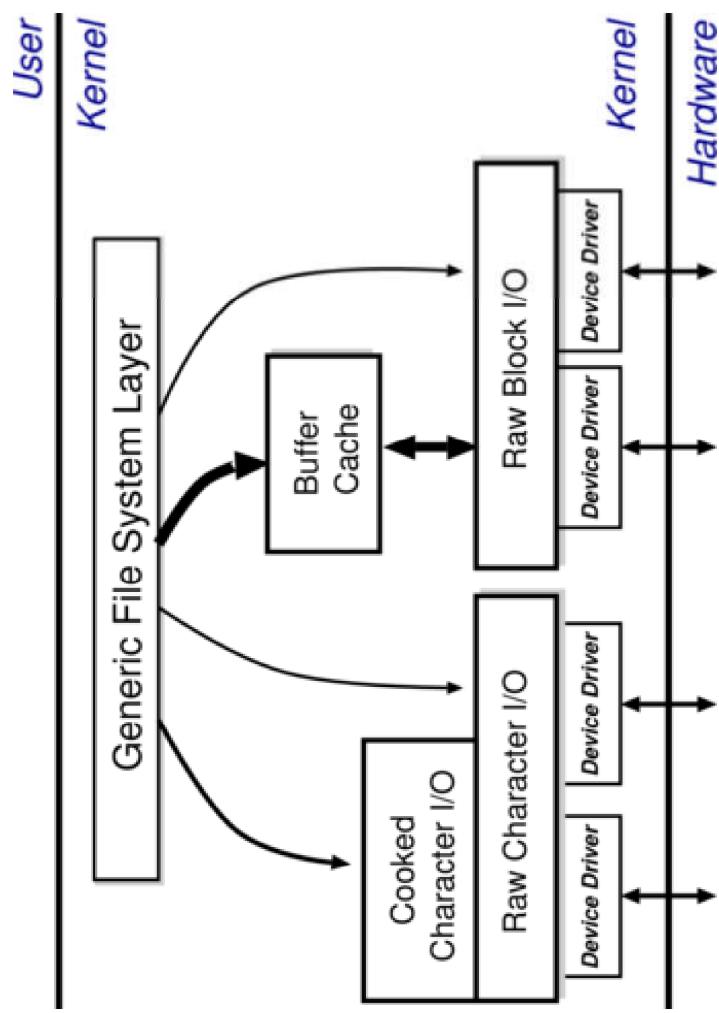
If crash: must check entire filesystem for any block unreferenced and any block double referenced

Crash detected as OS knows if crashed because root fs not unmounted cleanly

I/O

IMPLEMENTATION

- Everything accessed via the file system
- Two broad categories: block and character; ignoring low-level gore:
 - Character I/O low rate but complex —most functionality is in the "cooked" interface
 - Block I/O simpler but performance matters —emphasis on the buffer cache



THE BUFFER CACHE

Basic idea: keep copy of some parts of disk in memory for speed

On read do:

- Locate relevant blocks (from inode)
- Check if in buffer cache
- If not, read from disk into memory
- Return data from buffer cache

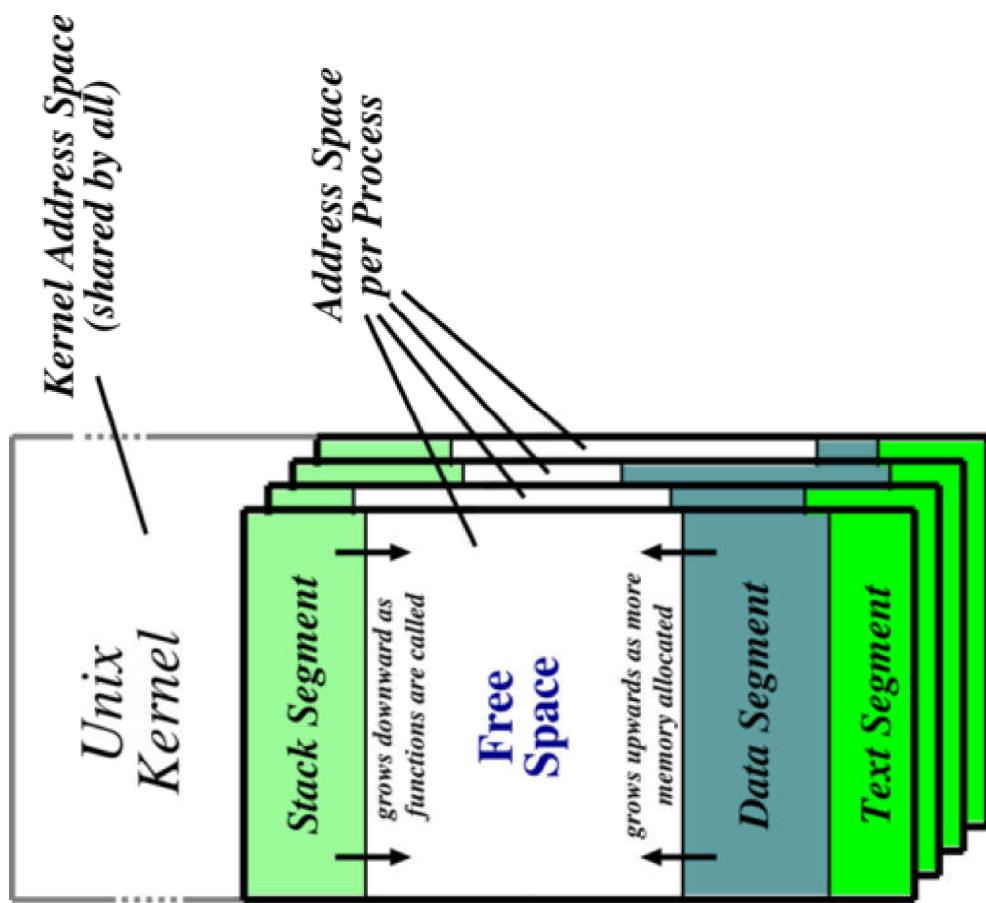
On write do same first three, and then update version in cache, not on disk

- "Typically" prevents 85% of implied disk transfers
- But when does data actually hit disk?
- Call sync every 30 seconds to flush dirty buffers to disk
- Can cache metadata too —what problems can that cause?

UNIX

PROCESSES

Recall: a process is a program in execution



Processes have three segments: text, data and stack. Unix processes are heavyweight

Text: holds the machine instructions for the program

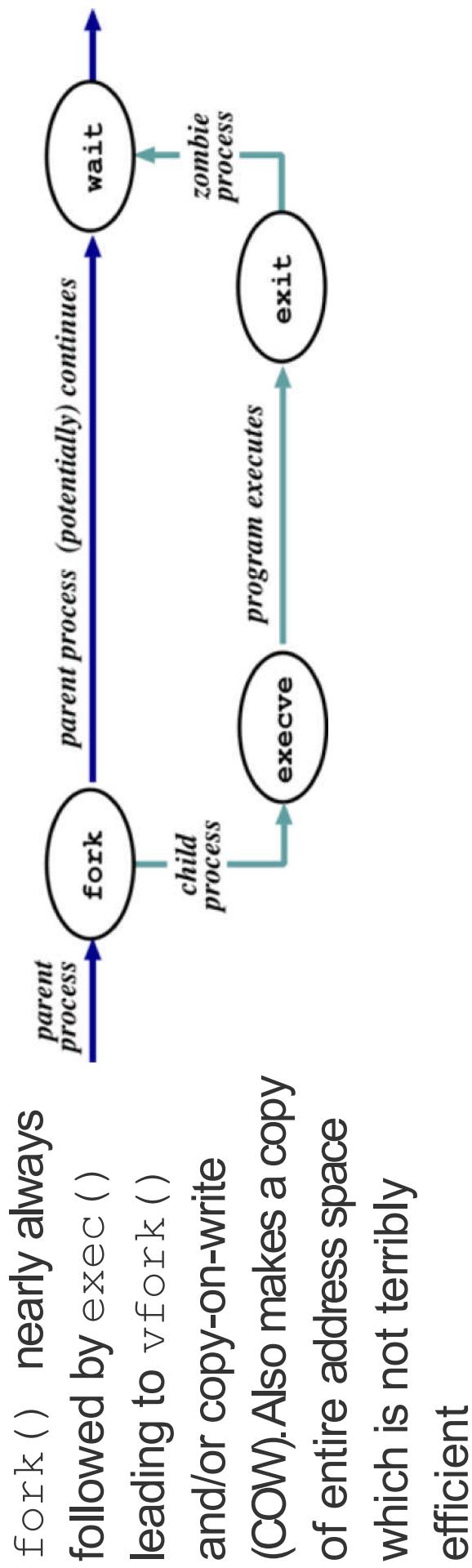
Data: contains variables and their values

Stack: used for activation records (i.e. storing local variables, parameters, etc.)

UNIX PROCESS DYNAMICS

Process is represented by an opaque process id (pid), organised hierarchically with parents creating children. Four basic operations:

- pid = **fork** ()
- reply = **execve** (pathname, argv, envp)
- **exit** (status)
- pid = **wait** (status)



UNIX PROCESS STATES

ru	=	running (user-mode)	rK	=	running (kernel-mode)
z	=	zombie	p	=	preempted
sl	=	sleeping	rb	=	runnable

c = created

NB. This is simplified — see *Concurrent Systems* section 23.14 for detailed descriptions of all states/transitions

THE SHELL

- Introduction
- Design Principles
- Filesystem
- IO
- Processes
- The Shell
 - Examples, Standard IO
- Summary

THE SHELL

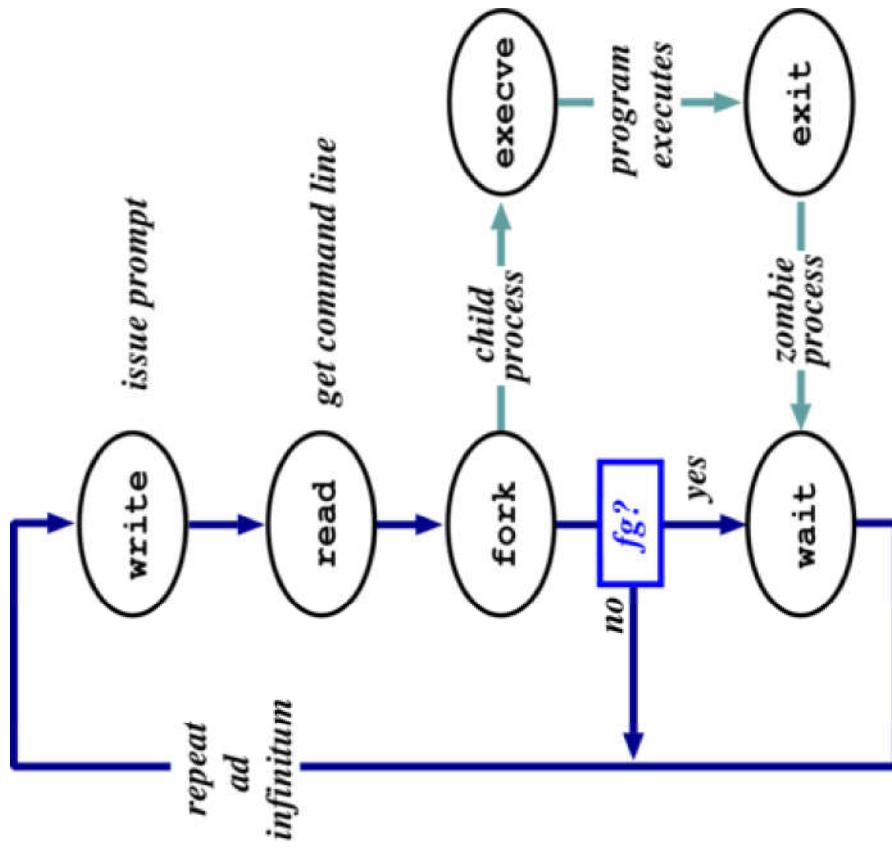
Shell just a process like everything else.

Needn't understand commands, just files

Uses path for convenience, to avoid needing
fully qualified pathnames

Conventionally & specifies background

Parsing stage (omitted) can do lots: wildcard
expansion ("globbing"), "tilde" processing



SHELL EXAMPLES

```
$ pwd  
/Users/mort/src  
$ ls -F  
awk-scripts/ karaka/  
backup-scripts/ mrt.0/  
bib2x-0.9.1/ ocal/  
c-utils/ ocaml/  
dtrace/ ocaml-libs/  
exapräxia-gae/ ocaml-mrt/  
external/ ocaml-pst/  
junk/ ocaml.org/  
$ cd python-scripts/  
/Users/mort/src/python-scripts  
$ ls -1F  
total 224  
-rw-r--r-- 1 mort staff 17987 2 Jan 2010 LICENSE  
-rw-rw-r-- 1 mort staff 1692 5 Jan 09:18 README.md  
-rwxr-xr-x 1 mort staff 6206 2 Dec 2013 bberry.py*  
-rwxr-xr-x 1 mort staff 7286 14 Jul 2015 bib2json.py*  
-rwxr-xr-x 1 mort staff 7205 2 Dec 2013 call.py*  
-rw-r--r-- 1 mort staff 1860 2 Dec 2013 cc4unifdef.py  
-rwxr-xr-x 1 mort staff 1153 2 Dec 2013 filebomb.py*
```

Prompt is \$. Use man to find out about commands. User friendly?

MAIN UNIX FEATURES

- File abstraction
 - A file is an unstructured sequence of bytes
 - (Not really true for device and directory files)
- Hierarchical namespace
 - Directed acyclic graph (if exclude soft links)
 - Thus can recursively mount filesystems
- Heavy-weight processes
- IO: block and character
- Dynamic priority scheduling
 - Base priority level for all processes
 - Priority is lowered if process gets to run
 - Over time, the past is forgotten
- But V7 had inflexible IPC, inefficient memory management, and poor kernel concurrency
- Later versions address these issues.

Case Study : LINUX

UNIX/Linux Goals

- Designed by programmers, for programmers
- Designed to be:
 - Simple
 - Elegant
 - Consistent
 - Powerful
 - Flexible

Interfaces to Linux

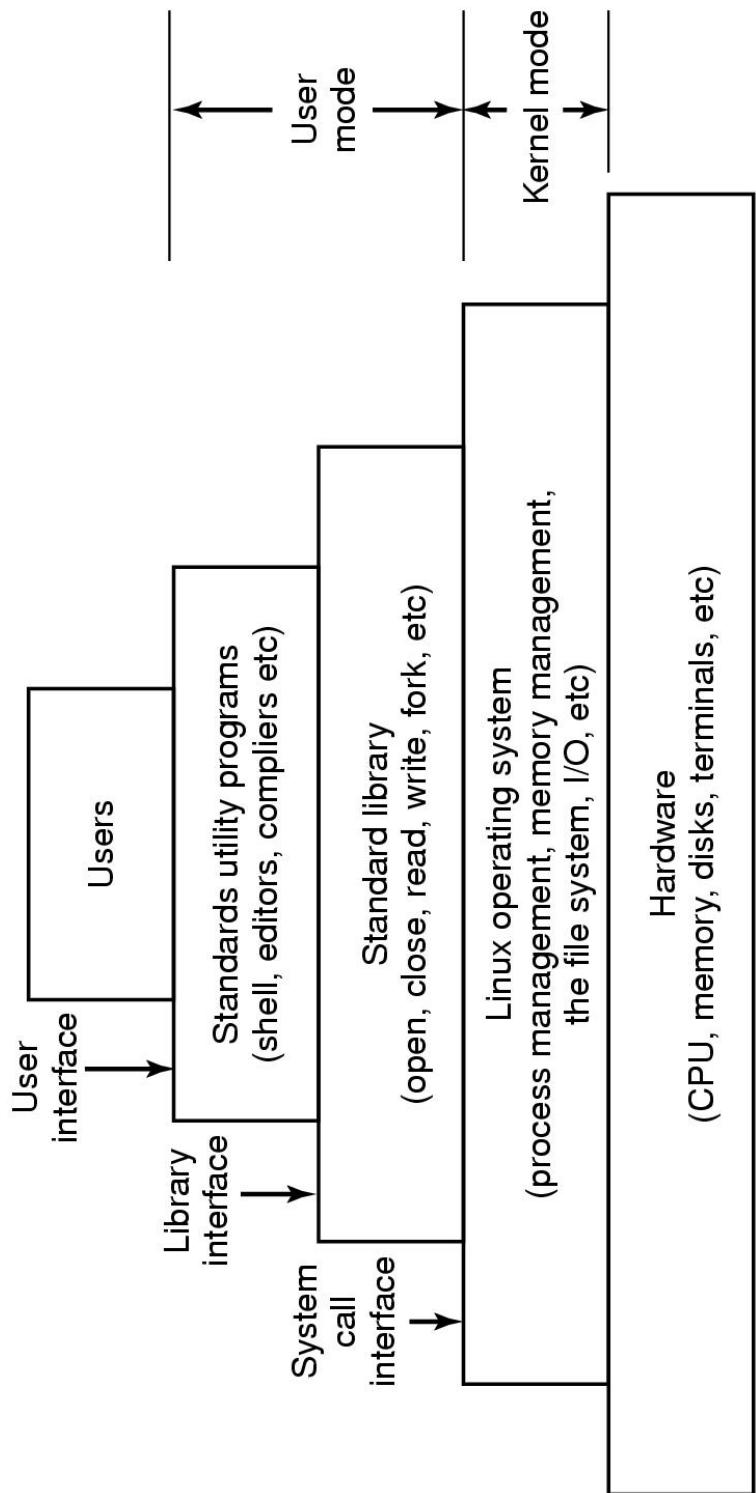


Fig: The layers in a Linux system.

Linux Utility Programs (1)

Categories of utility programs:

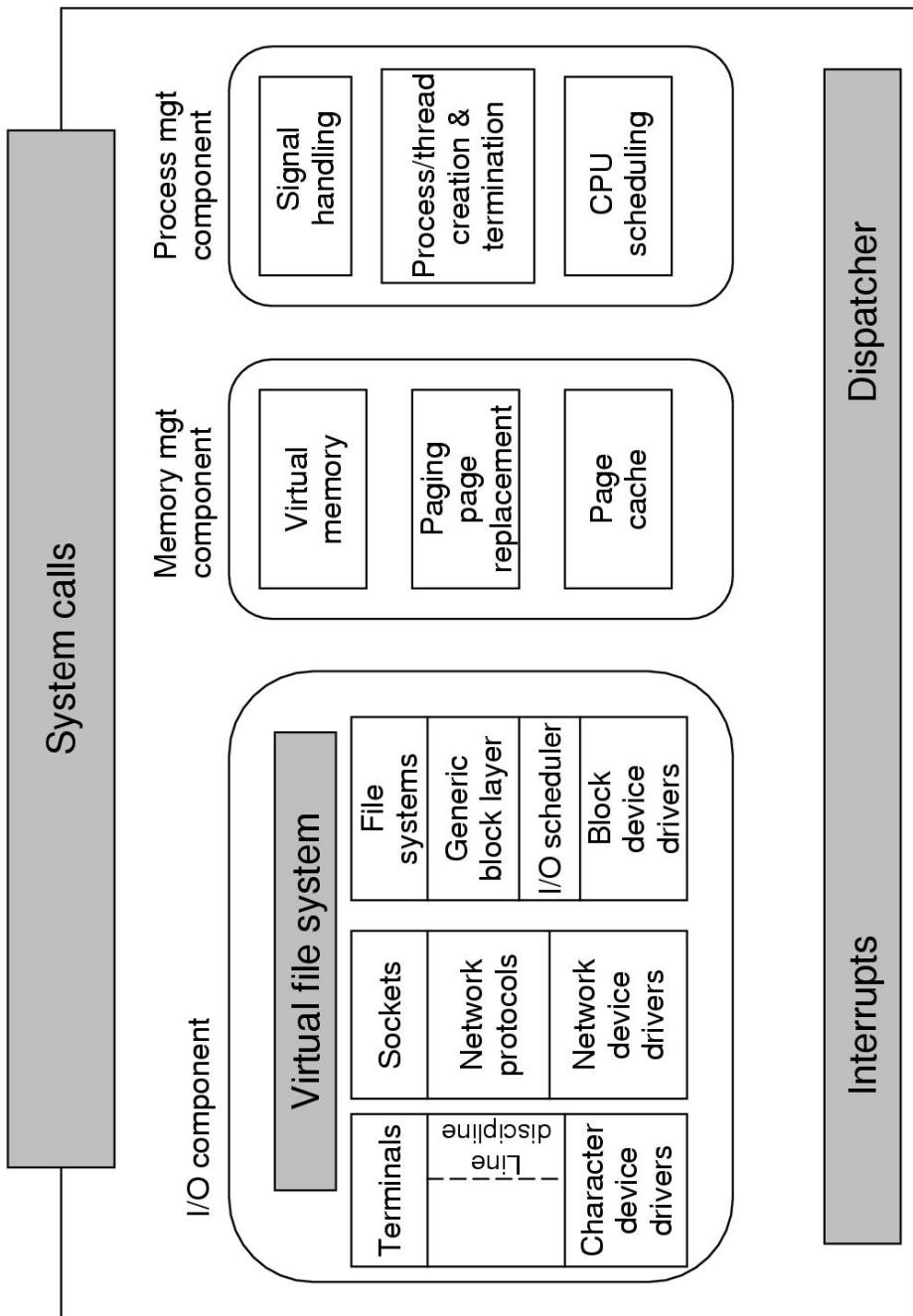
- File and directory manipulation commands.
- Filters.
- Program development tools, such as editors and compilers.
- Text processing.
- System administration.
- Miscellaneous.

Linux Utility Programs (2)

Program	Typical use
cat	Concatenate multiple files to standard output
chmod	Change file protection mode
cp	Copy one or more files
cut	Cut columns of text from a file
grep	Search a file for some pattern
head	Extract the first lines of a file
ls	List directory
make	Compile files to build a binary
mkdir	Make a directory
od	Octal dump a file
paste	Paste columns of text into a file
pr	Format a file for printing
ps	List running processes
rm	Remove one or more files
rmdir	Remove a directory
sort	Sort a file of lines alphabetically
tail	Extract the last lines of a file
tr	Translate between character sets

Fig: A few of the common Linux utility programs required by POSIX.

Kernel Structure



Structure of the Linux kernel

Processes in Linux

```
pid = fork();
if (pid < 0) {
    handle_error();
} else if (pid > 0) {
} else {
    /* child code goes here. */
}
```

/* if the fork succeeds, pid > 0 in the parent */
/* fork failed (e.g., memory or some table is full) */

/* parent code goes here. */

Fig: Process creation in Linux.

Signals in Linux (1)

Signal	Cause
SIGABRT	Sent to abort a process and force a core dump
SIGNALRM	The alarm clock has gone off
SIGFPE	A floating-point error has occurred (e.g., division by 0)
SIGHUP	The phone line the process was using has been hung up
SIGILL	The user has hit the DEL key to interrupt the process
SIGQUIT	The user has hit the key requesting a core dump
SIGKILL	Sent to kill a process (cannot be caught or ignored)
SIGPIPE	The process has written to a pipe which has no readers
SIGSEGV	The process has referenced an invalid memory address
SIGTERM	Used to request that a process terminate gracefully
SIGUSR1	Available for application-defined purposes
SIGUSR2	Available for application-defined purposes

Process Management System Calls in Linux

System call	Description
pid = fork()	Create a child process identical to the parent
pid = waitpid(pid, &statloc, opts)	Wait for a child to terminate
s = execve(name, argv, envp)	Replace a process' core image
exit(status)	Terminate process execution and return status
s = sigaction(sig, &act, &oldact)	Define action to take on signals
s = sigreturn(&context)	Return from a signal
s = sigprocmask(how, &set, &old)	Examine or change the signal mask
s = sigpending(set)	Get the set of blocked signals
s = sigsuspend(sigmask)	Replace the signal mask and suspend the process
s = kill(pid, sig)	Send a signal to a process
residual = alarm(seconds)	Set the alarm clock
s = pause()	Suspend the caller until the next signal

Fig: Some system calls relating to processes.

A Simple Linux Shell

```
while (TRUE) {
    /* repeat forever */
    /* display prompt on the screen */
    /* read input line from keyboard */

    pid = fork();
    if (pid < 0) {
        printf("Unable to fork0);
        continue;
    }

    if (pid != 0) {
        /* parent waits for child */
    } else {
        /* child does the work */
    }
}
```

Fig:10-7. A highly simplified shell.

Implementation of Processes and Threads

Categories of information in the process descriptor:

- Scheduling parameters
- Memory image
- Signals
- Machine registers
 - System call state
- File descriptor table
- Accounting
- Kernel stack
- Miscellaneous

Implementation of Exec

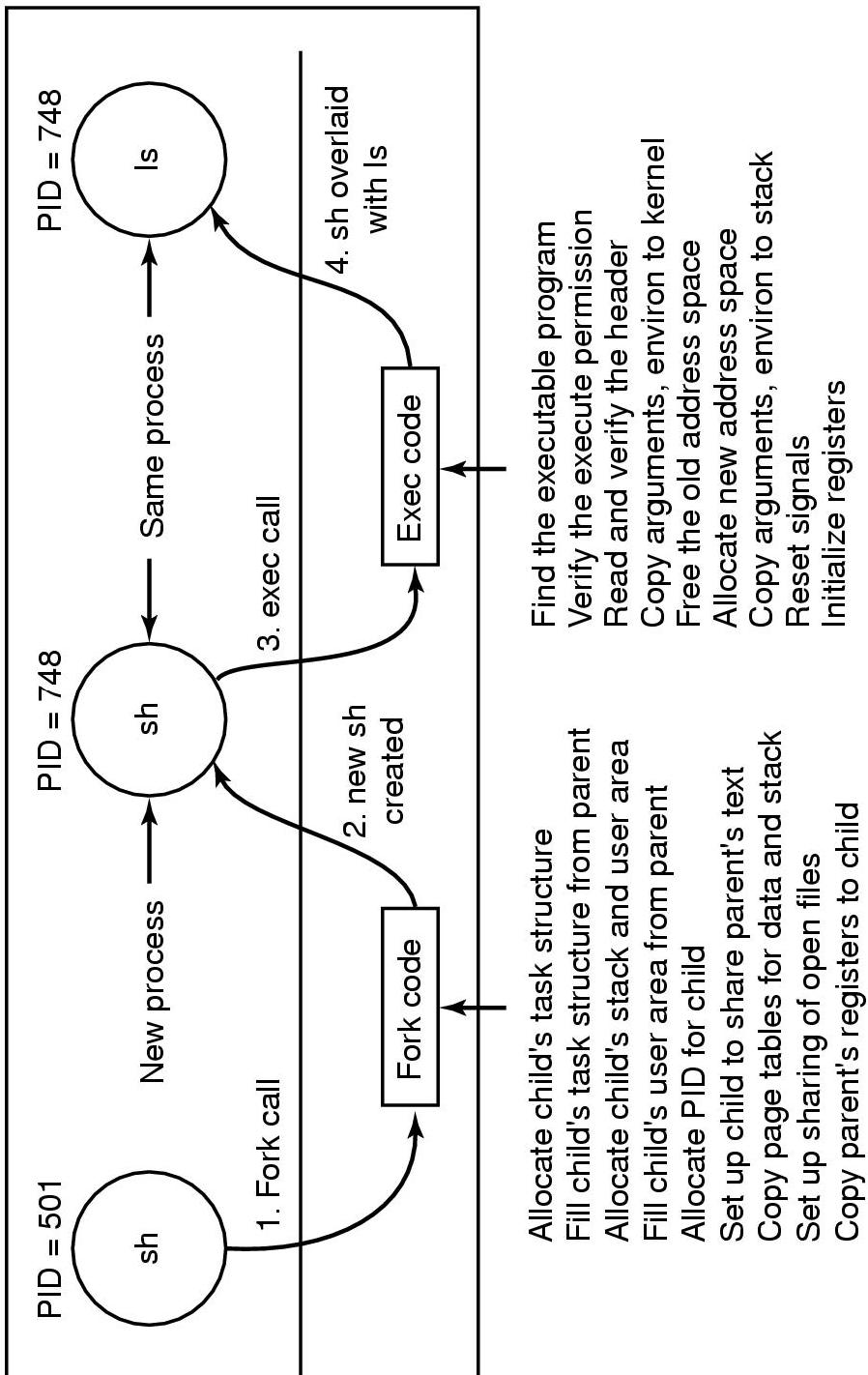


Fig: The steps in executing the command `/s` typed to the shell.

The Clone System Call

Flag	Meaning when set	Meaning when cleared
CLONE_VM	Create a new thread	Create a new process
CLONE_FS	Share umask, root, and working dirs	Do not share them
CLONE_FILES	Share the file descriptors	Copy the file descriptors
CLONE_SIGHAND	Share the signal handler table	Copy the table
CLONE_PID	New thread gets old PID	New thread gets own PID
CLONE_PARENT	New thread has same parent as caller	New thread's parent is caller

Scheduling in Linux (1)

Three classes of threads for scheduling purposes:

- Real-time FIFO.
- Real-time round robin.
- Timesharing.

Scheduling in Linux (2)

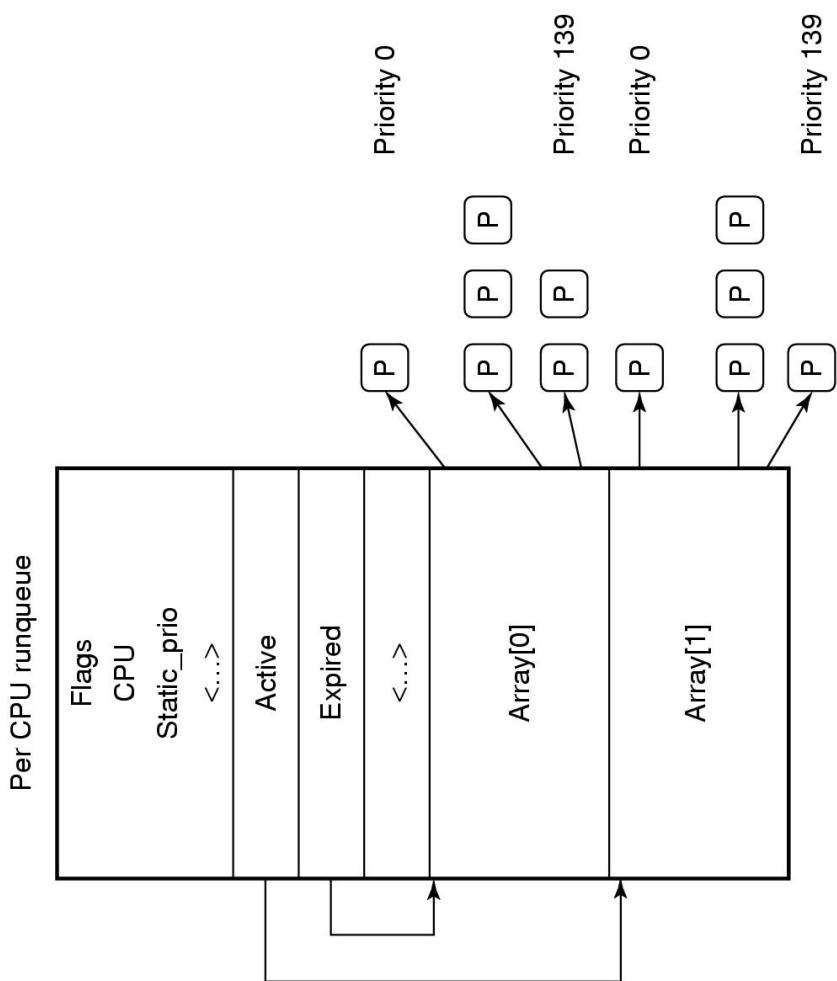


Fig: Illustration of Linux runqueue and priority arrays.

Booting Linux

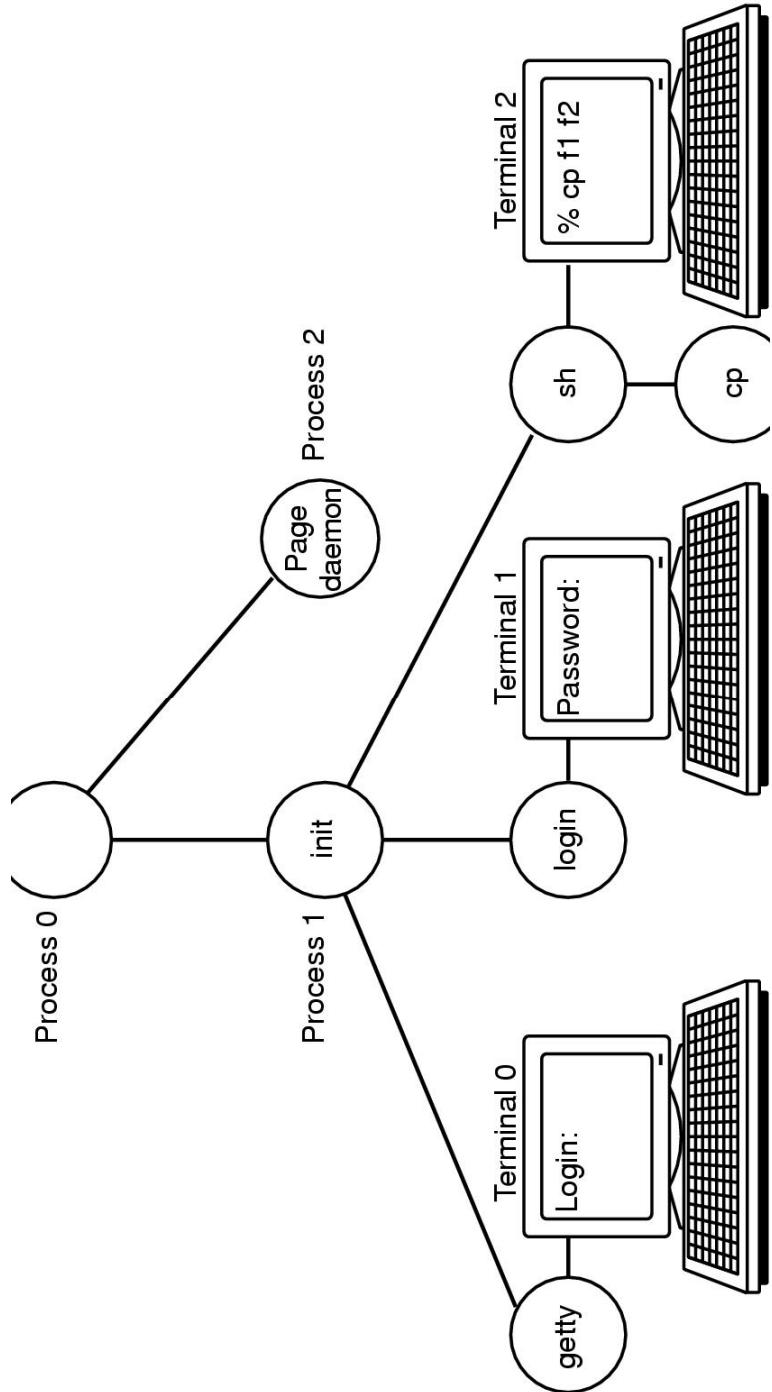


Fig: The sequence of processes used to boot some Linux systems.

Memory Management in Linux (1)

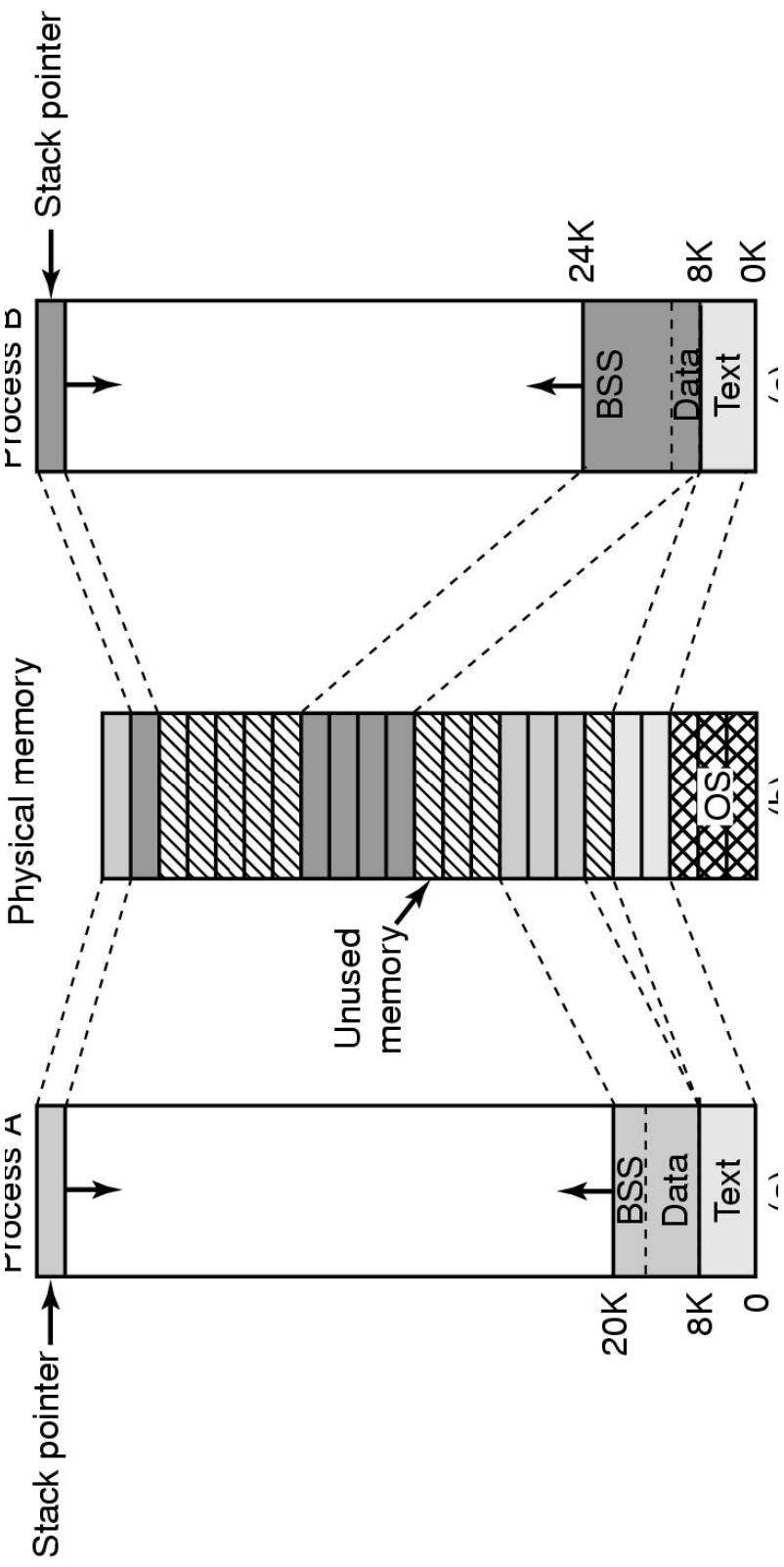


Fig: Process A's virtual address space. (b) Physical memory. (c)
Process B's virtual address space.

Memory Management in Linux (2)

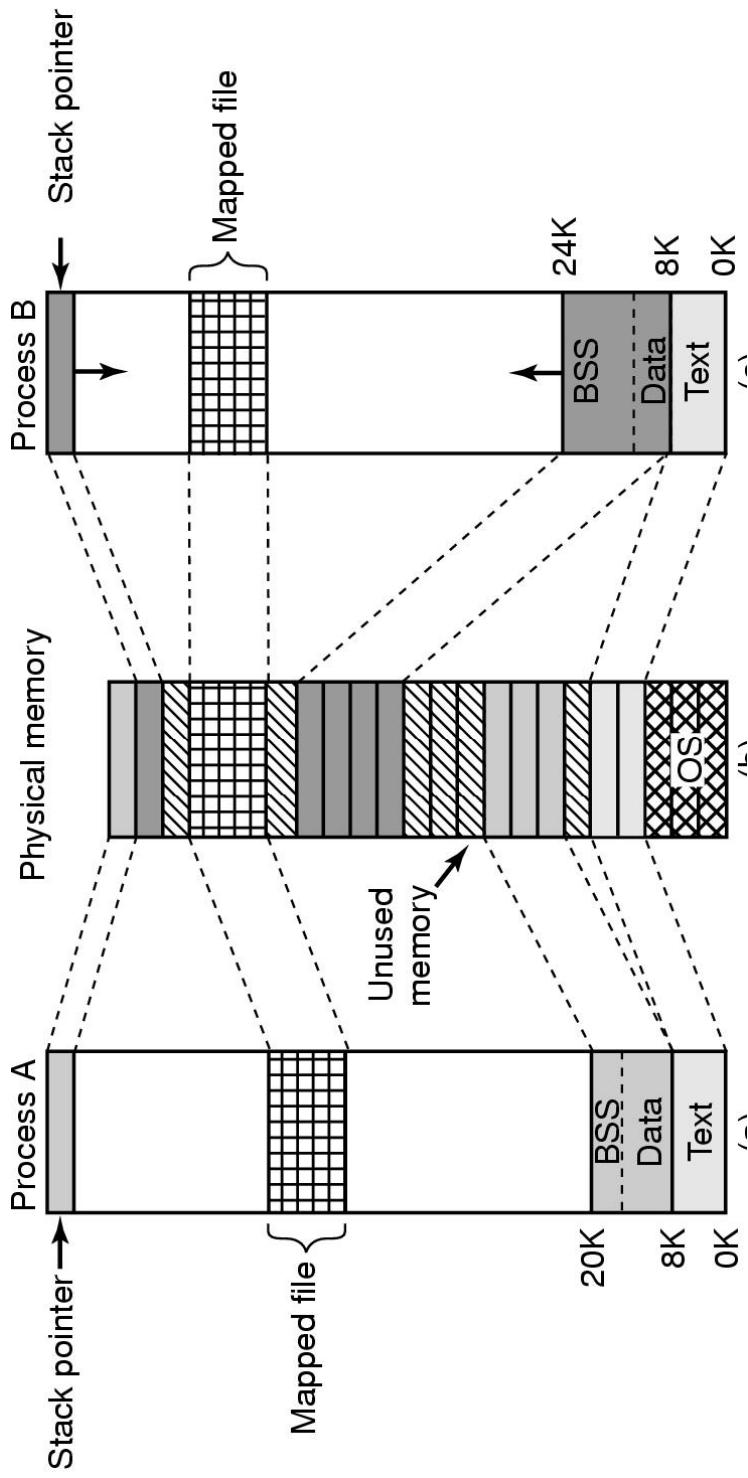


Fig: Two processes can share a mapped file.

Memory Management System Calls in Linux

System call	Description
s = brk(addr)	Change data segment size
a = mmap(addr, len, prot, flags, fd, offset)	Map a file in
s = unmap(addr, len)	Unmap a file

Fig: Some system calls relating to memory management.

Physical Memory Management (2)

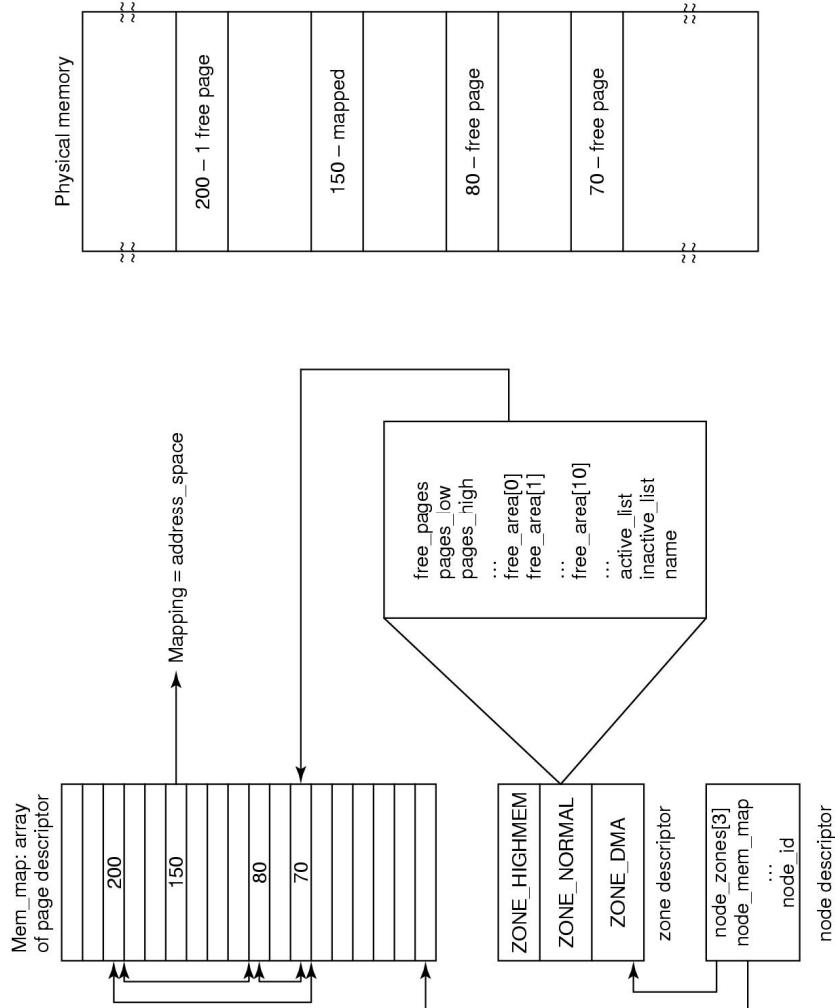


Fig: Linux main memory representation.

Physical Memory Management (3)

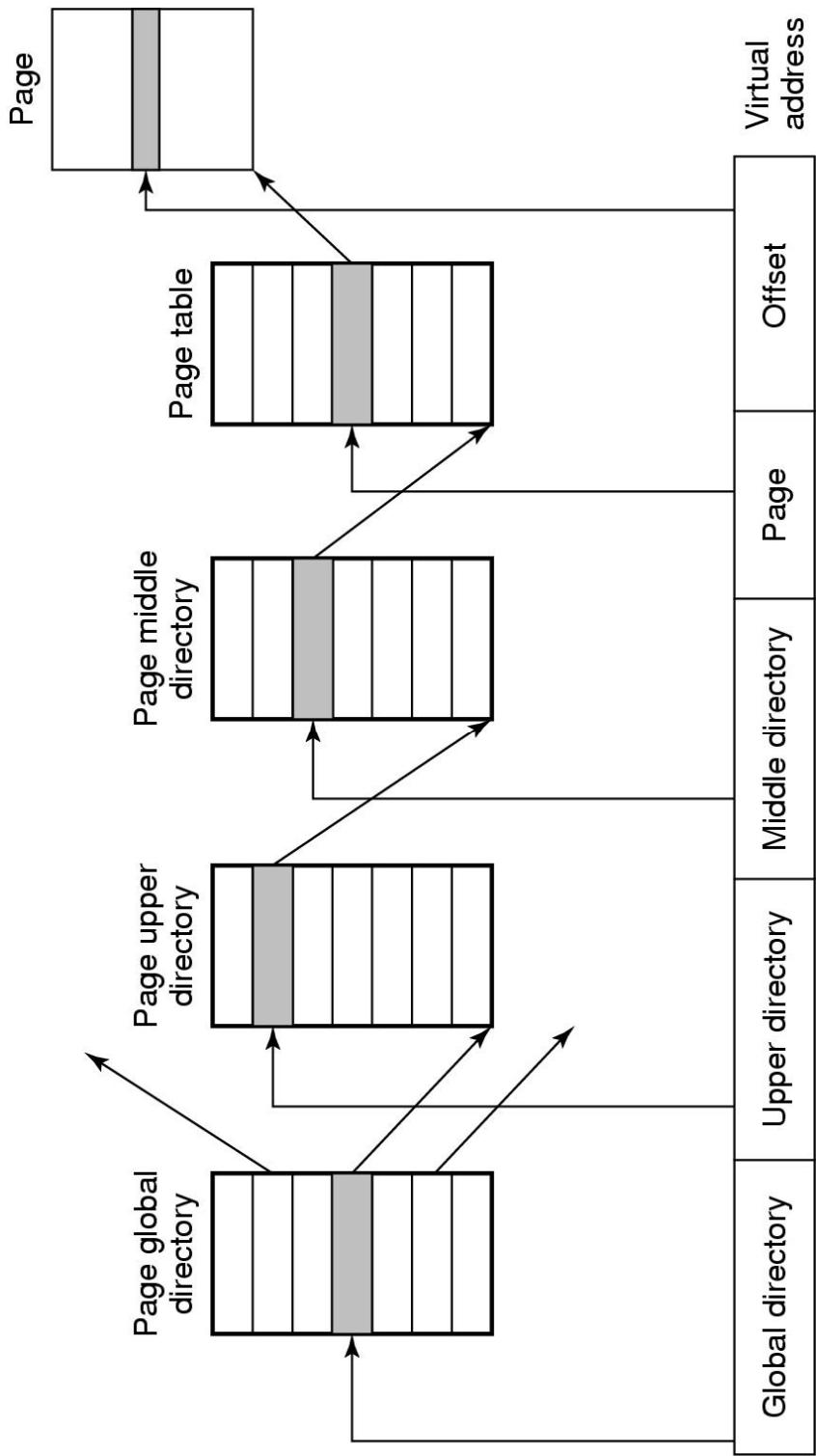


Fig: Linux uses four-level page tables

Memory Allocation Mechanisms

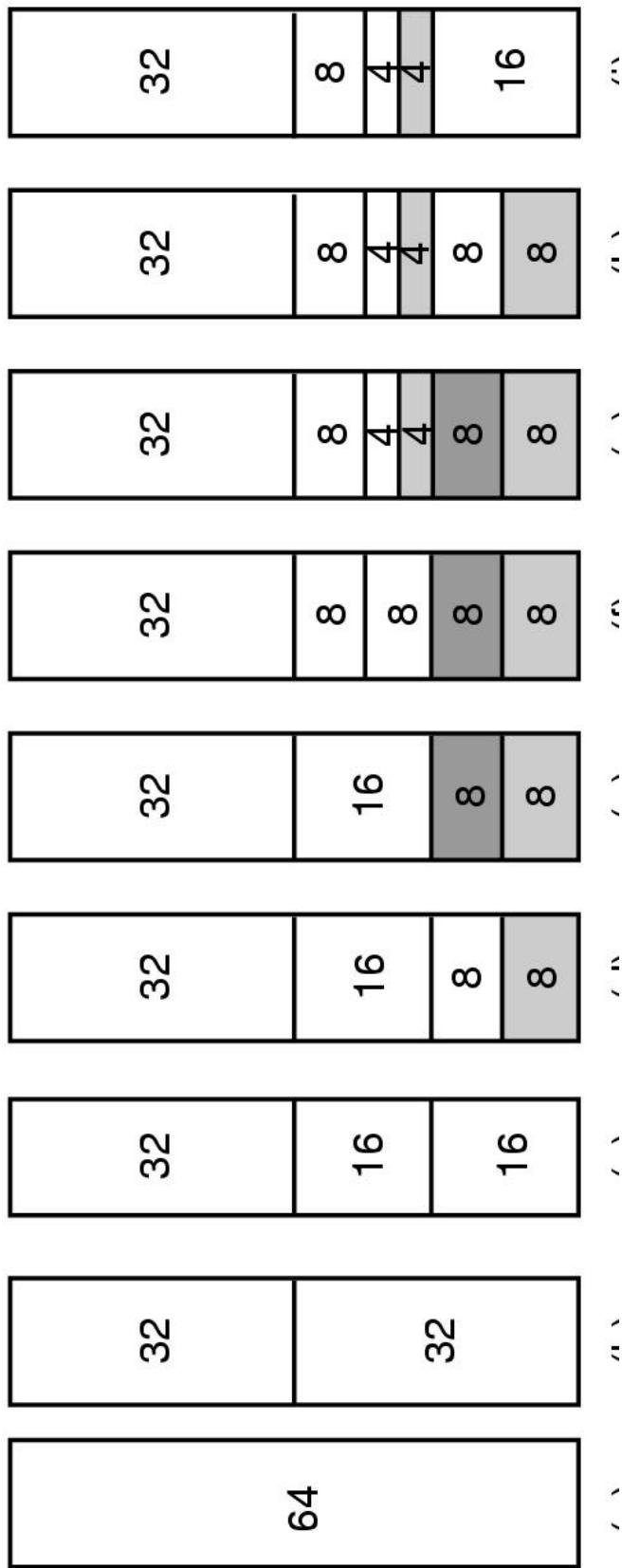


Fig: Operation of the buddy algorithm.

The Page Replacement Algorithm

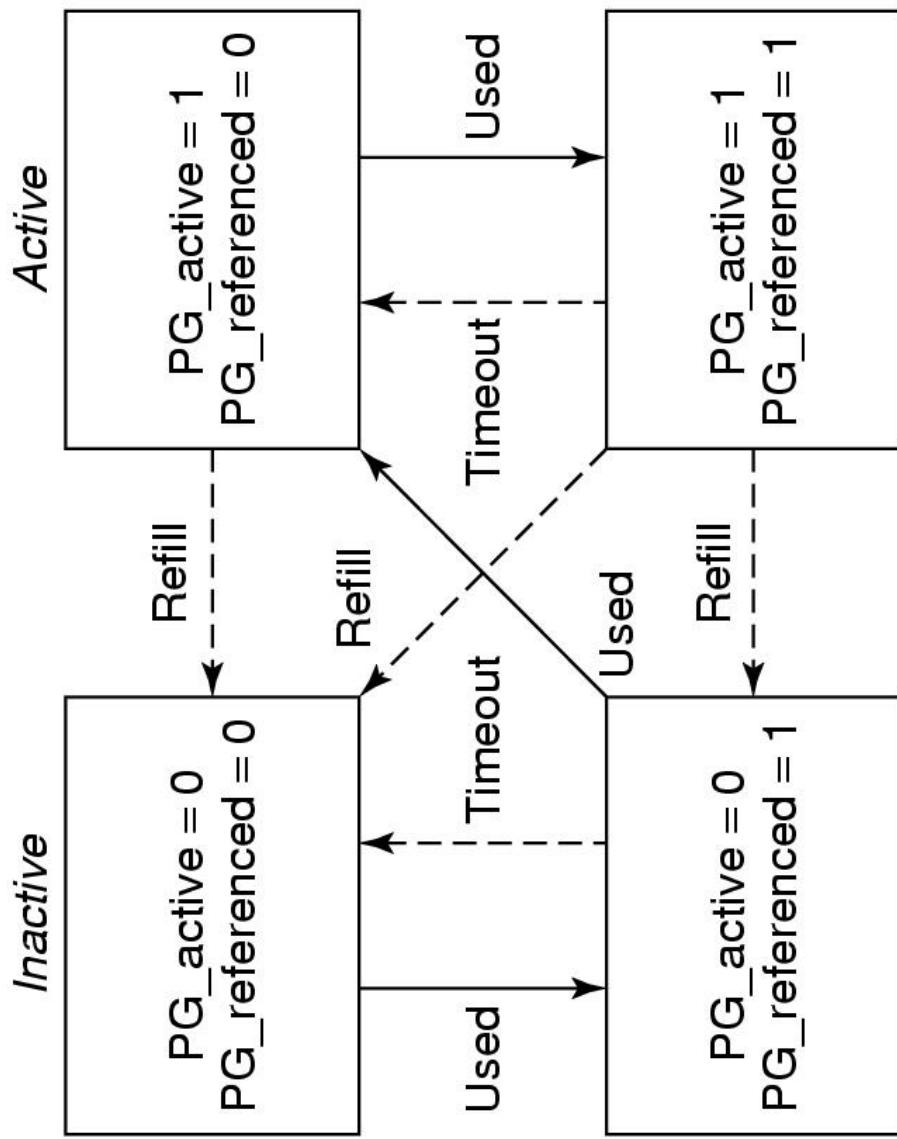


Fig: Page states considered in the page frame replacement algorithm.

Networking

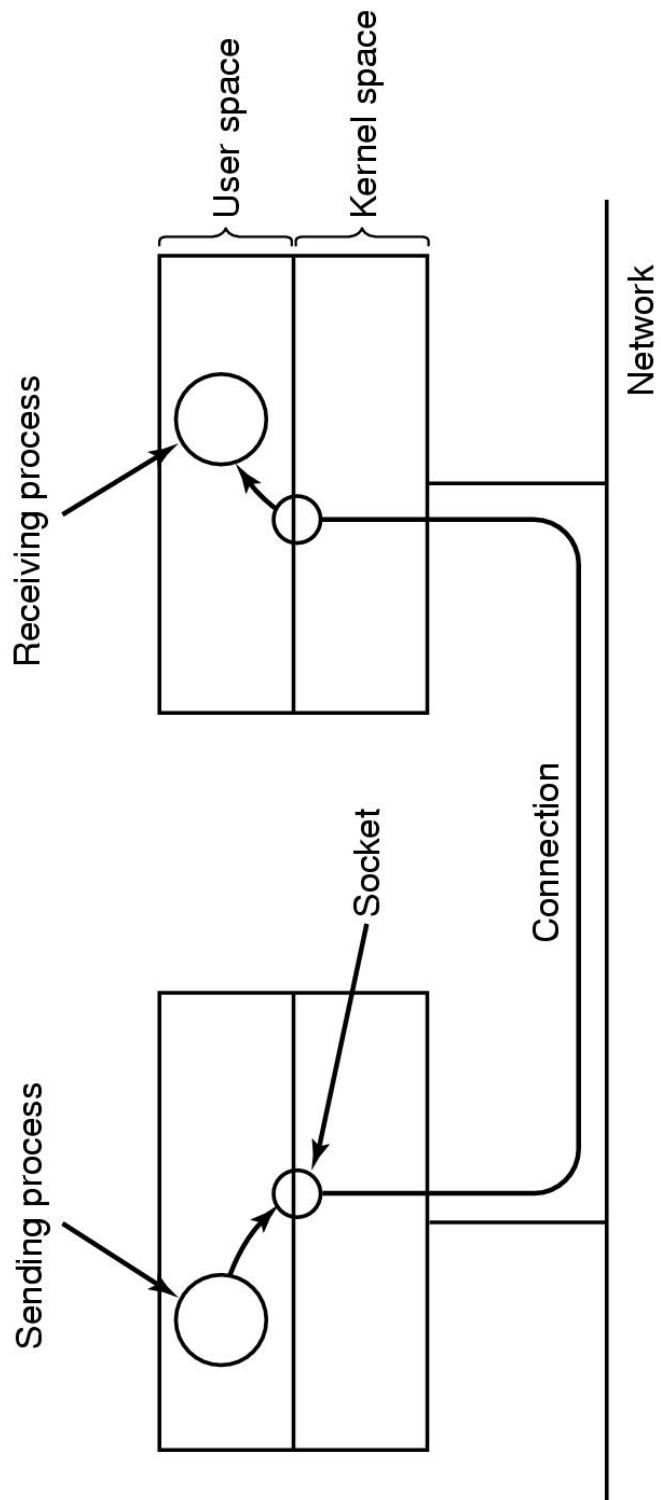


Fig: The uses of sockets for networking

Input/Output System Calls in Linux

Function call	Description
<code>s = cfsetospeed(&termios, speed)</code>	Set the output speed
<code>s = cfsetispeed(&termios, speed)</code>	Set the input speed
<code>s = cfgetospeed(&termios, speed)</code>	Get the output speed
<code>s = cfgetispeed(&termios, speed)</code>	Get the input speed
<code>s = tcsetattr(fd, opt, &termios)</code>	Set the attributes
<code>s = tcgetattr(fd, &termios)</code>	Get the attributes

The Major Device Table

Device	Open	Close	Read	Write	ioctl	Other
Null	Null	Null	Null	Null	Null	...
Memory	Null	Null	Null	Null	Null	...
Keyboard	Null	Null	Null	Null	Null	...
Printer	Null	Null	Null	Null	Null	...
Power	Null	Null	Null	Null	Null	...

Implementation of Input/Output in Linux (2)

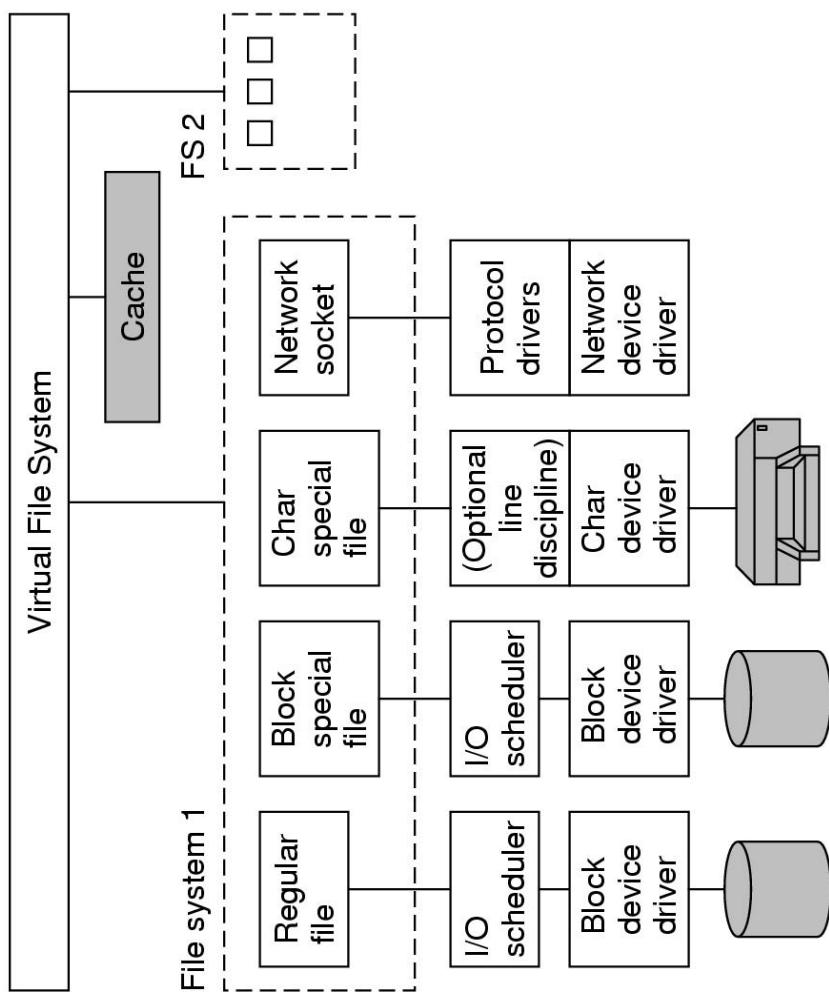


Fig: The Linux I/O system showing one file system in detail.

The Linux File System (1)

Directory	Contents
bin	Binary (executable) programs
dev	Special files for I/O devices
etc	Miscellaneous system files
lib	Libraries
usr	User directories

The Linux File System (2)

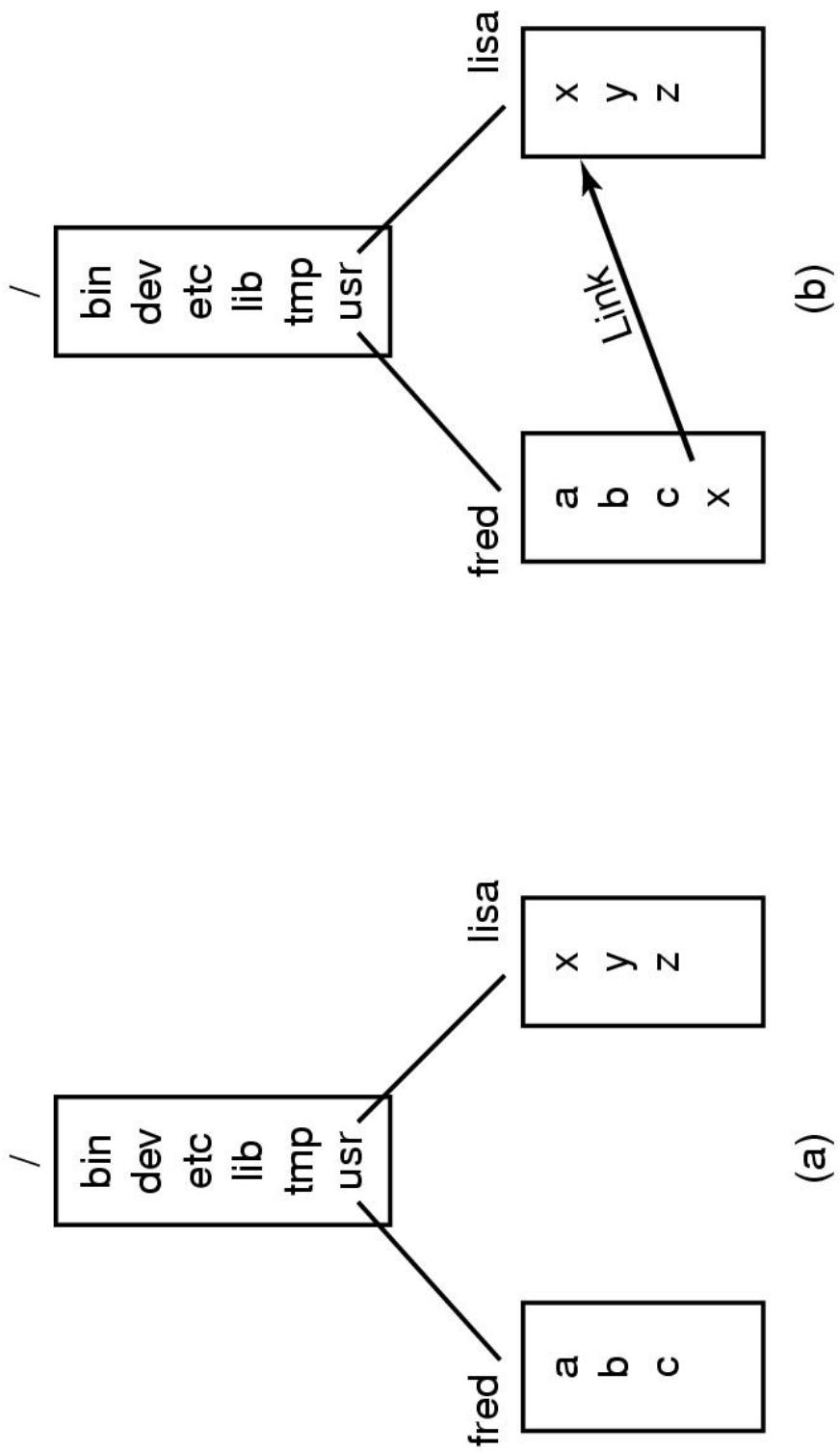


Fig: (a) Before linking. (b) After linking.

The Linux File System (3)

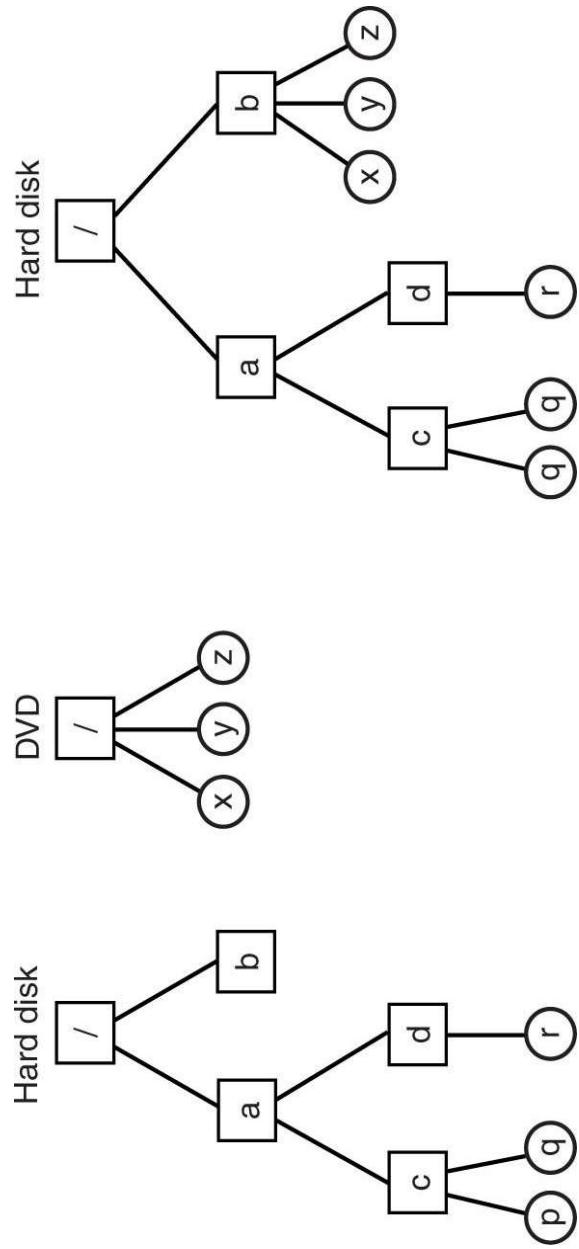


Fig: (a) Separate file systems. (b) After mounting.

The Linux File System (4)

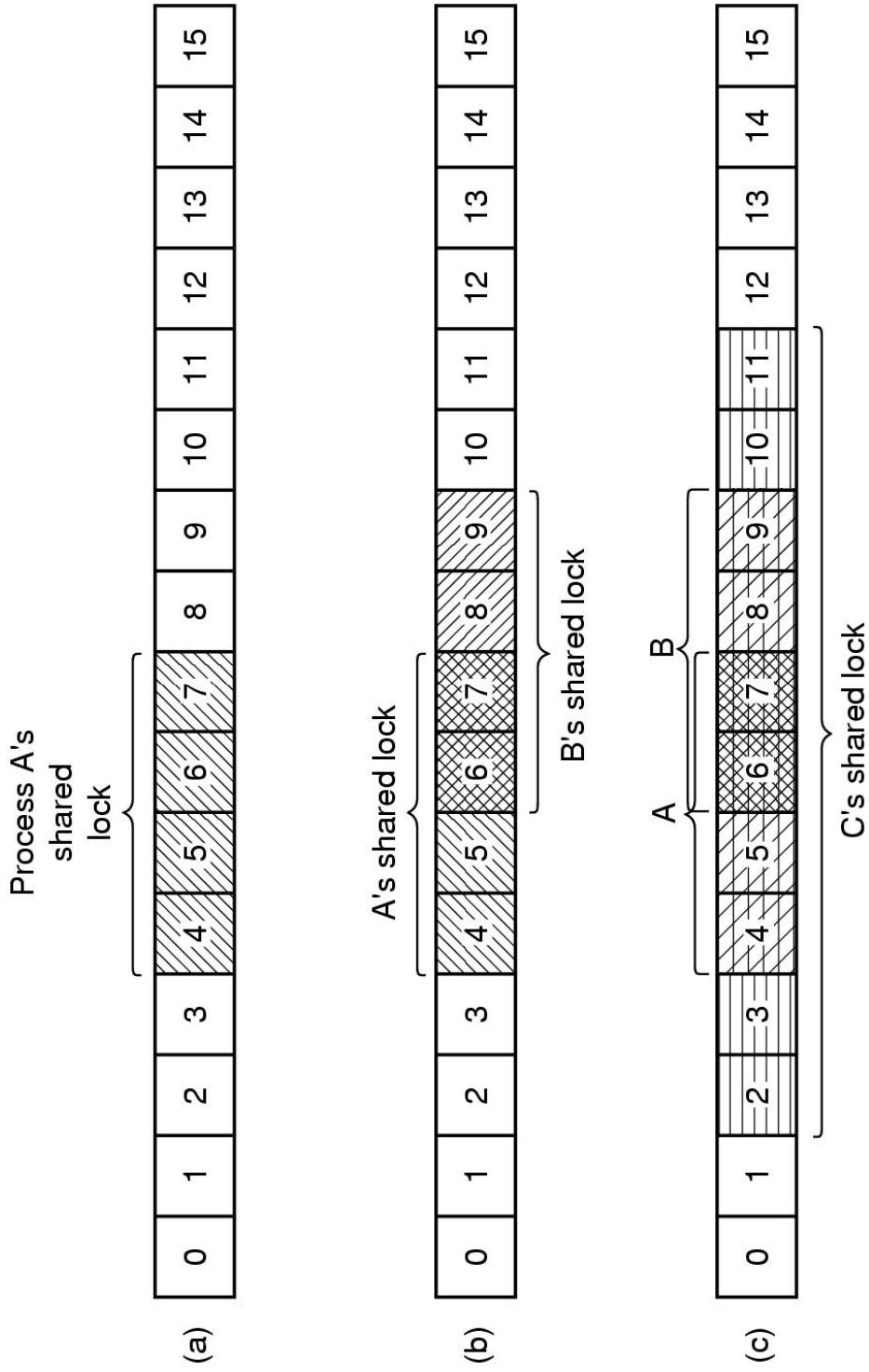


Fig: (a) A file with one lock.
(b) Addition of a second lock. (c) A third lock.

File System Calls in Linux (1)

System call	Description
fd = creat(name, mode)	One way to create a new file
fd = open(file, how, ...)	Open a file for reading, writing, or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information
s = fstat(fd, &buf)	Get a file's status information
s = pipe(&fd[0])	Create a pipe
s = fcntl(fd, cmd, ...)	File locking and other operations

Fig: System calls relating to files.

File System Calls in Linux (2)

Device the file is on
I-node number (which file on the device)
File mode (includes protection information)
Number of links to the file
Identity of the file's owner
Group the file belongs to
File size (in bytes)
Creation time
Time of last access
Time of last modification

File System Calls in Linux (3)

System call	Description
s = mkdir(path, mode)	Create a new directory
s = rmdir(path)	Remove a directory
s = link(oldpath, newpath)	Create a link to an existing file
s = unlink(path)	Unlink a file
s = chdir(path)	Change the working directory
dir = opendir(path)	Open a directory for reading
s = closedir(dir)	Close a directory
dirent = readdir(dir)	Read one directory entry
rewinddir(dir)	Rewind a directory so it can be reread

The Linux Virtual File System

Object	Description	Object	
Symbolic Link	A pointer to another file's inode.	Subprocess	A child process of the current process.
Character Device	A device that handles character data.	Device	A physical or logical connection to hardware.
Block Device	A device that handles block data.	File	A named file.
Directory	A container for files and other directories.		

NFS Protocols

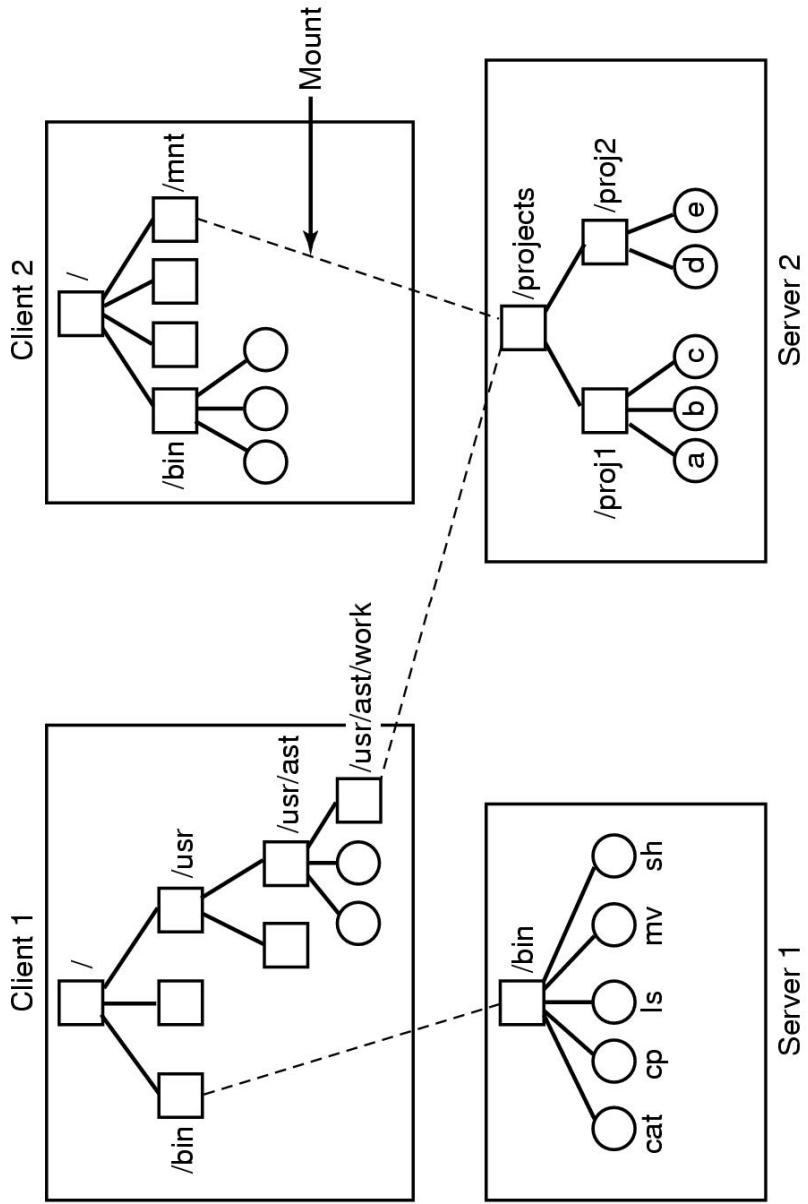


Fig: Examples of remote mounted file systems.
Directories shown as squares, files shown as circles.

NFS Implementation

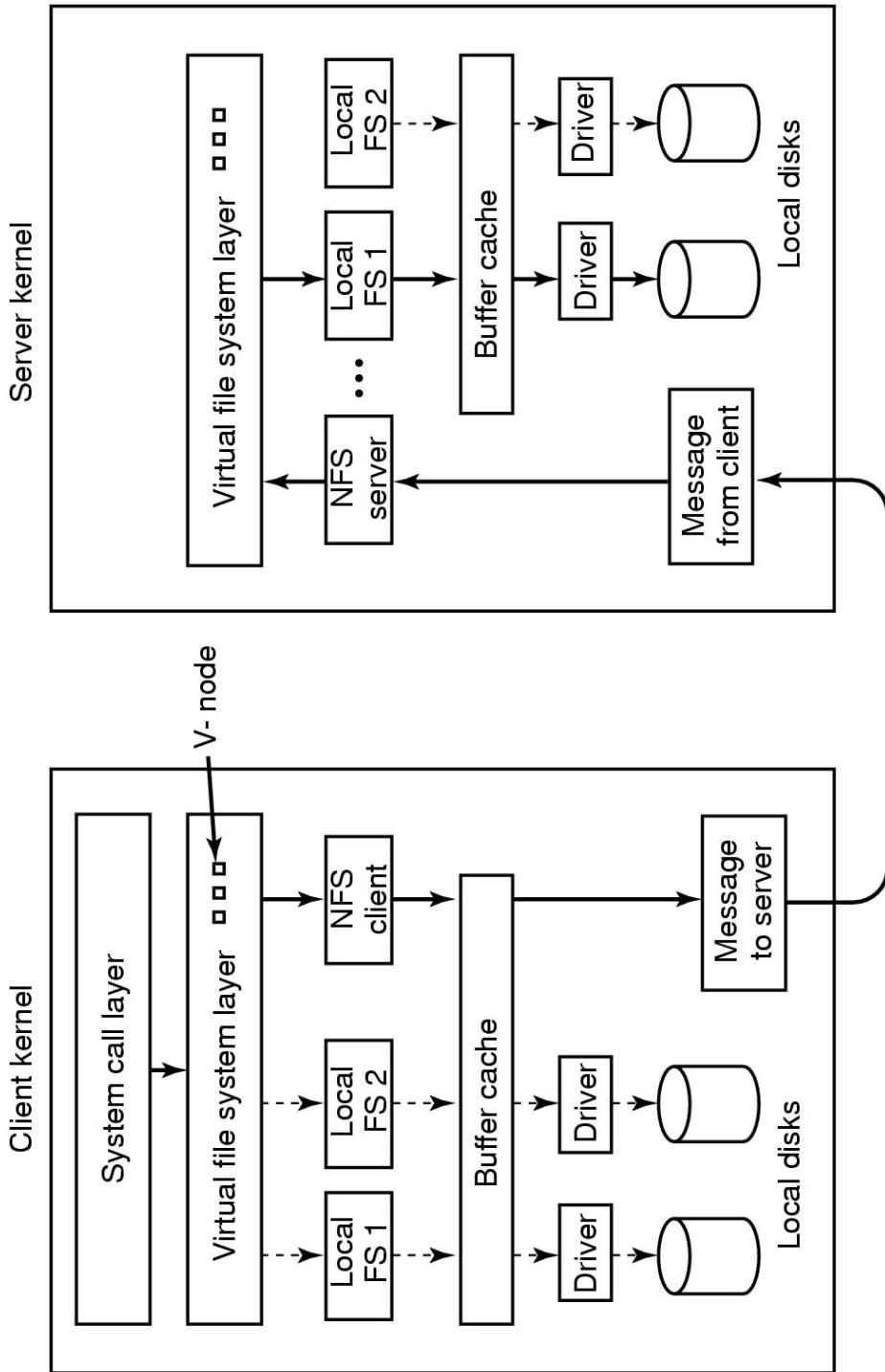


Fig: The NFS layer structure

Security In Linux

Binary	Symbolic	Allowed file accesses
111000000	rwx-----	Owner can read, write, and execute
111111000	rwxrwx----	Owner and group can read, write, and execute
110100000	rw-r-----	Owner can read and write; group can read
110100100	rw-r--r--	Owner can read and write; all others can read
111101101	rwxr-xr-x	Owner can do everything, rest can read and execute
000000000	-----	Nobody has any access
000000111	-----rwx	Only outsiders have access (strange, but legal)

Security System Calls in Linux

System call	Description
s = chmod(path, mode)	Change a file's protection mode
s = access(path, mode)	Check access using the real UID and GID
uid = getuid()	Get the real UID
uid = geteuid()	Get the effective UID
gid = getgid()	Get the real GID
gid = getegid()	Get the effective GID
s = chown(path, owner, group)	Change owner and group
s = setuid(uid)	Set the UID
s = setgid(gid)	Set the GID



**Swami Keshvanand Institute of Technology, Management & Gramothan,
Ramnagar, Jagatpura, Jaipur-302017, INDIA**

Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956

Tel. : +91-0141- 5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in

Tutorial Sheet

Process Management

Consider three processes (process id 0, 1, 2 respectively) with compute time bursts 2, 4 and 8 time units. All processes arrive at time zero. Consider the longest remaining time first (LRTF) scheduling algorithm. In LRTF ties are broken by giving priority to the process with the lowest process id. The average turn around time is:

- (A) 13 units
- (B) 14 units
- (C) 15 units
- (D) 16 units

Answer: (A)

Consider three processes, all arriving at time zero, with total execution time of 10, 20 and 30 units, respectively. Each process spends the first 20% of execution time doing I/O, the next 70% of time doing computation, and the last 10% of time doing I/O again. The operating system uses a shortest remaining compute time first scheduling algorithm and schedules a new process either when the running process gets blocked on I/O or when the running process finishes its compute burst. Assume that all I/O operations can be overlapped as much as possible. For what percentage of time does the CPU remain idle?

- (A) 0%
- (B) 10.6%
- (C) 30.0%
- (D) 89.4%

Answer: (B)

Consider three CPU-intensive processes, which require 10, 20 and 30 time units and arrive at times 0, 2 and 6, respectively. How many context switches are needed if the operating system implements a shortest remaining time first scheduling algorithm? Do not count the context switches at time zero and at the end.

- (A) 1
- (B) 2
- (C) 3
- (D) 4

Answer: (B)



Tutorial sheet

Memory Management

A CPU generates 32-bit virtual addresses. The page size is 4 KB. The processor has a translation look-aside buffer (TLB) which can hold a total of 128 page table entries and is 4-way set associative. The minimum size of the TLB tag is:

- (A) 11 bits
- (B) 13 bits
- (C) 15 bits
- (D) 20 bits

Answer: (C)

Consider the virtual page reference string

1, 2, 3, 2, 4, 1, 3, 2, 4, 1

On a demand paged virtual memory system running on a computer system that main memory size of 3 pages frames which are initially empty. Let LRU, FIFO and OPTIMAL denote the number of page faults under the corresponding page replacements policy. Then

- (A) OPTIMAL < LRU < FIFO
- (B) OPTIMAL < FIFO < LRU
- (C) OPTIMAL = LRU
- (D) OPTIMAL = FIFO

Answer: (B)

Let the page fault service time be 10ms in a computer with average memory access time being 20ns. If one page fault is generated for every 10^6 memory accesses, what is the effective access time for the memory?

- (A) 21ns
- (B) 30ns
- (C) 23ns
- (D) 35ns

Answer: (B)

A system uses FIFO policy for page replacement. It has 4 page frames with no pages loaded to begin with. The system first accesses 100 distinct pages in some order and then accesses the same 100 pages but now in the reverse order. How many page faults will occur?

- (A) 196
- (B) 192
- (C) 197
- (D) 195

Answer: (A)



Tutorial sheet

Deadlock

A counting semaphore was initialized to 10. Then 6 P (wait) operations and 4 V (signal) operations were completed on this semaphore. The resulting value of the semaphore is

- (A) 0
- (B) 8
- (C) 10
- (D) 12

Answer: (B)

Tutorial sheet

Disk Management

Consider a disk with 200 tracks and the queue has random requests from different processes in the order: 55, 58, 39, 18, 90, 160, 150, 38, 184

Initially arm is at 100. Find the Average Seek length using FIFO, SSTF, SCAN and C-SCAN algorithm.

Consider a disk pack with the following specifications- 16 surfaces, 128 tracks per surface, 256 sectors per track and 512 bytes per sector.

Answer the following questions-

1. What is the capacity of disk pack?
2. What is the number of bits required to address the sector?
3. If the format overhead is 32 bytes per sector, what is the formatted disk space?
4. If the format overhead is 64 bytes per sector, how much amount of memory is lost due to formatting?
5. If the diameter of innermost track is 21 cm, what is the maximum recording density?
6. If the diameter of innermost track is 21 cm with 2 KB/cm, what is the capacity of one track?
7. If the disk is rotating at 3600 RPM, what is the data transfer rate?
8. If the disk system has rotational speed of 3000 RPM, what is the average access time with a seek time of 11.5 msec?



Tutorial sheet

Deadlock

An operating system uses the Banker's algorithm for deadlock avoidance when managing the allocation of three resource types X, Y, and Z to three processes P0, P1, and P2. The table given below presents the current system state. Here, the Allocation matrix shows the current number of resources of each type allocated to each process and the Max matrix shows the maximum number of resources of each type required by each process during its execution.

	Allocation			Max		
	X	Y	Z	X	Y	Z
P0	0	0	1	8	4	3
P1	3	2	0	6	2	0
P2	2	1	1	3	3	3

There are 3 units of type X, 2 units of type Y and 2 units of type Z still available. The system is currently in a safe state. Consider the following independent requests for additional resources in the current state:

REQ1: P0 requests 0 units of X,
0 units of Y and 2 units of Z

REQ2: P1 requests 2 units of X,
0 units of Y and 0 units of Z

Which one of the following is TRUE?

- (A) Only REQ1 can be permitted.
- (B) Only REQ2 can be permitted.
- (C) Both REQ1 and REQ2 can be permitted.
- (D) Neither REQ1 nor REQ2 can be permitted

Answer: (B)

A system has 6 identical resources and N processes competing for them. Each process can request atmost 2 resources. Which one of the following values of N could lead to a deadlock?

- (A) 1
- (B) 2
- (C) 3
- (D) 4

Answer: (D)

Question Bank

SCSD

OS Assignment

Deadlock Gate Questions

1. A system has 6 identical resources and N processes competing for them. Each process can request at most 2 resources. Which one of the following values of N could lead to a deadlock? [2015]

- (A) 1 (B) 2 (C) 3 (D) 4

2. Consider a hard disk with 16 recording surfaces (0-15) having 16384 cylinders (0-16383) and each cylinder contains 64 sectors (0-63). Data storage capacity in each sector is 512 bytes. Data are organized cylinder-wise and the addressing format is . A file of size 42797 KB is stored in the disk and the starting disk location of the file is

:1200, 9, 40>. What is the cylinder number of the last sector of the file, if it is stored in a contiguous manner? [2013]

- (A) 1281 (B) 1282
(C) 1283 (D) 1284

3. Consider the following policies for preventing deadlock in a system with mutually exclusive resources.

I. Processes should acquire all their resources at the beginning of execution. If any resource is not available, all resources acquired so far are released

II. The resources are numbered uniquely, and processes are allowed to request for resources only in increasing resource numbers

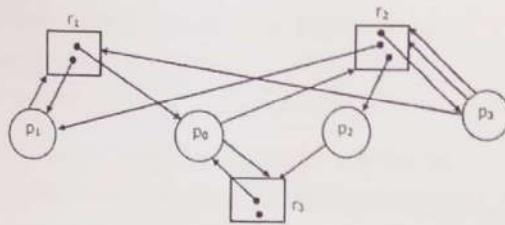
III. The resources are numbered uniquely, and processes are allowed to request for resources only in decreasing resource numbers

IV. The resources are numbered uniquely. A process is allowed to request only for a resource with resource number larger than its currently held resources

Which of the above policies can be used for preventing deadlock?

- (A) Any one of I and III but not II or IV
(B) Any one of I, III, and IV but not II
(C) Any one of II and III but not I or IV
(D) Any one of I, II, III, and IV

4. Consider the resource allocation graph given in the figure.



- (a) Find if the system is in a deadlock state.
(b) Otherwise, find a safe sequence.

5. A computer system has 6 tape drives, with n process competing for them. Each process may need 3 tape drives. The maximum value of n for which the system is guaranteed to be deadlock free is:

- (a) 2 (b) 3 (c) 4 (d) 1

6. Consider a system having m resources of the same type. These resources are shared by 3 processes A, B and C, which have peak demands of 3, 4 and 6 respectively. For what value of m deadlock will not occur?

- (a) 7 (b) 9 (c) 10 (d) 13 (e) 15

7. An operating system contains 3 user processes each requiring 2 units of resource R. The minimum number of units of R such that no deadlocks will ever arise is

- (a) 3 (b) 5 (c) 4 (d) 6

8. Which of the following is NOT a valid deadlock prevention scheme?

- (a) Release all resources before requesting a new resource
(b) Number the resources uniquely and never request a lower numbered resource than the last one requested.
(c) Never request a resource after releasing any resource
(d) Request and all required resources be allocated before execution.

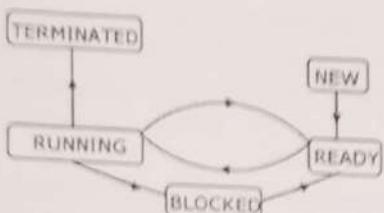
9. Which of the following is NOT true of deadlock prevention and deadlock avoidance schemes?

- (A) In deadlock prevention, the request for resources is always granted if the resulting state is safe.
(B) In deadlock avoidance, the request for resources is always granted if the result state is safe.
(C) Deadlock avoidance is less restrictive than deadlock prevention.
(D) Deadlock avoidance requires knowledge of resource requirements a priori.

1. Which scheduling policy is most suitable for a time shared operating system?

- (a) Shortest Job First (b) Round Robin
- (c) First Come First Serve (d) Elevator

2. The process state transition diagram in Fig is representative of



- (a) a batch operating system
- (b) an operating system with a preemptive scheduler
- (c) an operating system with a non-preemptive scheduler
- (d) a uni-programmed operating system.

3. Which of the following is an example of spooled device?

- (a) A line printer used to print the output of a number of jobs.
- (b) A terminal used to enter input data to a running program.
- (c) A secondary storage device in a virtual memory system.
- (d) A graphic display device.

4. Which of the following is an example of a spooled device?

- (a) The terminal used to enter the input data for the C program being executed.
- (b) An output device used to print the output of a number of jobs.
- (c) The secondary memory device in a virtual storage system.
- (d) The swapping area on a disk used by the swapper.

5. Consider n processes sharing the CPU in a round-robin fashion. Assuming that each process switch takes s seconds, what must be the quantum size q such that the overhead resulting from process switching is minimized but at the same time each process is guaranteed to get its turn at the CPU at least every t seconds?

$$(a) q \leq \frac{t - ns}{n-1} \quad (b) q \geq \frac{t - ns}{n-1} \quad (c) q \leq \frac{t - ns}{n+1} \quad (d) q \geq \frac{t - ns}{n+1}$$

6. System calls are usually invoked by using

- (a) a software interrupt. (b) polling.
- (c) an indirect jump. (d) a privileged instruction.

7. A processor needs software interrupt to

- (a) test the interrupt system of the processor.
- (b) implement co-routines.
- (c) obtain system services which need execution of privileged instructions.
- (d) return from subroutine.

8. A CPU has two modes-privileged and non-privileged. In order to change the mode from privileged to non-privileged

- (a) a hardware interrupt is needed.
- (b) a software interrupt is needed.
- (c) a privileged instruction (which does not generate an interrupt) is needed.
- (d) a non-privileged instruction (which does not generate an interrupt) is needed.

9. Consider a set of n tasks with known runtimes r_1, r_2, \dots, r_n to be run on a uniprocessor machine. Which of the following processor scheduling algorithms will result in the maximum throughput?

- (a) Round-Robin. (b) Shortest-Job-First.
- (c) Highest-Response-Ratio-Next.
- (d) First-Come-First-Served.

10. Which of the following scheduling algorithms is non-preemptive?

- (a) Round Robin. (b) First-In First-Out.
- (c) Multilevel Queue Scheduling.
- (d) Multilevel Queue Scheduling with Feedback.

11. Consider three CPU-intensive processes, which require 10, 20 and 30 time units and arrive at times 0, 2 and 6, respectively. How many context switches are needed if the operating system implements a shortest remaining time first scheduling algorithm? Do not count the context switches at time zero and at the end.

- (a) 1 (b) 2 (c) 3 (d) 4

12. A CPU generally handles an interrupt by executing an interrupt service routine

- (A) As soon as an interrupt is raised.
- (B) By checking the interrupt register at the end of fetch cycle.
- (C) By checking the interrupt register after finishing the execution of the current instruction.
- (D) By checking the interrupt register at fixed time intervals.

5CSD
OS Assignment

12. Consider the following statements about user level threads and kernel level threads. Which one of the following statements is FALSE?

- (A) Context switch time is longer for kernel level threads than for user level threads.
 - (B) User level threads do not need any hardware support.
 - (C) Related kernel level threads can be scheduled on different processors in a multi-processor system.
 - (D) Blocking one kernel level thread blocks all related threads.

13. Which of the following statements are true?
I. Shortest remaining time first scheduling may cause

14. A computer handles several interrupt sources of which the following are relevant for this question.

- Interrupt from CPU temperature sensor(raise interrupt CPU temperature is too high)
 - Interrupt from Mouse(raise interrupt if the mouse is moved or a button is pressed)
 - Interrupt from Keyboard(raise interrupt when a key is pressed or release)
 - Interrupt from Hard Disk(raise interrupt when a disk read is completed)Which one these will be handled at the HIGHEST priority?
 - (A) Interrupt from Hard Disk
 - (B) Interrupt from Mouse
 - (C) Interrupt from Keyboard
 - (D) Interrupt from CPU temp sensor

15. Let the time taken to switch between user and kernel modes of execution be t_1 while the time taken to switch between two processes be t_2 . Which of the following is TRUE?

- (A) $t_1 > t_2$ (B) $t_1 = t_2$ (C) $t_1 < t_2$
(D) Nothing can be said about the relation between t_1 and t_2 .

16. A thread is usually defined as a 'light weight process' because an operating system (OS) maintains smaller data structures for a thread than for a process. In relation to this, which of the following is TRUE?

- (A) On per-thread basis, the OS maintains only CPU register state.

- (B) The OS does not maintain a separate stack for each thread.

- (C) On per-thread basis, the OS does not maintain virtual memory state.
 - (D) On per thread basis, the OS maintains only scheduling and accounting information.

17. A process executes the code

fork 0; fork 0; fork 0;

The total number of child processes created is

- (A) 3 (B) 4 (C) 7 (D) 8

18. State the undesirable characteristic of each of the following criteria for measuring performance of an operating system:

- (a) Turn-around time (b) Waiting time

19. The highest-response ratio next scheduling policy favours.....jobs, but it also limits the waiting time ofjobs.

20. Which of the following is an example of a spooled device?

- (a) The terminal used to the input data for a program being executed.

(b) The secondary memory device in a virtual memory system

(c) A line printer used to print the output of a number of jobs.
(d) None of the above

(d) None of the above

21. Four jobs are waiting to be run. Their expected run times are 6, 3, 5 and x. In what order should they be run to minimize the average response time?

22. A multi-user, multi-processing operating system

(a) Address translation (b) DMA for disk transfer

- (c) At least two modes of CPU execution (privileged and non-privileged)

23. Semaphore operations are atomic because they are implemented within the OS

24. Consider a non-negative counting semaphore S . The operation $P(S)$ decrements S , and $V(S)$ increments S . During an execution, 20 $P(S)$ operations and 12 $V(S)$ operations are issued in some order. The largest initial value of S for which at least one $P(S)$ operation will remain blocked is _____.

25. At a particular time of computation the value of a counting semaphore is 7. Then 20 P operations and 15 V operations were completed on this semaphore. The resulting value of the semaphore is:

- (a) 42 (b) 2 (c) 7 (d) 12



Swami Keshvanand Institute of Technology, Management & Gramothan,

Ramnagar, Jagatpura, Jaipur-302017, INDIA

Approved by AICTE, Ministry of HRD, Government of India

Recognized by UGC under Section 2(f) of the UGC Act, 1956

Tel. : +91-0141- 5160400 Fax: +91-0141-2759555

E-mail: info@skit.ac.in Web: www.skit.ac.in

Text and Reference Book:

1. A. Silberschatz and Peter B Galvin: Operating System Principles, Wiley India Pvt. Ltd.
2. Achyut S Godble: Operating Systems, Tata McGraw Hill
3. Tanenbaum: Modern Opearting System, Prentice Hall
4. DM Damdhere: Operating Systems- A Concepts Based Approach, Tata McGraw Hill
5. Charles Crowley: Opearting System A Design Oriented Approach, Tata McGraw Hill