NEHRU COLLEGE OF ENGINEERING AND RESEARCH CENTRE
(NAAC Accredited)
(Approved by AICTE, Affiliated to APJ Abdul Kalam Technological University, Kerala)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

COURSE MATERIALS

EC 366 REAL TIME OPERATING SYSTEMS

VISION OF THE INSTITUTION

To mould true citizens who are millennium leaders and catalysts of change through excellence in education.

MISSION OF THE INSTITUTION

NCERC is committed to transform itself into a center of excellence in Learning and Research in Engineering and Frontier Technology and to impart quality education to mould technically competent citizens with moral integrity, social commitment and ethical values.

We intend to facilitate our students to assimilate the latest technological know-how and to imbibe discipline, culture and spiritually, and to mould them in to technological giants, dedicated research scientists and intellectual leaders of the country who can spread the beams of light and happiness among the poor and the underprivileged.

ABOUT DEPARTMENT

- Established in: 2002
- Course offered : B.Tech in Electronics and Communication Engineering

M.Tech in VLSI

- Approved by AICTE New Delhi and Accredited by NAAC
- Affiliated to the University of Dr. A P J Abdul Kalam Technological University.

DEPARTMENT VISION

Producing Highly Competent, Innovative and Ethical Computer Science and Engineering Professionals to facilitate continuous technological advancement.

DEPARTMENT MISSION

1. Impart Quality education by providing excellent teaching, learning environment.
2. Transform and adopting students in this knowledgeable era, where the electronic gadgets(things) are getting obsolete in short span.
3. To initiate multi-disciplinary activities to students at earliest and apply in their respectivefields of interest later.
4. Promote leading edge Research & Development through collaboration with academia &industry.

PROGRAMME EDUCATIONAL OBJECTIVES

PEO1: To prepare students to excel in postgraduate programmes or to succeed in industry/ technical profession through global, rigorous education and prepare the students to practice and innovate recent fields in the specified program/ industry environment.

PEO2: To provide students with a solid foundation in mathematical, Scientific and engineering fundamentals required to solve engineering problems and to have strong practical knowledge required to design and test the system.

PEO3: To train students with good scientific and engineering breadth so as to comprehend, analyze, design, and create novel products and solutions for the real life problems.

PEO4: To provide student with an academic environment aware of excellence, effective communication skills, leadership, multidisciplinary approach, written ethical codes and the life-long learning needed for a successful professional career.

PROGRAM OUTCOMES (POS)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSO)

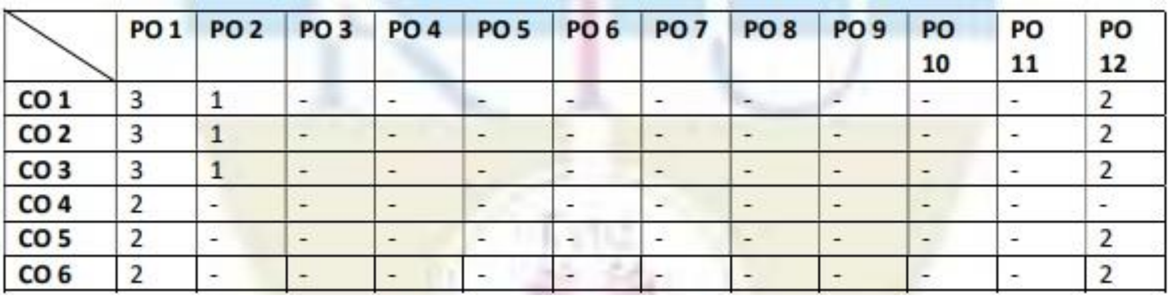
PSO1: Facility to apply the concepts of Electronics, Communications, Signal processing, VLSI,Control systems etc., in the design and implementation of engineering systems.

PSO2: Facility to solve complex Electronics and communication Engineering problems, using latesthardware and software tools, either independently or in team.

COURSE OUTCOMES

CO1	Understand the basics of operating systems tasks and basic OS architectures and develop these to RTOS
CO2	Understand the concept of task scheduling
CO3	Understand problems and issues related with multitasking
CO4	Learn strategies to interface memory and I/O with RTOS kernels
CO5	Impart skills necessary to develop software for embedded computer systems using a real time operating system
CO6	Understand the application of RTOS with a case study

MAPPING OF COURSE OUTCOMES WITH PROGRAM OUTCOMES



	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12
CO 1	3	1	-	-	-	-	-	-	-	-	-	2
CO 2	3	1	-	-	-	-	-	-	-	-	-	2
CO 3	3	1	-	-	-	-	-	-	-	-	-	2
CO 4	2	-	-	-	-	-	-	-	-	-	-	-
CO 5	2	-	-	-	-	-	-	-	-	-	-	2
CO 6	2	-	-	-	-	-	-	-	-	-	-	2

Note: H-Highly correlated=3, M-Medium correlated=2, L-Less correlated=1

SYLLABUS

COURSE CODE	COURSE NAME	L-T-P-C	YEAR OF INTRODUCTION
EC366	**Real Time Operating Systems**	**3-0-0-3**	**2016**

Prerequisite: EC206 Computer Organization

Course objectives:

- To understand the basics of operating systems tasks and basic OS architectures and develop these to RTOS
- To understand concepts of task scheduling
- To understand problems and issues related with multitasking
- To learn strategies to interface memory and I/O with RTOS kernels
- To impart skills necessary to develop software for embedded computer systems using a real-time operating system.

Syllabus:

Introduction to OS and RTOS, Process management of OS/RTOS, Process Synchronization, Memory and I/O management, Applications of RTOS

Expected outcome:

At the end of the course the students will be familiar with operating systems. They will have an in depth knowledge about the real time operating systems and its applications.

Text Books:

1. C.M. Krishna and G.Shin, Real Time Systems, McGraw-Hill International Edition, 1997.
2. Jean J Labrosse, Embedded Systems Building Blocks Complete and Ready-to-use Modules in C, CMP books, 2/e, 1999.

References:

1. Jean J Labrosse , Micro C/OS-II, The Real Time Kernel, CMP Books, 2011
2. Sam Siewert, V, Real-Time Embedded Components and Systems: With Linux and RTOS (Engineering), 2015
3. Tanenbaum, Modern Operating Systems, 3/e, Pearson Edition, 2007.
4. VxWorks: Programmer's Guide 5.4, Windriver, 1999
5. Wayne Wolf, Computers as Components: Principles of Embedded Computing System Design, 2/e, Kindle Publishers, 2005.

Course Plan

Module	Course content	Hours	End Sem. Exam Marks
I	Operating system objectives and functions, Virtual Computers, Interaction of O. S. & hardware architecture, Evolution of operating systems	2	15
	Architecture of OS (Monolithic, Microkernel, Layered, Exo-kernel and Hybrid kernel structures)	3	
	Batch, Multi programming, Multitasking, Multiuser, parallel, distributed & real –time O.S.	3	
II	Uniprocessor Scheduling: Types of scheduling	2	15
	Scheduling algorithms: FCFS, SJF, Priority, Round Robin	3	
	UNIX Multi-level feedback queue scheduling, Thread Scheduling, Multiprocessor Scheduling concept	3	
FIRST INTERNAL EXAM			

III	Concurrency: Principles of Concurrency, Mutual Exclusion H/W Support, software approaches, Semaphores and Mutex, Message Passing techniques	2	**15**
	Classical Problems of Synchronization: Readers-Writers Problem, Producer Consumer Problem, Dining Philosopher problem.	3	
	Deadlock: Principles of deadlock, Deadlock Prevention, Deadlock Avoidance, Deadlock Detection, An Integrated Deadlock Strategies.	3	
IV	Memory Management requirements, Memory partitioning: Fixed, dynamic, partitioning	3	**15**
	Memory allocation Strategies (First Fit, Best Fit, Worst Fit, Next Fit), Fragmentation, Swapping, Segmentation, Paging, Virtual Memory, Demand paging	2	
	Page Replacement Policies (FIFO, LRU, Optimal, clock), Thrashing, Working Set Model	3	
SECOND INTERNAL EXAM			
V	I/O Management and Disk Scheduling: I/O Devices, Organization of I/O functions	2	**20**
	Operating System Design issues, I/O Buffering, Disk Scheduling (FCFS, SCAN, C-SCAN, SSTF), Disk Caches	3	
VI	Comparison and study of RTOS: Vxworks and µCOS	3	**20**
	Case studies: RTOS for Control Systems.	3	
END SEMESTER EXAM			

Question Paper Pattern (End semester exam)

Maximum marks: 100 **Time: 3 hours**

The question paper shall consist of three parts. Part A covers modules I and II, Part B covers modules III and IV, and Part C covers modules V and VI. Each part has three questions uniformly covering the two modules and each question can have maximum four subdivisions. In each part, any two questions are to be answered. Mark patterns are as per the syllabus with 50 % for theory and 50% for logical/numerical problems, derivation and proof.

QUESTION BANK

MODULE 1				
Q.NO	Questions	CO	KL	Page No
1	What is the significance of a virtual computer?	CO1	K3	18
2	Compare parallel operating systems and distributed operating systems.	CO1	K2	22
3	List the functions of an operating system as a resource manager.	CO1	K6	22
4	Describe the virtual machine structure of operating system design	CO1	K6	22
5	Describe the function of an operating system as an abstract machine	CO1	K1	22
6	Explain in detail about the functions of an operating system	CO1	K5	30
7	Write a short note on serial processing	CO1	K4	30
8	Briefly explain the concept of simple batch system	CO1	K5	30
9	Give a detailed description on multi programmed batch system	CO1	K5	30
10	Write a short note on time sharing system	CO1	K1	24
11	Describe in detail about multi user operating system	CO1	K1	27
12	Briefly explain distributed operating system	CO1	K2	33
13	Compare and contrast hard real time operating system and soft real time operating system	CO1	K3	40
14	With suitable diagram explain the architecture of monolithic operating system	CO1	K1	44
15	Discuss in detail about the architecture of distributed operating system	CO1	K1	46
16	Write a short note on microkernel operating system	CO1	K1	49

17	With suitable diagram explain the architecture of Exo kernel operating system	CO1	K2	50

MODULE 2				
Q.NO	Questions	CO	KL	Page No
1	Compare FCFS and Round Robin algorithm	CO2	K2	60
2	Describe the problems associated with multiprocessor scheduling. How they can be solved?	CO2	K1	60
3	Compare SJF and Priority algorithms	CO2	K5	63
4	Differentiate preemptive and non preemptive scheduling schemes. Give examples	CO2	K1	61
5	Explain Round Robin algorithm for scheduling	CO2	K4	62
6	Describe the features of multilevel feedback queue scheduling	CO2	K2	59
7	With an example explain shortest job next algorithm	CO2	K1	65

MODULE 3				
Q.NO	Questions	CO	KL	Page No
1	Describe the principles of deadlock	CO6	K1	84
2	State and explain the Dining philosopher's problem.	CO6	K3	75
3	Illustrate a solution for Dining philosopher problem using fork function	CO6	K6	77
4	With proper code write in detail about producer consumer problem and suggest a suitable solution	CO6	K2	85
5	Discuss the different methods of preventing deadlock	CO6	K1	97
6	What is meant by critical section problem? Why is it	CO6	K1	88

	atomic in nature?			
MODULE 4				
Q.NO	Questions	CO	KL	Page No
1	Explain the concept of demand paging	CO4	K1	84
2	Consider the following page reference string: 0,2,1,6,4,0,1,0,3,1,2,1. Compute and compare the page fault rate for the following replacement algorithm, assuming frame size to be 4? Assume that the frames are initially empty. (i) FIFO replacement	CO4	K3	75
3	Consider the following page reference string: 0,2,1,6,4,0,1,0,3,1,2,1. Compute and compare the page fault rate for the following replacement algorithm, assuming frame size to be 4? Assume that the frames are initially empty (ii) Optimal replacement	CO4	K6	77
4	Explain the concept of dynamic partitioning using an example	CO4	K2	85
5	Using suitable examples, illustrate the idea behind resource allocation graph	CO4	K1	97
6	Give the structure of a page table entry used with virtual memory	CO4	K1	88
7	Give the solution of dining philosopher problem using semaphore	CO4	K5	97
8	Consider the following page reference string: 7,0,1,2,0,3,1,6,4,0,1,0,3,1,2,1. Compute and compare the page fault rate for the following replacement algorithm, assuming frame size to be 3.	CO4	K1	77

	Also assume that the frames are initially empty. (i) LRU replacement			
9	Consider the following page reference string: 7,0,1,2,0,3,1,6,4,0,1,0,3,1,2,1. Compute and compare the page fault rate for the following replacement algorithm, assuming frame size to be 3. Also assume that the frames are initially empty. (i) Optimal replacement	CO4	K3	89
MODULE 5				
Q.NO	Questions	CO	KL	Page No
1	Give a detailed description about the different I/O buffering schemes	CO5	K1	84
2	Explain the techniques for performing I/O function	CO5	K3	75
3	Write in detail about any three disk scheduling algorithm	CO5	K6	77
4	Explain the various I/O buffering schemes	CO5	K2	85
5	Write in detail about the evolution of I/O function	CO5	K1	97
6	Explain the various disk scheduling schemes	CO5		
MODULE 6				
Q.NO	Questions	CO	KL	Page No
1	Explain the inter various inter process communication techniques supported by VxWorks and MicroOS	CO6	K1	84
2	Explain how MicroC/OS 2 handles the critical section of code	CO6	K3	75
3	Using a block diagram explain how a real time	CO6	K6	77

	system is implemented. Describe a real life example of an RTOS control system			
4	Compare the characteristics of VxWorks and MicroOS	CO6	K2	85
5	Using a simple case study explain how real time system is implemented. Draw necessary diagram to depict the hardware and software implementation.	CO6	K1	97
6	Prepare suitable requirements table for an RTOS control system used in adaptive cruise control	CO6		

APPENDIX 1

CONTENT BEYOND THE SYLLABUS

S:NO;	TOPIC	PAGE NO:
1	Types of Antenna	100
2	Radiation Antenna	100

MODULE NOTES

13/1/19

MODULE 1

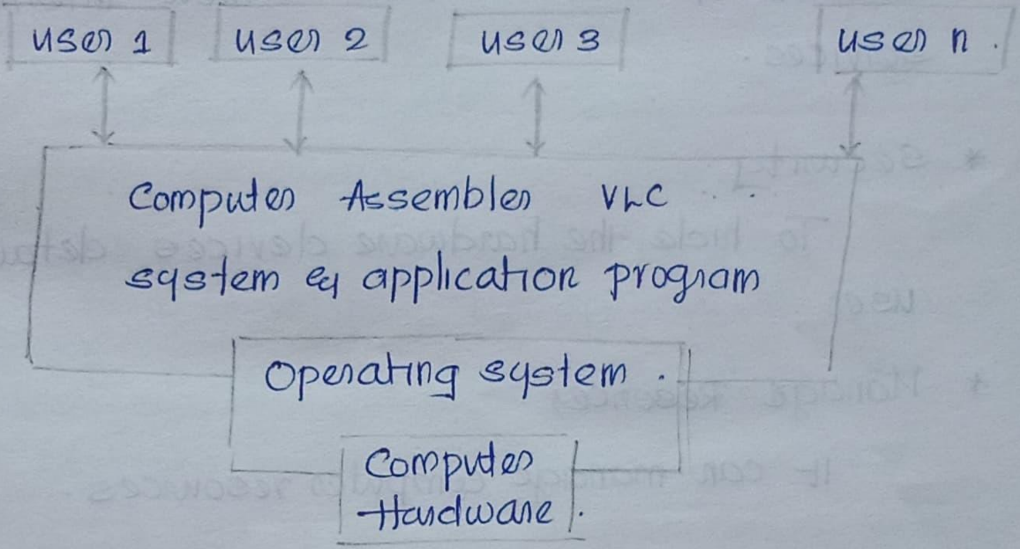
MODULE 1

Operating System.

It is a program that act as a intermediary - between users of a computer & computer hardware.

It is a program that controls the execution of application programs.

Abstract view of a System.



It act as an interface between applications & hardware.

Popular Operating systems are,

Windows, LINUX, Micro O.S ...

Main Objectives of o.s are,

Convenience
Efficiency
Ability to evolve.

* Convenience.

An o.s make a computer system more convenient to use.

* Efficiency.

An o.s allow the computer system resources (CPU, Memory, I/o devices) to be used in an eficient manner.

* Ability to evolve.

An o.s should be constructed in such a way as to permit efficient development, testing & introduction of new functions without interfering the services.

* Security.

To hide the hardware devices details from the user.

* Manage Resonces.

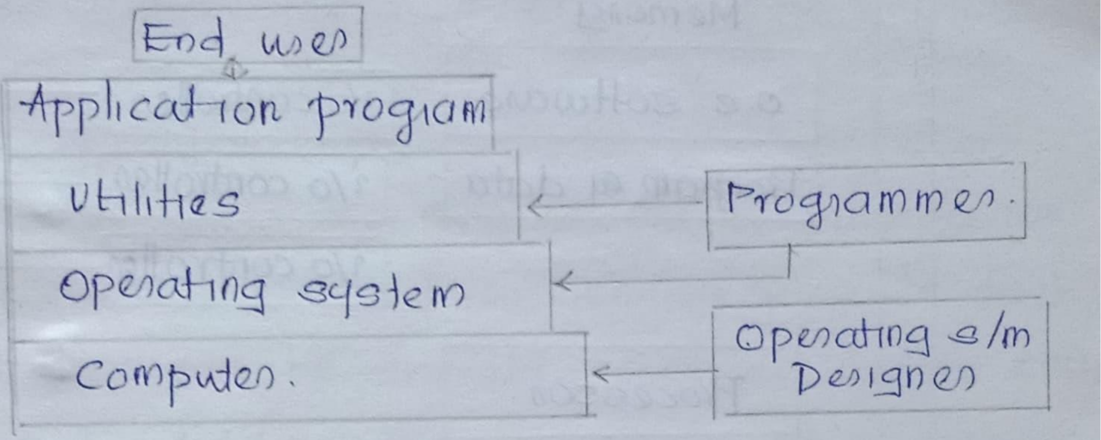
It can manage computer resources.

Services provided by o.s.

- Program development.
- Program execution.
- Access i/o devices.
- File system access.
- Error detection & responses.
- Accounting

15/01/20 Functions of O.S

1. O.S act as a user computer interface.



- End user :

View a computer system in terms of set of applications. End user may be a people or a computer.

- Application program :

It can be expressed in programming language. - It is developed by application program. It can be a compiler, assembler, file system.

- Utilities :

A set of system program which can be used to create a program, management of a file & control of i/o devices.

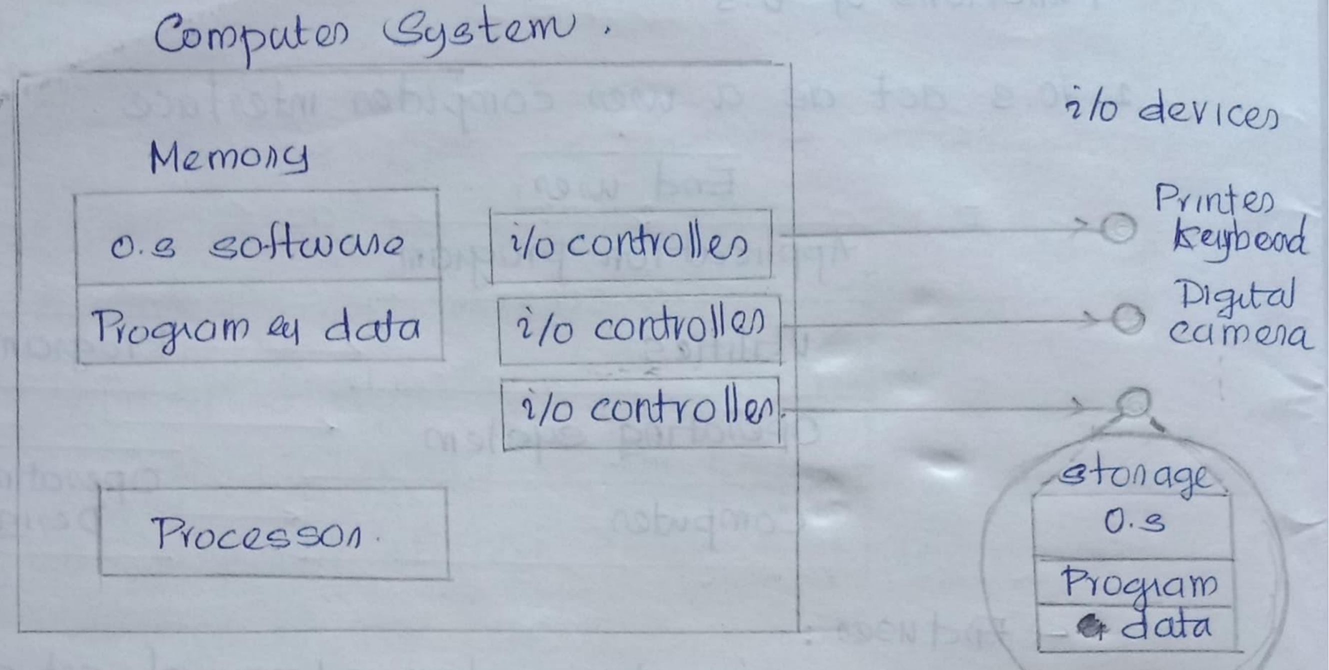
- Operating system :

It act as a mediator for the programmer & for the application program to access & use the services & facility.

- Hardware :

CPU, Memory, i/o device. which can be - access by operating system designer.

2. Resourse Manager / Resourse Allocator.



Resources like memory, cpu & all the input & output devices that are attached to a system are known as resource of an O.S.

O.S will manage all the resources of the system. A portion of O.S is in main memory which contain most frequently used functions. Other portion contain program & data

O.S decide which time CPU will perform its functions, how much processor time is to be needed for the execution of the program, which time the i/o devices ~~which~~ will respond to the request of user.

3. Storage Management.

O.S also control all the storage operations that means how the data or file will be stored in to the computer & how the files are accessed by the user.

O.s allows creation of files, creation of ~~dia~~. directories, reading & writing data of file and copy the content of file from one place to another.

4. Memory Management.

It refers to the management of primary memory or main memory. Main memory provide fast storage that can be directly accessed by the CPU. O.s keep track of primary memory that means - what part of it are in use by whom & what part not in use. It allocate the memory when a program required it to do so. It deallocate the memory when a process no long over need it.

5. Device Management.

O.s manage the device communication - through their respective drives. It also keep track of all the input & output devices. O.s decide - which process get the devices when & how much time. It allocate the devices to a process and deallocate the device when not in use.

6. Processor Management.

In multiprogramming, O.s decide which - program get the CPU, when & how much time. This function is called process scheduling. O.s keep track of processor & status of the program. It allocate CPU to a process & also deallocate when the process is no longer needed.

7. Security.

By means of passwond or similar other techniques o.s protect unauthorized access to programs & data.

8. Job Accounting.

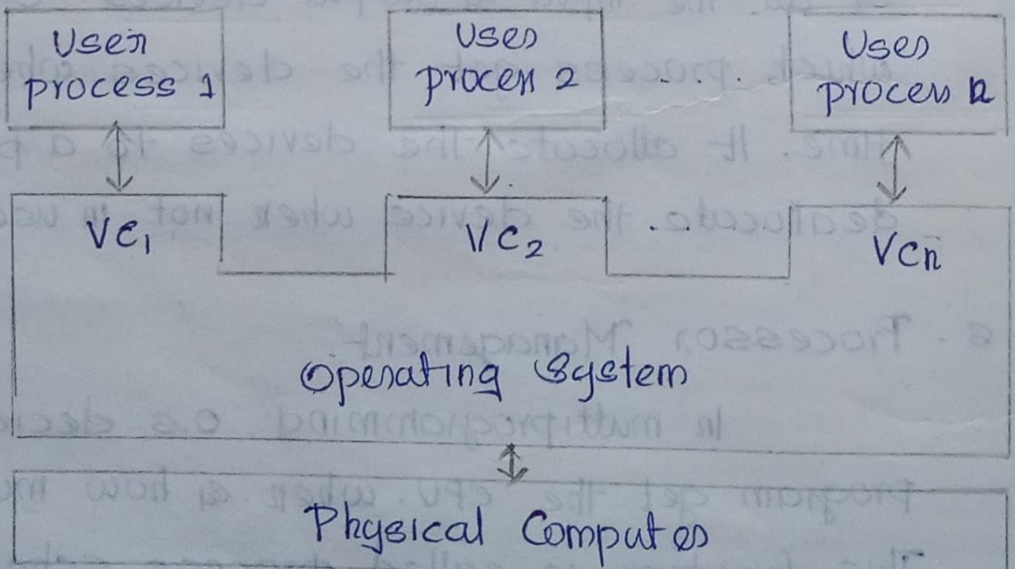
O.s keep track of time & resources used by various jobs & user.

9. Error Detecting Aids.

Virtual Computer.

It has 4 basic parts.

* Processor
* 1° memory
* 2° memory
* I/O devices.



O.s create a virtual computer from a physical - computer. The main ~~com~~ difference is that there are many virtual computer while there is only one physical computer O.s allow software copies of the processes and the memory. O.s implement file system & I/O system. It also ~~allow~~ create multiple address space for the memory.

* Virtual Processor.

It have same inteface to the user as the physical processor. It uses same set of instruction as physical processor. These instructions are called system call.

* Virtual 1° memory.

The memory of virtual computer is similar to that of hardware memory. The only difference is that O.s will divide the physical memory in to parts & give ~~each part~~ a part to each virtual computer

* Virtual 2° memory

The 2° storage provide long term storage of data.

* Virtual I/O

The I/O operation offer virtual computer entirly different from physical computer. The physical computer has devices with complex control & status devices. But in virtual I/O is simple & easy to use.

16/11/20

Evolution of Operating System.

stages include,

Serial processing
Simple batch systems
Multiprogrammed batch systems
Time sharing system.

1. Serial Processing.

~~No os~~ From the late 1940s 1950, the programmer interacted with the computer hardware directly, and they don't have an o.s. Thus computer runs from a console with display lights; toggle switches, input device & printer. Program in machine code were loaded through input devices like card reader. If an error occurs in the program the error ~~indication~~ condition was indicated by light. If a program is proceeded to a normal condition, output appears on the printer.

Disadvantages :

* scheduling.

If a user may sign up for 1 hour but finishes his job in 45 minutes, this may result in wasting computer ideal time.

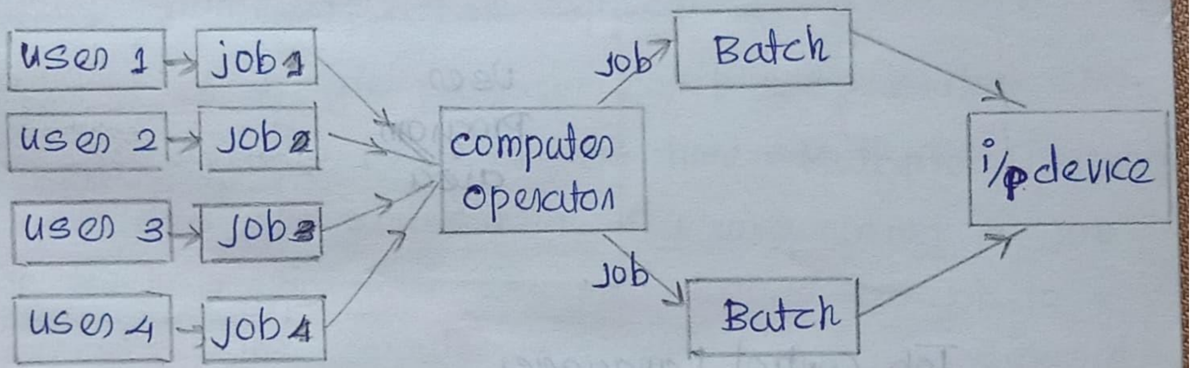
If the user can't be able to finish the job in allotted time, he is forced to stop that job before resolving the problem.

* setup time

A single program involve loading compiler & source program in a machine & save the compiled machine & compiled program. If an error has occu user has to go. to the beginning wasted the considerable amount of time for setting up a new program to run.

2. Simple Batch System.

The main aim of simple batch system is to improve the utilization of the processor.



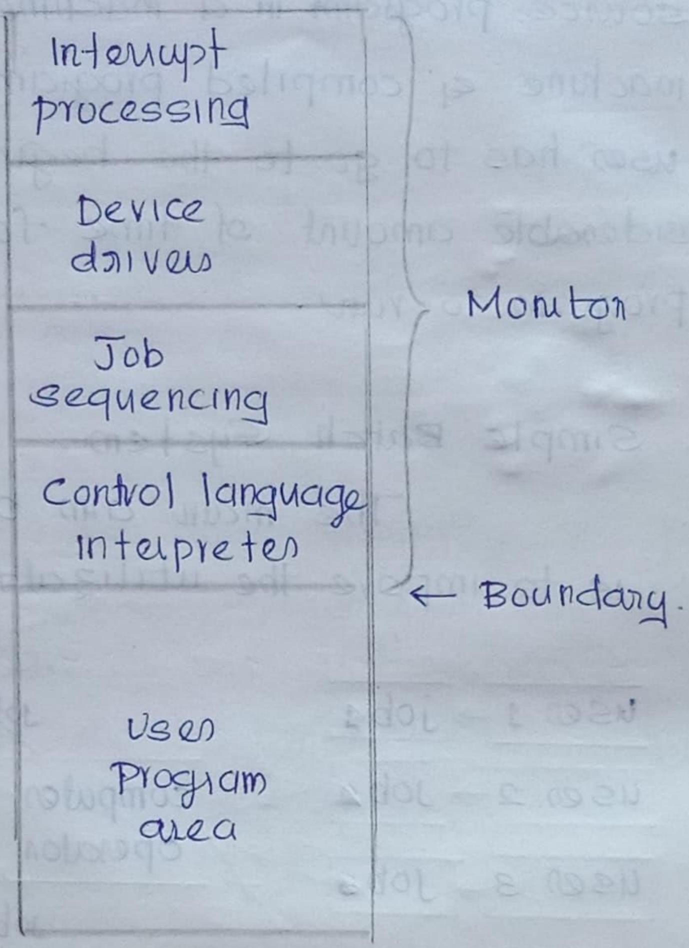
17/01/20

Batch is defined as a group of job - with similar need. In this o.s the user has no longer access to the processor instead the user submit the job on cards to a computer operator who batches the job sequencially & place the entire batch on the input device.

Monitor is the software that controls the sequence of events. It batches the job together & the program ~~#~~ returns the control to the - monitor when the task is finished.

'Resident Monitor' is the software always in memory

Memory Layout for a Resident Monitor.



Job Control Language.

It is a special type of programming language - to control job. It provide instruction to the monitor like what compiler to, what data to use . . .

There are two modes of operation in simple batch system.

1) User Mode

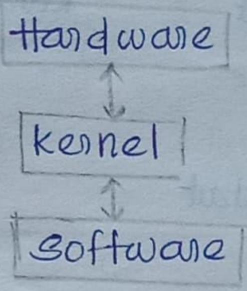
In user mode, user executes program
user program executes in user mode.

certain areas of memory protected from user access

certain instructions may not be executed.

2) Kernel Mode

Kernel :



It is the interface between hardware & software.

Monitor executes in Kernal mode.

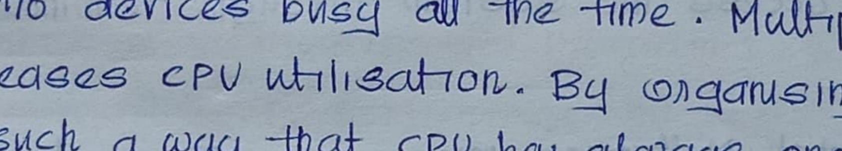
Privileged instructions may be executed, all memory accessible.

21/01/20

3. Multiprogrammed Batch System.

A single program can't keep either CPU or i/o devices busy all the time. Multiprogramming increases CPU utilisation. By organising a job in such a way that CPU has always one job to - execute. The computer is requred to run several program at the same time. ie, CPU could be always busy for most of the time by switching one program to another.

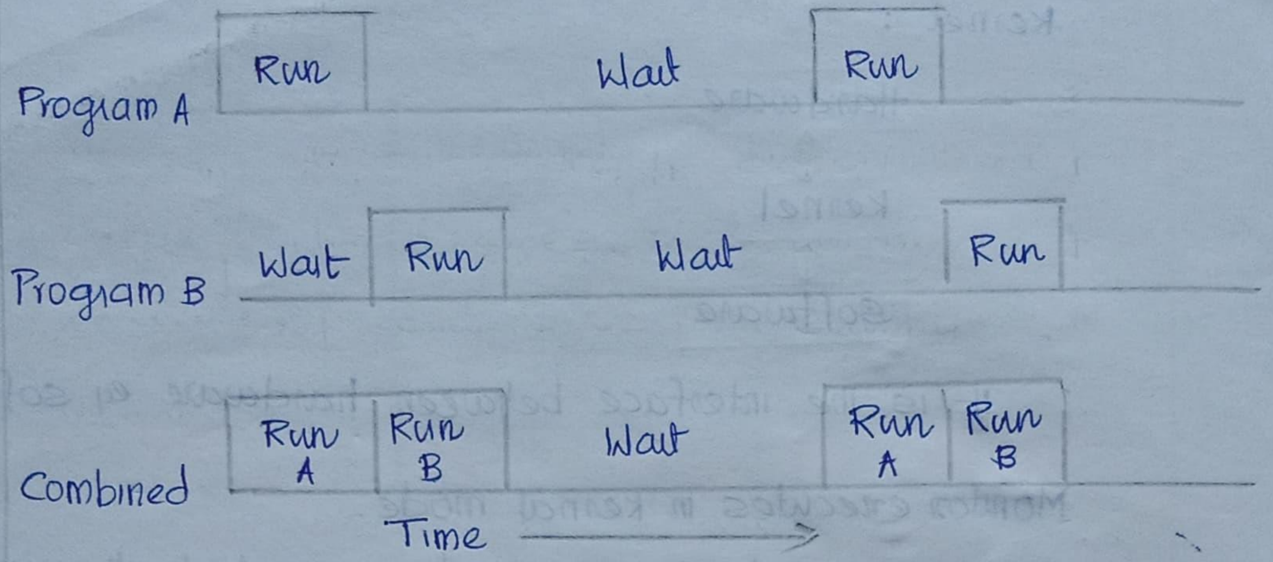
→ Uniprogramming.



Processor must wait for I/o instruction to complete before preceding.

→ Multiprogramming.

When one job needs to wait for I/O, the processor can switch to the other job.



4. Time sharing system

In time sharing system, processor time shared among multiple users. Multiple users simultaneously - access the system through terminals.

Batch Multiprogramming vs Time sharing.

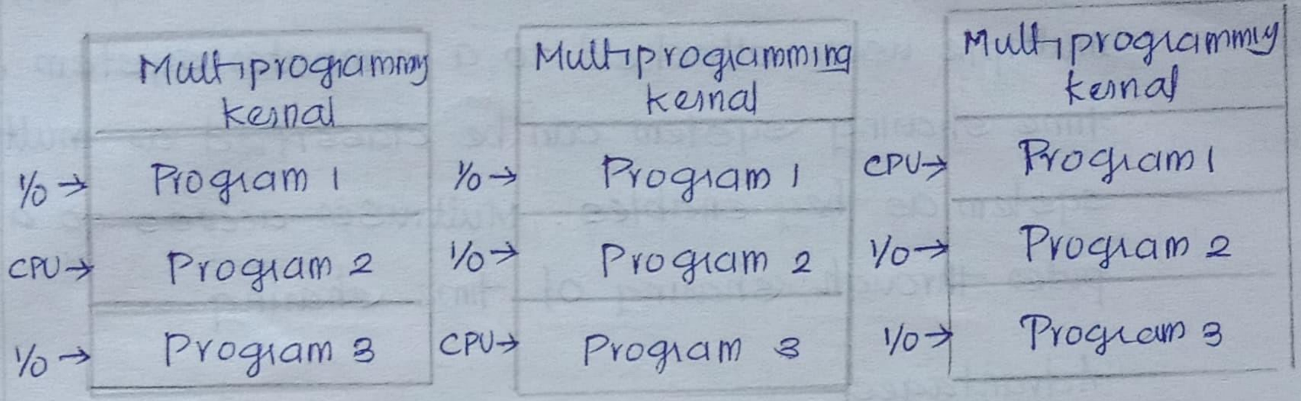
	Batch Multiprogramming	Time sharing.
Principle Objective	Maximize processor use.	Minimize response - time.
Source of directives to operating system.	Job control language commands provided with the job.	Commands entered at the terminal.

Types of Operating System.

* Batch O.S.

same fig & explanation of simple batch O.S

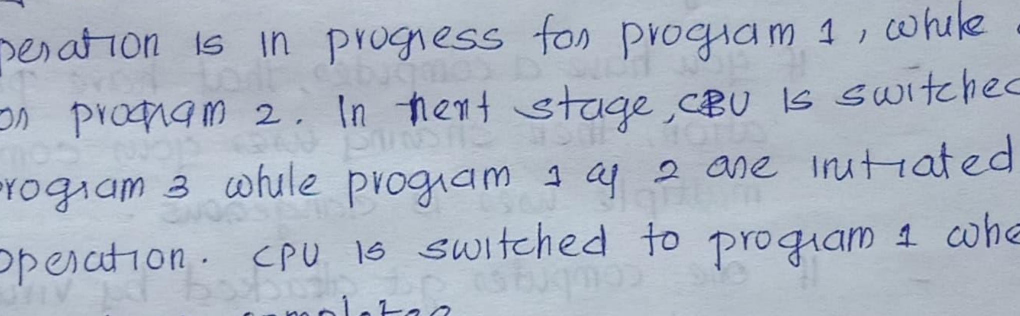
* Multiprogrammed O.S.



The most important aspect of job scheduling is the ability to multiprogramming. Let CPU execute instruction for one program while I/O sub system is busy with an I/O operation for another program. This technique is called multiprogramming. The above fig shows the memory contain 3 program. An I/O operation is in progress for program 1, while CPU is for program 2. In next stage, CPU is switched to program 3 while program 1 & 2 are initiated by I/O operation. CPU is switched to program 1 when I/O operation completes.

Multiprogramming is the first instant ~~for~~ where the O.S make the desicions for the user.

* Multi user Operating system.



Multiuser OS share processor time

A	B	C	A	B	A	B	C

Time slices.

Multiuser OS is that which handles & control – multiple users attached to a computer system. The time sharing system can be classified as multiuser system as they enables. Multiuser access to a computer through sharing of time sharing.

Advantages :

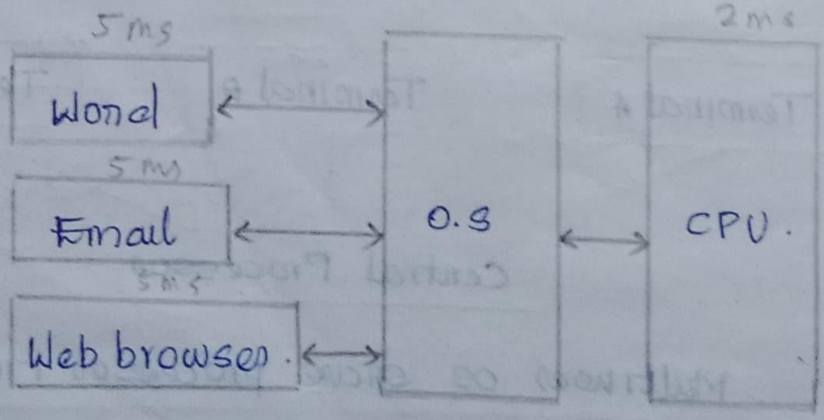
- Airline ticket reservation done thus type of O.S
- Printing jobs in the office or library, can be best handled by multiuser OS.
- If one computer, in a network get error, the other computer doesn't get effected.

Disadvantages :

- If you have a computer, that have private information, then sharing ~~user~~ your computer with multiple user is dangerous.
- If one computer get attacked by virus, sometimes others get affected

Examples : Linux, UNIX, Windows

22/01/20 * Multitasking Operating System (Time shared O.S)



In multitasking o.s, each process or each task - execute for a fixed amount of time. After that fixed time, CPU switches to another task. The fixed period for each task is called time quantum.

Here there is only one CPU, but switches between different processes, so quickly. So that it give an - illution that all processes at same time.

Time shared o.s or multitasking o.s is a logical extention of multiprogramming. In multitasting o.s - more than one user can interact with the system same time. CPU shared the time to different processor. So that the system is called time ~~tim~~ sharing o.s. Time sharing o.s is called use job scheduling, memory - management.

CPU scheduler select a job from the queue and switches CPU to that job. When the time slot is - experied, CPU switches to another job. Time slices is given by the o.s for sharing CPU time between - processes.

Eg: UNIX.

Advantages:

- Better response time
- Better CPU utilisation
- Execute multiple task together

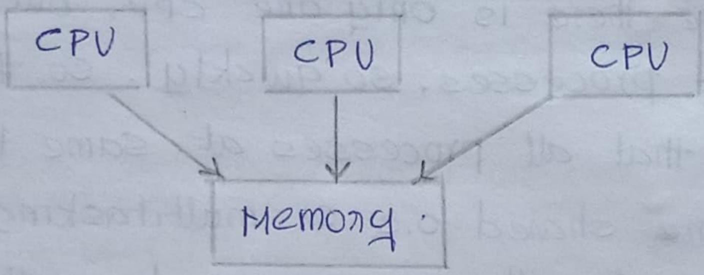
Disadvantages:

- Disk management is required.
- O.s must have memory management & protection. This is bcz several jobs are kept in memory at

same time.

- As more than one user interacting with processor at same time, the task become complex.

* Multiprocessor O.s (Parallel O.s)



Multiprocessor have more than one processor. This O.s also known as parallel O.s or tightly - coupled O.s, Because of no. of processor are - executing jobs in parallel. The processor share - computer bus, clock, peripheral devices..etc. This O.s control hardware & software resources such that user can view. The most common multiprocessor O.s uses symmetric multiprocessing (SMP). In SMP, one O.s control all the CPUs. And each CPU has equal rights. In SMP, each processor run on identical - copy of O.s & this copy communicate with each other.

Asymmetric Multiprocessing (ASMP) :

In which each processor assign a specified task. It defines, master slave relationship. Master processor schedule & allocate task ~~for~~ to slave processor.

Advantages :

- Increase throughput.
- Increase reliability

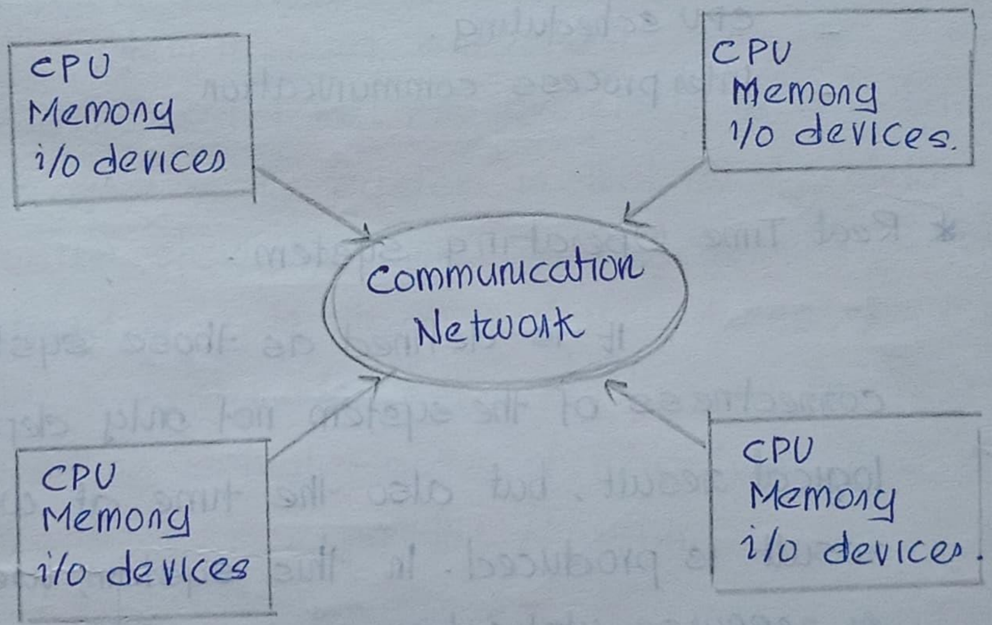
- Computation speedup
- Cost saving
- Efficient battery life.

Disadvantages :

- More complex
- It require large main memory.
- Security & protection
- Memory management

* Distributed Operating system.

Eg : Ameoba .



The distributed o.s are the O.s for a - network of autonomous computer connected by a communication network. That follow message - passing. In this system the processor can't share the memory or clock. each processor has its own local memory. The processor communicat~~es~~ each other through $comm^n$ lines known as high speed bus.

Distributed O.S control & manages the hardware & software resources of an operating system.

Advantages :

- Resource sharing
- Reliability
- Computation speed up
- Communication

Disadvantages :

- Process synchronisation.
- Dead loack
- Memory management
- CPU scheduling.
- Inter process communication

* Real Time Operating System.

It is defined as those system in which correctness of the system not only depend on the logical result, but also the time at which the result is produced. In this system, user convenience & resource utilisation are secondary concerned.

Applications :

Rocket launching
Flight control
Fire & smoke sensors
Robotics
Telephone switching equipment.

In real time system many events that must be -

accepted & processed in a short time on within - certain deadline. In the case of sensors, they bring ~~that~~ data to the computer & computer can analyze the data & adjust the control to modify sensor - input.

23/01/20

Real time operating system are classified into,

1. Hard Real Time O.S
2. Soft Real Time O.S

Hard Real Time O.S

In this O.S, O.S gaurntee that crytical task to be computed on time.

A system is said to be hard real time O.S, if the deadline is not met, the system is said to have failed. The goal of the system is that all the delay in the system should be time counted, from the retreval of stored data to the time that it takes, the O.S finish any request made off it. Penalty due to the missing deadline is higher order of magnitude than the - reward in meeting the deadline.

Eg: Rocket launching.

Soft Real Time O.S

A system is said to be soft O.S, if the deadline is missed, then the system doesn't fail. In this - O.S, each task get priority over the other task. Retail in that priority level until it completes. In this system penalty has lesser magnitude than reward. eg: railwaq ticket reservation, vedio on demand

Archutecture of Operating System.

Monolitic O.S

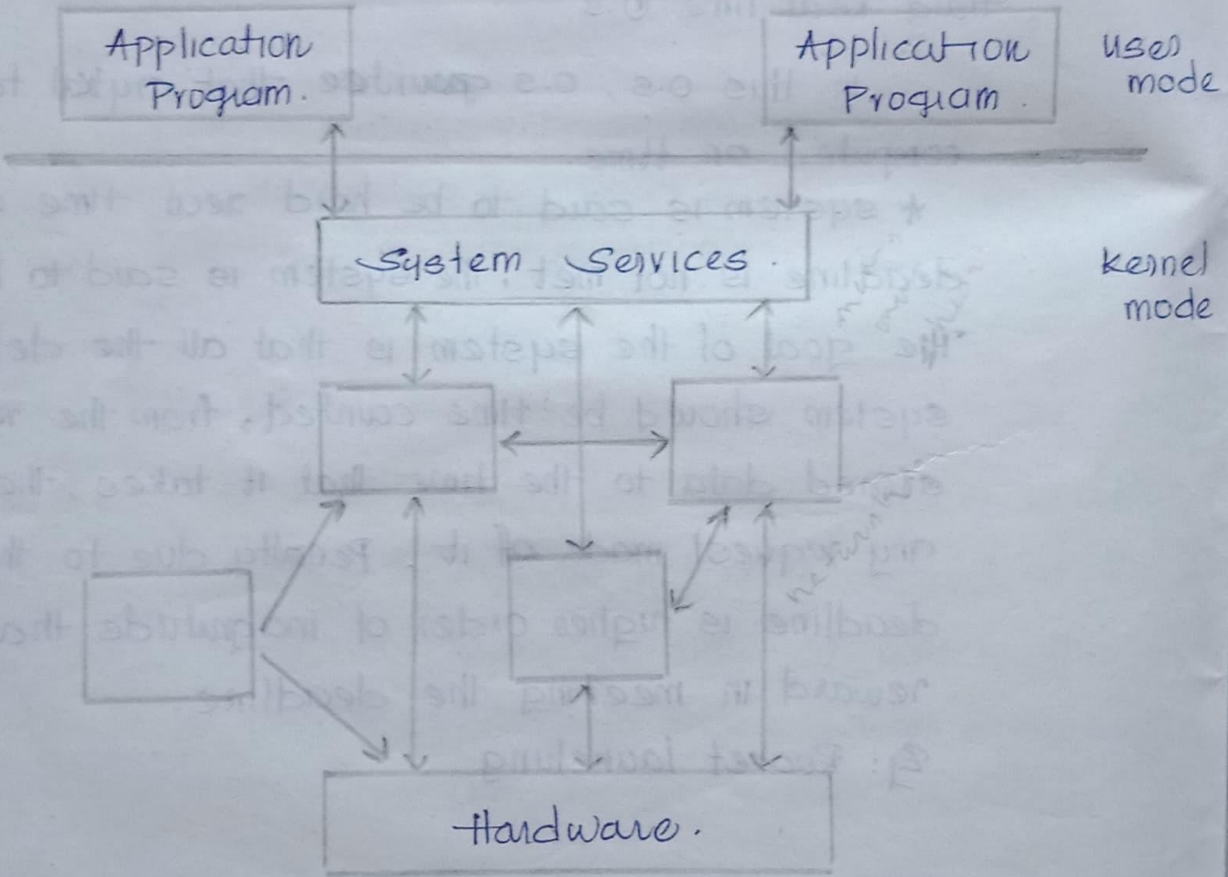
Layered O.S

Microkernel O.S

Exokernal O.S

Hybrid O.S.

→ Monolitic Operating System.



It consist of two modes ; user mode & kernel mode.

In user mode, application programs like airline ticket reservation, web browsing. . .

In kernal mode, system services like memory - management, schedules, process management, inter processor . . .

Hardware section consist of CPU, memory, I/O devices...

In between system services & hardware different networks are connected.

Working :

Operating system run in kernel mode with access to the system data & hardware. Application program run in user mode with a limited set of interfaces & limited access to the system data. When the user system call a system services, the processor trap the call & switches to kernel mode. When the services get completed the processor switches from kernel mode to user mode allow the caller to continue.

Advantages :

× Better application performance.

Disadvantages :

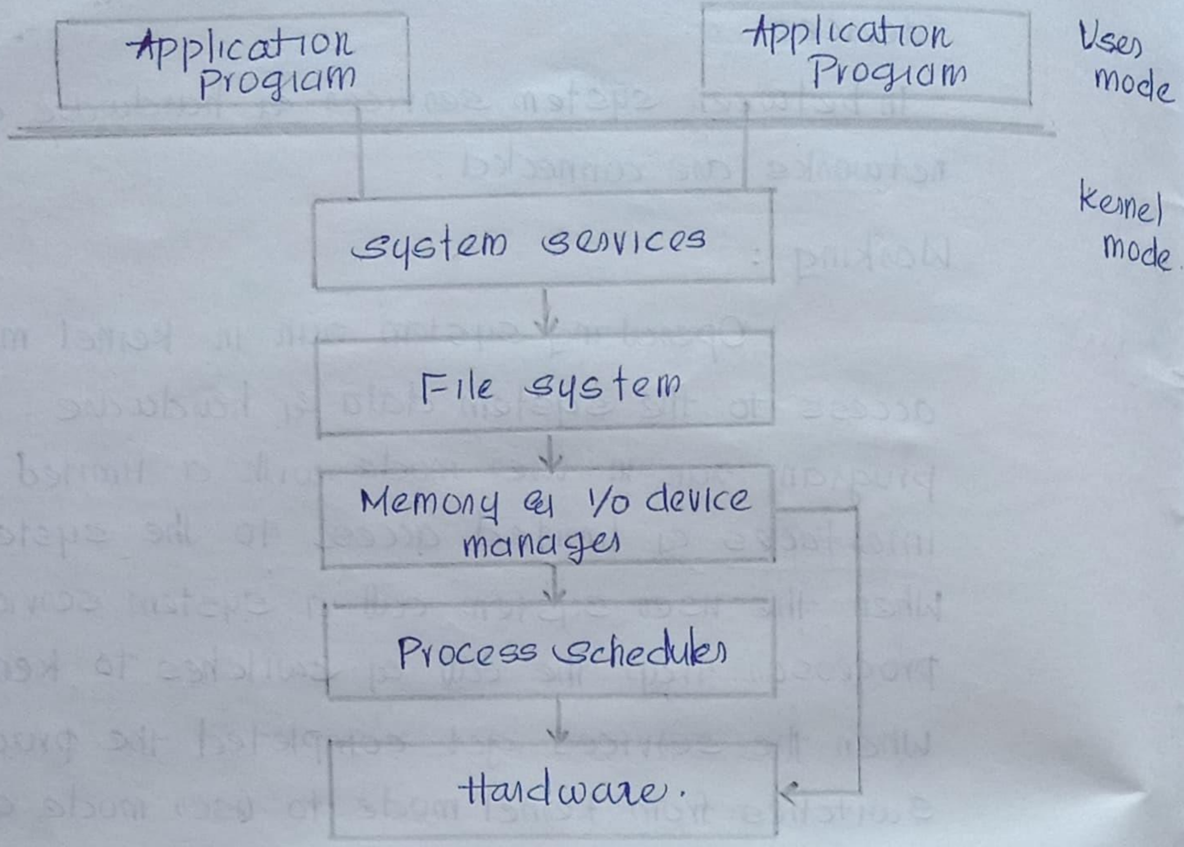
× Dificulty to entend.

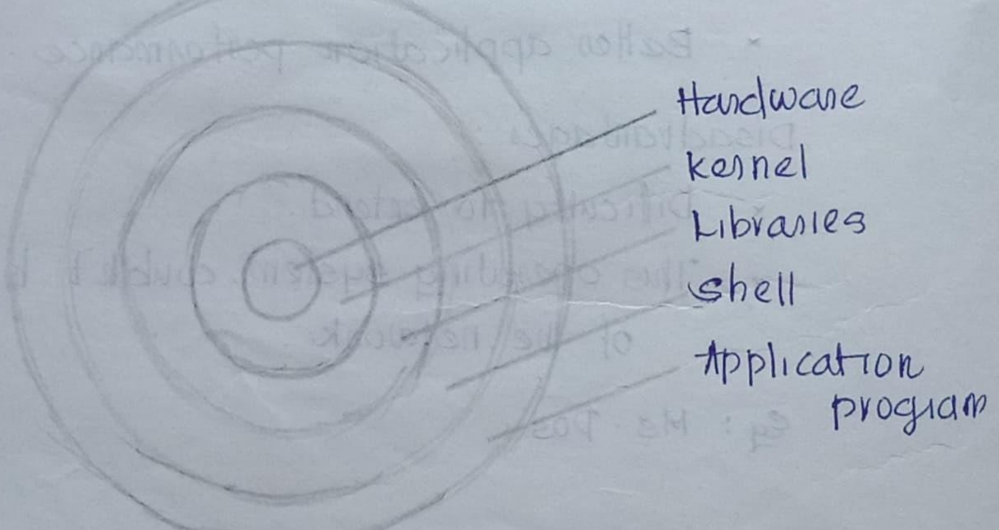
× Thus operating system, couldn't hide the details of the network.

Eg : Ms. Dos.

⇒ Layered Operating system.

The components of layered o.s are organised into modules & layered them one on the top of other. Each module provide a set of functions that the other module can call.

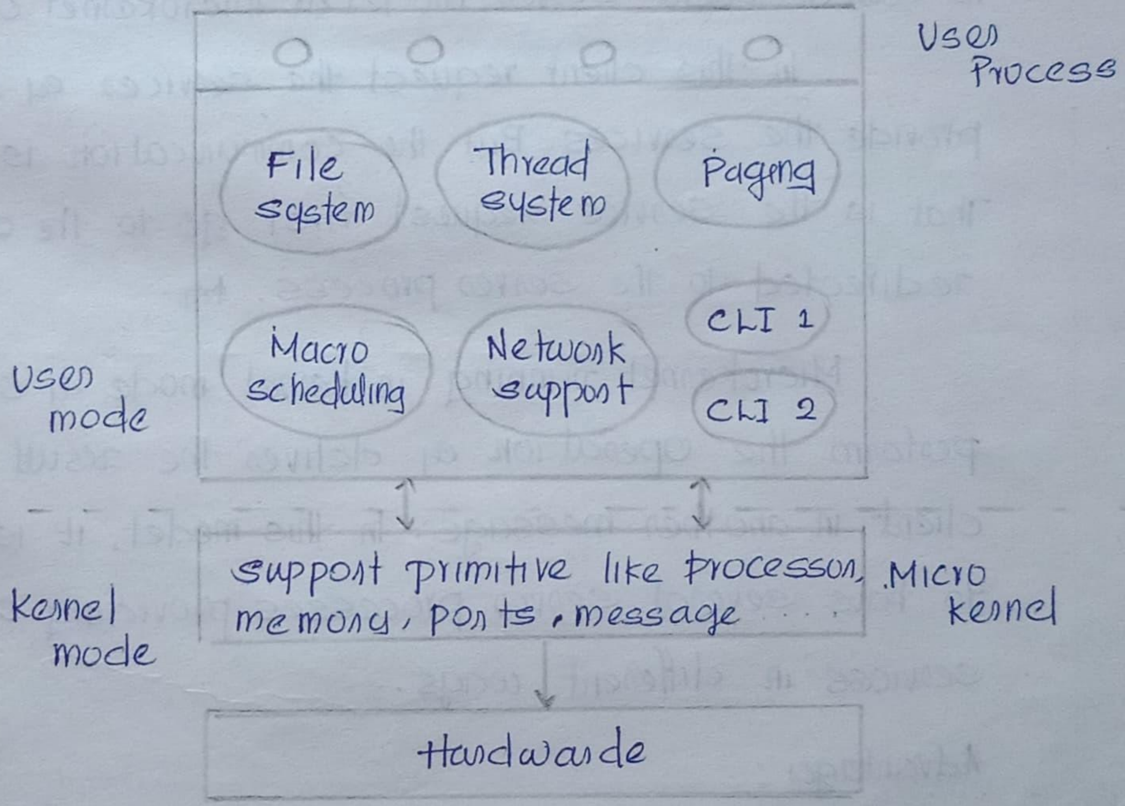




Interface function at any particular level can provide services by lower layers. The advantage of layered o.s is that each layer is given access to only the lower level interfaces. In this approach - nth layer, (eg:file system) can access services provided by (n-1)th layer (system services) & provide services memory & I/O de to (n+1)th layer. In this approach, it can enhance the operating system, ie; one entire layer can be replaced without affecting the other parts of the system. Compared to monolitic o.s, layered o.s delivers low application performance.

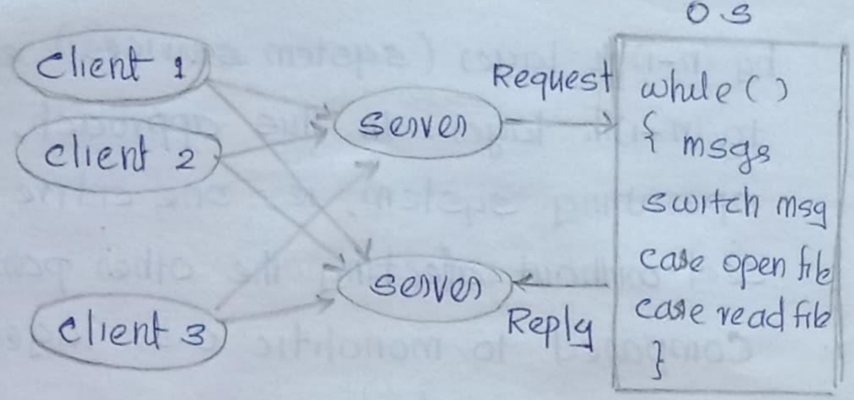
Eg: UNIX.

→ Microkernel Operating System.



24/01/20

In kernal mode, basic processes, memory - management, message passing between services - are included. This mode, also provide security and protection. But most services like file system, thread system are performed in user mode. Micro kernel os also known as client server model.



The above fig shows client-server communication. The system processes that do much of the work of the kernel is called server. And this type of system structure is called client-server model or microkernel O.S.

In this client request the services & server provide the services. But the communication is indirect. That is the service request first go to the O.S, then redirected to the server process. ~~M~~

Microkernel running in kernel mode & server perform the operation & deliver the result to the client in another message. In this model, it is possible to have several server processes providing similar services in different ways.

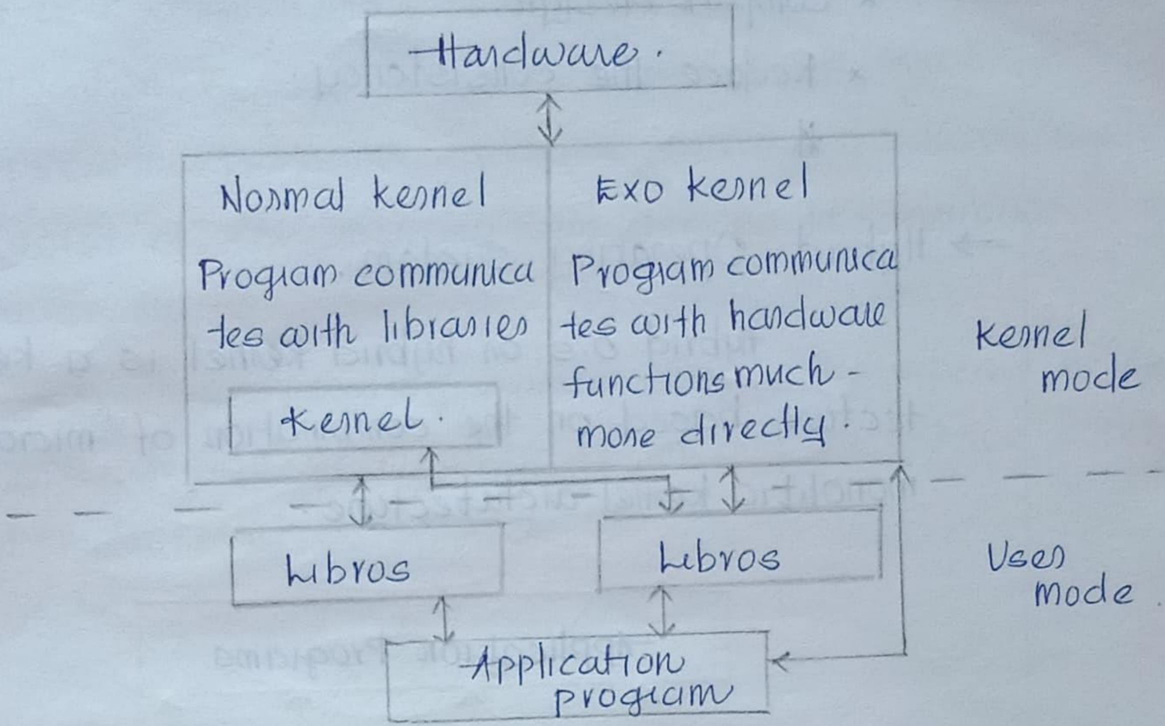
Advantages :

- This O.s can be used with a networked or distributed environment.
- Modularity : If there is any error, O.s determine & correct it.

Disadvantages :

- Speed : This model has low speed to sent message to another processes.

⟹ Exokernel Operating System.


Hardware.
Normal kernel
Exo kernel
Program communicates with libraries
Program communicates with hardware functions much more directly.
kernel.
Kernel mode
Libros
Libros
User mode.
Application program

Exo kernel O.s is developed by Massachusets institute of technology used to provide application level management of hardware resources. Thus O.s is typically small in size because of their limited operation.

Libros are library O.s. It works on the bottom of Exo kernel interface. There are two modes; user mode & kernel mode. Libros are in user mode. Thus O.s - perform 3 tasks.

- It track the ownership of the resources. ~~It~~
- It ensure protection by guarding all resources.
- Revoke access to the resources

Advantages :

x Improved performance on applications.

x More efficient use of hardware resources, through process resource allocation.

x Easier development & testing of new O.s.

Disadvantages :

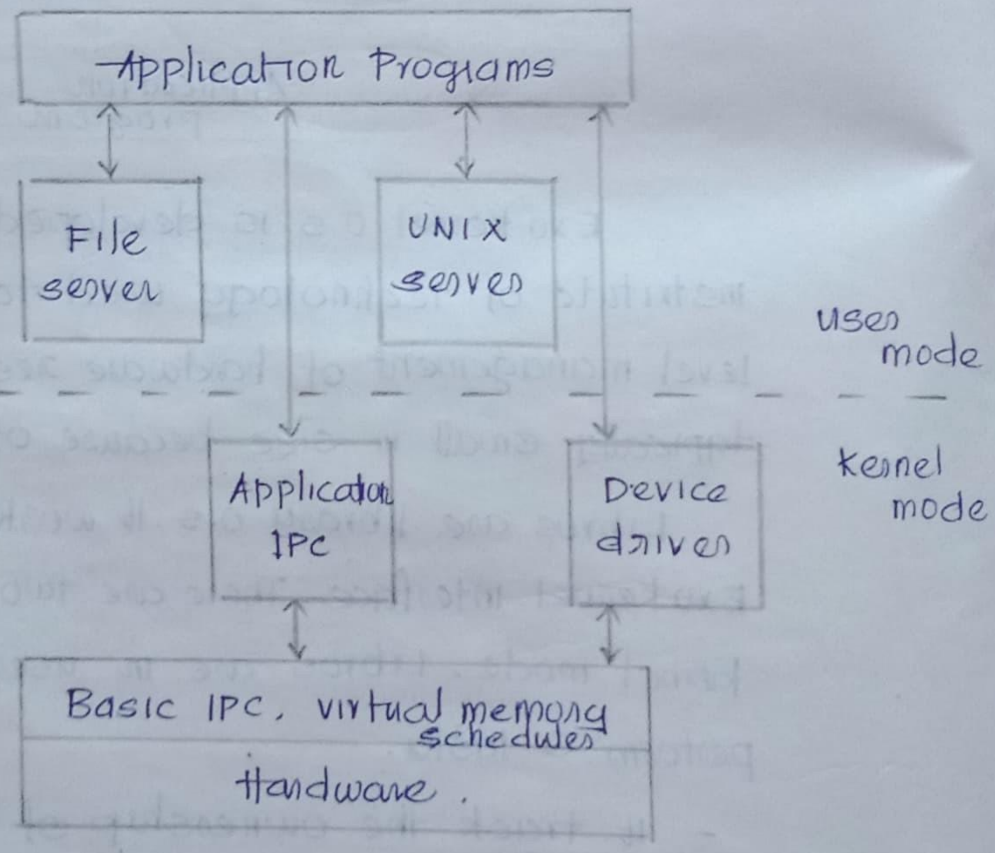
× Complex design

× Reduce the consistency.

→ Hybrid Operating System.

28/01/20

Hybrid o.s or hybrid kernel is a kernel architecture based on the combination of microkernel & monolitic kernel architecture.



IPC - Inter Process Communⁿ.

eg: Windows 2000, windows vista, windows XP.

Hybrid kernel consist of two modes; user mode & kernel mode. In user mode, application programs like banking, airline ticket reservation, web browser ..etc. It consist of two servers; File server & UNIX server. In kernel - mode, consist of system services like basic IPC,

Shortest Job First / Shortest Job Next.

· Out of all available (waiting) process, it selects the process with the smallest burst time to execute next.

. Two types

- . Pre emptive. Also known as Shortest Remaining Time First (SRTF)
- . Non Preemptive.

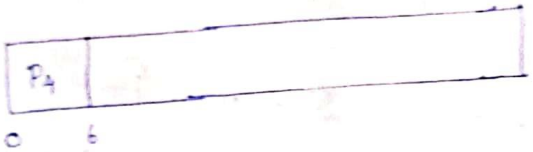
Non-preemptive Scheduling: A scheduling discipline is non-preemptive if once a process has been used the CPU, the CPU cannot be taken away from that process.

Pre emptive Scheduling: A scheduling discipline is preemptive if once a process has been used in the CPU, the CPU can be taken away.

Consider an example.

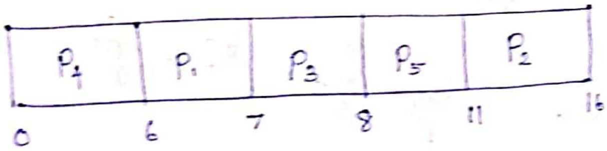
	Arrival time	Burst time
P_1	2	1
P_2	1	5
P_3	4	1
P_4	0	6
P_5	2	3

step 1: Prepare Granth chart.



when Process 4 was completed, P1, P2, P3 and P5 are in ready queue.

Granth chart:



After P4, P1 was selected. P1 and P3 have same Burst time which is the smallest amont the processes in the ready queue. Arrival time of $P_1 = 2$ & $P_3 = 4$. So as per the first come first serve basis P1 is selected after P4.

After P1, P3 is selected b'coz it has the smallest burst time among the Processes in the ready queue. The processes in the ready queue are P3, P5, P2.

After the selection of P3 by CPU, the processes in the ready queue are P5 and P2.

Among P5 and P2, P5 has got smallest Burst time. So P5 was selected and later on P2.

As per the Ganth chart, CPU was busy with the process till t = 16. Since CPU was not in an idle state Burst time = 1+5+1+6+3 = 16 will be equal to Ganth chart completion time is

t = 16.

	Arrival time	Burst time (BT)	CT	TAT	WT	RT
P_1	2	1	7	5	4	4
P_2	1	5	16	15	10	10
P_3	4	1	8	4	3	3
P_4	0	6	6	6	0	0
P_5	2	3	11	9	6	6

CT = completion time

TAT = Turn around time

= CT - Arrival time or

= WT + BT

WT = waiting time

WT = TAT - (BT)

RT = Response time

= The time at which CPU has been allocated to the process first time after the arrival of that process.

$\therefore$ RT for $P_1 = 6 - 2 = 4$

" $P_2 = 11 - 1 = 10$

" $P_3 = 7 - 4 = 3$

" $P_4 = 0 - 0 = 0$

" $P_5 = 8 - 2 = 6$

RT of P2:

at t = 1, P_2 has come to READY state, but CPU has been allocated to the process at t = 11 $\therefore$ RT of $P_2 = 11 - 1 = 10$.

If the algorithm is non preemptive WT = RT.

$\therefore$ Avg. TAT = $\frac{39}{5} = 7.8$

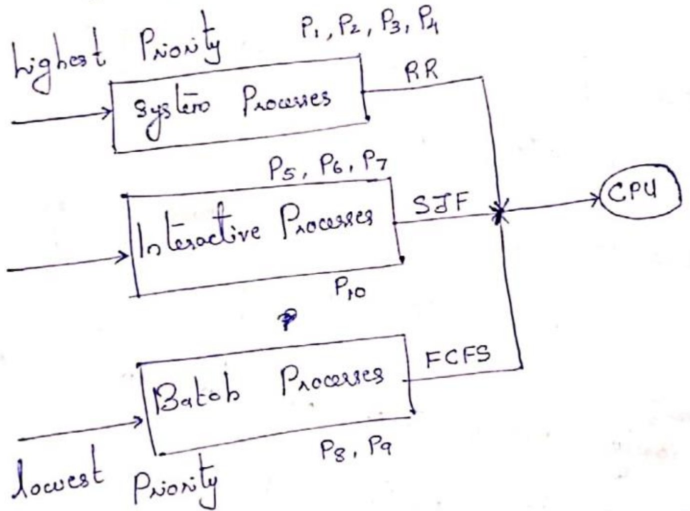
Avg. WT = $\frac{23}{5} = 4.6$

Multilevel Queue & Multilevel Feedback Queue Scheduling

Different types of processes are

1. System Processes: Processes used to run system pgms Highest priority.
2. Interactive Processes: Continuously working with an application eg: MS Word.
3. Batch Processes: Processes which run in the background.

eg:



- When a process comes, that process is permanently assigned to a particular queue. ie till the termination.
- Each processes has its own scheduling algorithm.
- After the completion of System processes, then only interactive processes can execute.
- Batch Processes can execute, only after the system queue & interactive queue are empty.
- While executing a batch process, assume a new interactive process arrived. Then CPU allocation will be given to the new interactive process.
- Sometimes process in Batch queue has to wait for indefinite amount of time. This is known as Starvation.

ie multilevel queue algorithm suffers from starvation problem.

. To overcome starvation, promote ageing.

. Ageing cannot be done in Multilevel Queue algorithm. B'coz processes cannot migrate to other queues.

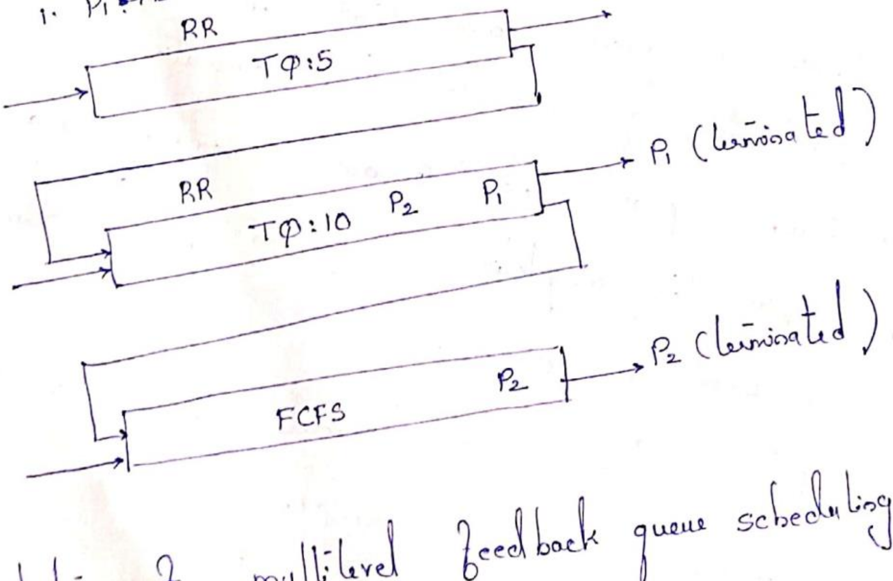
. So to avoid starvation in Multilevel Queue, we can use Multilevel feedback Queue Scheduling.

. Here processes from a lower priority queue can be promoted to a higher priority queue.

. Similarly processes from a higher priority queue can be demoted to a lower priority queue.

eg: Higher priority process demoted to lower priority queue.

1. P1 → BT=15   2. P_2 = 20(BT)



. Implementation of multilevel feedback queue scheduling is very tough.

Priority Scheduling:

- Each process has its own priority.
- Out of all available processes, highest priority process gets the CPU.
- If tie, then use CPU.
- Priority → Static (doesn't change throughout the execution of process)
  → Dynamic (changes after some interval of time)

- Version → Non Pre emptive
  → Preemptive.

Non Pre emptive Scheduling:

	Priority	AT	BT
P_1	3	0	8
P_2	4	1	2
P_3	4	3	4
P_4	5	4	1
P_5	2	5	6
P_6	6	6	5
P_7	1	10	1

- Lesser the number, higher the priority. (for this example only)

Gantt chart

P1	P5	P7	P2	P3	P4	P6

0 8 14 15 17 21 22 27

at t=8 : at t=8, P1 was terminated.

P2, P3, P4, P5 and P6 are in Ready queue.

P5 has highest priority. at t=14, P5 has terminated

at = 14 :

P2, P3, P4, P6 and P7 are in Ready queue

at t=10, all the processes are arrived in Ready queue.

P7 has highest priority. at t=15, P7 was terminated.

P2, P3, P4 & P6 are available in Ready queue.

P2 & P3 have same priority. AT of P2=1 & P3=3. ∴ P2 got CPU allocation. Later P3.

at t= 17, P2 was terminated. At t=21, P3 was terminated.

P4 and P6 are available in Ready queue.

at t= 22, P4 was terminated

at t= 27, P6 was terminated. So, at t=27 all processes were terminated.

Σ BT = 27, which is = 27 of Gantt. Since it is a non preemptive scheduling.

Scanned with CamScanner

	Priority	AT	BT	CT	TAT	WT	RT
P_1	3	0	8	8	8	0	0
P_2	4	1	2	17	16	14	14
P_3	4	3	4	21	18	14	14
P_4	5	4	1	22	18	17	~~23~~ 17
P_5	2	5	6	14	9	3	3
P_6	6	6	5	27	21	16	16
P_7	1	10	1	15	5	4	4

TAT = CT − AT , WT = TAT − BT

In non, pre emptive , WT = RT.

Avg. TAT = $\frac{95}{7}$ = 13.7

Avg. WT = $\frac{68}{7}$ = 9.7

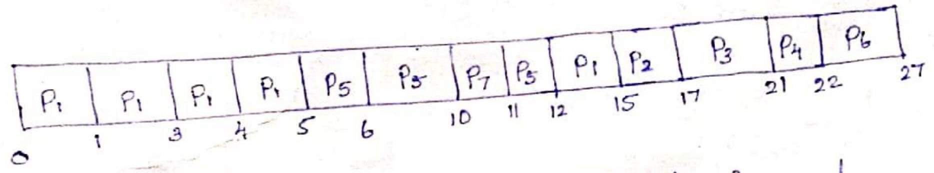
Avg. RT = $\frac{68}{7}$ = 9.7

Priority Scheduling (Pre emptive)

Lesser the number - higher the priority.

	AT	Priority	BT
P_1	0	3	8
P_2	1	4	2
P_3	3	4	4
P_4	4	5	1
P_5	5	2	6
P_6	6	6	5
P_7	10	1	1

Gantt chart



t=0 : P1 has CPU.

Ready queue = P1 . Run P1 till the arrival of next process with higher priority in Ready queue

at t=1

Ready queue = P1, P2 . P1 has CPU access. B'coz priority of P1 > new arrived P2.

at t=3

Ready queue = P1, P2, P3 . P1 has CPU access. B'coz priority of P1 > newly arrived P3.

at t=4

Ready queue = P1, P2, P3, P4. P1 has CPU access. B'coz priority of P1 > newly arrived P4

at t=5

Ready queue = P1, P2, P3, P4, P5. P5 has CPU access. B'coz priority of P5 > currently running P1. ie Context switching happens.

BT of P1 = 8-5 = 3.

at t=6

Ready queue = P1, P2, P3, P4, P5, P6. P5 has CPU access. B'coz priority of P5 > newly arrived P6.

at t=10

Ready queue = P1, P2, P3, P4, P5, P6, P7. P7 has CPU access. B'coz priority of P7 > currently running P5.

BT of P5 = 6-5 = 1.

All the processes ~~will~~ came at Ready queue. Now this algorithm works as non preemptive method.

at $t=11$

P_7 was terminated.

Ready queue = $P_1, P_2, P_3, P_4, P_5, P_6$.

P_5 has highest priority. P_5 has CPU access. P_5 was terminated at $t = 12$.

Ready queue $(t=12)$ = P_1, P_2, P_3, P_4, P_6

P_1 has highest priority. P_1 has CPU access. P_1 was terminated at $t = 15$.

Ready queue (at $t=15$) = P_2, P_3, P_4, P_6

P_2 & P_3 have highest priority. To break tie, AT of $P_2 = 1$ & AT of $P_3 = 3$. So P_2 has CPU access. P_2 was terminated at $t=17$.

Ready queue (at $t=17$) = P_3, P_4, P_6.

P_3 has CPU access. P_3 was terminated at $t=21$.

Ready queue (at $t=21$) = P_4, P_6.

P_4 has highest priority. P_4 has CPU access. P_4 was terminated at $t = 22$.

Ready queue (at $t=22$) = P_6

P_6 has CPU access. P_6 was terminated at $t=27$.

	AT	Priority	BT	CT	TAT	WT	RT
P1	0	3	8	15	15	7	0
P2	1	4	2	17	16	14	14
P3	3	4	4	21	18	14	14
P4	4	5	1	22	18	17	17
P5	5	2	6	12	7	1	0
P6	6	6	5	27	21	16	16
P7	10	1	1	11	1	0	0

TAT = CT − AT ; WT = TAT − BT

Avg. TAT = $\frac{96}{7}$ = 13.7 ; Avg. WT = $\frac{69}{7}$ = 9.8 ; Avg. RT = $\frac{61}{7}$ = 8.7

Drawbacks:

1. Starvation problem. - If a process is waiting for long amount of time to get CPU access.

Solution to Starvation problem is ageing.

Ageing means allocation of dynamic priority to the processes. Here, the priority of waiting processes can be decreased by a specific count in regular intervals of time

	AT	Priority	BT	CT	TAT	WT	RT
P_1	0	3	8	15	15	7	0
P_2	1	4	2	17	16	14	14
P_3	3	4	4	21	18	14	14
P_4	4	5	1	22	18	17	17
P_5	5	2	6	12	7	1	0
P_6	6	6	5	27	21	16	16
P_7	10	1	1	11	1	0	0

TAT= CT− AT ; WT= TAT− BT

Avg. TAT= $\frac{96}{7}$ = 13.7 ; Avg. WT= $\frac{69}{7}$ =9.8 ; Avg. RT= $\frac{61}{7}$ = 8.7

Drawbacks:

1. starvation problem. - If a process is waiting for long amount of time to get CPU access.

Solution to Starvation problem is ageing.

Ageing means allocation of dynamic priority to the processes. Here, the priority of waiting processes can be decreased by a specific count in regular intervals of time

Starvation & Ageing:

Starvation – Indefinite blocking.

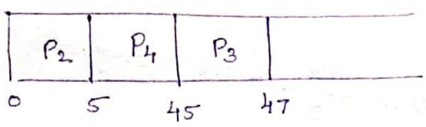
- a process which is ready to run can wait indefinitely b'cos of low priority.

- high priority processes prevent a low priority from ~~ever~~ ever get the CPU.

	BT	Priority
P_1	10	20
P_2	5	1
P_3	2	5
P_4	40	2

· Lesser the no: higher the priority.

Scenario:



new processes:

New Process ↓	Priority
P_3	3
P_6	4
P_7	7

So P_1 has to wait for idefinite amount of time.

Ageing: method to ensure that processes with lower priority will eventually complete their execution.

- by gradually increasing the priority of process that wait in the system for a long time.

Case I: after every 3 unit of time, priority of waiting processes will decrease by 1. (b'coz lesser the no: higher the priority).

$$P_1: \quad 20 \xrightarrow{3} 19 \xrightarrow{3} 18 \xrightarrow{3} 17 \xrightarrow{3} 16 \xrightarrow{3} 15 \cdots \xrightarrow{3} 0$$

may be some other processes are running in CPU

P_1 has highest Priority & get CPU access.

Convoy effect: FCFS

Starvation problem SJF, priority scheduling.

Round Robin (RR) CPU Scheduling algorithm.

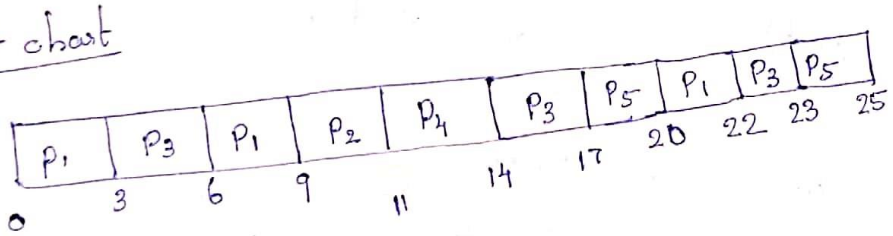
- used in time sharing systems / multi tasking operating systems
- similar to FCFS with time quantum.
- mode: pre emptive
- AT + TQ where TQ = Time Quantum

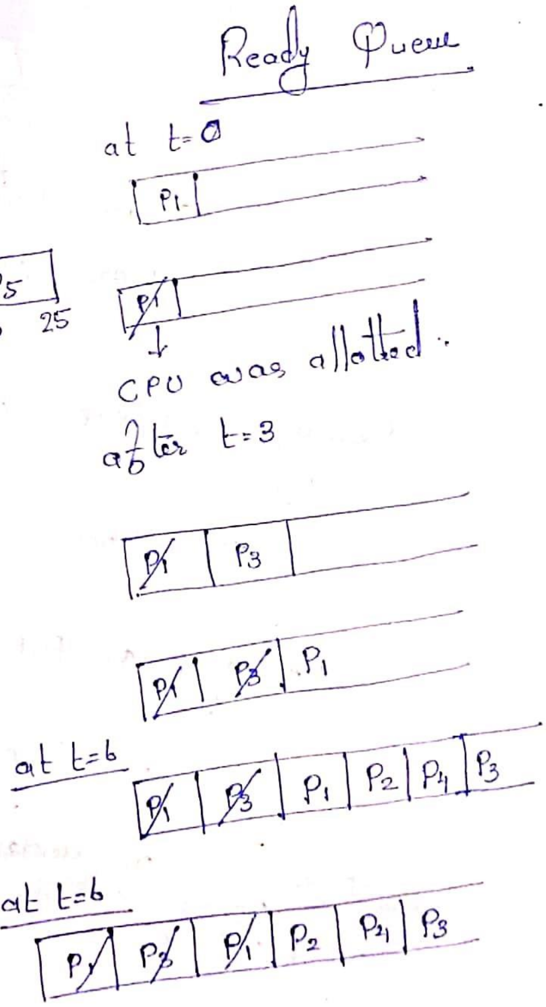
Example:

TQ = 3.

	AT	BT
P1	0	8
P2	5	2
P3	1	7
P4	6	3
P5	8	5

Gantt chart





P1

CPU allocation time = 3.

Remaining BT = 8-3 = 5

P3

Remaining BT = 7-3 = 4

P1

Remaining BT = 5-3 = 2

P2

P2 is terminated, so not in Ready queue.

P4

P4 is terminated

P3

Remaining BT = 4-3
= 1

P5

Remaining BT = 5-3 = 2

P1

P1 is terminated

P5

P5 is terminated

at t=25, no ~~nett~~ Processes waiting in Ready queue. So Gantt chart calculation is over.

at t=9.

~~P1~~	~~P3~~	~~P1~~	P2	P4	P3	P5	P1

at t=9

~~P1~~	~~P3~~	~~P1~~	~~P2~~	P4	P3	P5	P1

at t=11

~~P1~~	~~P3~~	~~P1~~	~~P2~~	~~P4~~	P3	P5	P1

at = 14

~~P1~~	~~P3~~	~~P1~~	~~P2~~	~~P4~~	~~P3~~	P5	P1

at t = 17

~~P1~~	~~P3~~	~~P1~~	~~P2~~	~~P4~~	~~P3~~	P5	P1	P3

~~P1~~	~~P3~~	~~P1~~	~~P2~~	~~P4~~	~~P3~~	~~P5~~	P1	P3

at t = 20

~~P1~~	~~P3~~	~~P1~~	~~P2~~	~~P4~~	~~P3~~	~~P5~~	P1	P3	P5

~~P1~~	~~P3~~	~~P1~~	~~P2~~	~~P4~~	~~P3~~	~~P5~~	~~P1~~	P3	P5

t = 22

~~P1~~	~~P3~~	~~P1~~	~~P2~~	~~P4~~	~~P3~~	~~P5~~	~~P1~~	~~P3~~	P5

at t= 23

~~P1~~	~~P3~~	~~P1~~	~~P2~~	~~P4~~	~~P3~~	~~P5~~	~~P1~~	~~P3~~	~~P5~~

	AT	BT	CT	TAT	WT	RT
P_1	0	8	22	22	14	0
P_2	5	2	11	6	4	4
P_3	1	7	23	22	15	2
P_4	6	3	14	8	5	5
P_5	8	5	25	17	12	9

$\text{Avg. WT} = \frac{50}{5} = 10$

$\text{Avg. TAT} = \frac{75}{5} = 15$

$\text{Avg. RT} = \frac{20}{5} = 4$

TAT = CT - AT

WT = TAT - BT

Advantages:

1. It gives deterministics Response time ie avg. RT is minimum in Round Robin.

Disadvantages:

1. If Tφ is very large, Round Robin works similar to FCFS. (eg: Tφ = 10}

2: If Tφ is very smaller, no: of context switching will be more. ie the no: of context switching will also take some time. in that case Avg. WT will be larger.

.. Idle time quantum is in b/w 10ms to 100ms.

2.

	BT	AT
P_1	8	0
P_2	2	0
P_3	7	0
P_4	3	0
P_5	5	0

Let Context switch time is 1 unit. Average WT, TAT, RT, no: of context switches & CPU utilization?

TQ = 3.

Ready Queue

at t=0, all the processes are in Ready queue.

ie

P_1	P_2	P_3	P_4	P_5	

Gantt chart :

P_1	▨	P_2	▨	P_3	▨	P_4	▨	P_5	▨	P_1	▨	P_3	▨	P_5	▨	P_1	▨	P_3

0 3 4 6 7 10 11 14 15 18 19 22 23 26 27 29 30 32 33 34

▨ . CPU is idle. b'coz of context switching.

at t=0 :

BT of P_1 = 8-3=5.

Ready Queue:

~~P_1~~	P_2	P_3	P_4	P_5	

~~P_1~~	P_2	P_3	P_4	P_5	P_1

at t=4

~~P_1~~	~~P_2~~	P_3	P_4	P_5	P_1

P_2 is terminated

at t=7

~~P_1~~	~~P_2~~	~~P_3~~	P_4	P_5	P_1	

BT of P_3 = 7-3 = 4

~~P_1~~	~~P_2~~	~~P_3~~	P_4	P_5	P_1	P_3	

at t= 11

~~P_1~~	~~P_2~~	~~P_3~~	~~P_4~~	P_5	P_1	P_3

P_4 is terminated

at t = 15

~~P_1~~	~~P_2~~	~~P_3~~	~~P_4~~	~~P_5~~	P_1	P_3	

BT of P_5 = 5 - 3 = 2

~~P_1~~	~~P_2~~	~~P_3~~	~~P_4~~	~~P_5~~	P_1	P_3	P_5	

at t= 19

~~P_1~~	~~P_2~~	~~P_3~~	~~P_4~~	~~P_5~~	~~P_1~~	P_3	P_5	

BT of P_1 = 5-3= 2

~~P_1~~	~~P_2~~	~~P_3~~	~~P_4~~	~~P_5~~	~~P_1~~	P_3	P_5	P_1	

at t = 23

~~P_1~~	~~P_2~~	~~P_3~~	~~P_4~~	~~P_5~~	~~P_1~~	~~P_3~~	P_5	P_1

B T of $P_3 = 4-3 = 1$.

~~P_1~~	~~P_2~~	~~P_3~~	~~P_4~~	~~P_5~~	~~P_1~~	~~P_3~~	P_5	P_1	P_3

at t = 30.

~~P_1~~	~~P_2~~	~~P_3~~	~~P_4~~	~~P_5~~	~~P_1~~	~~P_3~~	~~P_5~~	~~P_1~~	P_3

P_1 is terminated

at t = 27

~~P_1~~	~~P_2~~	~~P_3~~	~~P_4~~	~~P_5~~	~~P_1~~	~~P_3~~	~~P_5~~	P_1	P_3

P_5 is terminated.

at t = 33

~~P_1~~	~~P_2~~	~~P_3~~	~~P_4~~	~~P_5~~	~~P_1~~	~~P_3~~	~~P_5~~	~~P_1~~	~~P_3~~

P_3 is terminated.

	BT	A T	CT	TAT	WT	RT
P_1	8	0	32	32	24	0
P_2	2.	0	6.	6	4	4
P_3	7	0	34	34	27	7
P_4	3	0	14	14	11	11
P_5	5	0	29	29	24	15

Avg WT = $\frac{90}{5}$ = 18

Avg. TAT = $\frac{115}{5}$ = 23.

Avg. RT = $\frac{37}{5}$ = 7.4

TAT = CT - AT

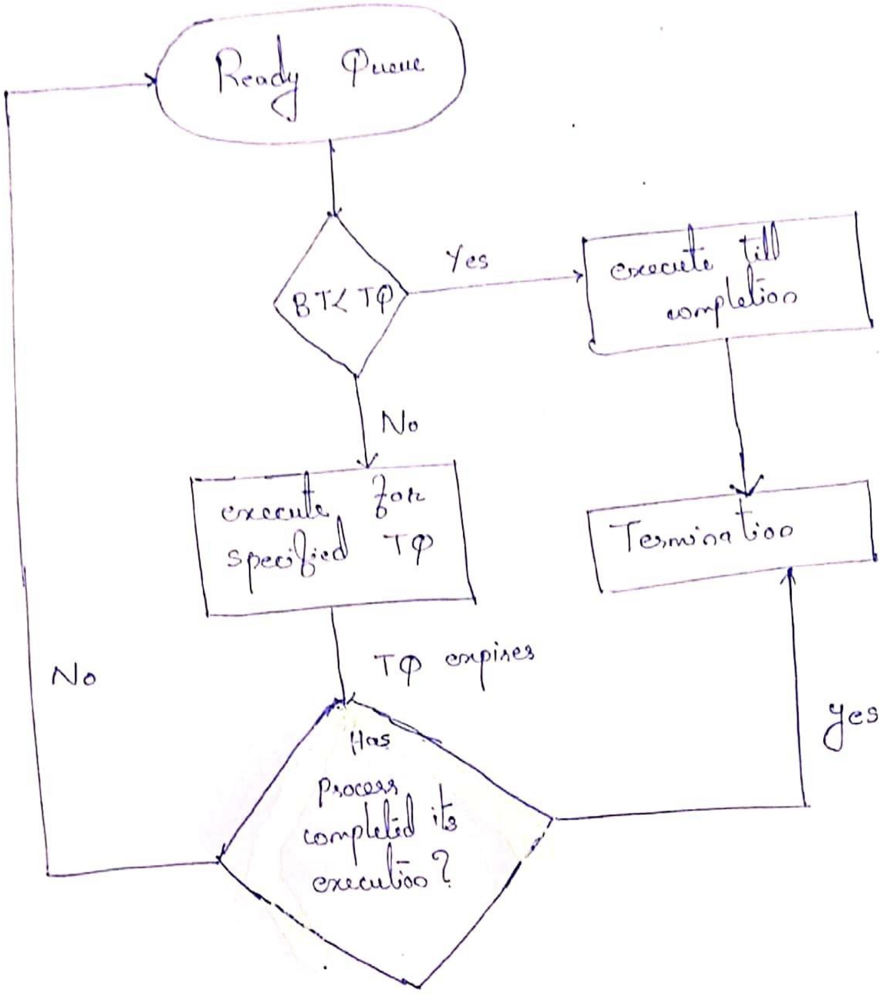
WT = TAT - BT

No: of context switches = 9.

CPU utilization = $\frac{\text{Expected time}}{\text{Actual time}} \times 100 = \frac{\text{Total BT}}{34} = \frac{8+2+7+3+5}{34} = \left(\frac{25}{34}\right) \times 100$

= 72.5 %.

Flow chart of Round Robin Algorithm:



Advantages :

- easy & simple to implement
- each process gets a fair share of CPU
- no starvation
- no convoy effect.
- deterministic response time.
- priority is same for each process.
- most frequently used.
- Also known as Time slicing algorithm.

Disadvantages:

- Throughput depends on time quantum
- If TQ is small - more overhead of context switching. & avg. waiting time increases

- if TQ is large - same as FCFS.
- deciding of TQ is very tough.

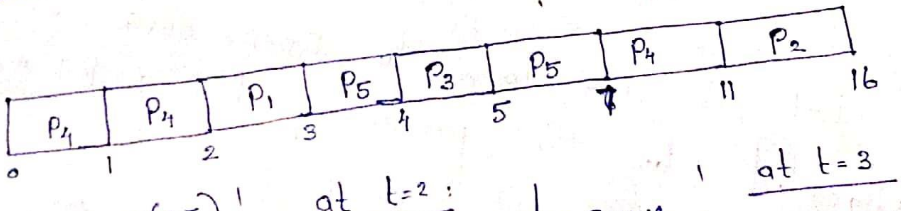
Shortest Job First (SJF) with Preemption / SRTF

- Whenever new process arrives, there may be preemption of the running process.
- ie when the newly arrived process has shorter burst time than the currently running processes, then only preemption will happen.
- Preemption is the phenomenon in which a process can be removed from CPU allocation before its termination.

Example:

	AT	BT
P_1	2	1
P_2	1	5
P_3	4	1
P_4	0	6
P_5	2	3

Gantt chart



at t=0
P_4 is only in Ready queue.

$P_4 = 6-1 = 5$ (BT)
at t=1
P_2 arrives
BT of $P_2 = 5$
To break tie, use FCFS, so P_4 continues

at t=2:
P_1 is in ready queue
P_5 is in ready queue.
BT of $P_4 = 5-1 = 4$.
∴ BT of $P_1 = 1$
$P_2 = 5$
$P_4 = 4$
$P_5 = 3$
∴ shortest burst time (BT) = P_1

at t=3.
P_1 = terminated.
- P_4, P_2, P_5 (to complete)

∴ BT of $P_4 = 4$
$P_2 = 5$
$P_5 = 3$
∴ shortest BT = P_5

at $t=4$

P_3 is in ready queue.

BT of $P_2 = 5$
$P_3 = 1$
$P_4 = 4$
$P_5 = 3-1 = 2$.

shortest BT = P_3

at $t=5$ (works as SJF algorithm).

P_3 = terminated

P_2, P_4, P_5 (to complete).

BT of $P_2 = 5$
$P_4 = 4$
$P_5 = 2$

shortest BT = P_5. Then P_4 will execute. Then P_2.

Now all the processes (P_1, P_2, P_3, P_4, P_5) arrived in Ready queue. Now this algorithm works as SJF algorithm. ie after all the process are arrived in Ready queue, the SRTF works as SJF only.

$\sum BT = 1+5+1+6+3$
$= 16$. = Final time of Gantt chart.

	AT	BT	CT	TAT	WT	RT
P_1	2	1	3	1	0	0
P_2	1	5	16	15	10	10
P_3	4	1	5	1	0	0
P_4	0	6	11	11	5	0
P_5	2	3	7	5	2	1

TAT = CT − AT

WT = TAT − BT

* In Pre emptive algorithm WT ≠ RT.

$$\text{Avg. TAT} = \frac{33}{5} = 6.6$$

$$\text{Avg. WT} = \frac{17}{5} = 3.4$$

SRTF will give minimal waiting time. So it is the optimal solution.

ie all the 4 conditions should simultaneously held in a system. Then Deadlock condition will occur.

Resource Allocation Graph

- How resources are allocated among processes ie represented using graphs. It is known as Resource allocation graph.

A graph has vertices and Edges

Set of Vertices:

V → set of Processes ($P_1, P_2, P_3 \ldots P_n$)
→ Set of Resources ($R_1, R_2 R_3 \ldots R_n$)

Edges:

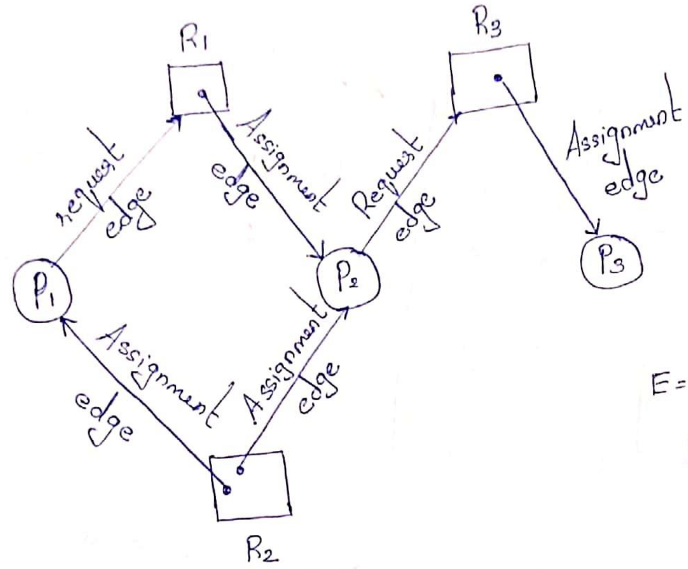
E → $P_i \rightarrow R_j$ (Request Edge)
→ $R_j \rightarrow P_i$ (Assignment Edge)

Request Edge: Process is requesting for a resource.

Assignment Edge: Resource is assigned to a process.

□ = Resource type ; ○ = Process ; ⊡ — instance of resource type.

eg: 5 printers ⇒ printer is a resource
5 is the instance of the resource.



P = {P1, P2, P3}

R = {R1, R2, R3, R4}

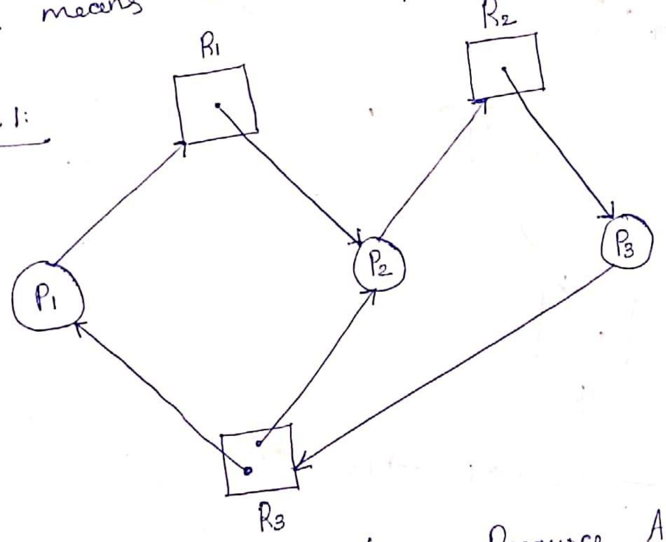
E = {P1 → R1, P2 → R3, R1 → R2,
R2 → P2, R2 → P1, R3 → P3}.

· If no cycle in Resource Allocation Graph, then no process in the system is deadlocked & if it contains cycle then deadlock may exist.

Method-1

cycle means a closed path.

Example 1:



· 2 cycles are in the above Resource Allocation Graph.

· P1 → R1 → P2 → R2 → P3 → R3 → P1 &

P2 → R2 → P3 → R3 → P2.

· Here P1 is waiting for R1
P2 " R2
P3 " R3. But P1 & P2 are holding two instances of R3. So no progress for P1, P2 & P3. Hence Deadlock for processes

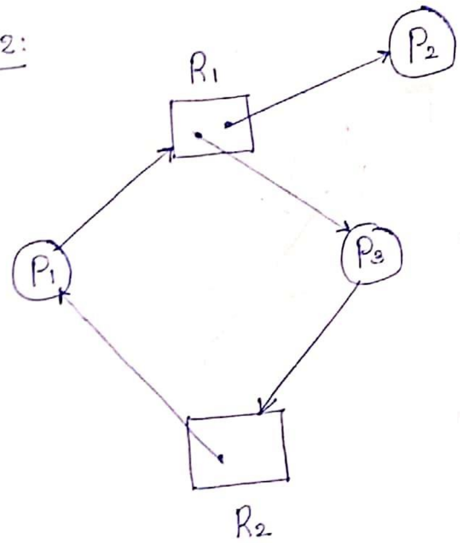
Method-2

	Allocation matrix			Request matrix			Availability matrix		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	0	0	1	1	0	0	0	0	0
P_2	1	0	1	0	1	0			
P_3	0	1	0	0	0	1			

· Availability matrix = available instances of Resources.

Zero availability of all resources. ∴ System is in **Deadlock.**

Example 2:



1 cycle: $P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

P_1 is waiting for R_1

P_2 is not waiting for any resources.

P_2 is holding R_1

P_1 is holding R_2

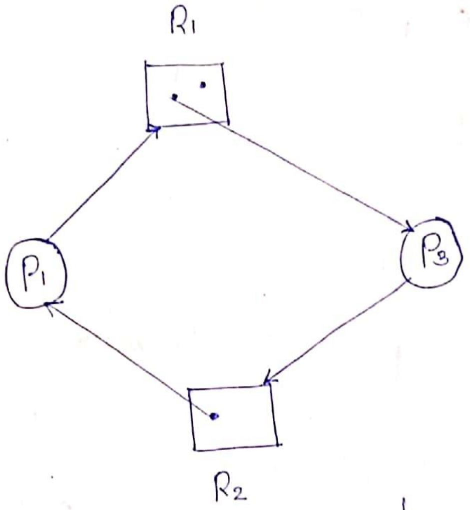
P_3 is waiting for R_2

P_3 is holding R_1

R_1 has 2 instances.
R_2 has 1 instance.

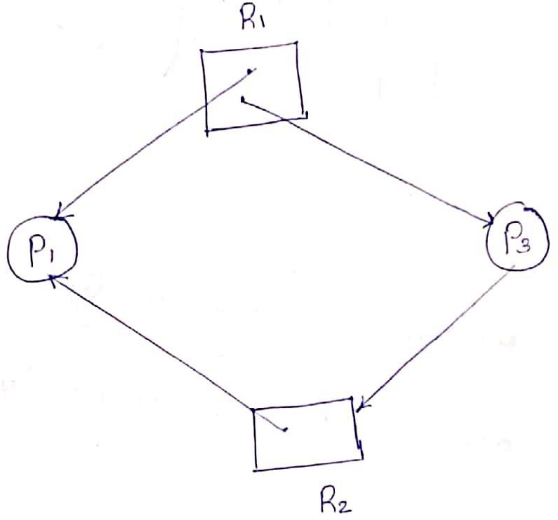
Since P_2 is not waiting, it can terminate at some time

∴ graph becomes

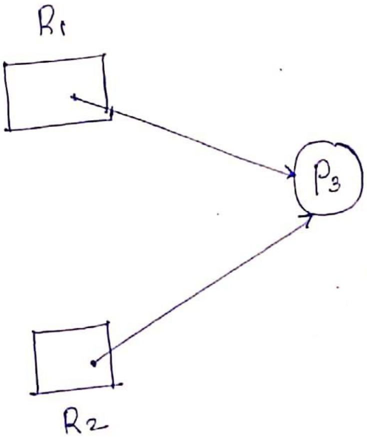


{One instance of R_1 is free.

That instance of R_1 can be assigned to P_1



∴ P_2 is terminated & one instance of R_2 is free



∴ P_3 is terminated.

System is not in Deadlock.

	Allocation matrix		Request matrix		Availability matrix	
	R1	R2	R1	R2	R1	R2
P1	0	1	1	0	0	0
P2	1	0	0	0		
P3	1	0	0	1		

Request of P2 can be fulfilled. So P2 releases R1. Hence

Availability matrix

	R1	R2
P1	0 1	0 0
✓P2	$\boxed{1}$	$\boxed{0}$
P3		

⇒

Availability matrix

	R1	R2
✓P1	0 1	0 0
✓P2	$\boxed{1}$ 0 $\boxed{1}$	$\boxed{0}$ 0 $\boxed{1}$
P3		

⇒

Availability matrix

	R1	R2
✓P1	0 1	0 0
✓P2	$\boxed{1}$ 0 $\boxed{1}$	$\boxed{0}$ 0 $\boxed{1}$
✓P3	1 $\boxed{2}$	0 $\boxed{1}$

So no <u>deadlock</u>

- Cycle + only one instance of resource type ⇒ Deadlock is there.
- Cycle + more than one instance of resource type ⇒ Deadlock may exist.

Deadlock handling in O.S.

4 methods : Depends on the nature of the problem.

1. Deadlock prevention
2. Deadlock Avoidance
3. Deadlock Detection & Recovery.
4. Deadlock Ignorance (Ostrich method).

1. Deadlock Prevention: prevent deadlock from occur.

Disadvantages: More effort.

2. Deadlock avoidance: Futuristic method.

3. Deadlock Detection & Recovery: Once the deadlock occurs, detect it and recover it.

4. Ostrich method: assume that deadlock will never occur.

Used by end users. eg: Reboot the system.

In Aircraft, Hospitals we cannot apply Deadlock Ignorance/ Ostrich method. Here we can use Deadlock Prevention method.

1. Deadlock Prevention: violate any of the four necessary conditions at any time & deadlock can never occur in the system.

→ Removal of mutual exclusion:

→ Removal of hold & wait: ie no hold & wait.

↳ A process must acquire all the necessary resources before execution. Disadv: Resource utilization will be very low. This approach is practically not implementable.

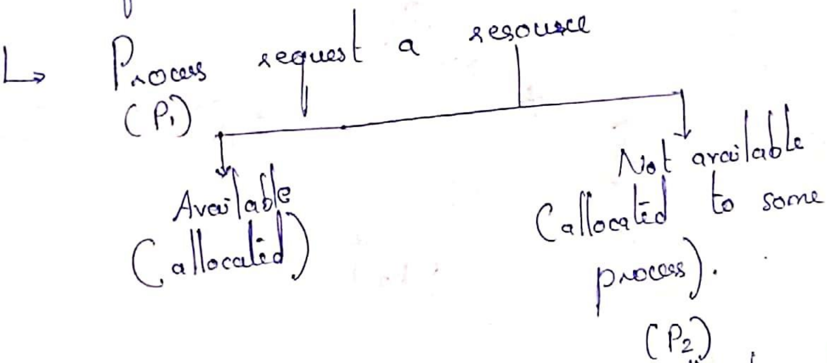
↳ Process holding some resources & requesting for additional resources, then it must release the acquired resources first. Drawback is starvation problem.

↳ wait time out: resources are time bound.

→ Removal of non-preemption: here a process can forcefully take a resource from a waiting process not a running process.

↳ Resources can be preempted from processes.

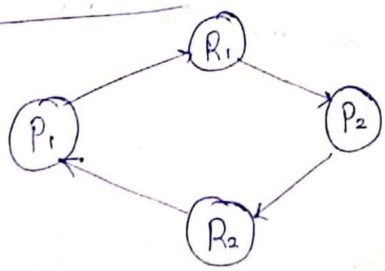
↳ process holding some resources & requesting for another resource that cant be immediately allocated, then all the acquired resources will be preempted.


↳ Process request a resource
(P1)
Available
(allocated)
Not available
(allocated to some other waiting process).
(P2)

. always the waiting process will be treated as victim.

So the resource of P2 is given to P1.

Removal of Circular wait:



increasing order / decreasing order.

Assume increasing order.

Printer	1
CPU	5
Memory	6
CD Drive	7

①

P1	P2
5	

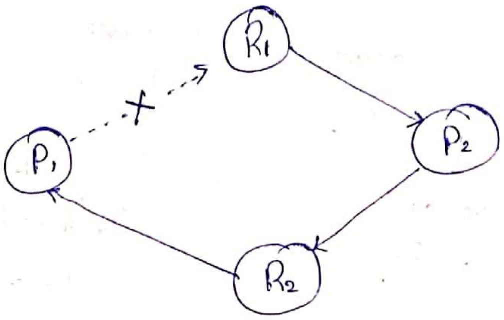
So P1 can request only 6 & 7. It cannot request 1.

P_1	P_2
1, 6	

. P_1 needs 5. For that P_1 has to release 6. Then P_1 can request 5.

- To request resource R_j, a process must first release all the acquired resources R_i such that $i >= j$.
- It can be implementable. But it is a tedious process ~~to~~ in ordering the resources.

∴



∴ no circular wait.

Deadlock Avoidance:

- System maintains some database using which it can take decision whether to entertain a request or not, just to be in safe ~~side~~ state. Unsafe state may lead to deadlock.
- System (Kernel) analyse the data base (allocation state) to determine whether granting a request can lead to deadlock in future.
- ↳ if not lead to deadlock, then the request is granted.

  ↳ otherwise keep pending until they can be granted. (process may face long delay for obtaining a resource).

Resource allocation graph algorithm

V → set of vertices → set of processes. / set of resources.

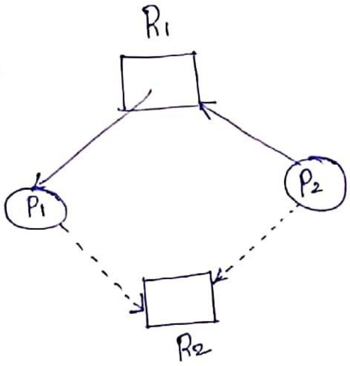
E → set of edges. → Request edge / Assignment edge / claim edge.

Request edge $(P_i \rightarrow R_j)$

Assignment edge $(R_j \rightarrow P_i)$

claim edge $(P_i \dashrightarrow R_j)$ ↓ P_i may request R_j in future.

eg:

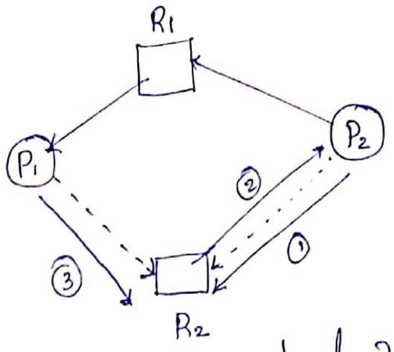


1. $P_1 \dashrightarrow R_1$: P_1 may request R_1 in future
2. $P_1 \longrightarrow R_1$: P_1 is requesting R_1
3. $P_1 \longleftarrow R_1$: if R_1 is free, it is allocated to P_1.
4. $P_1 \dashrightarrow R_1$: when R_1 is released, it became a claim edge.

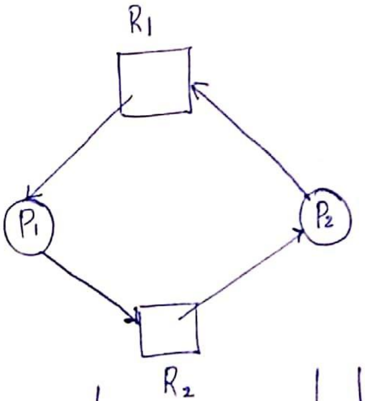
In this algorithm: all the resources have only one instance.

Before starting the execution, all the claim edges must be there in resource allocation graph. It means Kernel know in advance that, these resources may be used by the process in future.

Consider the case,



⇒



{ - if P_i request R_j then request edge can only be converted to assignment edge if it does not form a cycle in Resource allocation graph. }

a cycle occurs which result in **deadlock**.

Banker's algorithm

· handles multiple instances of same resources.

1. how many instances of each resource each process can max. request [MAX]. It is a 2D matrix/array.

2. how many instances of each resource each process currently holds. [Allocation]. It is a 2D array

3. how many instances of each resource is available in the system. [Available]. It is a 1D array

These 3 things should be known to apply Banker's algorithm.

eg:

	Allocation				Max				Available				Need				
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D	
P_0	0	0	1	2	0	0	1	2	1	5	2	0	0	0	0	0	✓
P_1	1	0	0	0	1	7	5	0					0	7	5	0	×
P_2	1	3	5	4	2	3	5	6					1	0	0	2	×
P_3	0	6	3	2	0	6	5	2					0	0	2	0	✓
P_4	0	0	1	4	0	6	5	6					0	6	4	2	×

1. Need matrix?

2. Is system in safe state? If yes then find safe sequence?

	A	B	C	D
Total:P0:	3	14	12	12

= Allocation + Available

Allocation

	A	B	C	D
P0	0	0	1	2
P1	1	0	0	0
P2	1	3	5	4
P3	0	6	3	2
P4	0	0	1	4
	2	9	10	12

+

Available

A	B	C	D
1	5	2	0

- Need : max − allocation.

Safe sequence: is not unique. It can start with P0 or P3.

	Allocation				Max				Available	Need	
	A	B	C	D	A	B	C	D	A B C D	A B C D	
P0	0	0	1	2	0	0	1	2	1 5 2 0 0 0 1 2	0 0 0 0	✓ 1.
P1	1	0	0	0	1	7	5	0	1 5 3 2 1 3 5 4	0 7 5 0	5.
P2	1	3	5	4	2	3	5	6	2 8 8 6 0 6 3 2	1 0 0 2	✓ 2.
P3	0	6	3	2	0	6	5	2	2 14 11 8 0 0 1 4	0 0 2 0	✓ 3
P4	0	0	1	4	0	6	5	6	2 14 12 12 1 0 0 0	0 6 4 2	✓ 4.
									3 14 12 12		

safe sequence: P0 P2 P3 P4 P1 or P0 P2 P3 P1 P4.

System is in safe state. After execution of all processes, available instances of resources = total instances of resources.

Banker's algorithm. Input – Processes

- any 2 out of 3 (Max, need, allocation)
- available or total no: of process.

step 1: flag[i]=0 for i=0 to (n-1) & find Need[n][m] = Max[n][m] – allocation[n][m]

step 2: find a process P_i such that: - flag[i]=0 & Need i <= Available

step 3: If such i exists then

flag[i]=1, available = available + allocation

goto step 2.

otherwise goto step 4.

step 4: if flag[i]=0 for all i then system is in safe state otherwise unsafe state.

m = no: of resources.

n = no: of processes.

- Banker's algorithm is also known as Safety algorithm.
- Time complexity of Banker's algorithm is $O(n^2m)$.

eg.

	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	2	0	0	1	4	2	1	2	3	3	2	1
P_1	3	1	2	1	5	2	5	2				
P_2	2	1	0	3	2	3	1	6				
P_3	1	3	1	2	1	4	2	4				
P_4	1	4	3	2	3	6	6	5				

1. Need matrix? 2. Is system in safe state? If yes find safe sequence? 3. If request from P_1 arrives for (1, 1, 0, 0), can request be immediately granted? 4. If request from P_4 arrives for (0, 0, 2, 0), can it be immediately granted?

Ans:-

	Allocation				Max				Available				Need				
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D	
P_0	2	0	0	1	4	2	1	2	3 2	3 0	2 0	1 1	2	2	1	1	✓ 1.
P_1	3	1	2	1	5	2	5	2	5 1	3 3	2 1	2 2	2	1	3	1	× 4
P_2	2	1	0	3	2	3	1	6	6 1	6 4	3 3	4 2	0	2	1	3	× 5
P_3	1	3	1	2	1	4	2	4	7 3	10 1	6 2	6 1	0	1	1	2	× 2.
P_4	1	4	3	2	3	6	6	5	10 2	11 1	8 0	7 3	2	2	3	3	× 3.
									12	12	8	10					

- Need = Max- allocation.
- Total = Allocation + Available

$$\underset{\text{A B C D}}{\text{Allocation}} \quad 9\ 9\ 6\ 9 \; + \; \underset{\text{A B C D}}{\text{Available}} \quad 3\ 3\ 2\ 1 \; = \; \underset{\text{A B C D}}{\text{Total}} \quad 12\ 12\ 8\ 10$$

Allocation					Available					Total			
A	B	C	D	+	A	B	C	D	=	A	B	C	D
9	9	6	9		3	3	2	1		12	12	8	10

Safe sequence: P0 P3 P4 P1 P2 ∴ system is in safe state.

If the system is able to execute all the processes without going to unsafe state, then we can system is in safe state.

Safe sequence: Sequence in which the processes execute in safe state is known as Safe sequence.

. System will not grant request even though the resources are available.

③. Requesting resource (1,1,0,0) should be less than or equal to (2,1,3,1) ie need matrix

. System also checks (1,1,0,0) is less than or equal to Available matrix ie (3,3,2,1).

. System will pretend to grant the resources.

. Then system will apply Banker's algorithm. If safe sequence can be obtained, then system will grant the resources.

Need

	A	B	C	D	
P1	2	1	3	1	−
	1	1	0	0	
	1	0	3	1	

Available

	A	B	C	D	
P1	3	3	2	1	−
	1	1	0	0	
	2	2	2	1	

Allocation

	A	B	C	D	
P1	3	1	2	1	+
	1	1	0	0	
	4	2	2	1	

after the updation,

	Allocation A B C D	max A B C D	Available A B C D	Need A B C D	
P0	2 0 0 1	4 2 1 2	2 2 2 1 2 0 0 1 4 2 2 2	2 2 1 1	✓ 1.
P1	4 2 2 1	5 2 5 2	1 3 1 2 5 5 3 4	1 0 3 1	× 4.
P2	2 1 0 3	2 3 1 6	1 4 3 2 6 9 6 6	0 2 1 3	× 5
P3	1 3 1 2	1 4 2 4	4 2 2 1 10 11 8 7	0 1 1 2	× 2.
P4	1 4 3 2	3 6 6 5	2 1 0 3 12 12 8 10	2 2 3 3	× 3.

Safe sequence: $\boxed{P_0 \quad P_3 \quad P_4 \quad P_1 \quad P_2}$ ie system is in safe state.

Yes, the request can be immediately granted..

4.

	Allocation				Max				Available				Need				
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D	
P_0	2	0	0	1	4	2	1	2	3	3	0	1	2	2	1	1	×
P_1	3	1	2	1	5	2	5	2					2	1	3	1	×
P_2	2	1	0	3	2	3	1	6					0	2	1	3	×
P_3	1	3	1	2	1	4	2	4					0	1	1	2	×
P_4	1	4	5	2	3	6	6	5					2	2	1	3	×

. System is in unsafe state. System will not grant this request immediately. System may be in deadlock if system grants the request.

Resource - Request algorithm (Banker's algorithm)

step 1: If $Request_i \le Need_i$ then go to step 2 otherwise error.

step 2: If $Request_i \le Available$ then go to step 3 otherwise P_i will wait.

step3: System pretend as if request has been granted by modifying the state as follows:

$$\left\{\begin{array}{l} Available \ -= Request_i \\ Allocation \ += Request_i \\ Need \ -= Request_i \end{array}\right.$$

. If modified resource-allocation state is safe then request granted.

. Otherwise P_i will wait & old allocation state is restored.

III. Deadlock Detection & Recovery:

. Allow the system to enter into deadlock state.

. Deadlock detection algorithms.

. Recovery techniques.

2 types of deadlock detection algorithms:

Single instance (wait for graph)

multiple Instances (Banker's algorithm)

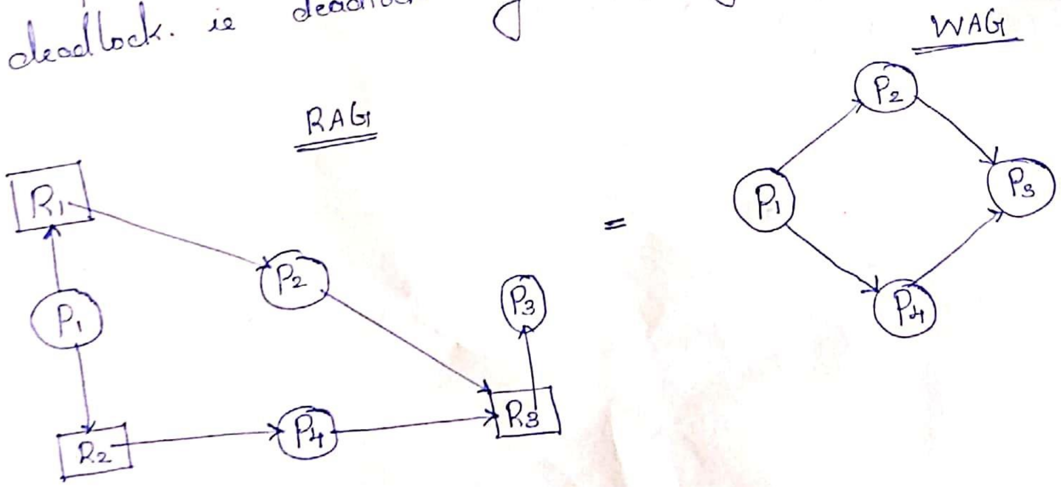
. Wait for graph is an enhanced version of Resource allocation graph.

Necessary & sufficient condition for deadlock in wait for graph:

. Single instance + Detect Cycle

. If multiple instances & existence of cycle ⇒ Necessary condition for deadlock. ie deadlock may occur/may not occur.

eg:

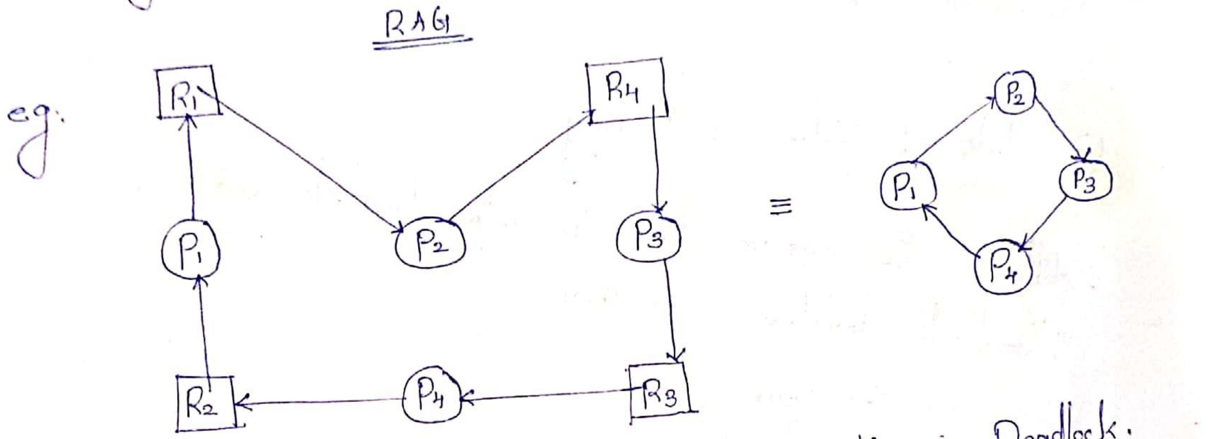


RAG has processes & resources.

Wait for graph has processes only.

No cycle in wait for graph. ∴ no deadlock.

eg.



Cycle exists in the wait for graph. It means there is Deadlock.

Banker's algorithm (Safety algorithm)

· for multiple instances of resources.

eg.

	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	0	0	0	0	0	0
P1	2	0	0	2	0	2			
P2	3	0	3	0	0	0			
P3	2	1	1	1	0	0			
P4	0	0	2	0	0	2			

Ans:

		Allocation			Request			Available
		A	B	C	A	B	C	A B C
1. ✓	P0	0	1	0	0	0	0	0 0 0 + 0 1 0
4. ✓	P1	2	0	0	2	0	2	0 1 0 3 0 3
2. ✓	P2	3	0	3	0	0	0	3 1 3 2 1 1
3. ✓	P3	2	1	1	1	0	0	5 2 4 2 0 0
5.	P4	0	0	2	0	0	2	7 2 4 0 0 2 7 2 6

- The request of P_0 and P_2 is equal to available resources.
- So we are taking P_0.

$< P_0\ P_2\ P_3\ P_1\ P_4 >$ is the safe sequence.

Also Total resources are :

Allocation			+	Available			=			
A	B	C		A	B	C		A	B	C
7	2	6		0	0	0		7	2	6

Here there is no deadlock

eg:

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

- P_0 – only P_0 can be executed. System cannot execute any other processes. So system is in deadlock.
- P_1, P_2, P_3 & P_4 are in deadlocked state.
- Deadlock will come, when system is not able to assign the request immediately.

Deadlock Recovery.

1. Optimistic approach
(Preemption of Resources & Processes)

↳ Preempt some resources from process & give these resources to other processes until the deadlock cycle is broken.

• The process to be preempted is based on the following factors

→ ↳ selecting a victim based on cost factors

→ ↳ Rollback: Victim process was Rollbacked. 2 methods.

↳ either roll back to the previous safe state.

↳ or roll back to the initial point - total Rollback.

Its better to rollback to previous safe state.

But system has to maintain the states of all running process. Then only process can be roll backed to previous safe state.

→ Starvation: occurs when same process has been selected for Roll back. To avoid starvation problem, finite no: of rollbacks should be given to a process.

2. Pessimistic approach.
(Process termination)

↳ abort all deadlocked processes.
But it is a costly approach

↳ Abort one process at a time & decide next to abort after deadlock detection.

Disadvantages: overhead of calling detection algorithm again & again.

Factors to consider when a system is decide to kill a process.

↳ Priority of processes

↳ How long the process has computed?

↳ How much longer a process will compute before completion?

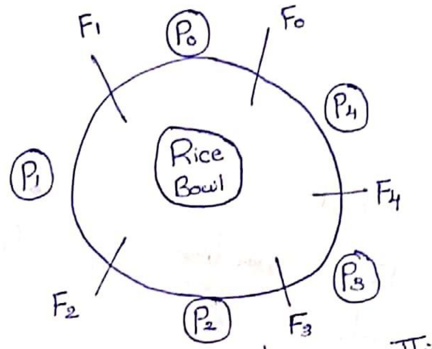
↳ How many & what types of resources, process has used?

↳ How many resources the process needs to complete before its execution?

↳ finite no: of Rollbacks.

Dining Philosophers Problem:

- 5 philosophers
- 5 forks.



Every philosopher has two actions: 1. Think 2. Eat

```
Void Philosopher (void)
{
  while (true)
   {
     Thinking();
     take_fork(i);          ← Left fork.
     take_fork((i+1) %N);  ← Right fork.
                               N = no: of forks.
     EAT();
     put fork (i);
     put fork ((i+1) %N);

   }
 }
```

Case 1: P0

4 actions each philosopher does. 1. Thinking 2. Take left fork 3. Take Right fork 4. Eat. Then put left fork & later on Right fork.

. 1. Thinking 2. F0 3. (0+1) mod 5 = F1 4. Eat 5. Put F0 6. put F1.

P_1.

1. F_1 2. $(1+1) \bmod 5 = 2 \rightarrow F_2$ 3. Eat 4. place F_1 5. place F_2.

Likewise P_2, P_3 & P_4. This code will perfectly run if the philosopher's arrives serially.

Case 2 : Firstly P_0 arrives. 1. F_0

Then P_1 arrives 1. F_1. P_0 is waiting for F_1. P_0 will get F_0 only after P_1 finishes his food.

So if more philosopher's arrives at the same time, then Race Around condition occurs.

So we have to use Binary Semaphores. We are using an array of Semaphores. ie S_0 S_1 S_2 S_3 S_4 ⇒ no. of semaphores = 5 = no: of philosophers's = no. of forks.

$S_0 = S_1 = S_2 = S_3 = S_4 = 1$.

```
Void philosopher (void)
  {
     while (true)
      {
        Thinking ();
Entry  { wait (takefork (Si))
Code   { wait (takefork (S(i+1) mod N)
     CS ← EAT ( );
Exit   ( Signal (put fork (i)) ;
code   ( signal (put fork((i+1) % N)) ;
      }
  }
```

P_0	S_0	S_1
P_1	S_1	S_2
P_2	S_2	S_3 .
P_3	S_3	S_4
P_4	S_4	$S_0 = (4+1) \bmod 5$ $= 5 \bmod 5 = 0$ $= F_0$

$S_0 = F_0$; $S_1 = F_1$; $S_2 = F_2$; $S_3 = F_3$;

$S_4 = F_4$.

We know $S_0 = S_1 = S_2 = S_3 = S_4 = 1$ (initialised)

Assume P_0 P_1 P_2

P_0 has $F_0 \not\& F_1$ ∴ $S_0 = 0$ $S_1 = 0$

P_1 comes. It was blocked.

P_2 comes. $F_2 \not\& F_3$ ∴ $S_2 = 0$ $S_3 = 0$

∴ $\boxed{\begin{array}{l}\text{EAT}\\ P_0\ P_2\end{array}}$ = CS

Two philosophers can EAT at a time if they are independent.

So in Critical section we have two philosophers at a time.

Case : Deadlock.

$S_0 = S_1 = S_2 = S_3 = S_4 = 1$.

P_0 comes first. $S_0 = \cancel{1}\ 0 = F_0$. P_0 cannot take F_1. P_0 was preempted.

B'coz P_1 arrives.

$P_1 \Rightarrow S_1 = \cancel{1}\ 0$. P_1 preempted. P_2 comes.

$P_2 \Rightarrow S_2 = \cancel{1}\ 0$. P_2 was pre empted . P_3 arrives.

$P_3 \Rightarrow S_3 = \cancel{1}\ 0$. P_3 was pre empted . P_4 arrives.
blocked .

$P_4 \Rightarrow S_4 = \cancel{1}\ 0$. P_4 was ∵ $S_0 = 0$ ∴ P_4 was blocked.

Now, $S_0 = S_1 = S_2 = S_3 = S_4 = 0$. Deadlock Occurs .

Solution: $P_0 \Rightarrow S_0 = \cancel{1}\ 0$ P_0 was pre empted. P_1 arrives.

$P_1 \Rightarrow S_1 = \cancel{1}\ 0$. P_1 was pre empted. P_2 arrives.

$P_2 \Rightarrow S_2 = \cancel{1}\ 0$. P_2 was pre empted. P_3 arrives.

$P_3 \Rightarrow S_3 = \cancel{1}\ 0$. P_3 was pre empted. P_4 arrives.

P_4 has to take Right fork first & then Left fork.

$P_4 \Rightarrow S_0 =$ already zero. ∴ P_4 is in blocked state. ie value of S_4 doesn't change.

But we have $\begin{array}{ccccc} S_0 & S_1 & S_2 & S_3 & S_4 \\ \Downarrow & \| & \| & \| & \| \\ 0 & 0 & 0 & 0 & \underline{1} \end{array}$

∴ $S_4 = \cancel{1}\ 0$. ∴ P_3 enters in CS ie in $\boxed{\text{EAT } P_3}$.

∴ P_3 have $S_4 = 1$

P_3 did exit code.

∴ P_3 makes $S_3 = 1$ & $S_4 = 1$

P_2 need S_3. P_3 releases S_3 ∴ $P_2 \Rightarrow S_3 = \cancel{1}\ 0$. ∴ P_2 enters into CS.

P_2 did exit code.

P_2 releases S_2 & S_3 ∴ $S_2 = 1$ & $S_3 = 1$.

|||ly P_1 & P_0 could be in Critical Section.

After P_0 releases semaphores S_0 & S_1. Then $S_0 = S_1 = 1$.

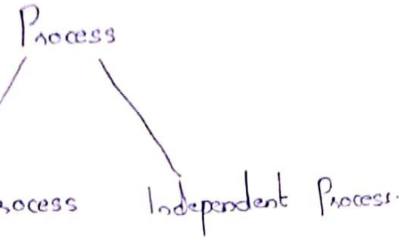
∴ P_4 could be in CS. (∵ P_4 needs $S_0 = 1$).

For (N-1) philosophers use the above code.

For N^{th} philosopher

Entry code ⇒ $\left\langle \begin{array}{l} \text{wait (takefork}(S_{(i+1)} \bmod N)) \\ \text{wait (takefork}(S_i)) \end{array} \right.$

Process Synchronization.



Co operative processes

- They share something like Variable, memory, code, Resources (CPU, Printer, Scanner)
- Here execution of one process affect other processes. eg: ATM transaction.

Independent processes

- Nothing common with these processes.

Co operative Processes.

- Process Synchronization is important. Otherwise it can create problems.

eg:

P₁

```
1.  int x = shared
2.      x++;
3.  sleep(1);
4.  shared = x ;
```

P₂

```
int y = shared;
y--;
sleep(1);
shared = y;
```

int shared = 5

We have only one CPU.

Assume CPU is allocated to Process 1 (P1) initially

∴

```
1. int x = shared ; x=5
2. x++; x=6
3. sleep(1); process is pre empted ie P1 = pause ; CPU is not idle.
4. shared = x;
```

. Process context occurs from instruction 3 of P_1 to instruction 1 of P_2.
Data of P_1 is stored in PCB.

P_2

1. int y = shared ; y = 5
2. y-- ; y = 4
3. sleep (1); P_2 is preempted. CPU is handed over to P_1 ie process context occurs.
4. shared = y;

Here process context occurs from 3rd instruction of P_2 to 4th instruction of P_1.

P_1

1. int x = shared ; x = 5
2. x -- ; x = 6
3. sleep (1); P_1 was pre empted.
4. shared = x ; x = 6.

P_1 was terminated. So, all resources handled by P_1 becomes free. CPU access will be given to 4th instruction of P_2.

P_2

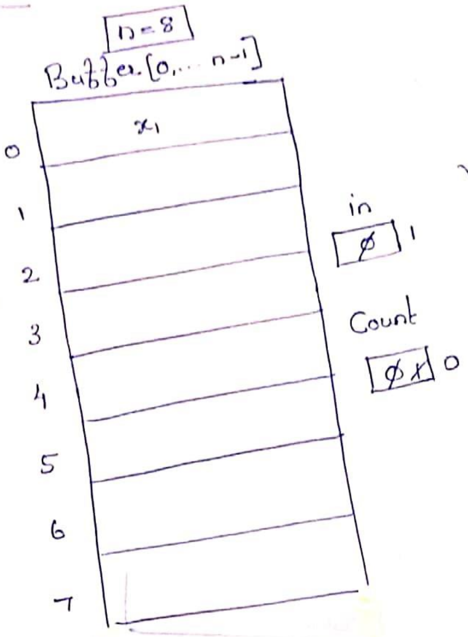
1. int y = shared ; y = 5
2. y-- ; y = 4
3. sleep (1).
4. shared = y ; y = 4.

shared = ~~5~~ ~~6~~ 4. This is a wrong answer. Exact answer should be 5. No process Synchronization. This problem is known as Race condition.

Producer Consumer problem:

Count = global variable; shared by both processes
Buffer = shared resource.

```
{ int itemc;
  while (true)
                              Out
  {                           Ø 1
Buffer empty ← while (count==0);
    itemC = Buffer(out);
    Out = (out+1) mod n;
    Count = Count -1 ;
    Process_item(itemc);
  }
}
```



```
int Count = 0;
Void Producer (void)
{
    int itemp;
    while (true)
    {
        Produce_item(itemp);
        while (count==n);
        Buffer [in] = itemp;
        in = (in+1)
        Count = Count +1 ;
    }
}
```

itemp = x_1

Case 1 : x_1

(0+1) mod 8

$\boxed{\text{Count = count +1;}}$
↓
1. load Rp , m[count];
2. INCR Rp;
3. store m[count], Rp;

- Count gives the no: of items in the count
- itemC = x_1
- (0+1) mod 8

  1 mod 8 = 1

$\boxed{\text{Count = count - 1}}$
↓
1. load Rc, m[count]; Rc = 1
2. DECR Rc ; Rc = 0
3. Store m[count], Rc ; Count = 0.

- Since count = 0, there is no item in the buffer.
- Case 1 is the best case. No issues.

$\boxed{n = 8}$

Buffer [0, . . . n-1]

Case 2 :

Out $\boxed{0}$

	Buffer
0	x_1
1	x_2
2	x_3
3	x_4
4	
5	
6	
7	

In $\boxed{3}$ becomes In $\boxed{4}$

Count $\boxed{3}$

New item produced = x_4

ie itemp = 4

$\boxed{\text{Count = count + 1 ;}}$

↓

1. load Rp, m[count];
2. INCR Rp; Rp=4
3. store m[count], Rp;

In: This variable = next empty slot.

After executing instruction 2, producer process was pre empted.

∴ Consumer process will execute. itemC = x_1

out $\boxed{~~0~~ 1}$

	Buffer
0	
1	x_2
2	x_3
3	x_4
4	
5	
6	
7	

Count = count -1

↓

1. load Rc, m[count]

   Rc = 3

2. DECR Rc ;

   Rc = 2.

After executing instruction 2, consumer process was pre empted. ∴ producer process will execute (ie process will resume). 3rd instruction of producer will execute.

3. store m[count], Rp;

Count = 4.

After executing instruction 3, producer process will terminate. Then consumer process will resume. 3rd instruction of consumer will execute.

3. store m[count], Rc; count = 2 (∵ Rc = 2).

So, after executing instruction 3, consumer process will terminate.

Hence two processes will terminate.

In buffer, we have 3 items (x_1, x_2, x_3). But count = 2 ∴ count value is wrong.

Race Condition occurs. Process Synchronization doesn't happen.

Flow of [In case 2, Producer: I_1, I_2, Consumer: I_1, I_2, Producer I_3, Consumer I_3]

Critical section is a part of the program where shared resources are accessed by various cooperative processes.

Critical section is a place where shared resources, variables are placed.

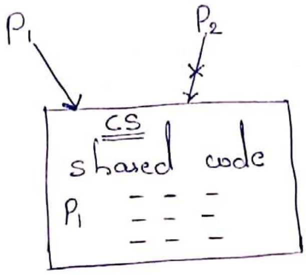
Synchronization mechanism:

4 conditions/Rules should satisfy.

mandatory {
1. Mutual Exclusion.
2. Progress

Secondary {
3. Bounded wait.
4. No assumption related to hardware, speed etc.

1. Mutual Exclusion.

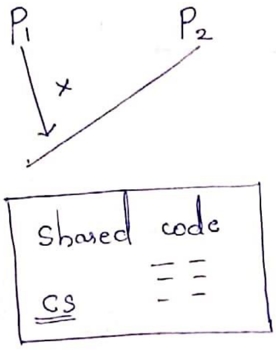


= mutual exclusion.

CS = critical section.

· vice versa can also happen.

2. Progress:



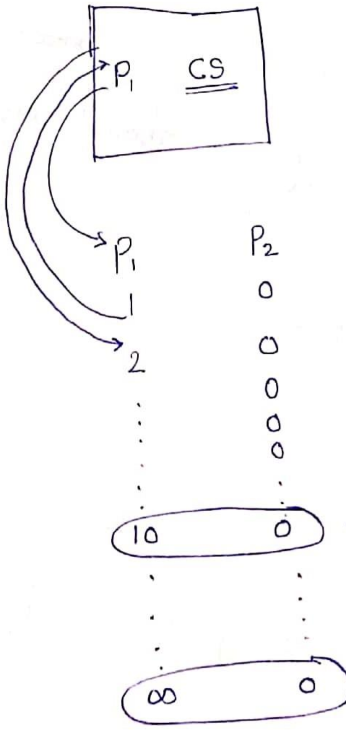
P1 is interested to enter critical section.
But P2 is blocking P1.

ie no progress.

∴ no process can execute.

vice versa also can happen.

3. Bounded Wait:



= Bounded wait

= unbounded wait for P2. Starvation problem for P2.

4. No assumption related to H/w, speed

Synchronization solution should not depend on hardware.

✓ ✗

eg: 32 bit, 64 bit

Solution must be universal.

Lock variable in OS.

Critical section Solution using Lock

```
do {
        acquire lock
        CS
        release lock
   }
```

Pseudo code

```
1. while (Lock==1);     Entry code
2. Lock =1
3. Critical section
4. Lock =0              Exit code
```

Lock=0 ⇒ CS is vaccant.
=1 ⇒ CS has some process.

- * Execute in user mode eg: application.
  ↳ no involvement of kernel.
- * multiprocesses solution.
- * No mutual exclusion guarantee.

Case 1

P₁ P₂ . (Lock==0)

Inst: 1 Lock==1

2

3. CS

4. Lock=0

1 ← inst.

2.

3. CS

4.

Case 2.

Lock=0

P₁	P₂
Inst. 1.	
P₁ preempted. Lock=0	inst: 1
	2 Lock=1 ⇒ Lock ~~0~~ 1
	3. CS
2. Lock = ~~0~~ 1	
3. CS	

So two processes P₁, P₂ were in Critical section. So no mutual exclusion is guaranteed.

* Lock variable does not guarantee mutual exclusion if the two processes are preempted.

Critical Section solution using 'Test_and_Set' instruction

```
while (test_and_set (& lock));

    CS
lock = false
```

```
boolean test_and_set (boolean *target)
 {
     boolean r = *target;
     * target = TRUE;
     return r;

 }
```

```
1. while (Lock ==1) ;     Entry
2. Lock =1                code
3. Critical Section
4. Lock = 0               Exit code
```

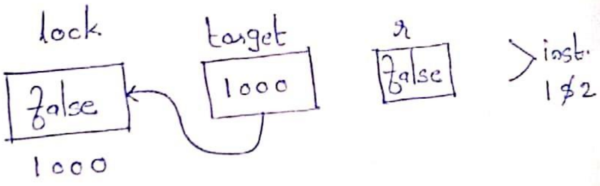
- - - - - -

Lock = 0 = false.

P₁. (1)

boolean test_and_set (boolean * target)

{



True

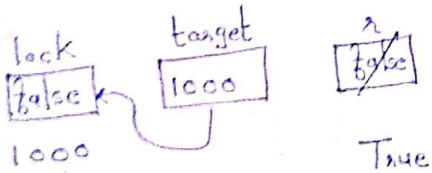
return r ; r = false

}.

P₂.

boolean test_and_set (boolean * target)

{



True (1).

~~False~~ True (2)

return r; r = true

} So P₂ doesn't CS. infinite loop.

{ P₁ is in CS. At that time P₂ is interested to access CS. }

So mutual exclusion acquired. Progress also acquired.

Turn Variable (strict Alteration method):

- * 2 process solution.
- * Runs in user mode. No need of kernel support.

Pseudo code:

	Process P_0	Process P_1
	No CS	
Entry code ⇒	while (turn!=0);	while (turn!=1);
	Critical Section	Critical section
Exit code ⇒	turn =1;	turn = 0;

Case 1: int turn = 0 (initial).

```
P0 :  while (turn != 0);  false
        Critical Section      ← P0 is in CS
      turn 1;
```

```
P1
while (turn != 1);   ← true ∴ P1 will be infinite loop. P1 doesn't get CS access.
   Critical Section
turn = 0;
```

⇒ P_0 in CS. At that time P_1 is interested to come into CS.

Case 2: Let int turn =1 (initially)

So P_1 will have $\boxed{CS}$ access. P_0 doesn't get $\boxed{CS}$ access.

So mutual exclusion acquired.

Case 3 : int turn = 0;

P_1 wishes to acquire $\boxed{CS}$. But P_1 enters into infinite loop. P_1 doesn't $\boxed{CS}$ access.

But P_0 get $\boxed{CS}$ access. After exiting from $\boxed{CS}$, P_0 will make turn = 1. Then P_1 can access $\boxed{CS}$.

So mutual exclusion acquired.

Assume there is no process in $\boxed{CS}$ initially. So depending on the values of turn, one process can block another process from acquiring $\boxed{CS}$. So no progress.

Bounded weight acquired. Since turn value always changes when a process exits from $\boxed{CS}$. ∴ turn by turn $\boxed{CS}$ is acquired.

This code is platform independent ie H/w independent.

Semaphore:

Semaphore is an integer variable which is used in mutual exclusive manner by various concurrent cooperative processes in order to achieve synchronization.

Semaphore is of two types 1. Counting ($-\infty$ to $+\infty$)
2. Binary (0,1)

Counting Semaphore

Assume P_1, P_2, P_3 are cooperative processes.

Entry code

$\boxed{CS \; \vdots}$

Exit code.

P( ), Down, wait ← Entry code

V( ), up, signal, post, Release ← Exit code.

Entry

```
Down (Semaphore s)
{
    S.Value = S.Value - 1 ;
    if (S.Value < 0)
    {
        Put Process (PCB) in
        Suspended list sleep ( );
    }
    else
        return;
}
```

CS ⇒ CS is vaccant.

P₁: CS ; S = 3 ; S = semaphore

S = 2 ⇒ S.value ≠ 0 ⇒ CS (P₁)

P₂: CS (P₁) ; S = 1 ⇒ S.Value ≠ 0 ; CS (P₁ P₂)

P₃: CS (P₁ P₂) ; S = 0 ⇒ S.Value ≠ 0 ; CS (P₁ P₂ P₃)

P₄: CS (P₁ P₂ P₃) ; S = -1 ⇒ S.Value < 0. CS (P₁ P₂ P₃)

P₄, P₅ ⇒ Block list = sleep.

S = -2

- If S = -4 ⇒ 4 processes are in block state / suspended.
- if S = 0 ⇒ further no other processes can enter into CS.

Exit

```
Up (Semaphore s)
{
    S.Value = S.Value +1 ;
    if (S.Value <= 0)
    {
        Select a process
        from Suspended list and
        Wake up ( );
    }
}
```

P₁ wishes to exit from CS. ∴ run exit code.

S = -1

Block state → wake up (Ready queue) → Running.

∴ P₄ can try for CS.

P₂ also wishes to exit from CS

∴ run exit code.

S = 0

∴ P₅ can try for CS. Hence in Ready queue we have P₄, P₅.

P_3 wishes to exit from CS. ∴ run exit code. Hence $\boxed{1}$.

- if S = 0 => no process is in the suspended list.
- if S = 10 => 10 processes can be in the critical section i.e. 10 successful operations can be performed. If a process is blocked, then it is an unsuccessful operation.

1. Assume S = 10. Perform 6P operations & 4V operations. What will be the final value of semaphore?

ans: S = 10.

6P operations = 10 − 6 ⇒ $\underline{S = 4}$.

4V operations = 4 + 4 ⇒ $\underline{S = 8}$.

2. Let S = 17. Perform 5P, 3V & 1P. Final value of S?

Ans: S = 17.

5P ⇒ 17 − 5 = $\underline{12 = S}$.

3V => 12 + 3 => S = 15.

1P => 15 − 1 => $\underline{S = 14}$

Counting Semaphore is rarely used.

Binary Semaphore : Two values 0 & 1.

```
Down (Semaphore s)
{
    if (s.value = =1)
    {
        S.Value = 0;
    }
    else
    {
        Block this process
        & place in suspend
        list, sleep ();
    }
}
```

```
Up (Semaphore s)
{
    if (Suspend List is Empty)
    {
        S.Value = 1 ;
    }
    else
    {
        select a process from
        suspend list & wake up();
    }
}
```

Down, P, wait
Up, V, Signal.

- Let S=1. Down operation performs & S becomes 0. It is a successful operation.
- Let S=0. Down operation performs & S =0 only. It is an unsuccessful operation.
- Let S=0. Up operation. S becomes 1. Assume suspend list is empty. If suspend list is not empty, then value of S=0 as such. It selects a process from suspend list & put it in Ready state i.e wake up().

- S=1. Assume suspend list is empty. The new value S=1 itself.
- S=1. Assume suspend list is not empty. Now value S=1 itself. It select a process from suspend list & wake that process

eg: Asssume two processes P₁ & P₂.

```
P₁            P₂

Down (s)      Down (s)

| CS |        | CS |

up (s)        up (s)
```

- P₁ & P₂ are co operative processes.

Initially S = 0.

P₁ comes first. P₁ blocked. Then control moves to P₂. Since S = 0, P₂ will also be in blocked state. Deadlock situati

So assume S=1 initially:

Assume P₂ comes first. S becomes 0. P₂ is in Critical section. Then P₁ comes. Since S=0, P₁ is in blocked state.

Mutual exclusion occurs.

P₂ wishes to come out of CS. Up operation should perform. It checks the suspended list. Suspended list is not empty. P₁ is in suspended list. So it puts P₁ in ready queue.

Solution of Producer Consumer problems

Problems:

1. inconsistency.
2. loss of data.

Synchronizing producer consumer problem using binary semaphore & counting semaphore.

```
Produce_item (item P):
Entry code   1.  down (Empty);
             2.  down (s);
             ┌─────────────────────────┐
             │ Buffer [IN] = itemp;    │
CS ←         │ In = (In+1) mod n;      │
             └─────────────────────────┘
Exit code    3.  up (s);
             4.  up (full);
```

IN=3

out=0

N=8

0	a
1	b
2	c
3	
4	
5	
6	
7	

```
Consumer
1. down (full);
2. down (s);
     ┌──────────────────────────┐
     │ itemC = Buffer [out];    │
CS ← │ Out = (out +1) mod n;    │
     └──────────────────────────┘
3. up (s)
4. up (Empty);
```

Counting semaphore → full = 0 = no. of filled slots.
→ Empty = N = no. of empty slots.

Binary semaphore = 1

Initally Empty = 5

Full = 3

Producer .

Producer produces an item ;

1. down (Empty); => Empty = $\cancel{5}$ 4.

2. down (S); => S = $\cancel{1}$ 0.

CS .

itemp = d.

Since IN=3

∴ d is placed in location 3.

N=8

0	a
1	b
2	c
3	d
4	
5	
6	
7	

IN = (IN+1) mod n

= 4 mod 8

IN = 4

3. up (S) ; => S = $\cancel{1}$ $\cancel{0}$ 1.

4. up (Full); => Full = $\cancel{3}$ 4.

Consumer

1. down (Full); Full = $\cancel{3}$ $\cancel{4}$ 3.

2. down (S); S = $\cancel{1}$ $\cancel{0}$ $\cancel{1}$ 0.

CS .

item C = a ∵ out = 0.

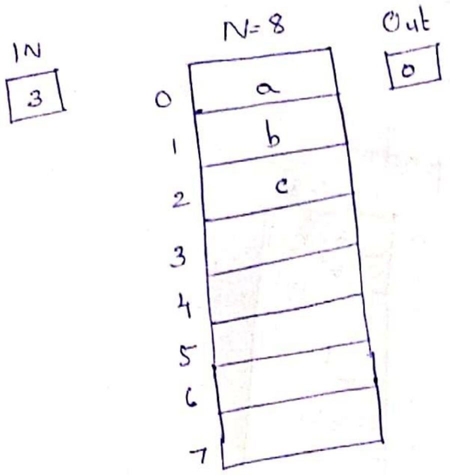
out = (out +1) mod n; => 0 + 1 = 1 mod 8 = 1.

3. up (S); S = $\cancel{1}$ $\cancel{0}$ $\cancel{1}$ $\cancel{0}$ 1.

4. up (Empty); Empty = $\cancel{5}$ $\cancel{4}$ 5.

Here we are assuming no preemption .

Preemption.



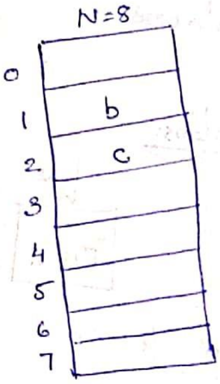
Assume Empty=5, full=3.
S=1

Producer:

1. down(Empty); ~~5~~ 4 = Empty.
Producer preempted.

Consumer

1. down(full); full = ~~3~~ 2.
2. down(s); s = ~~1~~ 0



* consumer enters into $\boxed{CS}$ *

item C = Buffer[out] ; item C = a

out = (out+1) mod n; out = (0+1)mod 8
= 1

Producer cannot enter into $\boxed{CS}$.

3. up(s); s = ~~1~~ ~~0~~ 1.

4. up(Empty); Empty = ~~5~~ ~~4~~ 5.

Consumer process was terminated. Since consumer was terminated, ∴ control goes to producer process.

Producer: Producer's PCB has information that instruction 1 was executed. So control moves to instruction 2.

2. down (s) ; $s = \cancel{1}\ \cancel{0}\ \cancel{1}\ 0$ → Processor was successfully executed

CS.

Buffer [IN] = itemp ; itemp = d

$In = (In + 1) \bmod n$;

$= (\overset{3}{\wedge} + 1) \bmod 8$

$= 4 \bmod 8 = \underline{\underline{4}}$

N = 8

0	
1	b
2	c
3	d
4	
5	
6	
7	

3. up(s) ; $s = \cancel{1}\ \cancel{0}\ \cancel{1}\ \cancel{0}\ 1$

4. up (full) ; $full = \cancel{1}\ \cancel{2}\ 3$.

Total slots = Empty + full = $5 + 3 = \underline{\underline{8}}$.

Reader Writer Problem:

- Database

Scenario:

Same data in data base.

Total 4 problems are there

R — W → problem = Read write problem

W — R → problem = write read "

W — W → problem = write write "

R — R → No problem. = any no: of readers can read but no problem.

- inconsistency
- loss of data.

To synchronize Reader & Writer. we are using Semaphore.

rc = Read count

= no. of Readers in buffer

$\boxed{DB}$ = critical section

```
int rc = 0;
                    Binary Semaphore mutex = 1;
                    Binary Semaphore db = 1;
Void Reader (void)
{
    while (true)
    {
```

Reader's Entry code

```
        down (mutex)
        rc = rc+1 ;
        if (rc==1) then down (db);
        up (mutex)
```

$\boxed{DB}$

Reader's Exit code

```
        down (mutex)
        rc = rc -1 ;
        if (rc==0) then up(db);
        up (mutex)
        Process_data
    }
}
Void Writer (void)
{
    while (true)
    {
```

Writer's Entry code → down(db);

$\boxed{DB}$

Writer's Exit code → up(db);

```
    }
}
```

Case 1 . (R-W problem).

Reader R1 comes first.

rc = 0 ; mutex = 1 ; db = 1

Entry code:

down (mutex) ; mutex = ~~1~~ 0

rc = rc + 1 ; rc = ~~0~~ 1

if (rc == 1) then down (db); db = ~~1~~ 0

up (mutex) ; mutex = ~~1~~ ~~0~~ 1.

Reader enters into DB. ie Reader has successful operation.

* Writer wishes to enter, this time.

down (db) ; db = ~~1~~ 0 ⇒ ∵ it is a binary semaphore 0-1=-1
blocked.

not possible . So writer's problem is

Case 1 is solved.

Case 2 : (W-R problem).

Writer's entry code.

1. down (db); db = ~~1~~ 0

Writer process enter into Critical Section. Successful operation.

Reader wishes to enter.

Reader's entry code

down (mutex) ; mutex = ~~1~~ 0.

rc = rc + 1 ; rc = ~~0~~ 1

if (rc == 1) then down db; db = ~~1~~ 0 ⇒ it is a binary semaphore.
So Reader is blocked.

Case 3 : W-W problem:

Writer (W_1)

Entry code:

down(db); db = $\cancel{1}$ 0

W_1 enters into Data base. ie successful operation.

Writer (W_2) wishes to enter:

Entry code:

down(db); db = $\cancel{1}$ 0. ie 0-1=-1 is not possible ∴ W_2 is in block state.

Case 4: R-R : No problem

Reader (R_1):

Entry code:

down(mutex); mutex = $\cancel{1}$ 0

rc = rc+1; rc = $\cancel{0}$ 1

if (rc==1) then down(db); db = $\cancel{1}$ 0

up(mutex); mutex = $\cancel{1}$ $\cancel{0}$ 1.

$\boxed{DB\ R_1}$ → R_1 enters into data base (DB). ie R_1 is in critical section now. Reader R_2 wishes to enter into critical section.

Reader (R_2):

Entry code:

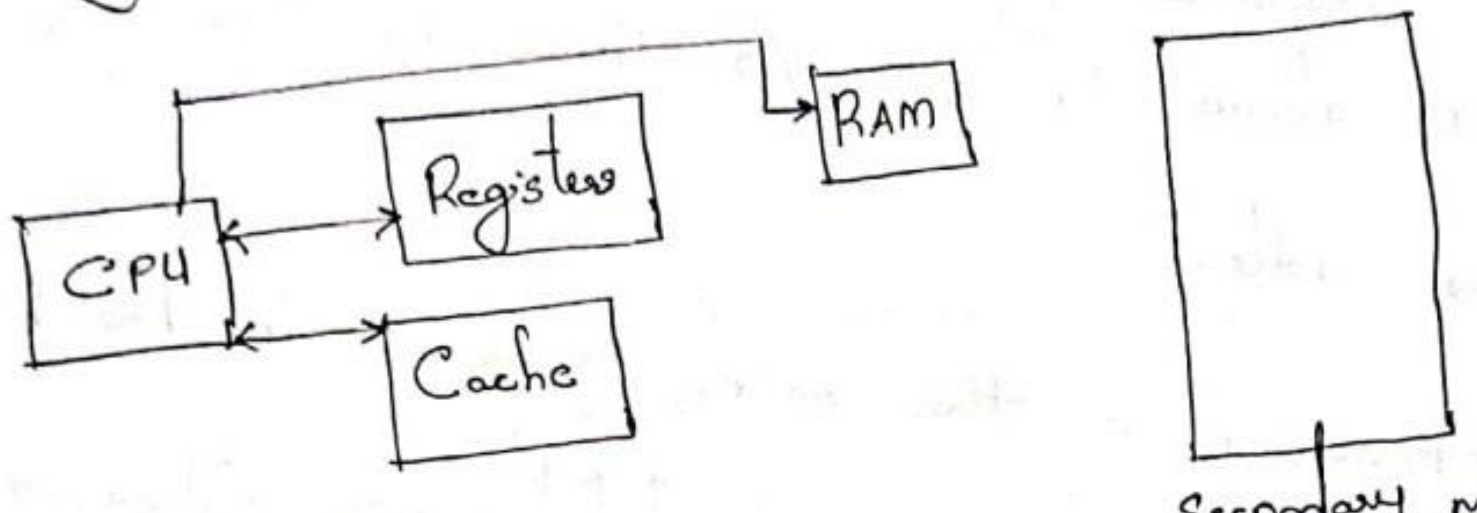
down(mutex): $\cancel{1}$ $\cancel{0}$ $\cancel{1}$ 0

rc = rc+1; rc = $\cancel{0}$ $\cancel{1}$ 2

if (rc==1) then down(db); rc ≠1 ∴

up(mutex); mutex = $\cancel{1}$ $\cancel{0}$ $\cancel{1}$ $\cancel{0}$ 1 ∴ $\boxed{DB\ R_1,R_2}$

<u>Memory Management:</u> method of managing the primary memory.



Secondary memory.

<u>Greater speed</u>

1. Registers 2. Cache

3. RAM

CPU is directly connected to Registers, Cache & RAM.

All programs are stored in secondary memory. For the pgms. to execute, programs should be loaded to RAM.

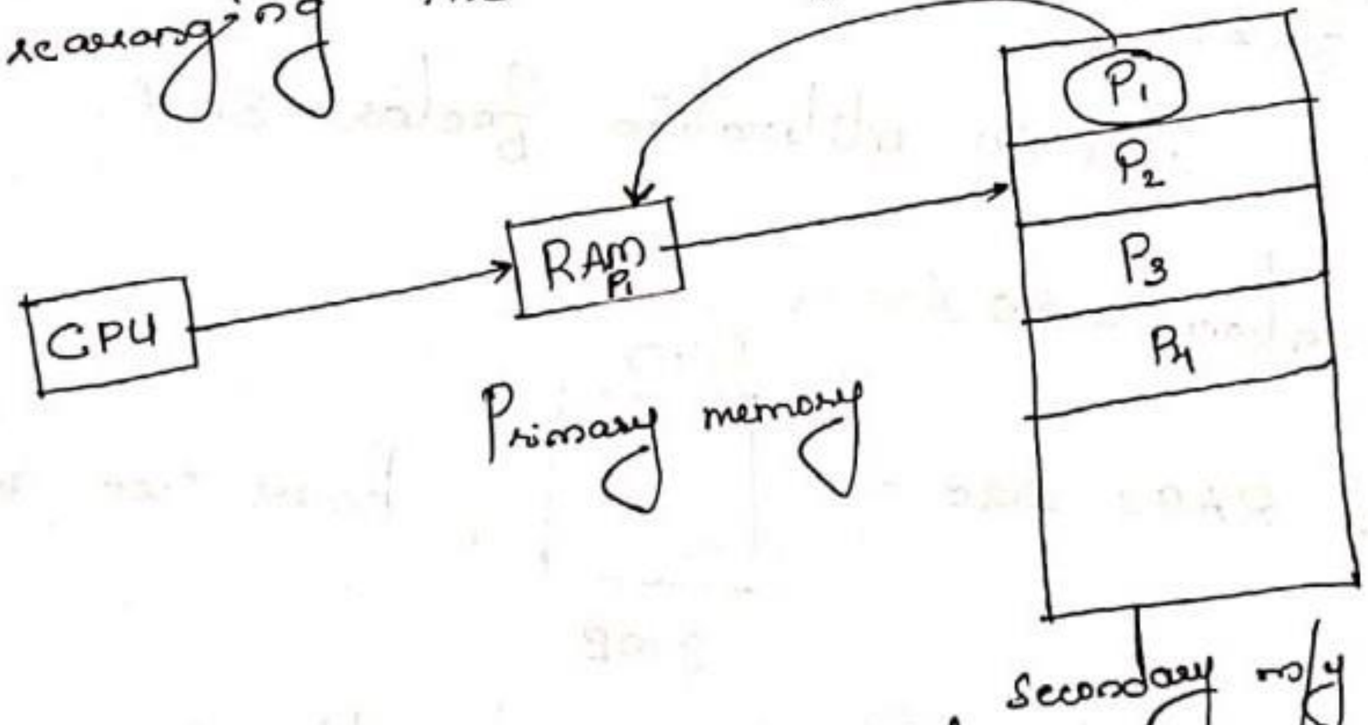
<u>Speed:</u>

Registers & Cache (kb) >> RAM (Gb).

CPU is not connected to secondary memory.

MODULE 4

So rearranging the above figure.

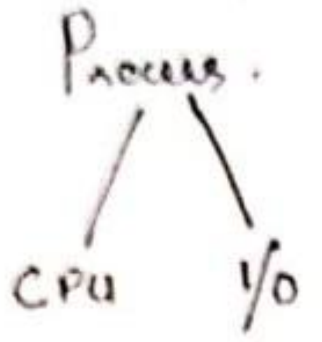


P_1, P_2, P_3, P_4 = programs

Now CPU can directly execute with the P_1. Similary all pgms. This is called the

<u>Degree of multiprogramming</u>

Multiprogramming means the execution of multiple programs. ie multiple pgms in primary memory (RAM). This will increase <u>cpu utilization factor.</u>



P_1 is executed by CPU. P_1 (process) request for I/O. So CPU allows P_1 for I/O. Hence P_1 has I/O. During this time CPU is idle.

Degree of multiprogramming is the no: of programs in the RAM (main memory). As Degree of multiprogramming ↑↑↑ ⇒ CPU utilization factor ↑↑ ⇒ System efficiency ↑↑↑.

1. RAM size is 4MB, process size 4MB. How many processes can be in the RAM?

$\Rightarrow \frac{4}{4} = 1.$

$\boxed{P_1}$. k = time for I/O operation by process (P_1). So CPU will execute P_1 for $(1-k)$ time
RAM

$\therefore$ CPU utilization factor $= (1-k)$.

Assume $k = 70\%$. $\therefore$ CPU utilization factor $= 30\%$

ie CPU utilization $= 30\%$

Increasing the RAM size = RAM □ 8MB, Process size is □ 4MB

No: of processes that can come into the RAM $= \underline{\underline{2}} = \left(\frac{8\text{MB}}{4\text{MB}}\right)$

Assume one process perform k amount of time for I/O operation. Since two processes are here, $\therefore$ k^2 time will be for I/O operation.

∴ CPU utilization will be = $(1-k^2)$

Let $k = 70\% = \frac{70}{100} = 0.7$.

∴ CPU utilization factor = $1-(0.7)^2 = 1 - .49 = .51 = \underline{\underline{51\%}}$

Increase RAM size ie 16MB. Process = 4MB.

No: of processes in RAM = $\frac{16}{4} = 4$.

If 'k' is the amount of time, 1 process is doing in the I/o operation.

Assume all 4 processes are doing the I/o operations at the same time, ie k^4!

∴ CPU utilization factor = $1-(k)^4$

Let k = 70% ∴ CPU utilization factor = $1-(0.7)^4 = \underline{76\%}$

$$\boxed{\text{Size of RAM} \uparrow\uparrow \Rightarrow \text{CPU utilization factor} \uparrow\uparrow}$$

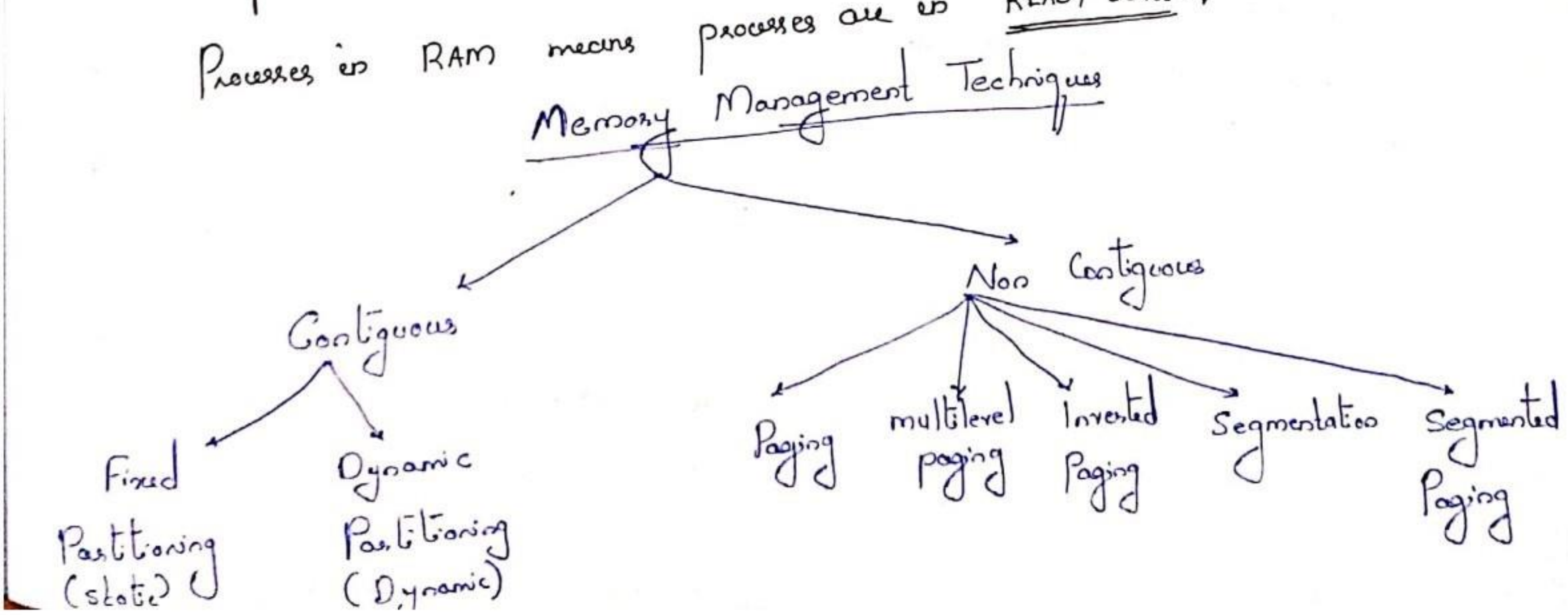
* Memory management of RAM (1° m/y) is more important.

* Memory management is ~~under~~ done by Operating System. Paging & segmentation is also done by O.S

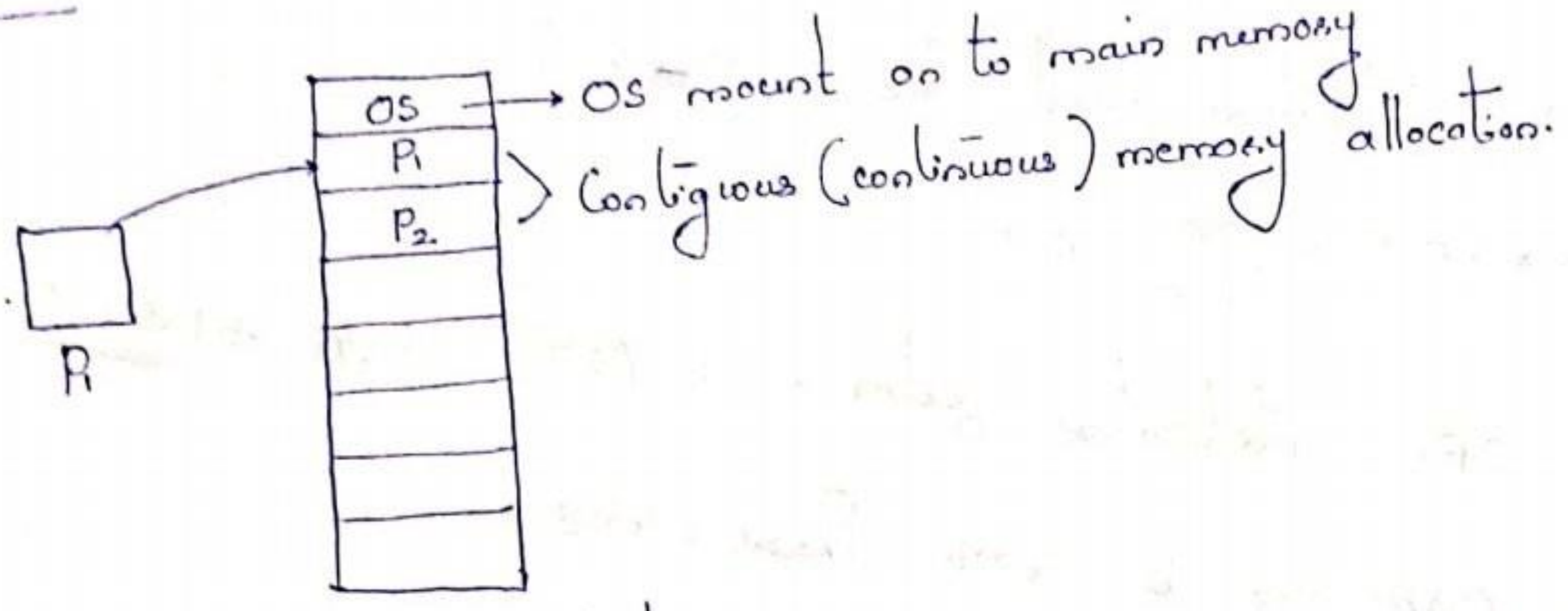
Memory management Techniques

Keep more no: of processes in the primary memory/RAM.

Processes in RAM means processes are in READY state.

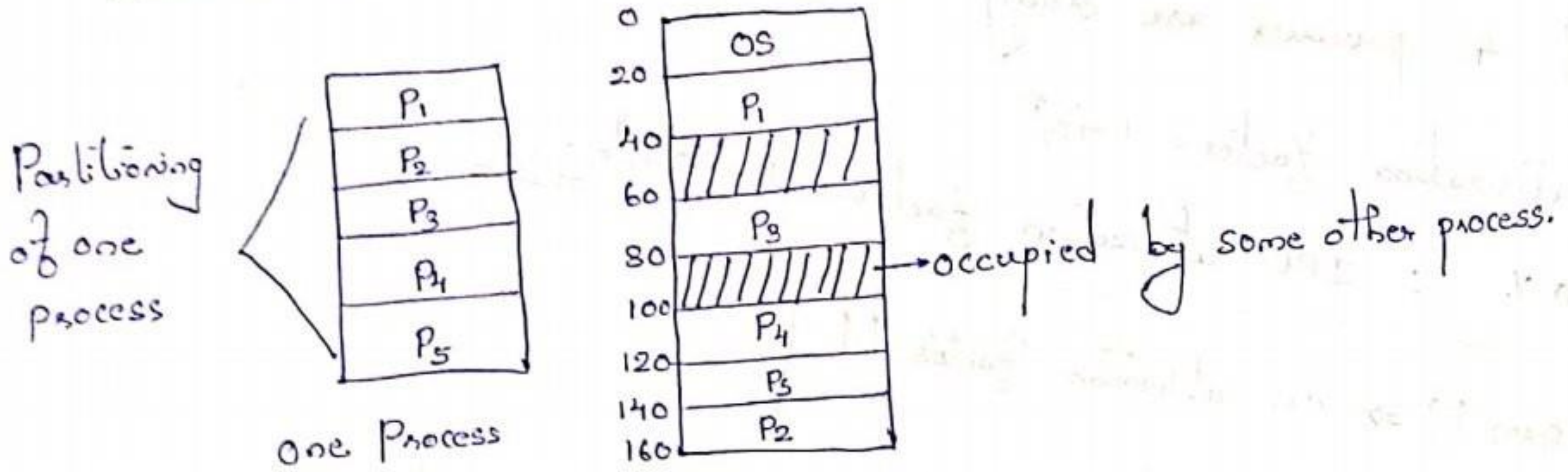


Contiguous


OS
P1
P2
P1
OS mount on to main memory
Contiguous (continuous) memory allocation.

Memory blocks
(RAM)

Non Contiguous

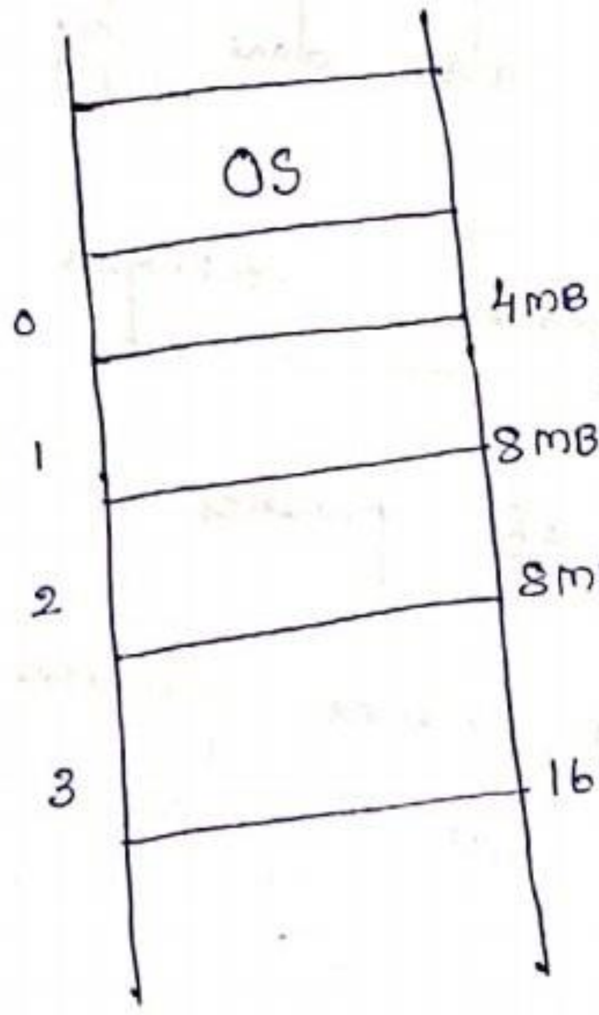


One Process

memory blocks
(RAM)

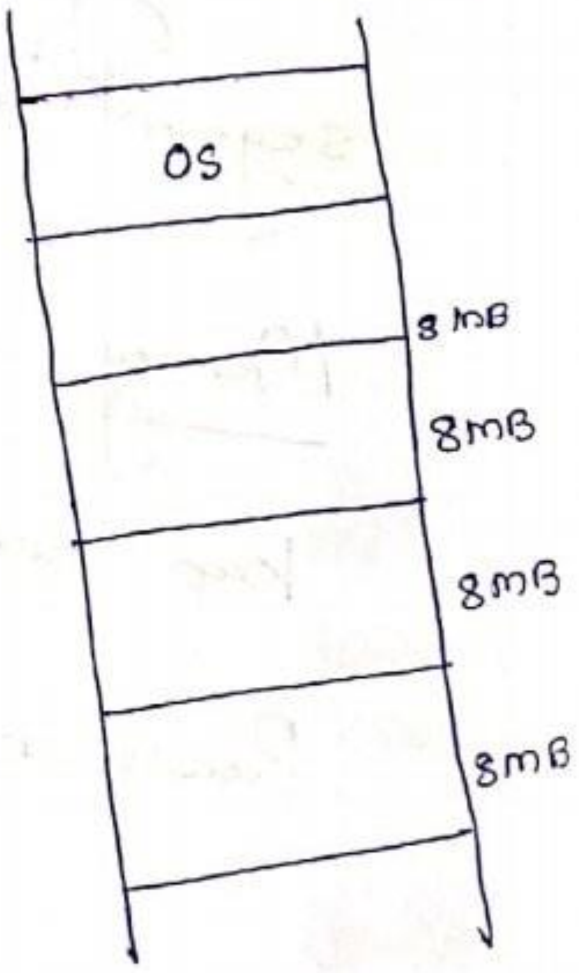
Fixed Partitioning (static Partition)

- No. of Partitions are fixed
- Size of each partition may or may not same.
- Continuous allocation so spanning is not allowed.



RAM

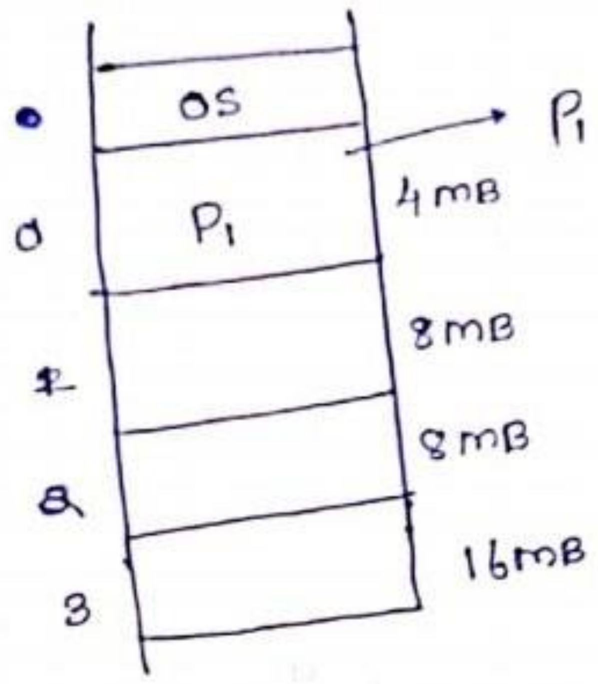
No: of Partitions
= 4



RAM

No: of Partitions
= 4.

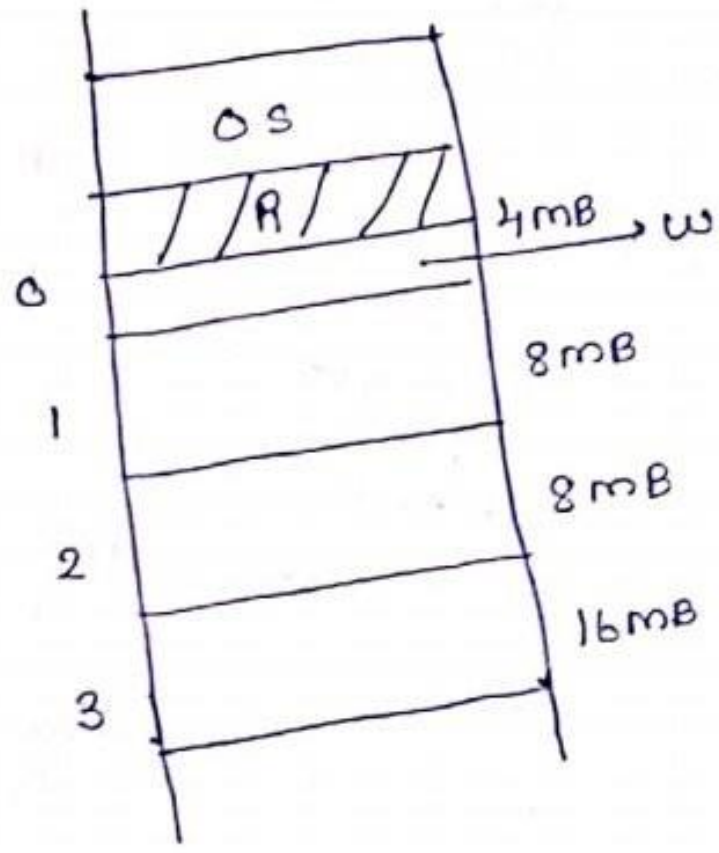
eg: Assume a process $P_1 = 4MB$ arrives to execute.



→ P_1 exactly fits in the partition.

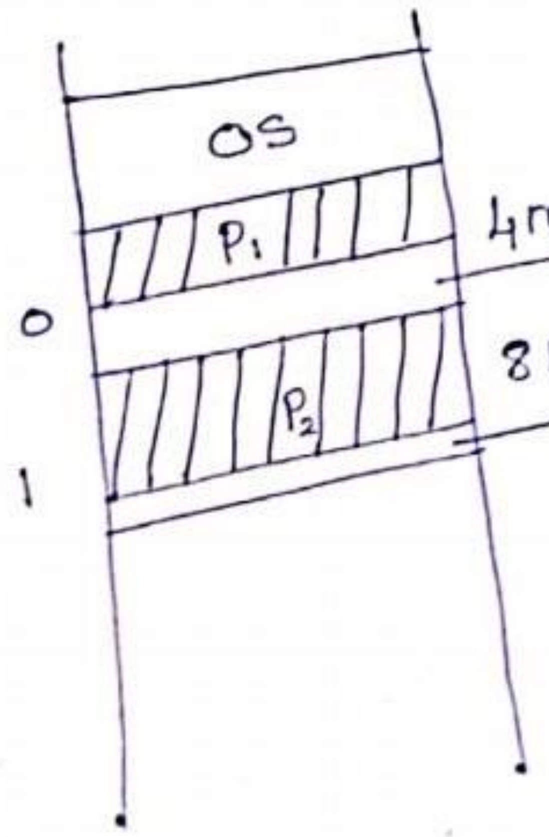
Best Case

$P_1 = 2MB$



→ waste = Internal Fragmentation

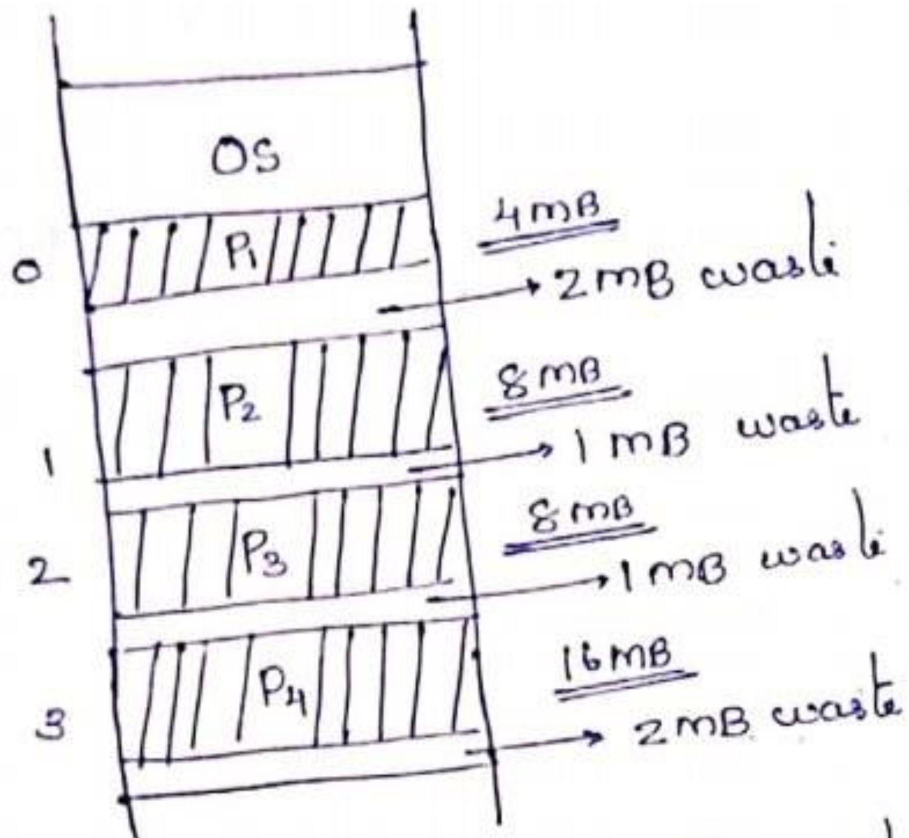
$P_2 = 7MB$



→ 2MB waste. }
→ 1MB waste. } ① Internal Fragmentation.

(2) Limit in process size: Let $P_3 = 32MB$. But P_3 cannot be accomodated in the RAM. B'coz max^m. size of partitioned memory in RAM is 16MB.

(3) Limitation on Degree of multiprogramming: Assume 6 processes are arrived. RAM can store only 4 processes, b'coz RAM was partitioned into 4. Remaining 2 processes cannot be accomodated



$P_1 = 2MB$.

$P_2 = 7MB$, $P_3 = 7MB$, $P_4 = 14MB$

Assume P_5 process came,
$P_5 = 5MB$

But here 6MB space is available. Since it is a contiguous allocation. This is known as (4) External Fragmentation. Whenever there is Internal Fragmentation in memory, External Fragmentation exists.

Fixed Partitioning was used in 1960's in Mainframes.

Advantages

1. Easy to implement

Disadvantages

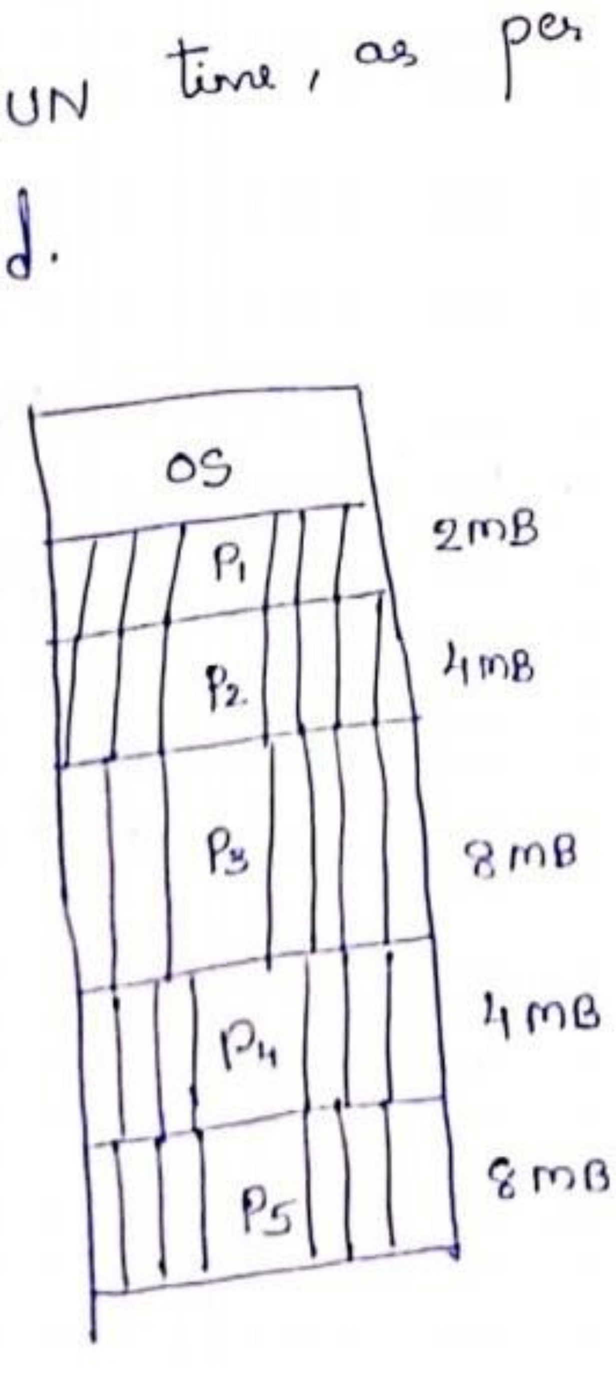
1. Internal Fragmentation
2. Limit in Process size.
3. Limitation on Degree of multiprogramming
4. External Fragmentation.

Variable Partitioning / Dynamic Partitioning.

Here RAM will partition as per process size.
In Fixed partitioning RAM was partitioned before process arrives.
→ So during RUN time, as per process requirement memory (RAM) will be allocated.

eg: $P_1 = 2MB$

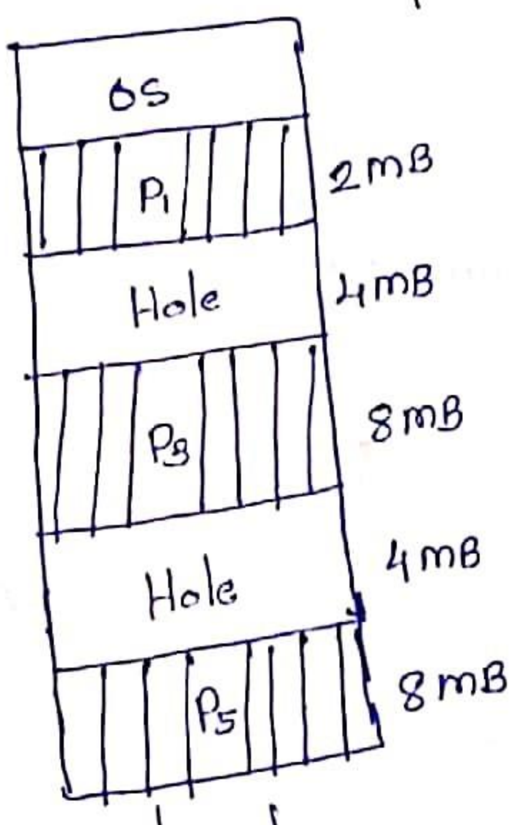
$P_2 = 4MB$, $P_3 = 8MB$



Advantages.

1. No Internal Fragmentation.
2. No limitation on no: of processes.
3. No limitation on the process size.

Assume P2,P4 were terminated. Let a new process P6 arrives ie P6=8MB.

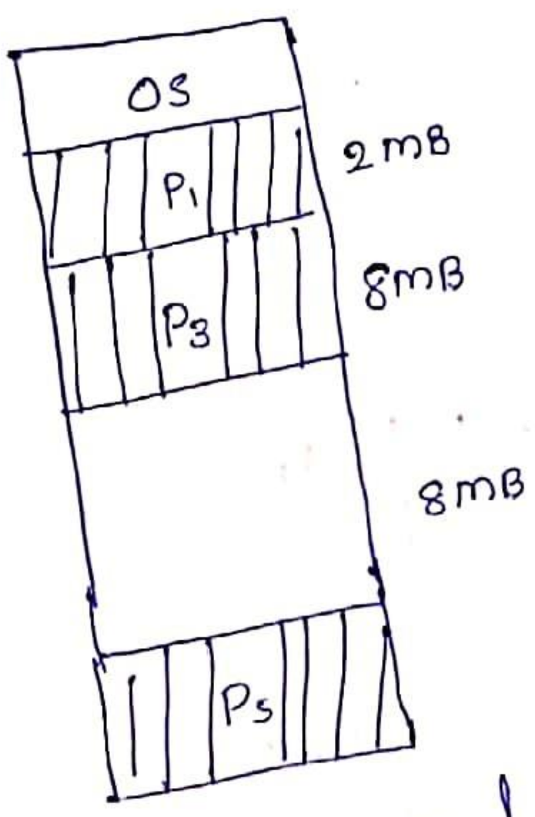


Since Dynamic partitioning comes under Contiguous memory management technique, P6 cannot be added to RAM.
∴ P6 has to wait.

This is known as

① External Fragmentation. → Disadvantage

To solve External Fragmentation, a new method known as Compaction can be used ie



But Compaction is an undesirable method, b'coz a running process has to be stopped. → Disadvantage.

(2) Allocation/Deallocation is complex: Since in variable partitioning more no: of processes arrives which increases more holes. ∴ allocation/deallocation of processes will be complex.

Contiguous Memory allocation algorithms:

To manage the holes, 4 algorithms are used. They are First fit, Next fit, Best fit and worst fit.

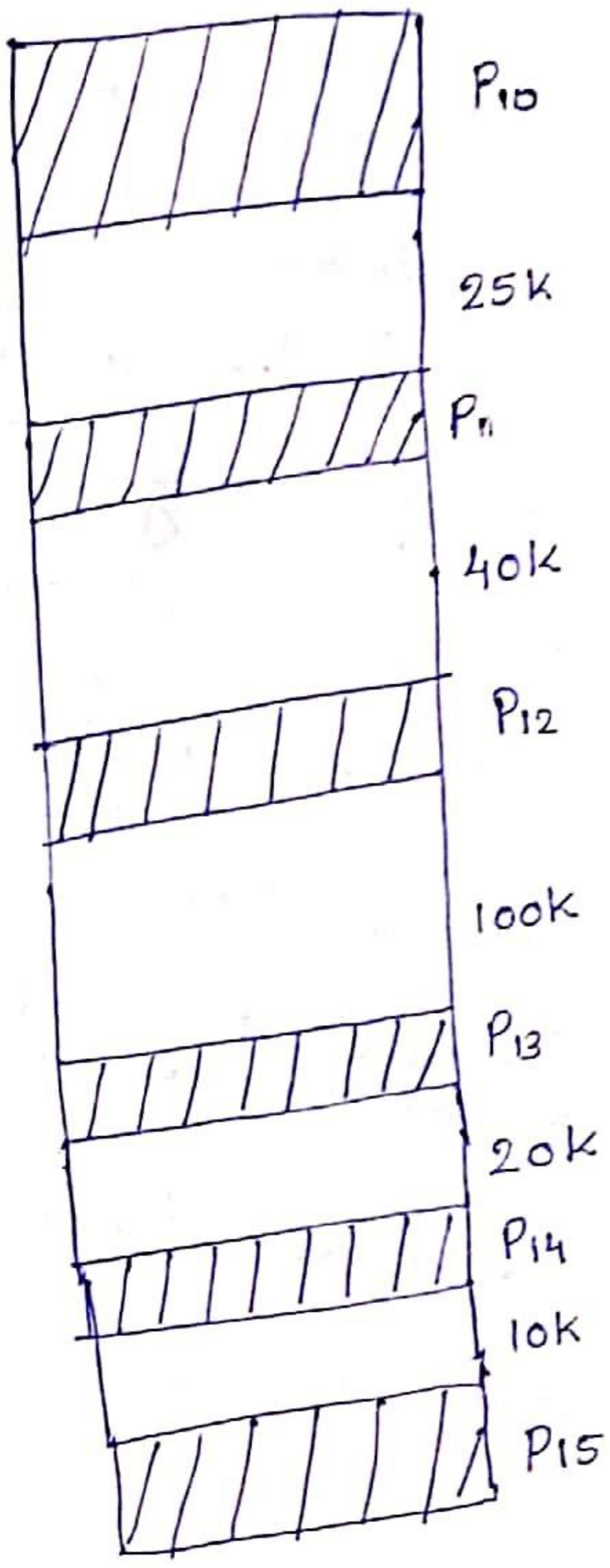
First fit: Allocate the first hole ie big enough.

Next fit: Same as first fit but start search always from last allocated hole.

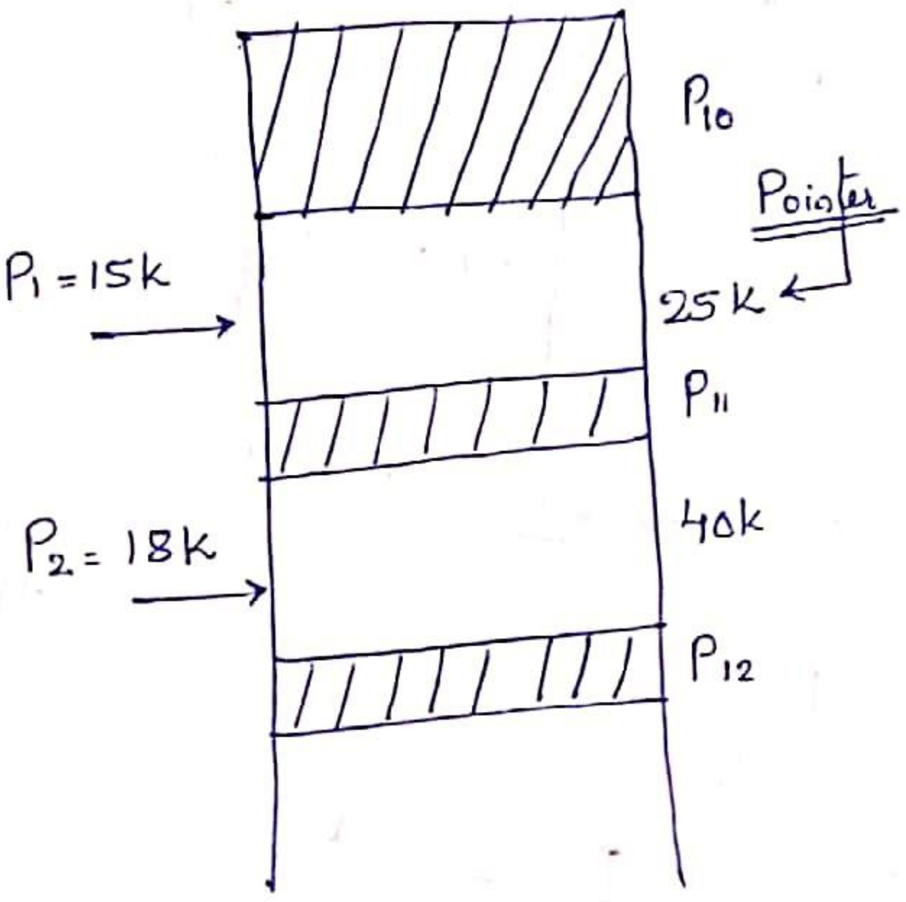
Best fit: Allocate the smallest hole ie big enough. Less internal fragmentation. It will search for the entire hole list.

First Fit: easiest algorithm. Less searching.

eg: Let $P_1 = 15k$

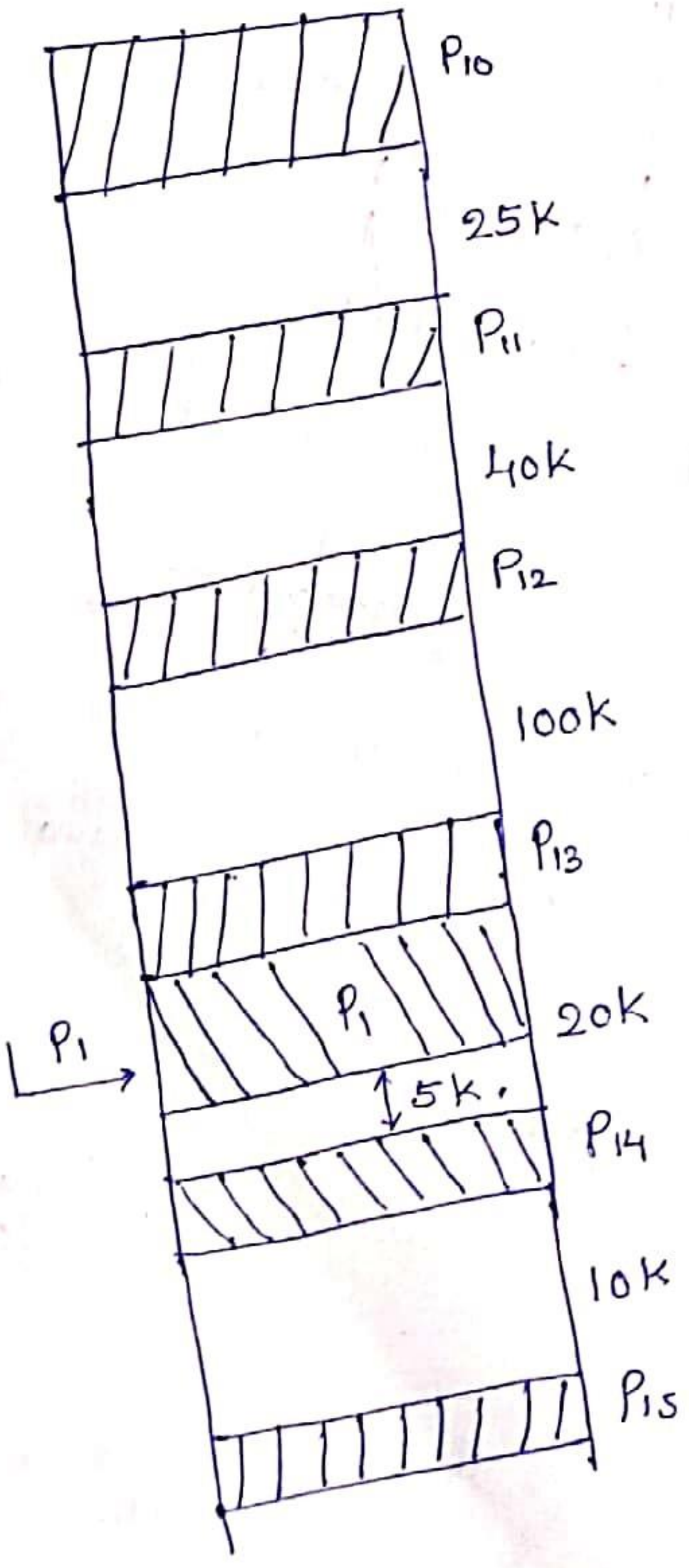


Next fit: Uses the pointer.



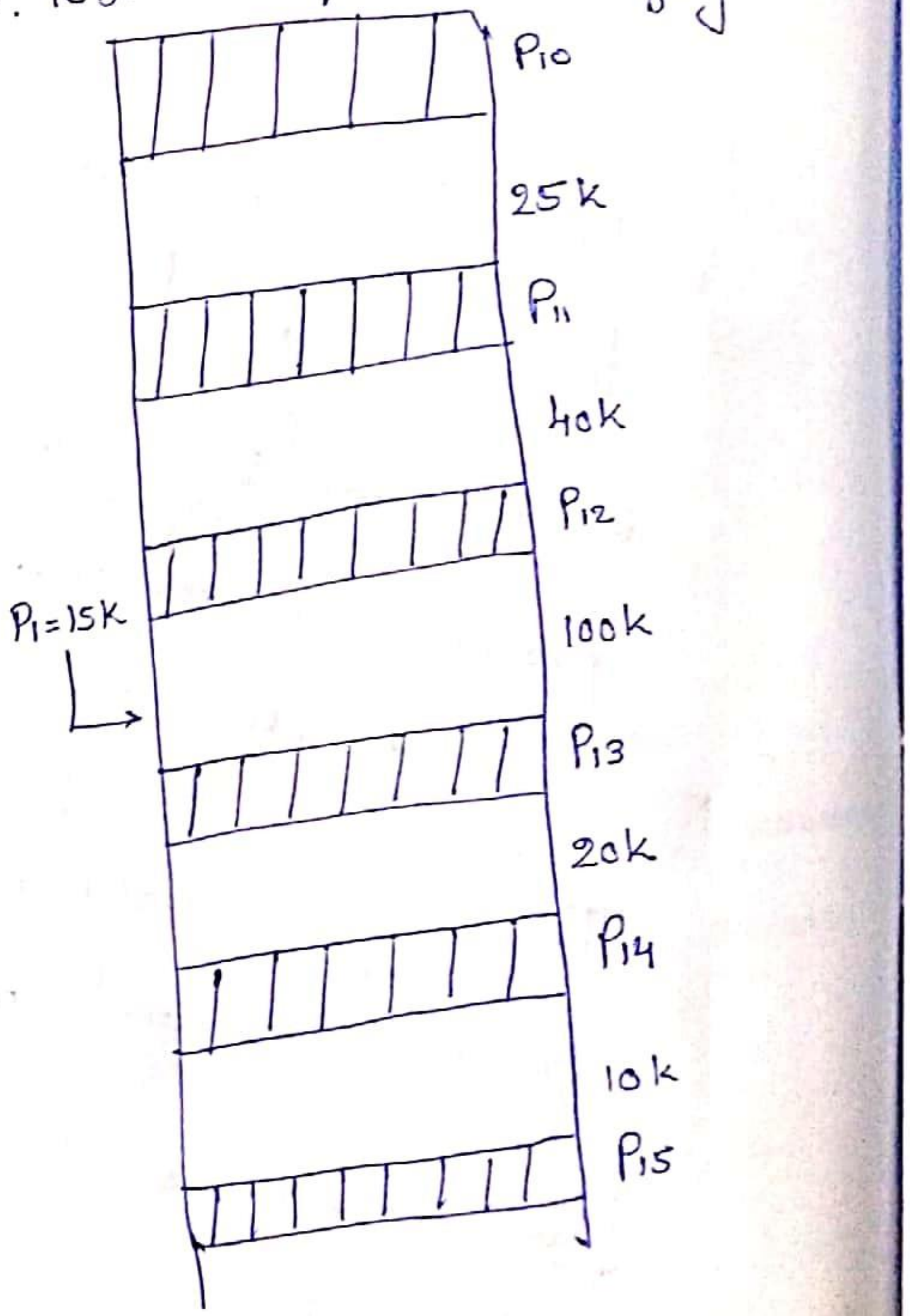
Best fit:

$P_1 = 15k$



Worst fit: allocate the largest hole. Opposite of Best fit. Let $P_1 = 15K$

Max^m. leftover m/y. Largest internal fragmentation



Comparison.

First fit

- Simple
- fast
- It can create big holes.

Next fit:

- fast. It doesn't start from the start.

Best fit

- Very less internal fragmentation
- It creates very small holes. mostly new processes cannot be added to these small holes
- It has to search the entire list of holes. ∴ it is very ~~slow~~ slow.

Worst fit.

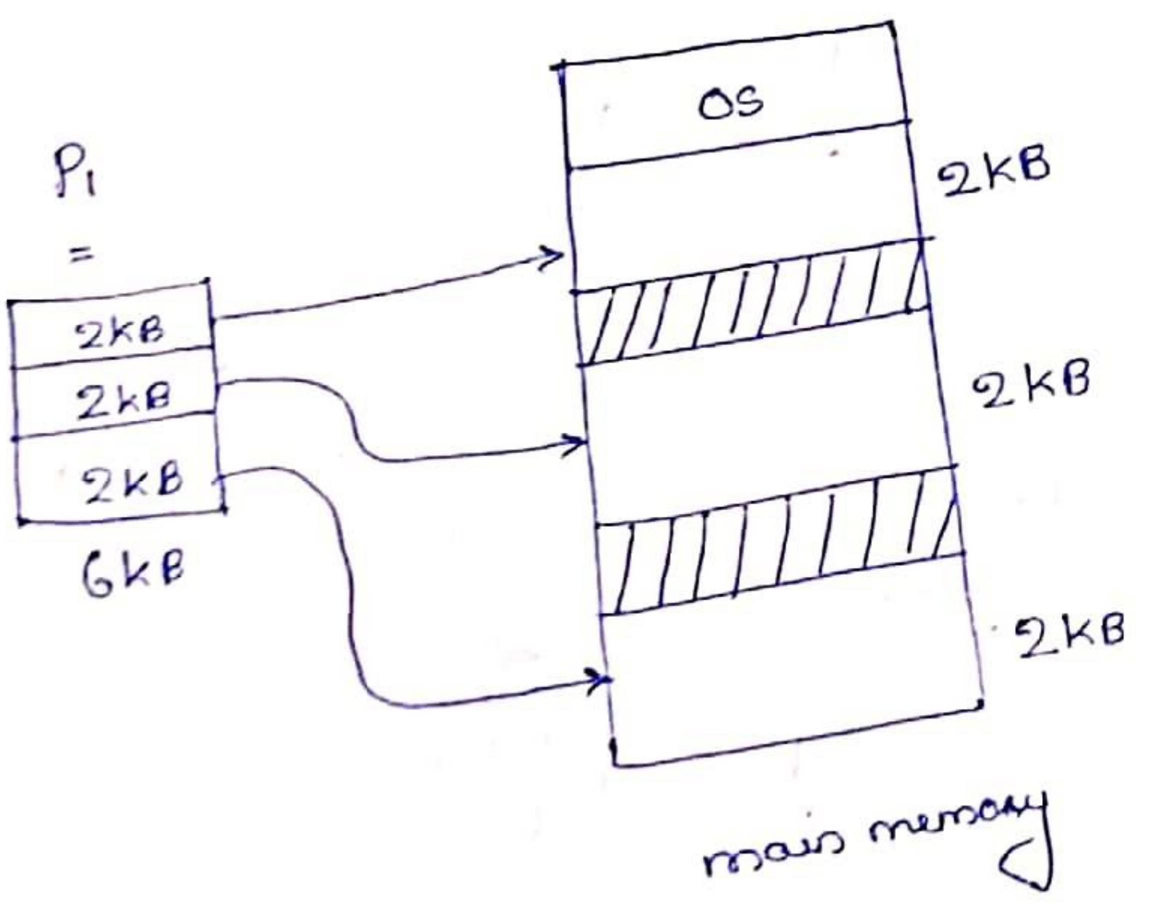
- Since the leftover space is much larger, more processes can be added to the leftover processes.
- It is slow ∵ it has to search for the entire list.
- Large internal fragmentation.

Next fit is the modified version of first fit.

So while comparing first fit, Best fit & worst fit, the best one in real life is first fit ∵ it ~~has~~ is fast. ie it requires less time.

Need of Paging: (Non Contiguous memory allocation).

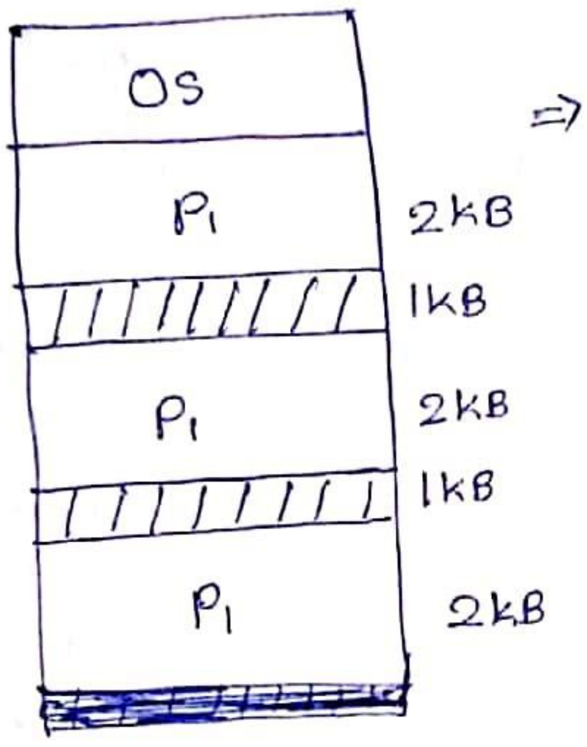
Let P_1 =



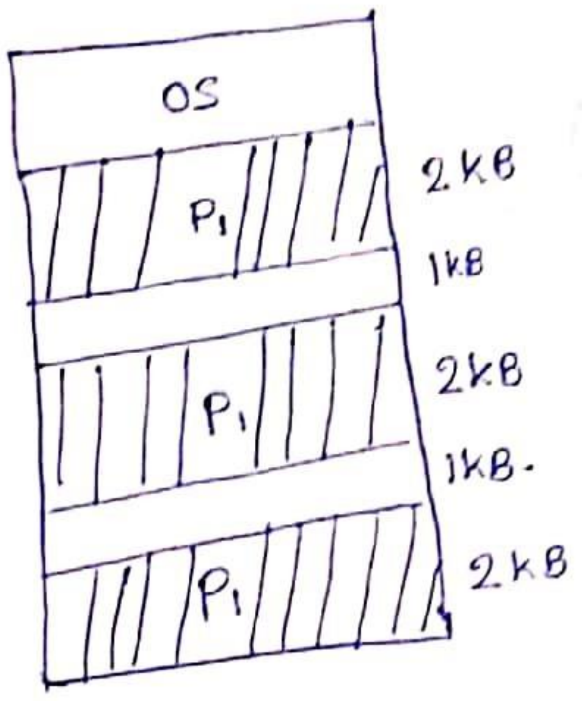
. External Fragmentation will not occur in Non Contiguous memory allocation

According to the size of holes, the ready to be process has to be divided.

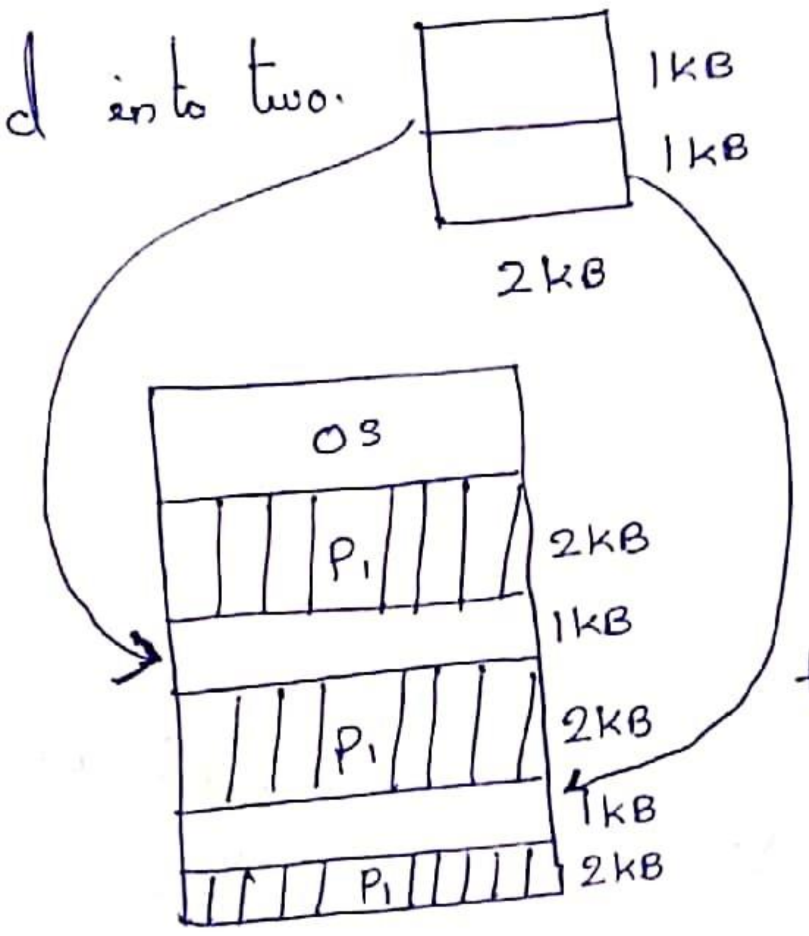
⇒



⇒ After some time,

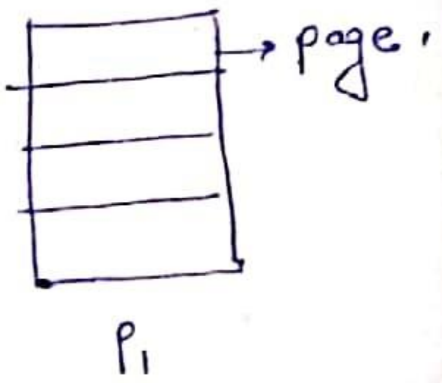


Let P_2 come, $P_2 = 2$ kB. Available hole size = 1kB + 1kB = 2kB. So P_2 is divided into two.

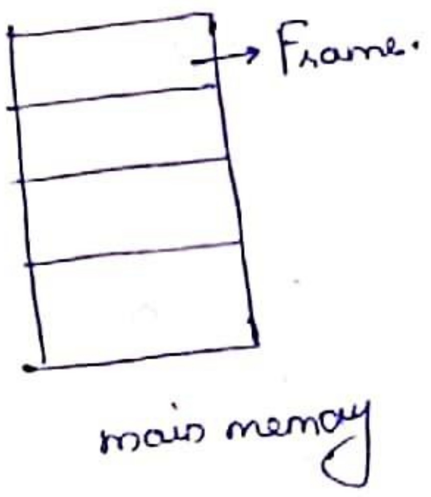


Holes are dynamically created. So it is time consuming to analyze the no: of size of holes. Hence before assigning to main memory, processes are divided. This division of process is known as Pages.

This partition of process happens in secondary memory.

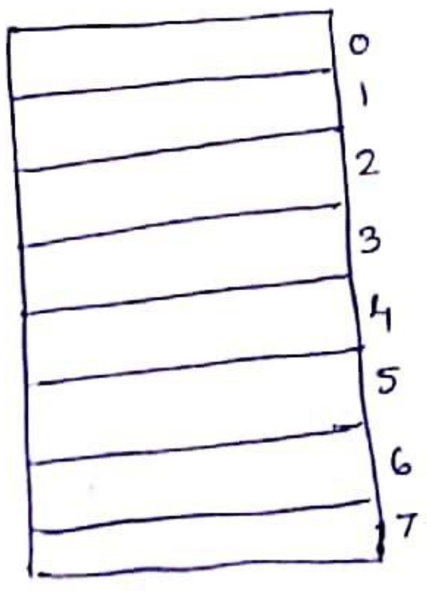


Similarly in main memory/RAM, also, the entire memory is divided into different Frames



Page size = Frame Size

eg:

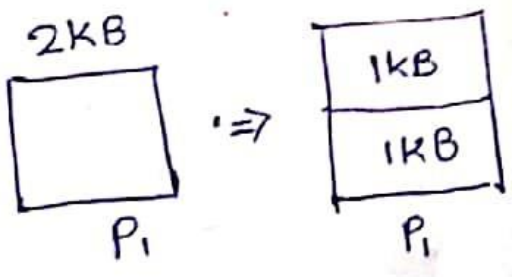


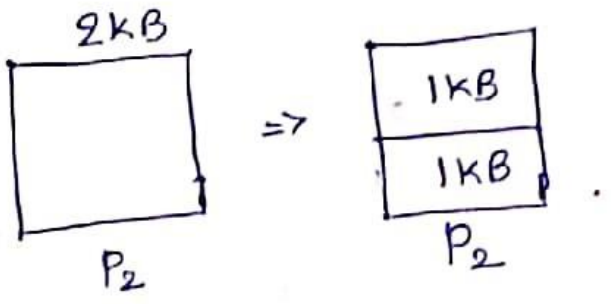
RAM = 8 KB

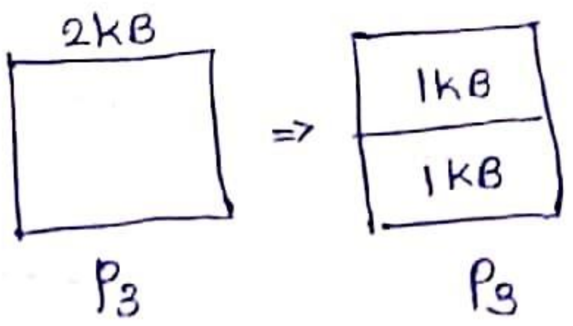
Frame Size = 1KB

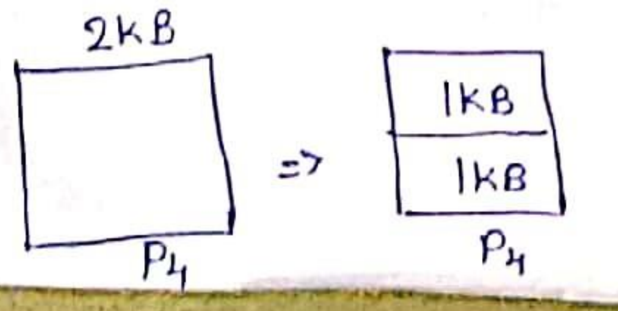
Total no: of frames = 8

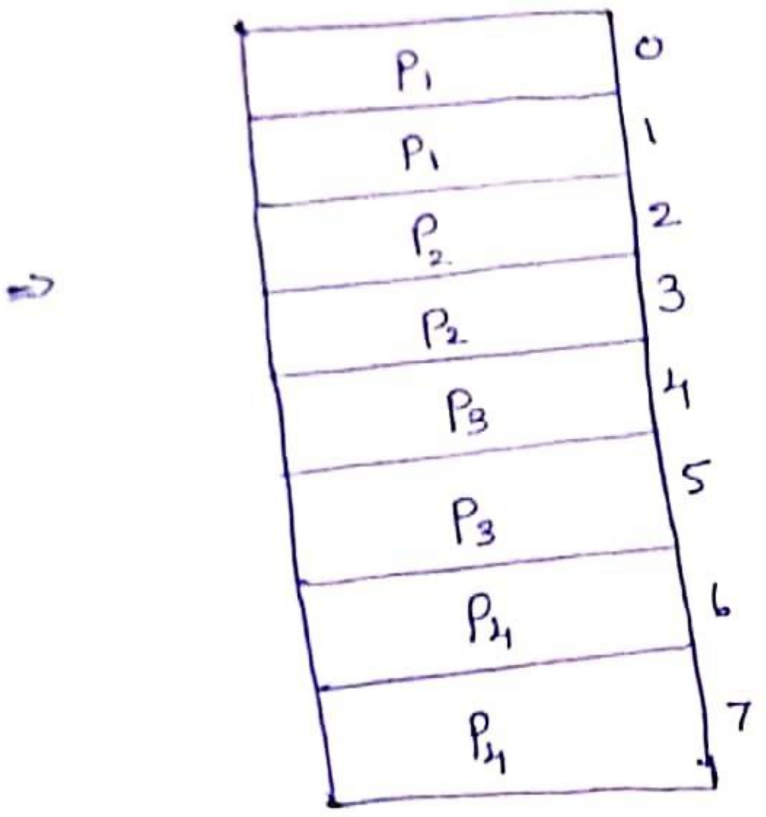
Let a process P_1 comes,





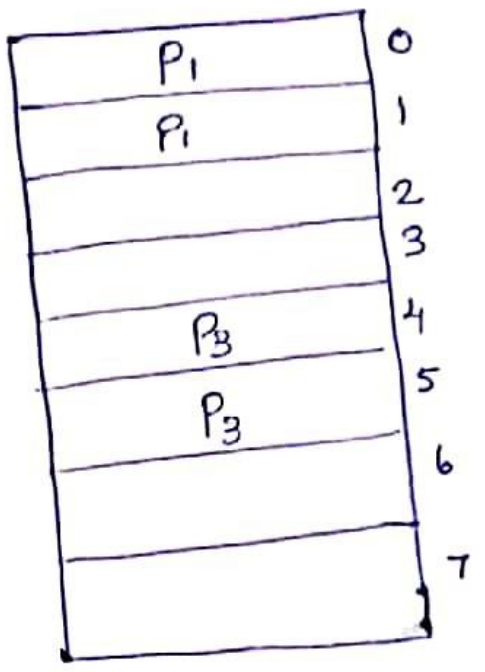


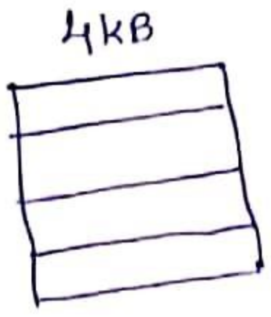


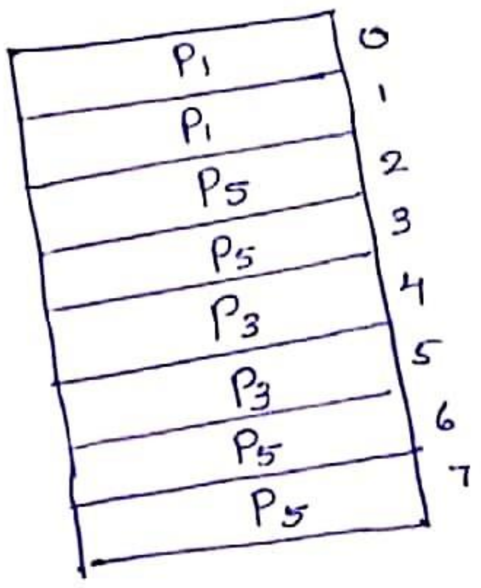
⇒ 

- Assume P2 & P4 are terminated ∴

∴ holes size = 4KB.



Let a new process come, P5 = 

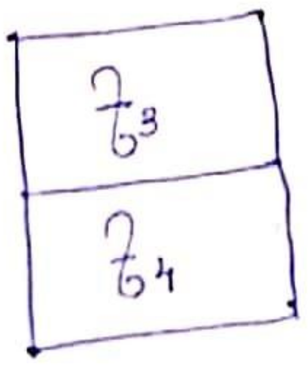
=7 

Here no External Fragmentation. This is why we need paging.

Segmentation:

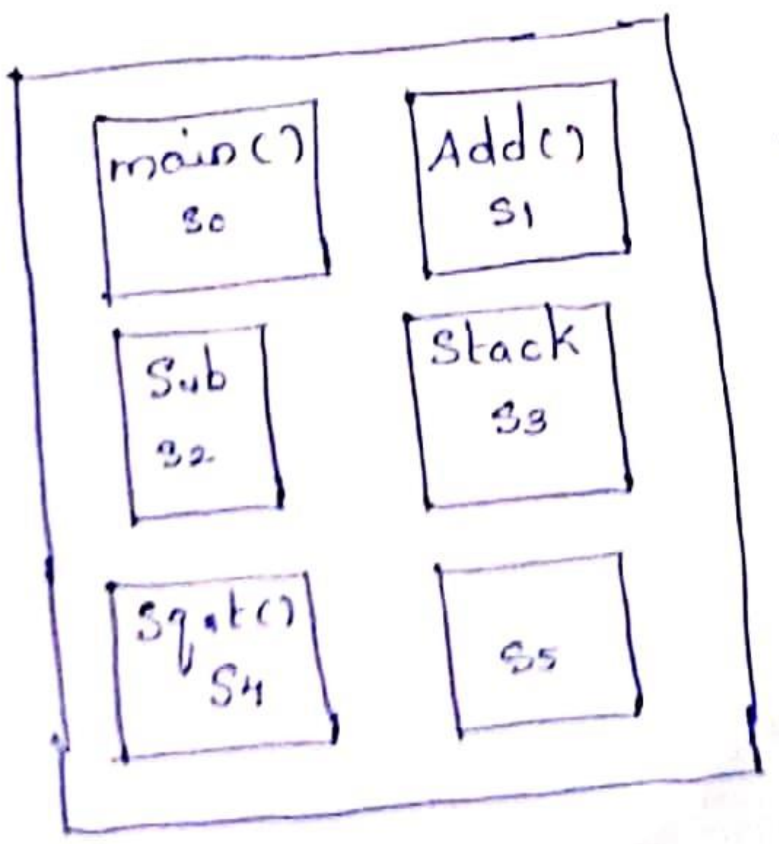
Without knowing what's inside the code, it is divided into fixed size pages.

eg: Add()



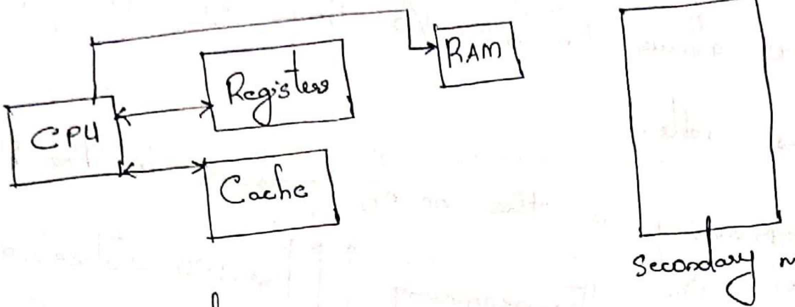
- ~~Some~~ One half of add() function is in f3 (frame 3) & the next half is in f4. For the CPU to execute add(), it needs the code in f3 & f4. f3 & f4 are different frames that means they are stored in separate memory locations in RAM. ie Page fault occurs in paging.

But Segmentation works on user point of view.



- Segments are created from program.
- Segments are of various sizes.

Memory Management: method of managing the primary memory.



Greater speed

1. Registers 2. Cache

3. RAM

CPU is directly connected to Registers, Cache & RAM.

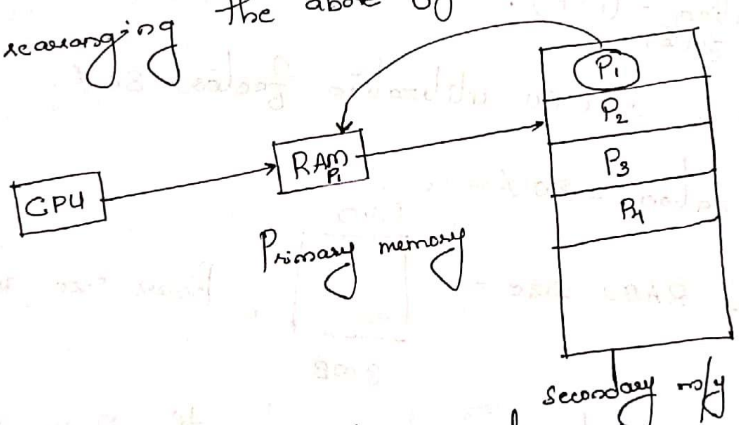
All programs are stored in secondary memory. For the pgms. to execute, programs should be loaded to RAM.

Speed:

Registers & Cache (kb) >> RAM (Gb).

CPU is not connected to secondary memory.

So rearranging the above figure.

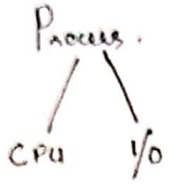


P_1, P_2, P_3, P_4 = programs

Now CPU can directly execute with the P_1. Similary all pgms. This is called the

Degree of multiprogramming

Multiprogramming means the execution of multiple programs. ie multiple pgms in primary memory (RAM). This will increase CPU utilization factor.



P_1 is executed by CPU. P_1 (process) request for I/O. So CPU allows P_1 for I/O. Hence P_1 has I/O. During this time CPU is idle.

Degree of multiprogramming is the no: of programs in the RAM (main memory). As Degree of multiprogramming ↑↑↑ ⇒ CPU utilization factor ↑↑ ⇒ System efficiency ↑↑↑.

1. RAM size is 4MB, process size 4MB. How many processes can be in the RAM?

$\Rightarrow \frac{4}{4} = 1.$

$\boxed{P_1}$. k = time for I/O operation by process (P_1). So CPU will execute P_1 for $(1-k)$ time

RAM

$\therefore$ CPU utilization factor $= (1-k)$.

Assume $k = 70\%$. $\therefore$ CPU utilization factor $= 30\%$

ie CPU utilization $= 30\%$

Increasing the RAM size = RAM $\boxed{\ \cdot\ }$ 8MB , Process size is $\boxed{\ \ }$ 4MB

No: of processes that can come into the RAM $= \underline{\underline{2}} = \left(\frac{8\text{MB}}{4\text{MB}}\right)$

Assume one process perform k amount of time for I/O operation. Since two processes are here, $\therefore$ k^2 time will be for I/O operation.

∴ CPU utilization will be = $(1-k^2)$

Let $k = 70\% = \frac{70}{100} = 0.7$.

∴ CPU utilization factor = $1-(0.7)^2 = 1-.49 = .51 = \underline{51\%}$

Increase RAM size ie 16MB. Process = 4MB.

No: of processes in RAM = $\frac{16}{4} = 4$.

If 'k' is the amount of time, 1 process is doing in the I/o operation.

Assume all 4 processes are doing the I/o operations at the same time, ie k^4!

∴ CPU utilization factor = $1-(k)^4$

Let k=70%. ∴ CPU utilization factor = $1-(0.7)^4 = \underline{76\%}$

$$\boxed{\text{Size of RAM} \uparrow\uparrow \Rightarrow \text{CPU utilization factor} \uparrow\uparrow}$$

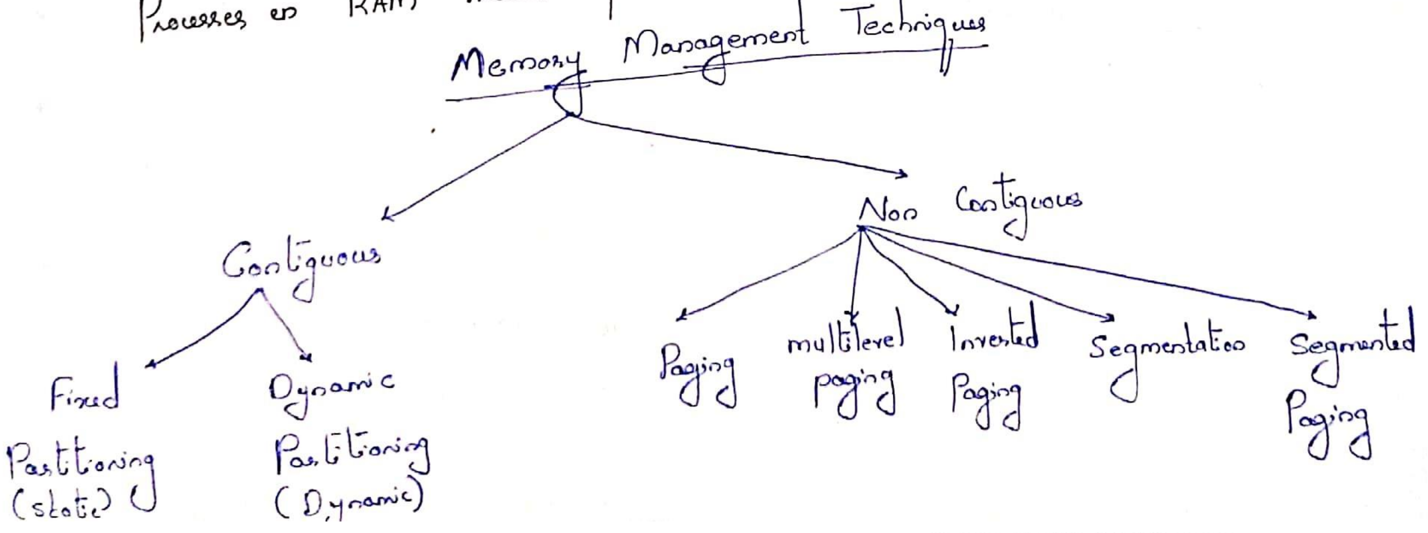
* Memory management of RAM (1° m/y) is more important.

* Memory management is ~~under~~ done by Operating System. Paging & segmentation is also done by O.S

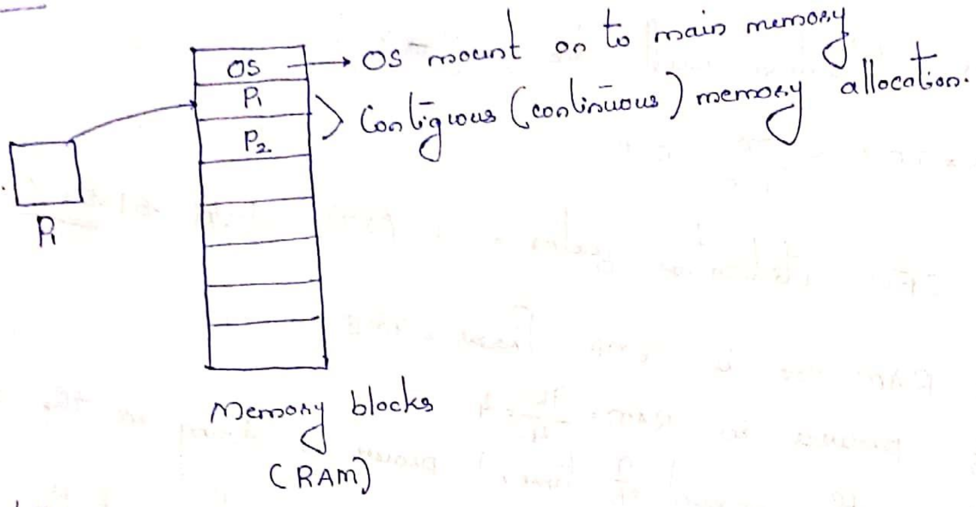
Memory management Techniques

Keep more no: of processes in the primary memory/RAM.

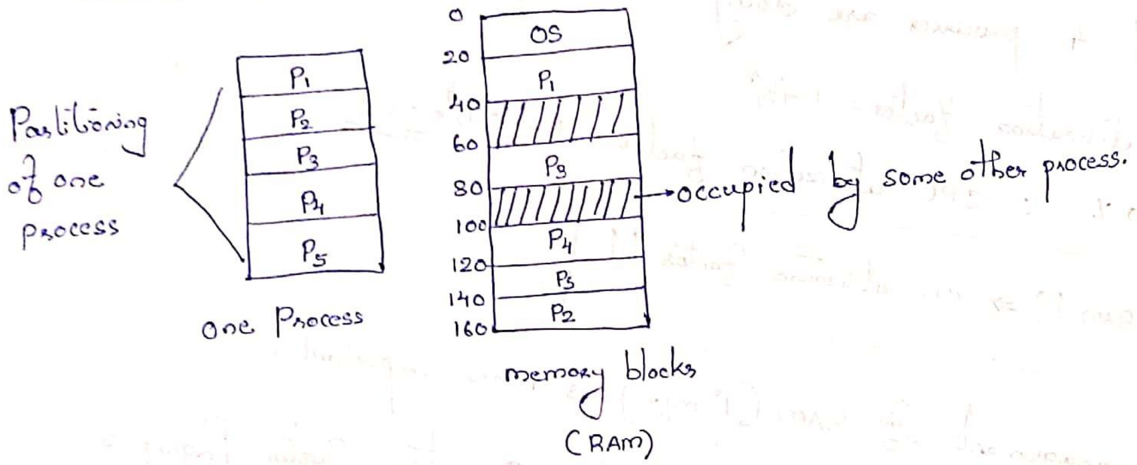
Processes in RAM means processes are in READY state.



Contiguous

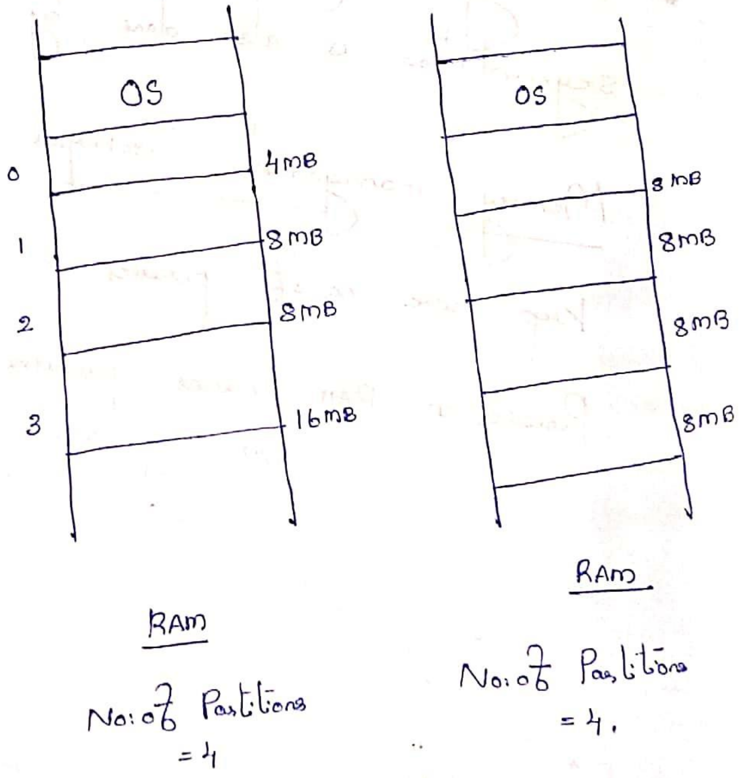

OS
P1
P2
P1
OS mount on to main memory
Contiguous (continuous) memory allocation.
Memory blocks
(RAM)

Non Contiguous

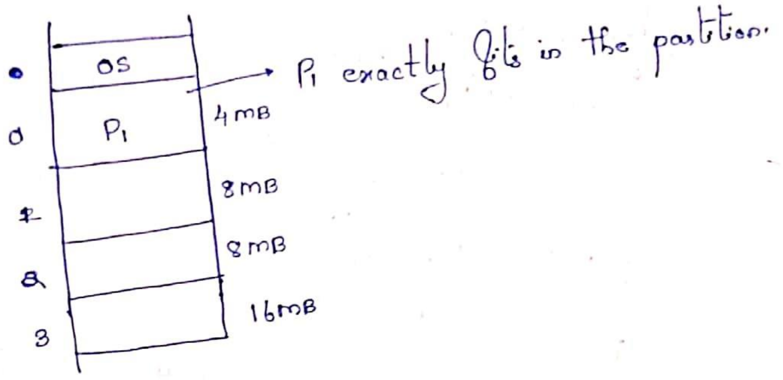


Fixed Partitioning (static Partition)

- No. of Partitions are fixed
- Size of each partition may or may not same.
- Continuous allocation so spanning is not allowed.

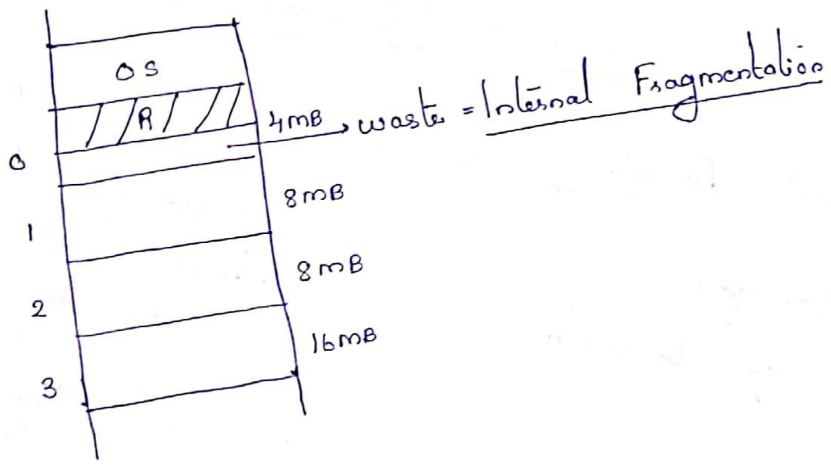


eg: Assume a process $P_1 = 4MB$ arrives to execute,

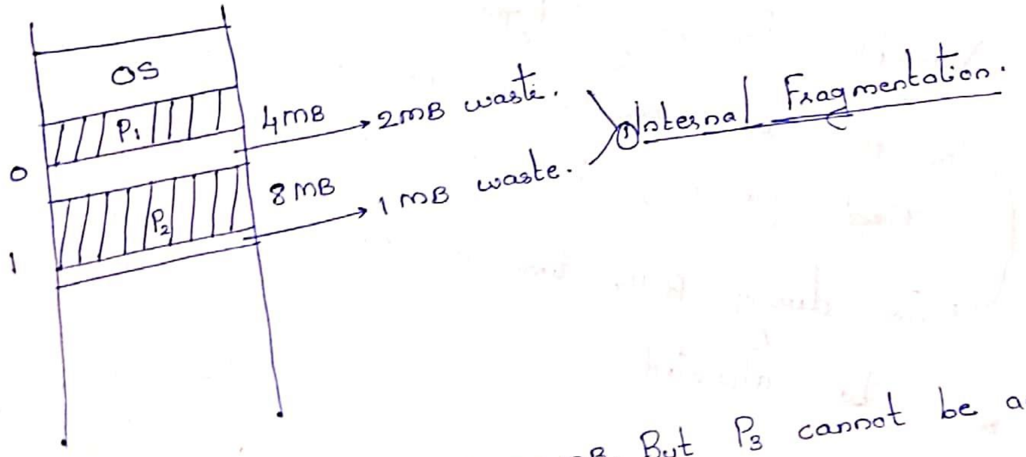


Best Case

$P_1 = 2MB$

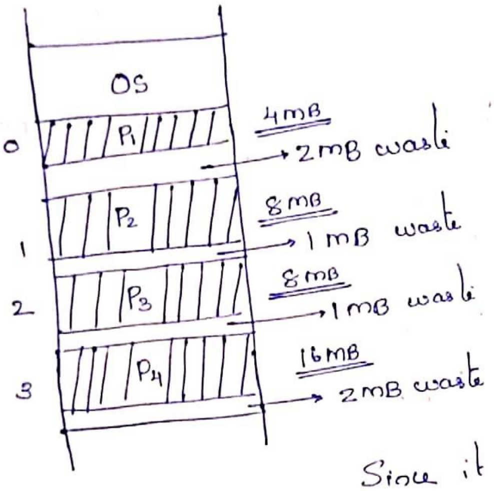


$P_2 = 7MB$



(2) Limit in process size: Let $P_3 = 32MB$. But P_3 cannot be accomodated in the RAM. B'coz maxm. size of partitioned memory in RAM is 16MB.

(3) Limitation on Degree of multiprogramming: Assume 6 processes are arrived. RAM can store only 4 processes, b'coz RAM was partitioned into 4. Remaining 2 processes cannot be accomodated



$P_1 = 2MB.$

$P_2 = 7MB, \quad P_3 = 7MB, \quad P_4 = 14MB$

Assume P_5 process came,

$P_5 = 5MB$

But here 6MB space is available. Since it is a contiguous allocation. This is known as (4) External Fragmentation. Whenever there is Internal Fragmentation in memory, External Fragmentation exists.

Fixed Partitioning was used in 1960's in mainframes.

Advantages

1. Easy to implement

Disadvantages

1. Internal Fragmentation
2. Limit in Process size.
3. Limitation on Degree of multiprogramming
4. External Fragmentation.

Variable Partitioning / Dynamic Partitioning.

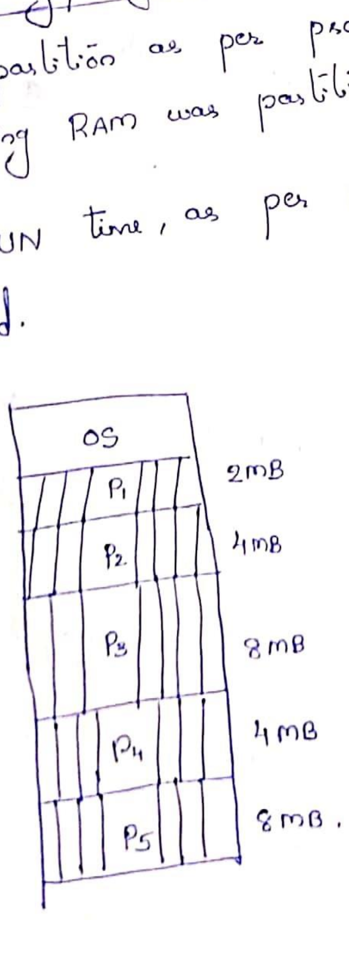
Here RAM will partition as per process size.

In Fixed partitioning RAM was partitioned before process arrives.

So during RUN time, as per process requirement memory (RAM) will be allocated.

eg: $P_1 = 2MB$

$P_2 = 4MB, \; P_3 = 8MB$



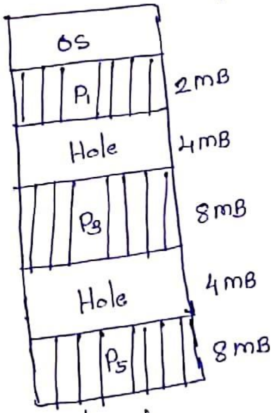
Advantages.

1. No Internal Fragmentation.
2. No limitation on no: of processes.
3. No limitation on the process size.

Assume P_2, P_4 were terminated. Let a new process P_6 arrives ie $P_6 = 8MB$.

Since Dynamic partitioning comes under Contiguous memory management technique, P_6 cannot be added to RAM.

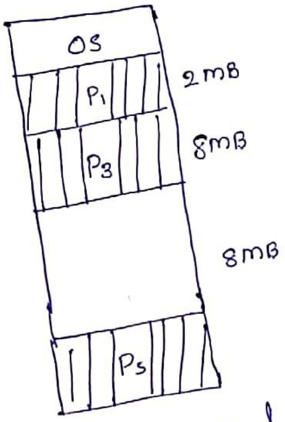
∴ P_6 has to wait.



This is known as

① External Fragmentation. → Disadvantage

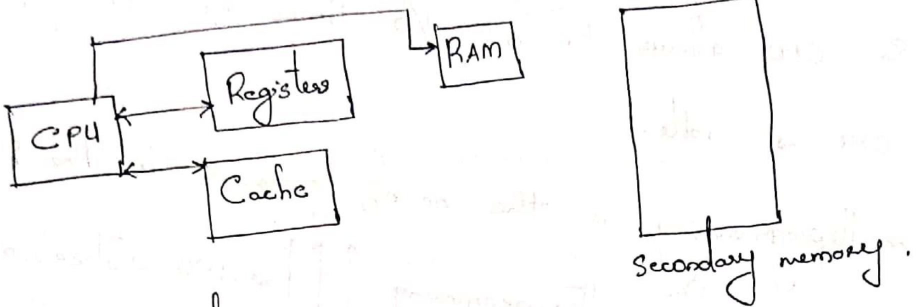
To solve External Fragmentation, a new method known as Compaction can be used ie



But Compaction is an undesirable method, b'coz a running process has to be stopped. → Disadvantage

(2) Allocation / Deallocation is complex: Since in variable partitioning more no: of processes arrives which increases more holes. ∴ allocation/deallocation of processes will be complex.

Memory Management: method of managing the primary memory.



Greater speed

1. Registers 2. Cache

3. RAM

CPU is directly connected to Registers, Cache & RAM.

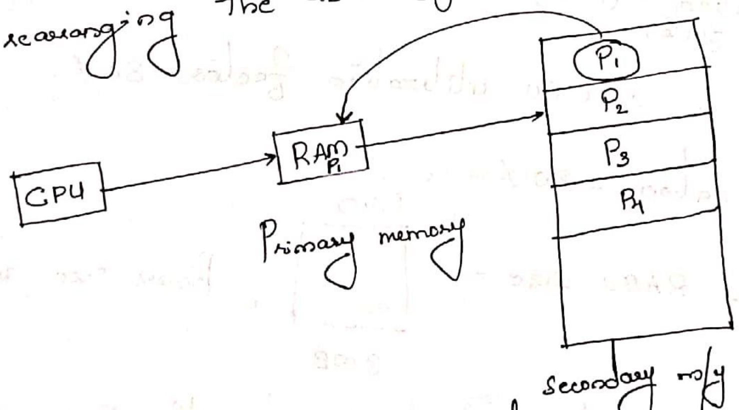
All programs are stored in secondary memory. For the pgms. to execute, programs should be loaded to RAM.

Speed:

Registers & Cache (kb) >> RAM (Gb).

CPU is not connected to secondary memory.

So rearranging the above figure.

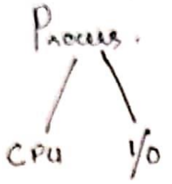


P_1, P_2, P_3, P_4 = programs

Now CPU can directly execute with the P_1. Similarly all pgms. This is called the

Degree of multiprogramming

Multiprogramming means the execution of multiple programs. ie multiple pgms in primary memory (RAM). This will increase CPU utilization factor.



P_1 is executed by CPU. P_1 (process) request for I/O. So CPU allows P_1 for I/O. Hence P_1 has I/O. During this time CPU is idle.

Degree of multiprogramming is the no: of programs in the RAM (main memory). As Degree of multiprogramming ↑↑↑ ⇒ CPU utilization factor ↑↑ ⇒ System efficiency ↑↑↑.

1. RAM size is 4MB, process size 4MB. How many processes can be in the RAM?

$\Rightarrow \frac{4}{4} = 1.$

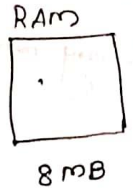
$\boxed{P_1}$. k = time for I/O operation by process (P_1). So CPU will execute P_1 for $(1-k)$ time

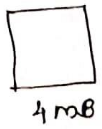
RAM

∴ CPU utilization factor $= (1-k)$.

Assume $k = 70\%$. ∴ CPU utilization factor $= 30\%$

ie CPU utilization $= 30\%$

Increasing the RAM size = 

, Process size is 

No: of processes that can come into the RAM $= \underline{\underline{2}} = \left(\frac{8\,MB}{4\,MB}\right)$

Assume one process perform k amount of time for I/O operation. Since two processes are here, ∴ k^2 time will be for I/O operation.

∴ CPU utilization will be = $(1-k^2)$

Let $k = 70\% = \frac{70}{100} = 0.7$.

∴ CPU utilization factor = $1-(0.7)^2 = 1-.49 = .51 = \underline{51\%}$

Increase RAM size ie 16MB. Process = 4MB.

No: of processes in RAM = $\frac{16}{4} = 4$.

If 'k' is the amount of time, 1 process is doing in the I/O operation.

Assume all 4 processes are doing the I/O operations at the same time, ie k^4!

∴ CPU utilization factor = $1-(k)^4$

Let k = 70 %. ∴ CPU utilization factor = $1-(0.7)^4 = \underline{76\%}$

Size of RAM ↑↑ ⇒ CPU utilization factor ↑↑

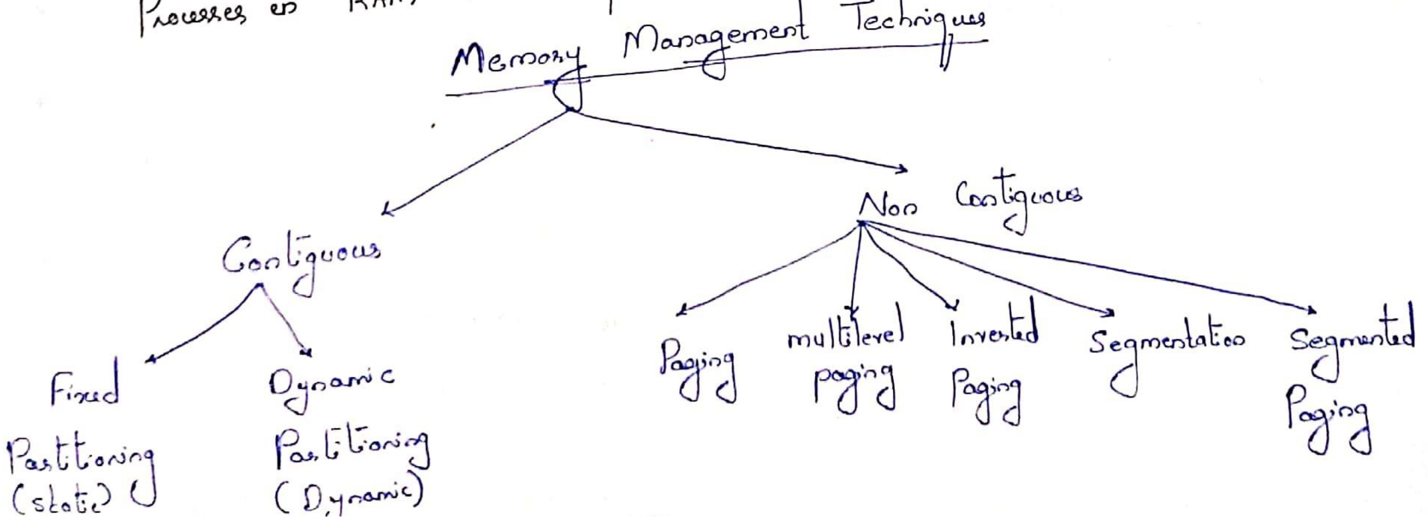
* Memory management of RAM (1° m/y) is more important.

* Memory management is ~~under~~ done by Operating System. Paging & segmentation is also done by O.S

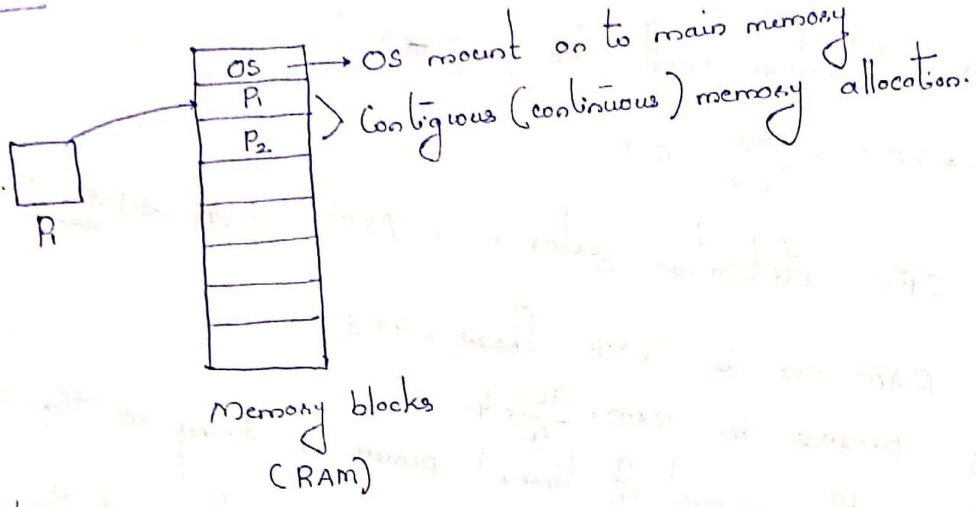
Memory management Techniques

Keep more no: of processes in the primary memory/RAM.

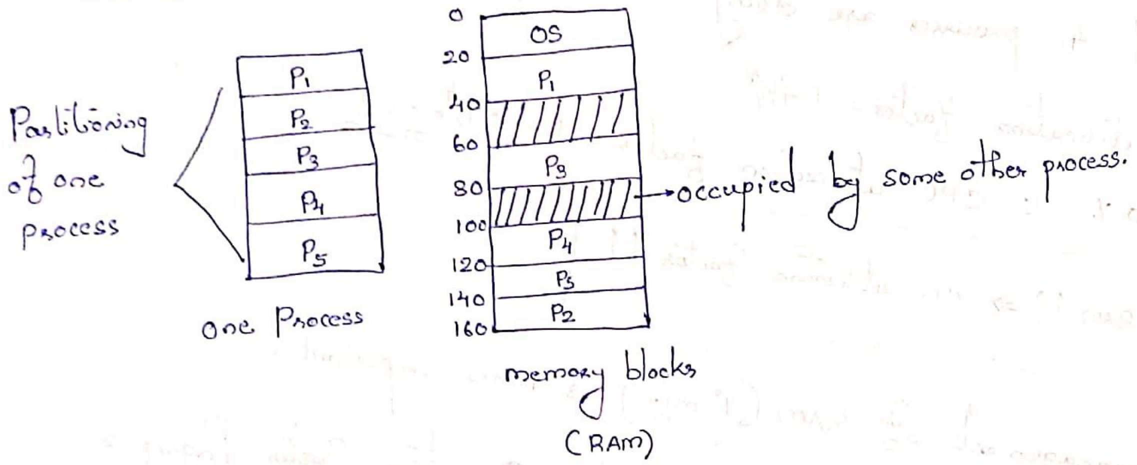
Processes in RAM means processes are in READY state.



Contiguous


OS
P1
P2
P1
OS mount on to main memory
Contiguous (continuous) memory allocation.
Memory blocks
(RAM)

Non Contiguous

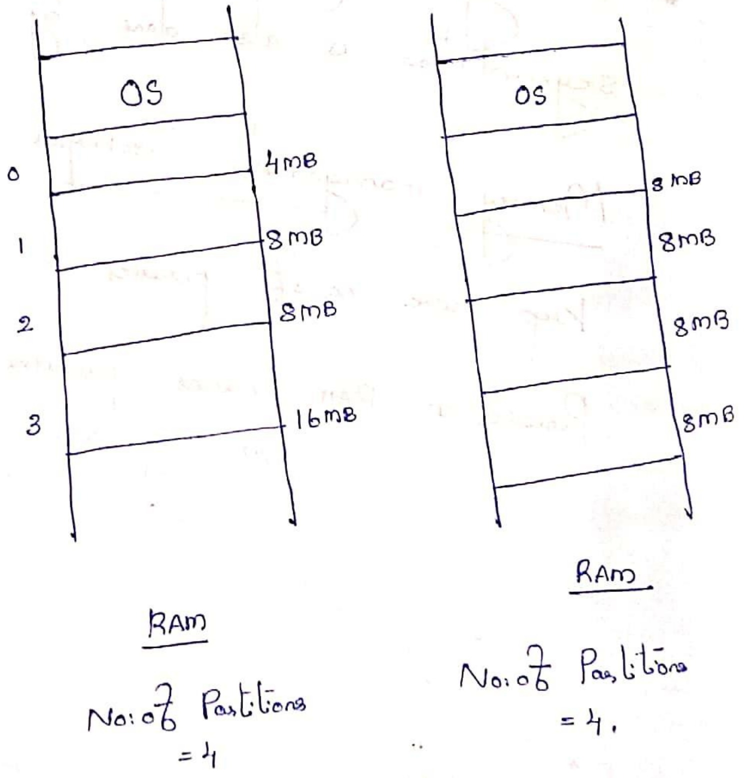


Fixed Partitioning (static Partition)

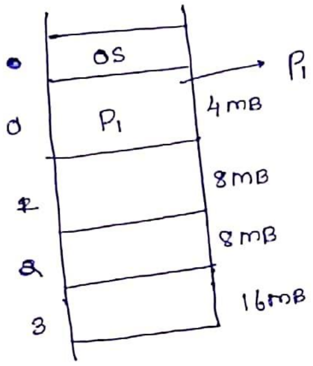
. No. of Partitions are fixed

. Size of each partition may or may not same.

. Continuous allocation so spanning is not allowed.



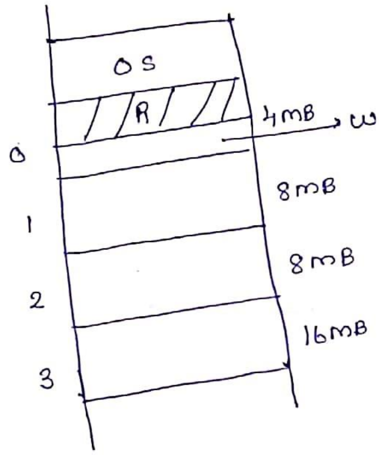
eg: Assume a process $P_1 = 4MB$ arrives to execute,



P_1 exactly fits in the partition.

Best Case

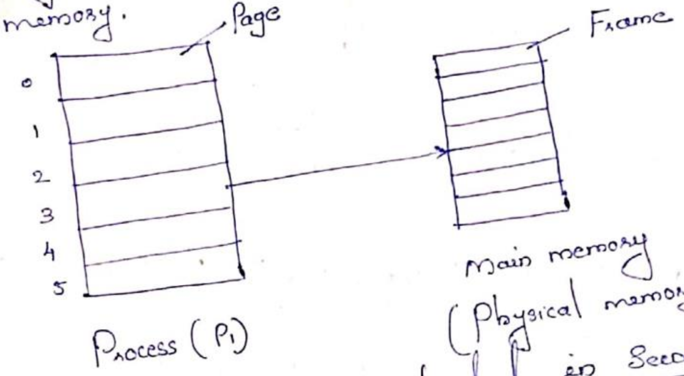
$P_1 = 2MB$



waste = Internal Fragmentation

Page Replacement Algorithms:

- For execution, a process has to be in main memory (physical memory). Size of main memory is finite. Sometimes size of Process is larger than main memory.
- Process is loaded into equally sized portions called pages. Page of that process to be executed is loaded into main memory.
- Memory management process. Hence more no: of process can be executed in main memory.



- Initially, all pages of process was loaded in Secondary memory. Page to be executed by CPU is loaded into the main memory. This is known as Demand Paging.
- When Demand Paging is used, more no: of processes can be there in main memory. ∴ it looks like size of the main memory is larger. This is known as Virtual Memory.
- If the needed page is not in the main memory, then it is known as Page Fault.
- Main memory is divided into equally sized portions called Frames.
- Suppose all the frames in main memory are full. Then a page from 2° m/y has to be swapped in to the main memory. For that some frame has to be swapped out of the main memory. The page to be swapped out of main memory is decided by Page Replacement Algorithm.

Page Table : Information about the swapped in & swapped out processes are stored in page Table.

- Page Table is located in main memory.

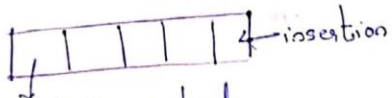
Page 1	I/v
Page 2	I/v
Page 3	I/v

FIFO Page Replacement Algorithm

Pages:	3	2	1	3	4	1	6	2	4	3	4	2	1	4	5	2	1	3	4
f_1	3	3	3	3	4	4	4	4	4	3	3	3	3	3	5	5	5	5	4
f_2		2	2	2	2	2	6	6	6	6	4	4	4	4	4	2	2	2	2
f_3			1	1	1	1	1	2	2	2	2	2	1	1	1	1	1	3	3
	×	×	×	✓	×	✓	×	×	✓	×	×	✓	×	✓	×	×	✓	×	×

Page fault / Page miss

Queue :



Page to be replaced.

No: of page faults = 13.

Miss ratio = $\frac{13}{19}$

hit ratio = $\frac{6}{19}$

LRU Page Replacement Algorithm

←

Sequence string (Pages)	3	2	1	3	4	1	6	2	4	3	4	2	1	4	5	2	1	3	4
f_1	3	3	3	3	3	3	6	6	6	3	3	3	1	1	1	2	2	2	4
f_2		2	2	2	4	4	4	2	2	2	2	2	2	2	5	5	5	3	3
f_3			1	1	1	1	1	1	4	4	4	4	4	4	4	4	1	1	1
	×	×	×	✓	×	✓	×	×	×	×	✓	✓	×	✓	×	×	×	×	×

LRU = Least Recently Used. Replace a page that was not used for a long period of time.

No. of page faults = 14.

No. of hits = 5.

Hit ratio = $\frac{5}{19}$

Miss ratio = $\frac{14}{19}$.

Optimal Page Replacement Technique :

→

Sequence string	3	2	1	3	4	1	6	2	4	3	4	2	1	4	5	2	1	3	4
f_1	3	3	3	3	4	4	4	4	4	4	4	4	4	4	5	5	5	5	5
f_2		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	3	3
f_3			1	1	1	1	6	6	6	3	3	3	1	1	1	1	1	1	4
	×	×	×	✓	×	✓	×	✓	✓	×	✓	✓	×	✓	×	✓	✓	×	×

Replace a page that will not be used in near future.

apply FIFO

No. of page faults = 10

No. of hits = 9

Hit ratio = $\frac{9}{19}$

Miss ratio = $\frac{10}{19}$.

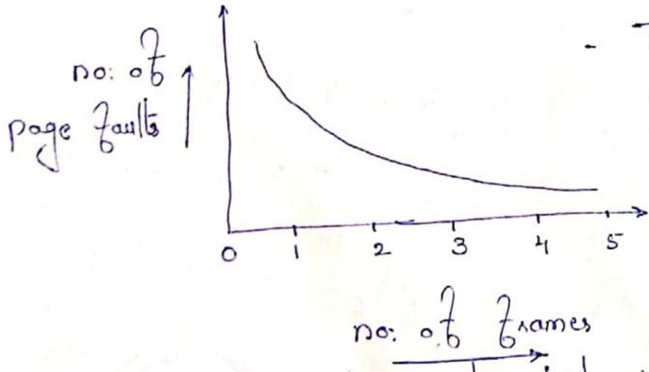
Optimal approach is the best one. It gives less no: of page faults. Not implementable. B'coz we have to predict the future. This technique can be used as a standard/benchmark. It is a theoretical concept.

For the given question,

	Page faults	hit ratio
FIFO	13	6/19
LRU	14	5/19
Optimal	10	9/19

For this reference string, FIFO is not good, b'coz it gives largest no: of Page faults.

Normally,



- This is the expected outcome. But this graph is not always true for every reference string. Sometimes, as no: of frames increases, no: of page faults also increases. This is b'coz of the abnormal behaviour of page replacement algorithm. This abnormal behaviour is known as Belady's Anomaly. It commonly occurs in FIFO algorithm.

Belady's anomaly doesn't come in LRU / Optimal replacement algorithm.

Increasing the no: of frames will increase the no: of page faults is known as Belady's anomaly.

Eg:. of Belady's algorithm.

Case 1 :

Reference strings	5	4	3	2	1	4	3	5	4	3	2	1
f_1	5	5	5	2	2	2	3	3	3	3	3	1
f_2		4	4	4	1	1	1	5	5	5	5	5
f_3			3	3	3	4	4	4	4	4	2	2
	×	×	×	×	×	×	×	×	✓	✓	×	×

No: of page faults = 10. ; Miss ratio = $\frac{10}{13}$

No: of hits = 3 Hit ratio = $\frac{3}{13}$.

Case 2 :

Reference strings	5	4	3	2	1	4	3	5	✓ 4	✓ 3	✓ 2	1	5
f_1	5	5	5	5	1	1	1	1	1	1	2	2	2
f_2		4	4	4	4	4	4	5	5	5	5	1	1
f_3			3	3	3	3	3	3	4	4	4	4	5
f_4				2	2	2	2	2	2	3	3	3	3
	×	×	×	×	×	✓	✓	×	×	×	×	×	×

No: of page faults = 11 ; Miss ratio = $\frac{11}{13}$

No: of hits = 2 ; Hit ratio = $\frac{2}{13}$

FIFO

Advantages

- easy to understand.
- easy to implement.

Disadvantages:

- Belady's anomaly. ie not very effective in all time.
- It replace an active page, to bring a new one ie why immediate page fault occurs.