(Brief note)

Applied Operating System (AOS)
↳ Intro [5hrs]
↳ process & Threads [13 hrs]
↳ Scheduling
↳ Deadlock & starvation
↳ Memory management.

class code
purmi 5q

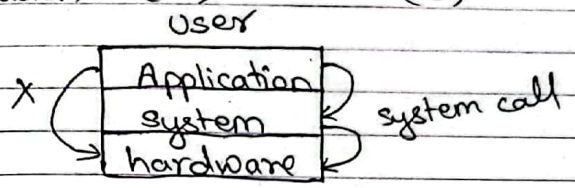
Chap 1 - Introduction [5 hrs] {Theory}
2 - Process, Threads & Scheduling [13 hrs]
3 - storage Management [12 hrs]
4 - Input/output Management [7 hrs] } T+N
5 - Case study [5 hrs]
6 - Emerging Trends in OS [3 hrs] {Theory}

ch:-1 :- Introduction

1/5

AOS

ROM. (boot loader) → (slow) Hard disk (OS) → (fast) Ram (OS) → Register → CPU



Evolution of OS / types of OS :-

1. Batch O.S
   ↳ punch card, compiter operator
   ↳ time consuming

2. Multiprogramming:
   ↳ placing different program in Ram

3. Time-sharing / Multi-tasking:
   ↳ CPU share time with the program stored in RAM

4. Distributed O.S vs centralized O.S
   ↳ fault tolerant ↳ fault intolerance
   ↳ lack of security

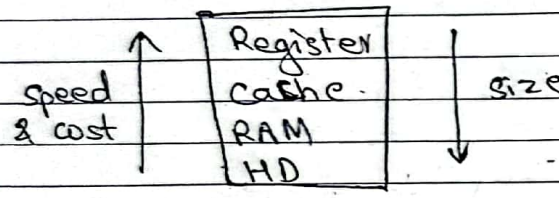
middle level language.

1/5

C programming
- low level language → for system software
- high level language → for application software

1st O.S using C language is UNIX.

Memory hierarchy



1/17

Goals of OS:-
primary Goal: User Convenience
secondary Goal: ~~Resou~~ Efficient use of resources

1.3 Computer Organization & Architecture

CO	CA
→ Operational unit	→ no. of bits used to represent data
→	→ Instruction set
→ Low level	→ High level
	→ Memory Organization

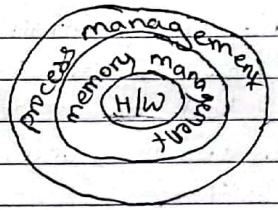
1/17

1.4 Operating system structure
1. ~~mondrit~~ monolithic
2. Layered
3. Virtual memory
4. client server

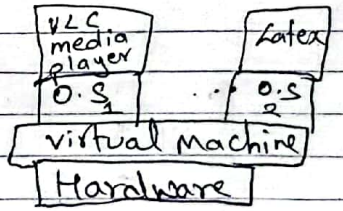
Dual mode of CPU

- User mode: for user program in which I/O device and certain other instruction aren't allowed
- Kernel mode: for os in which all instruction is allowed

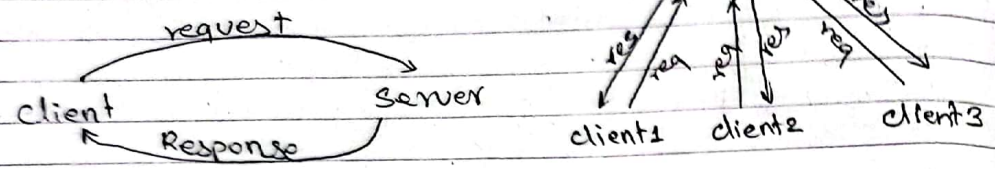
1. Monolithic → ~~0.~~ oldest architecture. every core functionality is on the same place that causes overhead → kernel
2. Layered →



3. Virtual memory →



4. client server



Operating System Structure:

Some of the main structure used in OS are:-

1. Monolithic architecture :-

It is the oldest architecture used for developing operating system. OS resides on kernel for anyone to execute. System call is involved ie switching from user mode to kernel mode and transfer control to operating system as shown as event 1. Two operating system then examines the parameter of the call to determine which system call is to be carried out shown in event 2. Next, the OS indexes into a table that contains procedure that carries out system call. This operation is shown in event 3. Finally, it is called when the work has been completed and the system call is finished, control is given back to the user mode as shown in event 4.

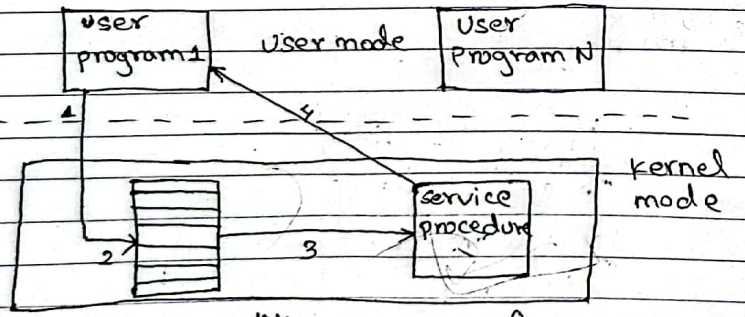


Fig:- monolithic structure of OS

2. Layered Architecture

The layered Architecture of OS was developed in 60's. In this approach, the OS is broken up into number of layers. The (bottom or) core layer is the ~~or~~ hardware layer and the last layer is the user interface layer as shown in the figure.

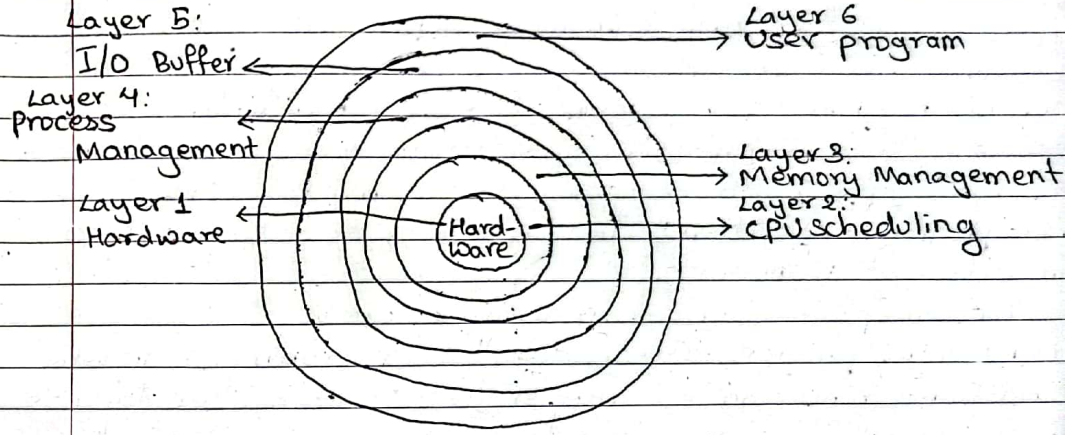


fig:- Layered structure of OS.

The layered are selected such that each user functions and services of only lower level layer. The 1st layer can be debugged without any concern for the rest of the system.

The main disadvantage of this architecture is that it requires an appropriate definition of the various layer and a careful planning of the proper placement of the layer.

Virtual memory architecture:

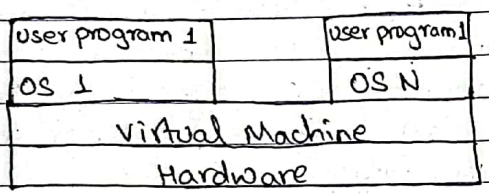


fig:- virtual memory architecture of OS

Virtual machine is an illusion of a real machine. It is created by a real machine OS which make a single real machine appears to be several real machine. Its multiprogramming shares the resource of a single machine in different manner.

The concepts of virtual machines are:-

a. CP (control program): It creates the environment in which virtual machine can execute. It gives to each user facilities of real machine such as processor, storage I/O devices.

b. conversation monitor system (conse, CMS): It is a system application having features of developing program ~~tech~~ which consists editor, language translator and various application package

c. Remote spooling communication system (RSCS): Provide virtual machine with the ability to transmit and receive file in distributed system.

IPCS (interactive problem control system): used to fix the virtual machine software problems.

4. Client/server architecture:

A trend in modern OS is to move maximum code into the higher level and remove as much as possible from OS, minimizing the work of the kernel. The basic approach is client always send request to this side in order to perform the task and on the other side, server gates complementing that request send back response.

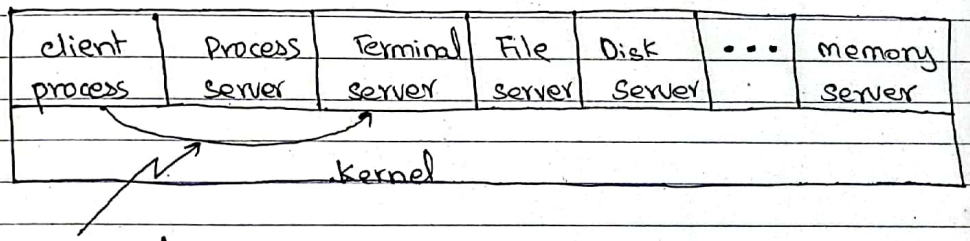


fig:- client/server architecture

Here, the main task of kernel is to handle all the communication between client and server by splitting the OS into number of parts.

Its advantage is its adaptability to user in distributed system.

1/20

1.5

Generation of OS

(i) 0th - programs loaded directly into the computer's memory

(ii) 1st - simple Batch OS

(iii) 2nd - multiprogramming EX IBM's OS/360

(iv) 3rd - Time sharing / multitasking Ex. UNIX

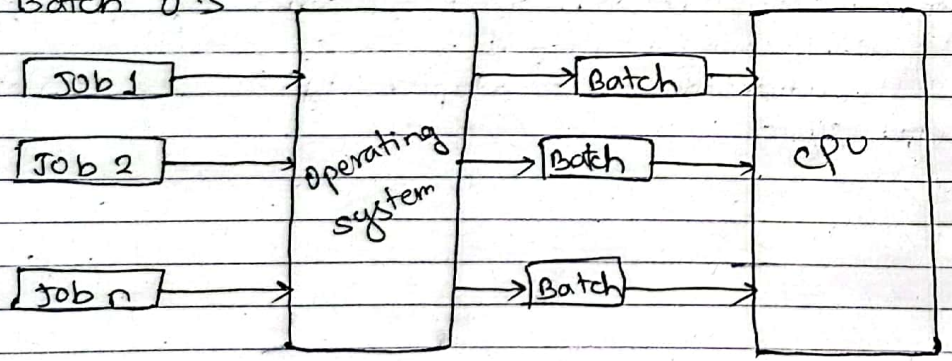
(v) 4th - Distributed computing EX windows, Mac

1.6 (scaler.com) or (javatpoint.com)

Types of OS:

1. Batch OS
2. Multiprogramming
3. Multiprocessing
4. Time sharing
5. RTOS
6. Network OS
7. Distributed OS
8. OS for embedded devices.

1. Batch O.S



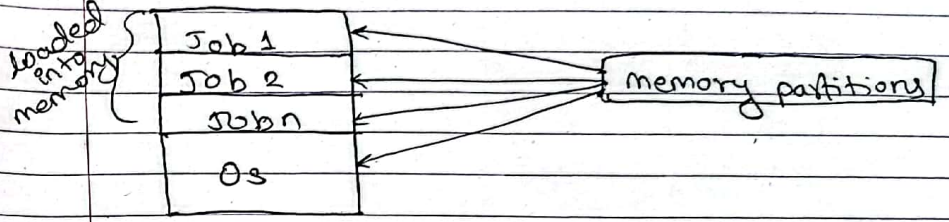
adv.

→ has mechanism to jump from 1st job to 2nd job and from area batch to another.

→ disadv - time consuming, system deadlock, CPU idle, lacks job priority.

1/20

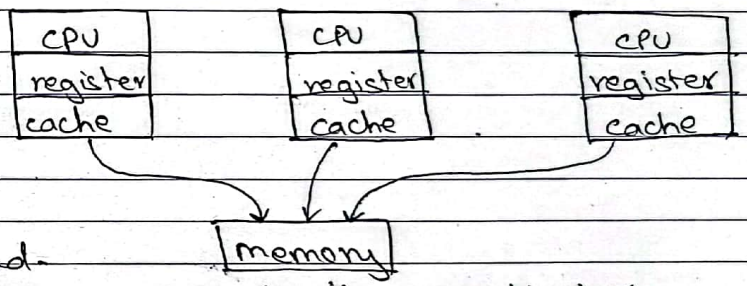
2. Multiprogramming:-



→ load all execute one program

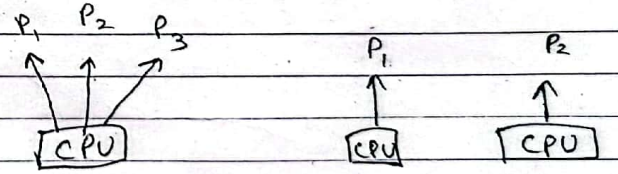
Advantage :- fast response time

3. Multiprocessing:



disad.

- Increase complexity, costlyier

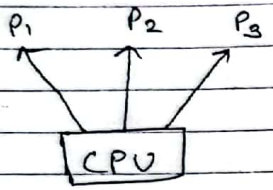


concurrent (multiprograming)

parallel (multiprocessing)

1/20

4. Multitasking / time sharing :- Eg :- multics, unix



Adv.
↳ Time response, no starvation, sharing resource

5 RTOS (real time OS)

Real Time OS

Soft RTOS Hard RTOS

1/21

Distributed OS	Network OS
- loosely coupled system - If one-component fails then the requested service is provided by the other components - manage and coordinate hardware resources accross multiple computers. - Eg. Amoeba - Authentication can be done whereever required	- tightly coupled system - If server is down then the entile system is affected - manage network resources and provide services to remote client. - Windows NT - Authentication is done only at server.

1/21

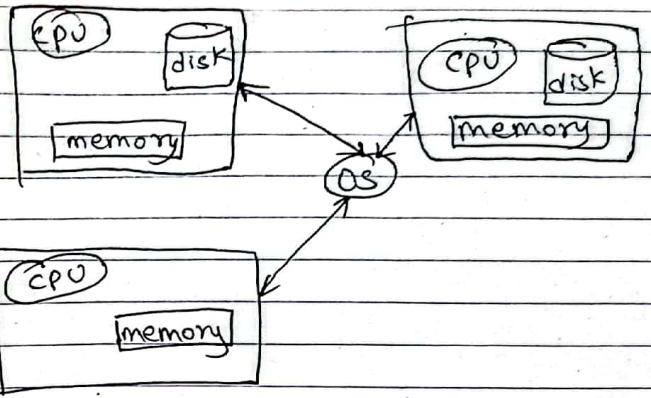


Fig : Distributed OS

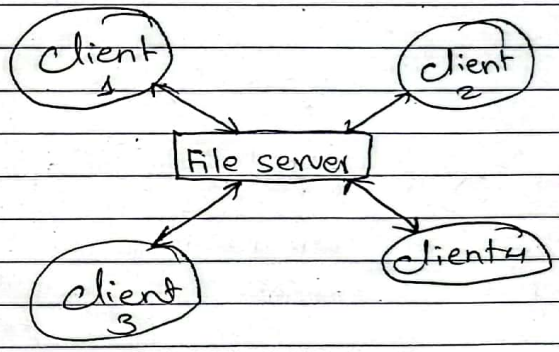


fig :- Network OS

(8) OS for embedded device
Eg :- Linux, micro controller OS,
- combination of computer processor, memory and I/O peripherals that has a dedicated function within a larger mechanical or electronic system.

1/21

1.7. Functions of OS

1. Memory management
2. Process management
3. Device management
4. Security
5. Error handling
6. Storage management
7. control over system performance.

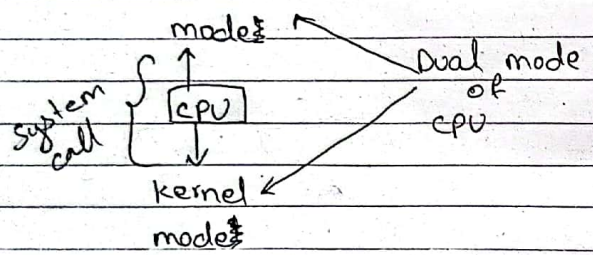
~~8.~~

1.8 Kernel Data structure

kernel = core OS

fast fault Tolerance (crash)

user space ← Program / application



kernel space

Hardware

~~1/23~~ 1/21

1.9 User & OS Interface

User

User Interface

Application software ↓ ↕ system call (comm. bet" As & OS)

OS

Hardware

→ To excess hardware, we ~~to~~ should go ~~with~~ through OS.

Aspect	CO	CA
defn	refers to the internal structure and components of computer system.	refers to conceptual design & system level structure of cs.
Focus:	hardware implementation details & functional units	system-level design principl & behaviour.
Goal	Optimize hardware organization for efficient operation	design efficient computer system to meet performance g

1/24

1.8 Kernel Data Structure:

The kernel data structure refers to the data structure used by the kernel of an OS to manage various aspects of the systems operations. These data structures are essential for organizing and accessing information about the processes, memory, devices, file systems and other system resources.

Example of data structure:-
1. PCB (process control block)
2. TCB (Thread control block)
3. Page tables
4.

1/24

1.9 User and OS Interface

A) Character User Interface (CUI)
- command user Interpreter (CLI)
- Interactions using command
- faster than GUI & consumes less memory during execution

B) Graphical user Interface (GUI)
- Interactions using icons, mouse, menu etc.
- slower & consumed more memory during execution.
- User friendly

1/24

1.10 System calls, Types of system calls and Handling system calls:

System calls:
- System calls provide the interface betn a process and the OS.
- These calls are generally available as assembly language instructions and are usually listed in the manuals used by assembly language programmers.
- Several languages such as C, C++ and PERL have been defined to replace assembly language for programming. These languages allow system calls to be made directly.
- For eg: i) UNIX system calls may be invoked directly from C or C++ program.
  ii) System calls for modern ms-windows platforms are part of the Win-32 API, which is available for use by all the compilers written for ms-windows
- Java doesn't allow system calls to be made directly because a system call is specific to an OS and results in platform specific code. However, if an application requires system specific features, a java program can call a method that has been written in another language, typically, C or C++, that can make the system calls. Such methods are known as native methods.

System calls occurs in different ways, depending
n the computer in use. Often more information
s required than simply the identity of the
desired system call. For eg:- to get input, we
may need to specify the file or device to use
s the source and the address and length of
the memory buffer into which the input should
e read.
Three general methods are used to pass param-
eters to the OS:
(i) The simplest approach is to pass the
arameters in registers.
(ii) If there are more parameters ~~than~~ then
egister then, the parameters are generally
tored in a pt block or table in memory and
the address of the block is passed as parameters
n a register.
(iii) Parameters can also be placed or pushed
n to the stack by the program and popped
ff the stack by the OS. This approach doesn't
imit the number or length of parameters being
assed.

Types of system calls:

System calls can be grouped roughly into 5 major categories:-

1. Process control:

These system call manage processes which are running instances of programs.
Examples:-
- fork(): create a new process (child) that is the copy of current process (parent)
- exec() family: replace the current process's image with a new program
- wait(): wait for a child process to terminate.
- exit(): Terminate the current process

2. File manipulation:

These system calls allow process's to create, read, write and manipulate files and directories (folder)
Examples:-
- open(): open a file for reading or writing
- read(): read data from a file
- write(): write data to a file
- close(): close an opened file

3. Device manipulation:
These system calls provide access to hardware devices and manage i/o operations.
Examples:
- read() and write() can also be used for ~~read and write~~ device i/p, o/p.
- ioctl() (i/o control): control device specific operations

4. Information maintainance:
These system calls retrieve info about system and process status.
Examples:
- getpid(): get the process id of the current process.
- getuid(): get the user id of the current user
- getgid(): get group id of current user
- time(): get current time.

5. communications:
These system calls enable communication and synchronisation betn processes.
eg:
pipe(): create a communication channel (pipe) betn two process
socket(): create a network socket for network communication
- send() and ~~receive()~~: recv(): send and receive data over sockets
- semaphore functions: mangge semaphores for process synchronization,

1/31

1.11 Virtual Machine

1/31

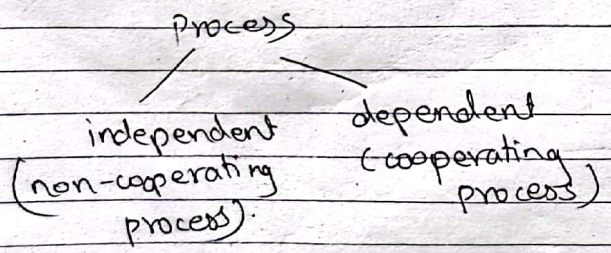
IMP Unit II: Process, Thread & Scheduling

2.1.1 Process:

A process is an instance of a program in execution. A program becomes process when executable file is loaded in main memory.

Difference betⁿ program and process

Basis	Program	Process
Basic	Program is a set of instruction	When a program is executed, it is known as process
nature	It is passive in nature	It is active in nature
Life span	Its life span is longer.	Its life span is limited
required resources	program is stored on disk on some file and doesn't require any other resources	process holds resources such as CPU, memory address, disc, i/o etc



1/31

VIMP 2.1.2

Process state:

When a process execute, it passes through different states. These stages may differ in different operating system and the name of these states are also not standardized. In general, a process can have one of the following 5 states at a time.

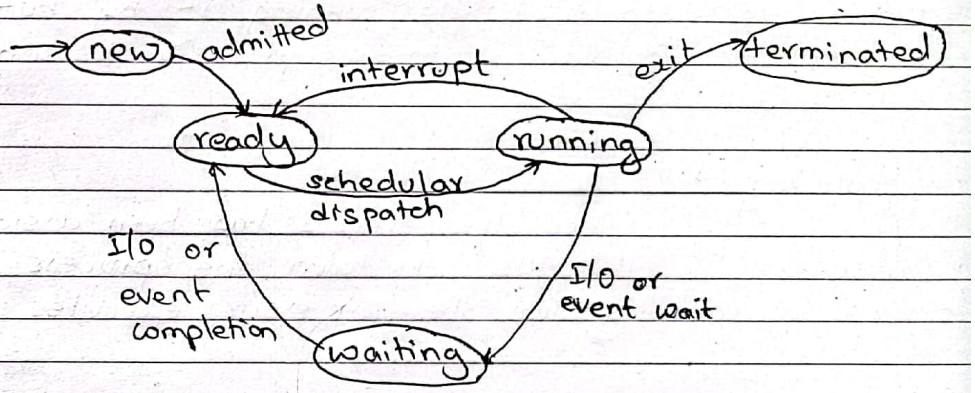
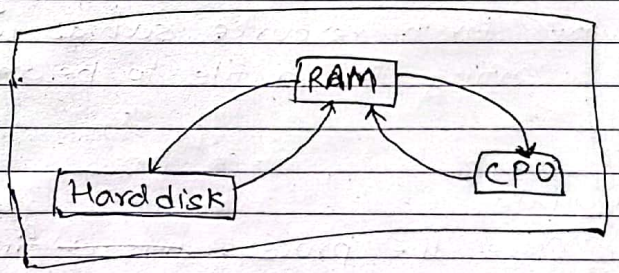


Fig: state diagram



1. new state / start state:
This is the initial state when a process is first started or created.

2. Ready state:
The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the OS so that they can run. Process may come into this state after new state or while running it by but interrupted by the schedular to assign CPU to some other process.

3. Running state:
Once the process has been assigned to a processor by the OS schedular, the process state is set to running and the processor executes its instruction.

4. Waiting state:
Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input or waiting for a file to become available.

5. terminated / exit state:
Once the process ~~misses~~ finishes its execution or it is terminated by OS, it is moved to terminated state where it waits to be removed from the main memory.

2.1.3 Operation on Processes:
1. Process Creation
2. Process Termination → Normal
→ abnormal

प्रक्रिया नियंत्रण → PCB.

2.1.4 PCB (Process control block):

A process control block is a data structure maintained by OS for every process. The PCB is identified by an integer process id (pid). The PCB is maintained for a process throughout its life time, and is deleted once a process terminates. A PCB keeps all the information needed to keep track of a process as listed in a figure.

Process ID
State
Pointer
Priority
Program counter
CPU register
I/O information
Accounting information
:

Process ID:

Unique identification for each of the process in the OS.

Process state:

The current state of the process i.e whether it is ready, waiting, terminating, running or new.

1/31

Pointer:

A pointer to a parent process

Priority:

Containts the priority number of each process

Program counter:

It is a pointer to a address of the next instruction to be executed for this process.

CPU registers:

various CPU registers where process need to be stored for execution during running state.

I/O ~~in~~ status information:

This includes a list of i/o devices allocated to the process.

Accounting information:

This includes the amount of CPU used for process execution, time limits, execution id etc.

CPU scheduling information:-

Process, priority and other scheduling info which is required to schedu the process.

memory management information:
This includes the info of page table, memory limits, segment table depending on memory used by OS

2.1.5 Implementation of processes:-
In an OS processes are implemented through various components such as PCB, schedular, memory management and i/o management

PCB:

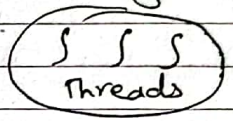
schedular:-
The schedular determines which process gets CPU time and when.

Memory management:
Allocates and deallocates ~~process~~ memory for processes.

I/O management:
Handles ~~i/o~~ i/p and o/p operations ensuring processes can interact with the devices ~~effici~~ efficiently.

2.2 Threads

2.2.1 Thread concepts and usage
↳ Light weight process



Thread concepts:
- A thread is a smallest unit of processing that can be performed in an OS. A process can consists of several threads, each of which execute separately. It is also known as light weight process.

For eg: One thread could handle screen refresh and drawing, another thread printing, another thread the mouse and keyboard

- A thread is the basic unit of CPU utilization It comprises a thread id, program counter, registe set and a stack. It shares with other thread belonging to same process. Its code section, data section and other operating resources such as open files and signals. A traditional process has multiple threads of control, it can perform more than one tasks at a time.

Usage:-
Threads were invented to allow parallelism to be combined with sequential execution and blocking

system calls. Some examples of situations where we might use threads:-

1. Doing lengthy processing:
When a window application is calculating, it cannot process any more messages. As a result, the display cannot be updated.

2. Doing background processing:
Some task may not be time critical but needed to execute continuously

3. Concurrent execution on multiprocessors

4. Manage i/o more efficiently:
Some threads wait for i/o while others compute.

5. It is mostly used in large server applications.

2.2.2 Threads or multithreading models:

A relationship must exist betⁿ user thread and kernel threads. There are 3 models related to user and kernel threads They are:-

1. Many to one model:

It maps many user thread to one kernel thread. Thread management is done by the thread library in user space, so it is efficient but the entire process will block if a thread makes a blocking system call. Because only one thread can access the kernel at a time, multiple thread are unable to run in parallel on multiprocessor.

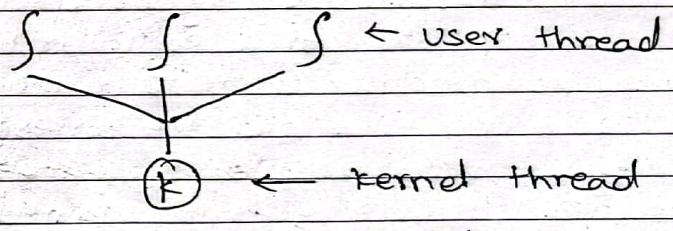


Fig:- many to one model

2. One to one model:

It maps each user thread to a corresponding kernel thread. It provides more concurrency than the many to one model by allowing another thread to run when a thread makes a blocking system call. i.e it also allows multiple threads to run in parallel on multiprocessor

The drawback is that creating a user thread requires creating the corresponding kernel thread and the overhead of creating the corresponding kernel thread can burden the performance of an application.

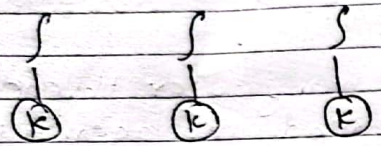


Fig: One to one model

9. Many to many model:

It multiplexes many user level threads to a smaller or equal number of kernel threads. The no. of kernel threads may be specific to either a particular application or a particular machine. This model allows the developer to create as many ~~as~~ user thread as the user wishes and the corresponding kernel threads can run in parallel on a multiprocessor. In such case true concurrency is not gained because the kernel can schedule only one thread at a time.

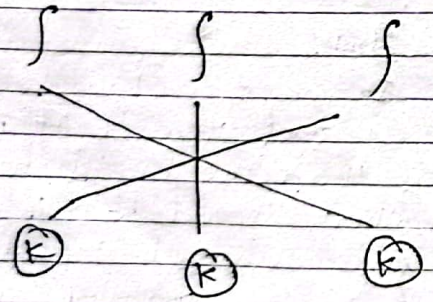


Fig:- many to many model

2.2.3 User space and kernel space thread:

User level thread are the thread that are visible to the programmer and unknown to kernel. such thread are supported above the kernel and managed without kernel support.

The OS directly supports and manages kernel level threads. In general user level threads are faster to create and manage than are kernel threads, as no intervention from the kernel is required. Linux, windows xp, mac osx supports kernel threads.

	User level thread	Kernel level thread
1.	User level threads are faster to create and manage	kernel level threads are slower to create & manage
2.	Implementation is done by a thread library at user level	OS supports creation of kernel threads
3.	User level threads is generic and can run on any OS.	kernel level thread is specific to the OS.
4.	Multithreaded application cannot take advantages of multiprocessing.	kernel routines themselv can be multithreaded

2.2.4 Hybrid implementations:

A hybrid threading model integrates the benefits of both user level thread and kernel level thread, aiming to optimize performance and resource utilization.

2.2.5 Thread vs process

process	Thread
① It is heavy weight	It is light weight, which takes lesser resources than a process.
② Process switching needs interaction with OS.	Thread switching doesn't need interaction with os.
③ In multiple processing environment, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
④ If one process is blocked then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, second thread in the same task can run
⑤ Multiple processes without using threads use more resources.	Multiple threaded processes uses fewer resources.

Process
- Independent (non-cooperating)
- Dependent (cooperating)
  - Shared variable
  - memory pass

⑥ In multiple processes each process operates independently of others	One thread can read, write or change another thread data.

2.3 Interprocess communication and synchronization

2.3.1 Cooperating processes:-

A cooperating process is one that can affect or be affected by other processes executing in the system.

2.3.2 Race condition:-

A race condition occurs when two processes (or threads) access the same variable / resource without doing any synchronization that means when several processes access and manipulate the same data concurrently and the outcome of the execution depend on the particular order in which access takes place is called a race condition.

For eg: One process is changing the bank balance in bank account while another is simultaneously observing the account balance and the last activity date. Now consider the scenario where the process changing the balance gets interrupted after updating the last activity date but before updating the balance. If the other process reads the data at this point, it doesn't get accurate information

2.3.3 Critical section (region) problem:

Suppose two or more processes require access to a single non-sharable resource such as printer. During the course of execution, each process will send command to the i/o device, receives status information, sending data and/or receiving data. Such resource is called critical resource and the portion of the program that uses it is called critical section of the program. However, only one program at a time is allowed in its critical section.

When two or more process access share data, often the data must be protected during access. Typically, a process that reads data from a shared queue cannot read it whilst the data is currently being written or its value being changed.

When a process is considered that it cannot be interrupted at the same time as performing a critical function such as updating data, it is prevented from being interrupted by the OS if it till it has completed the update. During this time, the process is said to be in its critical section. That means the part of program where the shared memory is accessed is called critical section or critical region. Once the process has written the data, it can then be interrupted and other processes can also run.

Critical section of a process should be small so that they don't take long to execute and thus other processes can run.

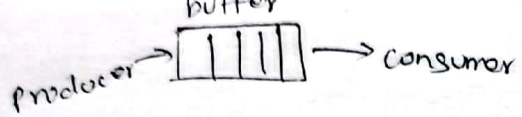
2.3.4(i) Mutual Exclusion with Busy waiting/strict alternation

→ while (condition);

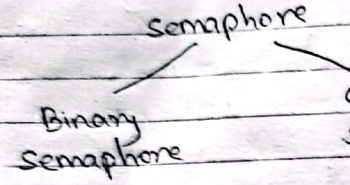
process 1	process 2

while (1)
{
while (turn != 0);
turn = 1;
noncritical_region();
}
process 1
	while (1)
	{
	while (turn != 1);
	turn = 0;
	noncritical_region();
	}
	process 2

- In this purpose solution, the integer variable 'turn' keeps track of whose turn is to enter the critical section.
- Initially turn = 0, process 1 inspects turn, finds it to be zero and enter in its critical section. Process 2 also finds it to be 0 and therefore sits in a loop continually testing 'turn' to see when it becomes 1.
- continuously testing a variable waiting for some value to appear is called busy waiting.
- When process 1 exit from the critical region is set to turn to 1 and now process 2 can find it to be 1 and enters into critical region
- Also, in this way both the process get the alternate turn to enter in the critical region.
- Taking turn is not a good idea when one of the ~~perso~~ processes is much slower than other
- Can used only when the waiting period is expected to be short.



(11) semaphore:




```
while(1)
{
  wait(s)
  critical section
  signal(s)
}
```

P_1

```
while(1)
{
  wait(s) or p(s)
  critical section
  signal(s) or v(s)
}
```

P_2

2/10

(2) Mutual Exclusion with Disabling Interrupt

It is the simplest solution to disable all interrupts to each process just after entering its critical section and enable them just before leaving it.

The CPU is only switched from process to process as a result of clock or other interrupt. If the interrupts are disable clock interrupts can't occur and the CPU can't be switched to another process. So, once a process has disabled interrupts, it can examine and update the shared memory without fear that any other process will interfere. However, this approach is not attractive because if one of the process disabled interrupt but it never turned on the interrupt, then that will be the end of the system.

(3) Mutual Exclusion with Lock variable:-

It is a software solution. In this approach shared variables are locked (1) or unlocked. When a process wants to enter its critical section, it first checks the lock. If the lock is zero, the process sets it to 1 and enters in the critical region, and a one means that some process is in its critical region.

This also has a problem: suppose that one process reads the lock

and sees that it is zero (0). Before it can set lock to one, another process is scheduled, runs and set the lock to one(1). When the 1st process runs again it will also set the lock to '1' and two process will enter in the critical region at the same time. Now the race will occur if the 2nd process modifies the lock just after the 1st process has finished its 2nd check.

④ Mutual exclusion with test and set lock.

TSL instruction is used for reading from a location or writing to a location in the memory and then saving a non-zero value at an address in memory. It is implemented in a hardware and is used in a system with multiple processors. When one processor is accessing the memory, no other processor can access the same memory location until TSL instruction is finished. Locking of a memory bus in this way is done in the hardware.

It reads the contents of memory word LOCK into register RX (TSL RX, LOCK) and then stores a non-zero value at the memory address LOCK. The operations of reading the word and storing into it are guaranteed to be invisible and no other process can access the memory word until the instruction is finished. To use the TSL instruction, a shared variable LOCK is used to access to shared

2/10

memory. When lock is 0 any process may set it to one '1' using the TSL instruction and then read or write the shared memory. When it is done, the process sets lock back to '0' using an ordinary MOVE instruction.

```
Enter_Region:
    TSL  REGISTER, LOCK
    CMP  REGISTER, #0     (compare)
    JNE  Enter_Region     (Jump when not equa
    RET                   (Return)

Leave_Region:
    MOV LOCK, #0
    RET
```

2/10

Peterson's Algorithm:

```
#define FALSE 0
#define TRUE 1
#define N 2
int turn;
int interested [N];
void enter_region (int process)
{
  int other;
  other = 1 - process;
  interested [process] = TRUE;
  turn = process;
  while (turn == process && interested [other] == TRUE)
}
void leave_region (int process)
{
  interested [process] = FALSE;
}
```

Initially neither process ~~in~~ is in its critical region. Now, process 0 calles enter_region(). It indicates its interest by setting its array element and sets turn to 0. Since process 1 is not interested, enter_region() return immediately. If process one now calls enter_region(), it will hang ~~their~~ ~~v~~ there until interested [0] goes to false, an event that only happens when process zero calls leave_region() to exit the critical region.

When both process calls enter_region(almost simultaneously. Both will store their process in turn. Which ever store is done last is the one that counts, the first one is lost. Suppose that process one stores last, so dot is one. When both process come to while statement, process zero execute it zero times and enter its critical region. Process one loop and doesn't enter in its critical region.

2.3.5 Sleep & Wakeup:

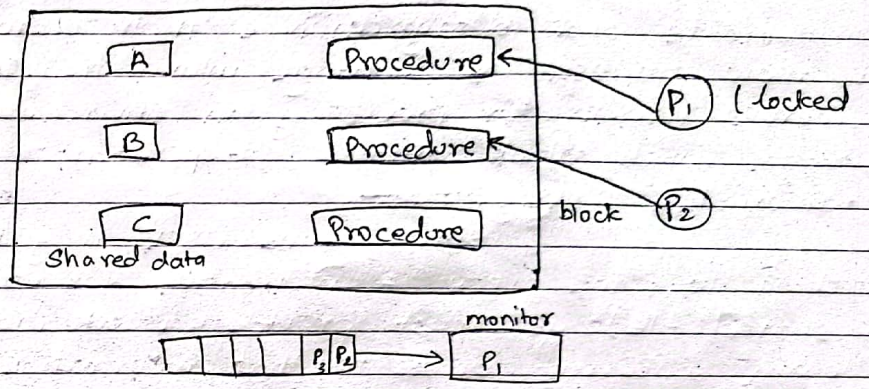
	Characteristic	Sleep & wakeup	Semaphore
1.	Granularity	Typically used for higher level conditions (eg. waiting for a condition to be true)	More fine grained control often used for both resource counting & mutual exclusion.
2.	Ease of use	Requires careful handling of cond^n and potential for missed signals if not properly managed.	Encapsulate signaling within the semaphore operations, reducing the risk of missed signals
3.	Complexity	more complex due to the need for additional cond^n variable management.	simpler for basic counting & mutual exclusion
4.	Scalability	can be less scalable due to the overhead of cond^n variables.	Generally more stable & efficient for high contention scenarios.

Sleep and wake up:

- When a process wants to enter its critical region, it checks to see if entry is allowed, if it is not, the process just sits in a tight loop waiting, this approach wastes CPU time.
- We could use an IPC primitives that blocks a process instead of wasting CPU time when they are not allowed to enter into their critical section.
- such a simple primitive pair is sleep and wakeup.
- Essentially, when a process is not permitted to access its critical section, it uses a system call known as sleep, which causes that process to block.
- The process will not be scheduled to run again, until another process ~~or~~ uses the wakeup system call.
- wake up is called by a process when it leaves its critical section if any other processes have blocked.

2.3.6 Semophores

2.3.7 Monitors:



- A monitor is a collection of procedures, variables and data structures that are grouped together in a special kind of module or package
- Once the process enters in monitor, the process gets locked and remaining process are kept in queue.
- Monitor is a type, or abstract data type, encapsulates private data with public methods to operate on the data
- Processes may call the prodeaures in a monitor whenever they want to but they cannot directly access the monitor's internal data structures from procedures declearred outside the monitor.

2.3.8. Interprocess communication (IPC) mechanisms:

IPC is the mechanism for process to communicate and to synchronize their actions or to allow processes to exchange information. There are numerous reasons for providing an environment or situation which allows process co-operation:

- Information sharing: Since

since some users may be interested in the same piece of information (for example, a shared file), you must provide a situation for allowing concurrent access to that information.

- Computation speedup: If you want a particular work to run fast, you must break it into sub-task where each ~~other~~ of them will get executed in parallel with other task. Note that such a speed-up can be attainted only when the computer has compound or various processing elements like CPUs or I/O channels

- Modularity:

You may want to build the system in a modular way by dividing the system functions into split processes or threads.

- Convenience:

Even a single user may work on many tasks at a time. For eg: a user may be editing, formatting, printing and compiling in parallel

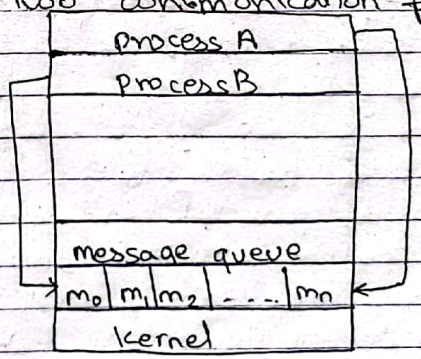
There are two primary models of inter-process communication:

(i) shared memory and
(ii) message passing.

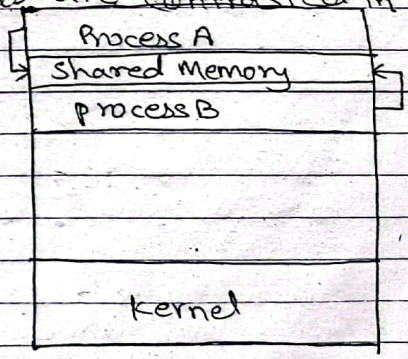
(i) In the shared-memory model, a region of memory which is shared by cooperating processes gets established. Processes can be then able to exchange information by reading & writing all the data to the shared region. In the message-passing form, communication takes place by way of ~~a~~ messages exchanged among the cooperating process.

Advantage of shared memory model is that memory communication is faster as compared to the message passing model. However, shared memory model may create problems such as synchronization & memory protection that need to be addressed.

Two communication ~~pr~~models are ~~contrasted~~ in fig:



a) message passing



b) shared Memory

Message passing is mechanism for processes to communicate and to synchronize their actions without reporting to a shared variable.

This method of IPC uses two primitives.

- send (destination, message);
- receive (source, message);

→ Send calls sends a message to a given destination
→ Receive receives a message from a given source.

If process P and Q needs to communicate, they need to

1. establish a communication link (physical or logical) between them
2. exchange message via send/receive

Processes must name each other explicitly:

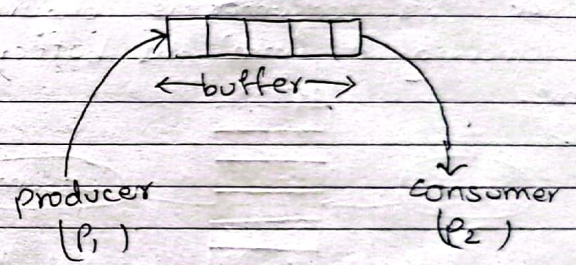
1. Send (P, message); send a message to process P
2. receive (Q, message); receive a message from process Q.

An advantage of message passing model is that it is easier to build parallel hardware. This is because message passing model is quite tolerant of higher communication latencies. It is also much easier to implement that shared memory model. However message passing model has slower communication than shared because the connection ~~to~~ setup takes time.

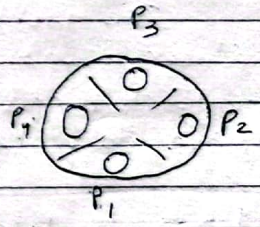
2.3.9 Classical IPC problems

1. Producer consumer problem
2. Dining philosopher problem
3. Sleeping Barber problem
4. Readers and writers problem.

1. Producer consumer problem
   ↳ semaphore.



2. Dining philosopher problem:



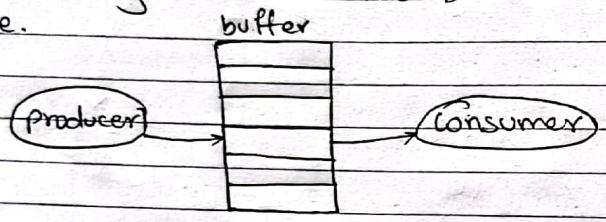
2.3.9 classical IPC problems

(a) Producer consumer problem:

Producer-consumer problem (also known as the bounded-buffer problem) is a multiprocessing synchronization problem.

producer :- The producer's job is to generate a piece of data, put it into the buffer and start again.

consumer: The consumer is consuming the data (i.e removing it from the buffer) one piece at a time.



problem:

The problem is to make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer.

solution to Producer-consumer problem

Producer either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer and it notifies the producer who starts to fill the buffer again consumer can go to sleep if it finds the buffer to be empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer.

(b) Dining philosopher problem:

The dining philosophers problem is a classic synchronization problem which is used to evalute situations where there is a need of allocating multiple resources to multiple processes.

- problem: scenario: Five philosophers sit around a circular table. Each philospher alternates between thinking and eating. There are five forks on the table, one between each pair of adjacent philosophers. To eat, a philosopher needs both the fork on their left and the fork on their right.

- Constraints: A philosopher can only use the forks adjacent to them, and they cannot eat unless they have both forks. After eating, they put down both forks and start thinking again.

Solutions:

- Number the forks (or philosophers) and enforce an order for picking them up. For example, always pick up the lower-numbered fork first. This prevents circular wait condition and thus avoids deadlock.

- Introduce a waiter or an arbitrator that philosopher must ask for permission before picking up fork. The waiter ensures that only a limited number of philosopher can pick up fork simultaneously.

- Use semaphores to control access to the forks. For eg, use a semaphore for each fork and a global semaphore to limit the number of philosophers who can attempt to eat at the same time to four.

(c) Sleeping Barber Problem:

problem: The analogy is based upon a hypothetical barber shop with one barber. There is a barber shop which has one barber, one chair, and n chairs for waiting for customers if there are any to sit on the chair.

- If there is no customer, then the barber sleeps in his own chair.
- When a customer arrives, he has to wake up the barber
- If there are many customers and the barber is cutting then the remaining either wait in the waiting room or leave if no chairs are empty.

Solution: The solution to this problem includes three semaphores. First is for the customer which counts the number of customers present in the waiting room. Second, the barber 0 or 1 is used to tell whether the barber idle or is working. And the third mutex is used to provide the mutual exclusion which is required for the process to execute. In this solution, the customer has the record of the number of chairs in the waiting room then the upcoming customer leaves the barbershop.

When the barber shows up in the morning, he executes the procedure barber, causing him to block on the semaphore customers because it is initially 0. Then the barber goes to sleep until the first customer comes up.

When a customer arrives, he executes customer producer the customer acquires the mutex for entering the critical region, if another customers enters thereafter, the second one will not be able to anything until the first one has released the mutex. The customer then checks the chair in the waiting room if waiting customers are less then the number of chairs then he sits otherwise he leaves and release the mutex.

If the chair is available then customer sits in the waiting room and increments the variable waiting value and also increase the customers semaphore this wakes up the barber if he is sleeping.

At this point, customer and barber are both awake and the barber is ready to give that person a haircut. When the haircut is over, the customer exits the procedure and if there are no customers in waiting room barber sleeps.

(d) Readers and writers problem:

problem:

Consider a situation where we have a file shared between many people.

- If one of the people tries editing the file, no other person should be reading or writing at the same time, otherwise changes will not be visible to him/her.
- However, if some person is reading the file, the

other may read it at the same time.
Precisely in OS we call this situation as the readers-writers problem.

Problem parameters:

- One set of data is shared among a number of process
- Once a writer is ready, it performs its write. Only one writer may write at a time.
- If a process is writing, no other process can read it.
- Readers may not write and only read.

Solution when Reader has the priority over writer.

Here priority means, no reader should wait if the share is currently opened for reading.

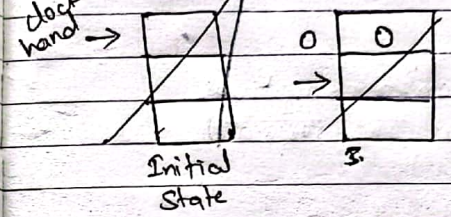
- Semaphore muter is used to ensure mutual exclusion when readcnt is updated i.e when any reader enters or exit from critical section and semaphore wrt is used by both readers and writers.
- readcnt tells the no. of processes performing reading in critical section, initially 0.

writer process:

- writer requests the entry to critical section
- If allowed i.e wait() gives a true value, it enters and performs the write. If not allowed, it keeps on waiting.
- It exists the critical section

0, 4, 1, 4, 2, 4, 3, 4, 2, 4, 0, 4

page frame = 3



2/14

VVImp

2.4 Process scheduling:

2.4.1 Basic concept & Types:-

Types of resources

- preemptive (Share जरूर किसी)
- non-preemptive (Share जरूर नहीं किसी)

2.4.2 Scheduling criteria:-

- ↳ Arrival Time
- ↳ Brust Time (Execution Time)
- ↳ Priority
- ↳ Time quantum

2.4.3 Scheduling Algorithms:-

1. FCFS
   - ↳ First Come First serve
   - ↳ Criteria : Arrival Time
   - ↳ Mode : Non-preemptive
   - ↳ Starvation may arrive or cause

[if arrival time is not given in any algorithm then it is always 0]

disadvantage: convoy effect.

TAT = CT − AT
= completion time − Arrival time

WT = TAT − BT

Using FCFS solve;

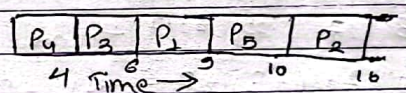
Q.

Process	Arrival time	Brust time
P_1	2	3
P_2	4	6
P_3	1	2
P_4	0	4
P_5	3	1

Soln,

process	AT Arrival time	BT Brust time	WT waiting time	TAT Turn around time
P_1	2	3	7−3=4	9−2=7
P_2	4	6	12−6=6	16−4=12
P_3	1	2	5−2=3	6−1=5
P_4	0	4	4−4=0	4−0=4
P_5	3	1	7−1=6	10−3=7

Gantt chart



Avg TAT = $\frac{7+12+5+4+7}{5} = \frac{35}{5} = 7$

Avg WT = $\frac{4+6+3+0+6}{5} = \frac{19}{5} = 3.8$

Adv: easy to implement

dis: convoy effect

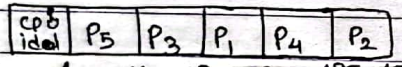
Q.

process	AT	BT
P_1	9	14
P_2	5	12
P_3	2	9
P_4	4	8
P_5	1	3

soln,

process	AT	BT	TAT	WT
P_1	9	14	24	10
P_2	5	12	42	30
P_3	2	9	11	2
P_4	4	8	31	23
P_5	1	3	3	0

Gantt chart



Avg TAT = $\frac{24+42+11+31+3}{5} = \frac{111}{5} = 22.2$

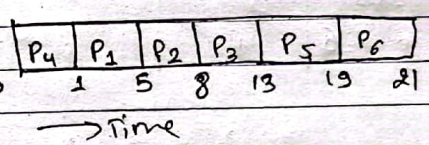
Avg WT = $\frac{10+30+2+23}{5} = \frac{65}{5} = 13$

8.

Process	AT	BT	TAT	WT
P_1	1	4	5-1 = 4	0
P_2	1	3	8-1 = 7	4
P_3	2	5	13-2 = 11	6
P_4	0	1	1-0 = 1	0
P_5	2	6	19-2 = 17	11
P_6	3	2	21-3 = 18	16

~~Gnt~~ $P_4, P_1, P_2, P_3, P_5, P_6$

Grant chart:



Avg TAT = $\frac{4+7+11+1+17+18}{6} = \frac{58}{6} = 9.667$

Avg WT = $\frac{0+4+6+0+11+16}{6} = \frac{37}{6} = 6.167$

(2) SJF:

↳ shortest Job First

↳ criteria : Brust Time

↳ Mode: non-preemptive

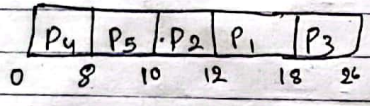
Q.

Process	AT	BT	TAT	WT
P_1	2	6	18-2=16	10
P_2	5	2	12-5=7	5
P_3	1	8	26-1=25	17
P_4	0	8	8-0=8	0
P_5	4	2	10-4=6	4

Ram

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

Gantt chart



Time →

Avg TAT = $\frac{16+7+25+8+6}{5}$ =

" WT = $\frac{10+5+17+0+4}{5}$ =

Ad: no convoy effect

dis: can't be implemented in real-time due to BT.

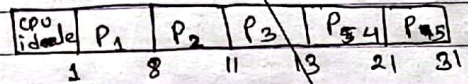
Q.

Process	AT	BT	TAT
P_1	1	7	8-1=7
P_2	3	3	11-3=8
P_3	6	2	13-6=7
P_4	7	10	31-7=24
P_5	9	8	

Gantt chart

Ram

~~P_2, P_3, P_5~~, P_4, P_5



→ Time

Avg TAT :-

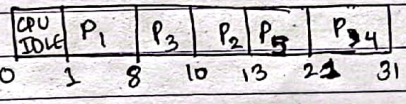
Q

Process	AT	BT
P_1	1	7
P_2	3	3
P_3	6	2
P_4	7	10
P_5	9	8

Gantt chart

Ram

$P_2, P_3, P_4, 5$



2/17

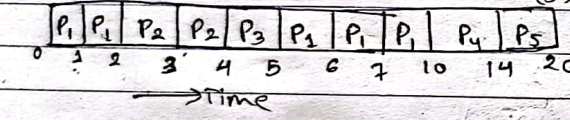
(3) SRTF ↳ Also known as shortest process next
↳ Shortest ~~Time~~ Remaining Time First
↳ SJF with preemptive mode
↳ Criteria: Burst Time
Mode: Preemptive

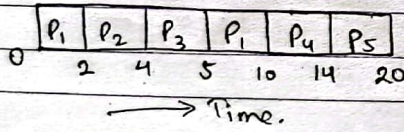
process	Burst Time	Arrival Time	TAT	WT
P_1	7	0		
P_2	2	2		
P_3	1	4		
P_4	4	6		
P_5	6	5		

RAM

0: P_1, 1: , 2: P_1 (3) P_2 (2) 3: P_1 (5) P_2 (2)
4: P_1 (5) P_3 (1) 5: P_1 (5) P_5 (6) 6: P_1 (4) P_5 (6) P_4 (4)
7: P_1 (3) P_5 (6) P_4 (4)

Gantt chart:





Avg TAT:

Avg WT:

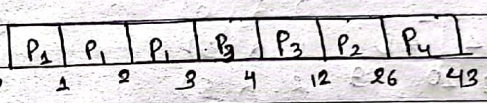
Same as SJF

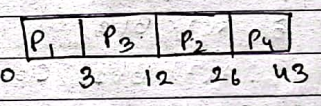
2/17

Q.

Process	BT	AT	Priority
P_1	3	0	2
P_2	14	1	1
P_3	9	2	3
P_4	17	3	4

Gantt chart





RAM

0: P_1 (3)
1: P_1 (2) P_2 (14)
2: P_1 (3) P_2 (14) P_3 (9)
3: ~~P_1~~ (2) P_2 (14) P_3 (9) P_4 (17)
4: P_2 (14) P_3 (8) P_4 (17)

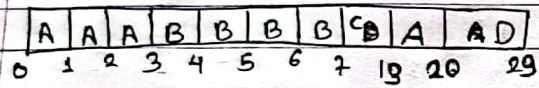
Avg TAT

TAT = CT - AT WT = TAT - BT

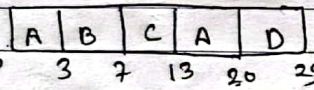
Q.

Process	BT	AT	TAT	WT
A	10	0	20-0 = 20	10
B	4	3	7-3 = 4	0
C	6	6	13-6 = 7	1
D	9	7	29-7 = 22	13

Gantt chart



→ Time



→ Time

RAM

0: A(10)
1: A(9)
2: A(8)
3: A(7), B(4)
4: A(7), B(3)
5: A(7), B(2)
6: A(7), B(1), C(6)
7: A(7), C(6), D(9)

Avg TAT = $\frac{20+3+7+22}{4} = 13$

Avg WT = $\frac{10+0+1+13}{4} = 6$

TAT = CT - AT WT = TAT - BT

(4) Priority Scheduling
↳ Criteria: Priority
↳ Mode: Both
preemptive non-preemptive

Ad:
Dis: can cause starvation if priority is not shared fairly.

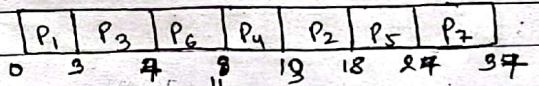
From non-preemptive.

Process	AT	BT	Priority
P_1	0	3	2
P_2	2	5	6
P_3	1	4	3
P_4	4	2	5
P_5	6	9	7
P_6	5	4	4
P_7	7	10	10

RAM
0: $P_1(3)$, 1: $P_3(4)$, 2
RAM:
$P_1, P_3, P_6, P_4, P_2, P_5$

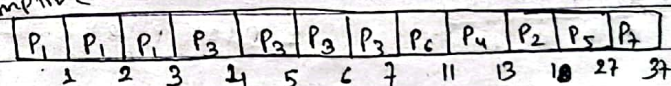
Gantt chart

Non-preemptive

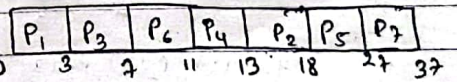


→ Time

preemptive



→ Time



→ Time

RAM
0: $P_1(1)$
1: $P_1(2)$ $P_3(4)$
2: $P_1(1)$ $P_3(4)$ $P_2(5)$
3: $P_3(4)$ $P_2(5)$
4: $P_3(3)$ $P_2(5)$ $P_4(2)$
5: $P_3(2)$ $P_2(5)$ $P_4(2)$ $P_6(4)$
6: $P_3(1)$ $P_2(5)$ $P_4(2)$ $P_6(4)$ $P_5(9)$
7: $P_2(5)$ $P_4(2)$ $P_6(4)$ $P_5(9)$ $P_7(10)$

2/17

TAT = C.T − A.T

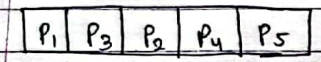
WT = TAT − BT

~~From preemptive~~

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

non-preemptive:

Gantt chart



RAM

0: $P_1(4)$

1: $P_1(3)$ $P_2(3)$

2: $P_1(2)$ $P_2(3)$ $P_3(1)$

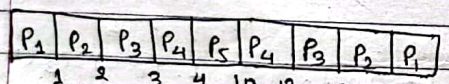
3: $P_1(1)$ $P_2(3)$ $P_3(1)$ $P_4(5)$

4: $P_2(3)$ $P_3(1)$ $P_4(5)$ $P_5(2)$

2/18

Priority with preemptive mode:

process	AT	BT	Priority
P_1	0	2	5
P_2	1	8	4
P_3	2	4	3
P_4	3	3	2
P_5	4	6	1 (high)



Avg TAT :

Avg WT :

RAM

0: $P_1(2)$

1: $P_1(1)$, $P_2(8)$

2: $P_1(1)$, $P_2(7)$, $P_3(4)$

3: $P_1(1)$, $P_2(7)$, $P_3(3)$, $P_4(3)$

4: $P_1(1)$, $P_2(7)$, $P_3(3)$ $P_4(2)$ $P_5(6)$

2/18

VAMP

(5) Round Robin

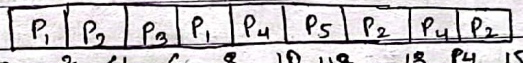
Criteria : Time Quantum

Mode : Preemptive

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

Time quantum = 2

Gantt chart



Avg TAT :

Avg WT :

$P_2 \mid P_3 \mid P_1$

ready queue

RAM

0: P_1

1: P_2

2: P_2 P_3 $P_1(2)$

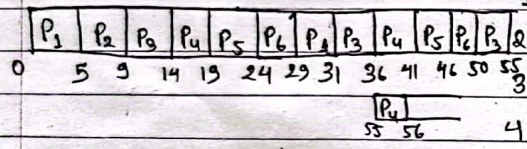
3: P_3 P_3 $P_1(2)$ $P_4(3)$ $P_5(1)$

4: $P_3(2)$ $P_1(2)$ $P_4(3)$ $P_5(1)$ $P_2(3)$ $(P_4(1))$ $P_2(1)$

2/18

Process	AT	BT
P_1	0	7
P_2	1	4
P_3	2	15
P_4	3	11
P_5	4	20
P_6	4	9

Time quantum = 5



RAM

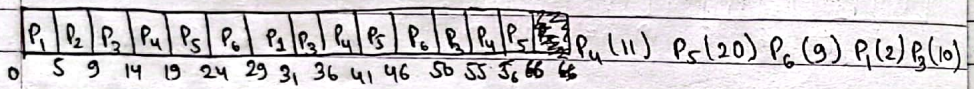
0: P_1 ✗

1: $P_2(4)$ ~~$P_3(15)$ P~~

2: $P_2(4)$ $P_3(15)$

3: P_2 ✗ P_3 P_4

4: $P_2(4) P_3(15) P_4(11) P_5(20) P_6(9) P_1(2)$



$P_4(11)$ $P_5(20)$ $P_6(9)$ $P_1(2)$ $P_3(10)$

$P_5(20)$ $P_6(9)$ $P_1(2)$ $P_3(10)$ $P_4(6)$

$P_6(9)$ $P_1(2)$ $P_3(10)$ $P_4(6)$ $P_5(15)$

Avg TAT :

$P_1(2)$ $P_3(10)$ $P_4(6)$ $P_5(15)$ $P_6(4)$

$P_4(6)$ $P_5(15)$ $P_6(4)$ $P_3(5)$

Avg WT :-

$P_5(15)$ $P_6(4)$ $P_3(5)$ $P_4(1)$

$P_6(4)$ $P_3(5)$ $P_4(1)$ $P_5(10)$

2/18

(6) Real Time scheduling
(7) Guaranteed scheduling
(8) Lottery scheduling

Priority Fair share

2/18

2/18

2.5 Thread scheduling:

Thread scheduling in OS involves managing the execution of multiple threads within a process, ensuring efficient use of cpu resources and meeting performance requirements. Threads are light weight units of execution within a process, sharing the same memory space but having separate execution states.

Types of Thread scheduling:
↳ Preemptive scheduling
↳ Non-preemptive scheduling

Algorithms are same as process scheduling.

VVIMP

2.6 Deadlock:

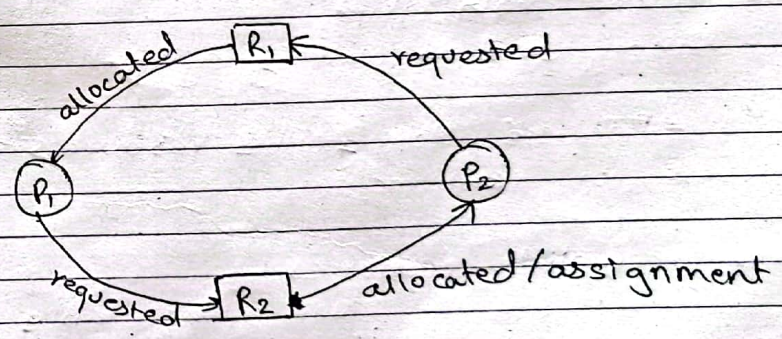
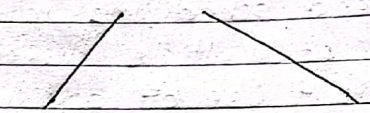


Fig:- deadlock cond^n

2/18

2.6.1 System Resources:



In OS, resources can be classified as preemptable and non-preemptable based on wet whether they can be taken away from a process without causing issue

1. Preemptable resources:

- Preemptable resources are those that can be taken away from the process without causing any problems. The process can continue executing once the resource is available again. Eg. of preemptable resource include
  - CPU Time (multitasking)
  - Memory (virtual memory system)

- Processes holding the resources can be paused or resumed.

2. Non-preemptable Resources:

Non-preemptable resources are those that cannot be safely taken away from the process without causing potential problems or inconsistencies. The process must voluntarily release these resources.

2/18

Eg: includes:- Printer
- file locks
- database transaction

2/21

2.6 Deadlock:

When several processes ~~complete~~ compete for a finite no. of resources, a situation may arise where a process request a resource and the resource is not available at that time, in which case the process enters a wait state. It may happen that waiting processes will never again change state, because the resources that they have requested are held by other waiting processes. This situation is called a deadlock.

IMP:

Necessary Condition:

A deadlock situation can arise if the following 4 cond^{n}s hold simultaneously in a system:

(1) Mutual Exclusion:

At least one resource must be held in a non-sharable mode i.e only one proce at a time can use the resource. If another proce request that resource, the requesting process must be delayed until the resource has been releas

(2) Hold and Wait:

There must exist a process

that is holding atleast one resource and is waiting to acquire additional resources that are currently being held by other processes.

③ No preemption:

Resources cannot be preemptive i.e a resource can be released only voluntarily by the process holding it, after that process has completed its task.

④ Circular wait:

There must exist a set $\{P_0, P_1, P_2 \ldots P_n\}$ of waiting processes, such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by P_2, ..., P_{n-1} is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0.

All ~~for~~ four condition must hold for deadlock

2.6.2 Methods for handling deadlocks:

There are three different methods for dealing with deadlock problem:

① We can use a protocol to ensure that the system will never enter a deadlock state.

② We can allow the system to enter a deadlock state and then recover.

③ We can ignore the problem and pretend that deadlocks never occur in the system. (Ostrich algorith

soln for ①: i Deadlock prevention
ii Deadlock avoidance
② i. Deadlock detection
ii. Deadlock recovery
③ i. Ostrich algorithm)

①

(i) Deadlock Prevention:

Deadlock prevention is a set of methods for ensuring that atleast one of the nece any conditions cannot hold. These methods prevent deadlocks by constraining (restricting) how request for resources can be made.

④ Mutual ~~cond~~ exclusion:

- The mutual exclusion con must hold for non-sharable resources. For eg:-

A printer can't be simultaneously shared by several processes.

- Sharable resources, on the otherhand, donot require mutually exclusive access, and thus can't be involved in a deadlock. Read only files are good eg. examples of a sharable resource.

- In general, it is not possible to prevent deadlocks by denying the mutual exclusion condition i.e some resources are intrinsically non-sharable.

(b) Hold and wait:

- To ensure that the hold and wait cond^n never occurs in a system we must guarantee that whenever a process request a resource, it doesn't hold any other resources.

- One protocol that can be used requires each process to request and be allocated all its resources before it begins execution.

- An alternative protocol allows a process to request resources only when the process has none. A process may request some resources and use them. Before it can request any additional resources, however, it must release all the resources that it is currently allocated.

- There are 2 main disadvantages of these protocols: i. resources utilization may be low
ii. Starvation

(c) No preemption:

- The no preemption cond^n states that resources allocated to a process can't be forcibly taken away from that process. To prevenot deadlock, this cond^n must be addressed.

- If a process holding some resources request ~~additional~~ additional resources that are not immediately available, the system can preempt the resources that the process is currently holding and allocate them to other processes.

- The process whose resources were preempted is then put into a waiting state. When the resources is originally held become available again, the process can resume its execution.

(d) Circular wait:-

- One way to ensure that the circular wait condition never holds is to impose a total ordering of all resource types and to require that each process request resources in an increasing order of enumeration.

- For eg: If the set of resource type R includes ~~tape~~ tape drives, disk drive and printers then the function F might be defin as follows:

F (tape drives) = 1
F (disk drives) = 5
F (printers) = 12

claim edge ----->

(ii) Deadlock avoidance:-

It requires that the OS be given in advance additional information concerning which resources a process will request and use during that process lifetime.

Deadlock prepent prevention algorithm prevent deadlock by restraining how request can be made. The restraints ensure that atleast one of the necessary conditions for deadlock cannot occur. Possible side effects of preventing deadlocks by this method are:-
low ~~devid~~ device utilization, reduce system through.puter and potential process starvation.

An alternative method for avoiding deadlocks is to require that a user (process) supply additional information about how requested resources are to be requested. The various algorithm differ in the amount and type of information required. If we have a resource allocation system with only one ~~s~~ instance of each resource type, we can use the variant of the resource allocation graph (RAG) for deadlock avoidance. In addition to the request and assignment edges, we introduce new type of edge called a claim edge.
A claim edge $P_i \rightarrow R_j$ indicates that process P_i may request ~~resore~~ resource R_j at some time in the future. This edge resembles a request

edge in direction but is represented by a ~~be~~ dash line (- - - ->). When process 'P_i' request resource 'R_j', the claim edge $P_i \rightarrow R_j$ is converted to a request edge. Similarly when a resource 'R_j' is released by 'P_i', the assignment as $R_j \rightarrow P_i$ is reconverted to a claim edge ($P_i \rightarrow R_j$)

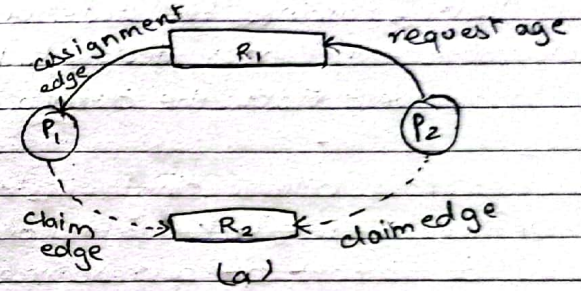
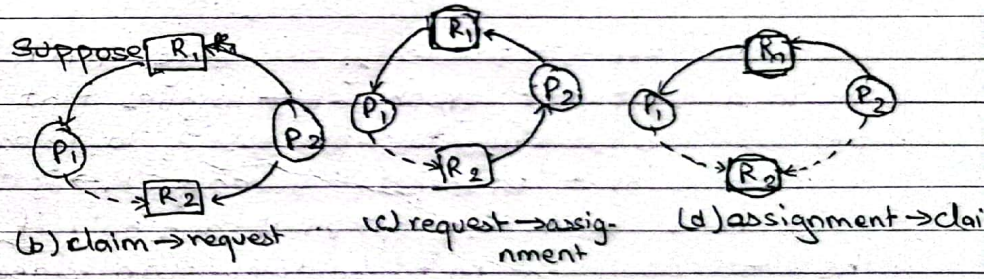


Fig: RAG for Deadlock Avoidance



Suppose that process P_i request resources (R_j). The request can be granted only if ~~co~~converting the request as $P_i \rightarrow R_j$ to an assignment as $R_j \rightarrow P_i$ does~~r~~ result in the ~~for~~ formation of a cycle in the resource allocation graph. If no cycle exist then the allocation of resource will leave the system in a safe state.

2(i) Deadlock Detection:

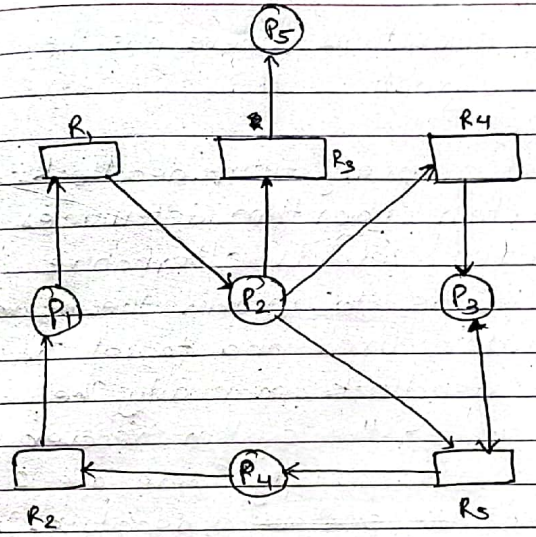
If a system doesn't employ either a deadlock prevention or deadlock avoidance algorithm, then a ~~deto~~ deadlock situation may occur. In this environment, the system must provide of of these ~~s~~ option

- An algorithm that examines the state of a system to determine whether a deadlock has occured
- An algorithm to recover from the deadlock.

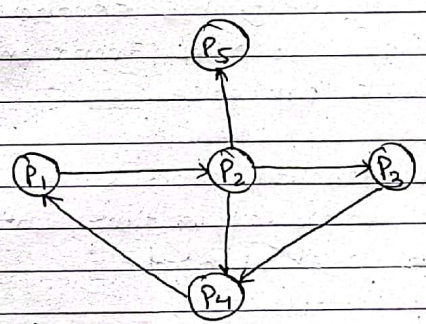
If all resources have only a single instance, then we can define a deadlock detection algorithm that uses a variant of a resource allocation graph, called a wait-for graph.

More ~~precisi~~ precisely, a edge from P_i to P_j in a wait for graph, ~~emp~~ implies that process P_i is waiting for process P_j to release a resource that P_i needs. An edge $P_i \rightarrow P_j$ exist in a wait for graph if and only if the corresponding RAG contains two edges $P_i \rightarrow R_q$ and $R_q \rightarrow P_j$ for some resource R_q.

2/24



(a) RAG



(b) corresponding wait-for graph

A deadlock exist in the system if and only if the wait-for graph contains a cycle. To detect deadlock the system needs to maintain wait-for graph and ~~pre~~ periodically to invoke an algorithm that serches for a cycle in the graph.

An algorithm to detect a cycle in a graph requires

2/24

an order of $O[n^2]$ operations where n is the no. of vertices in a graph.

(ii) Deadlock recovery or rerovery from deadlock:-

When a detection algorithm determines that a deadlock exist, several alternative exist. One possibility is to inform the operator that a deadlock has occured and to let the operator deal with deadlock manually. The other possibility is to let the system recover from the deadlock automatically. There are 2 options for breaking ~~the system~~ a deadlock: One soln is simply to abort one or more processes to break the circular wait. The second option is to preempt some resources from one or more of the deadlock processes.

a. Process termination:
   - ↳ Abort all deadlocked process
   - ↳ Abort one process at a time until the deadlock cycle is eliminated.
   - ↳

b. Resource preemption:
   - ↳ If preemption is required to deal with deadlocks, then three matters need to be addressed.
     - Selection of a victim
     - Roll back
     - Starvation

2/25

Need = MAX - allocation
need <= Available — to check if it is safer or not

Banker's Algorithm:-

Q. Consider a system that contains five processes P_1, P_2, P_3, P_4, P_5 and the three ~~region~~ resources type A, B and C. following are the resource types:
A has 10, B has 5 and resource type C has 7 instanc

Process	Allocation			MAX			Available		
	A	B	C	A	B	C	A	B	C
P_1	0	1	0	7	5	3	3	3	2
P_2	2	0	0	3	2	2			
P_3	3	0	2	9	0	2			
P_4	2	1	1	2	2	2			
P_5	0	0	2	4	3	3			

(i) Find the need matrix
(ii) Is it in safe state? If yes, what is the sequence soln,

need = max - allocation

Process	MAX			allocation			need			available		
	A	B	C	A	B	C	A	B	C	A	B	C
P_1	7	5	3	0	1	0	7	4	3	3	3	2
P_2	3	2	2	2	0	0	1	2	2	5	3	2
P_3	9	0	2	3	0	2	6	0	0	7	4	3
P_4	2	2	2	2	1	1	0	1	1	7	4	5
P_5	4	3	3	0	0	2	4	3	1	7	5	5

For process P_1,
(7, 4, 3) <= (3, 3, 2) (False)
For process P_2,
(1, 2, 2) <= (3, 3, 2) (True)

Safe sequence : P_2

Available = available + Allocation of P_2

$= (3,3,2) + (2,0,0)$

$= (5,3,2)$

For process P_3.

$(6,0,0) <= (5,3,2)$ (False)

For process P_4

$(0,1,1) <= (5,3,2)$ (True)

Safe sequence: $P_2 \rightarrow P_4$

available = available + allocation of P_4

$= (5,3,2) + (2,1,1)$

$= (7,4,3)$

For process P_5

$(4,3,1) <= (7,4,3)$ (True)

safe sequence: $P_2 \rightarrow P_4 \rightarrow P_5$

available = available + allocation of P_5

$= (7,4,3) + (0,0,2)$

$= (7,4,5)$

For process P_1.

$(7,4,3) <= (7,4,5)$ (True)

Safe sequence: $P_2 \rightarrow P_4 \rightarrow P_5 \rightarrow P_1$

available = available + allocation of P_1

$= (7,4,5) + (0,1,0)$

$= (7,5,5)$

For process P_3

$(6,0,0) <= (7,5,5)$ (True)

Safe sequence: $P_2 \rightarrow P_4 \rightarrow P_5 \rightarrow P_1 \rightarrow P_3$

available = available + allocation of P_3

$= (7,5,5) + (3,0,2)$

$= (10,5,7)$

Yes, the snapshot is in safe state.

The safe sequence is $P_2 \rightarrow P_4 \rightarrow P_5 \rightarrow P_1 \rightarrow P_3$.

Q. Consider a system that contains four processes P_1, P_2, P_3, P_4 and the 3 resource type A, B and C. Following are the resource types:

A

Process	Max			allocation			available		
Process	A	B	C	A	B	C	A	B	C
P_1	8	6	3	2	1	0	4	3	2
P_2	9	4	3	1	2	2			
P_3	5	3	3	0	2	0			
P_4	4	2	3	3	0	1			

A has 10, B has 8 and C has 5 instances.

(i) Find need state.

(ii) Is it in safe state? If yes what is the sequence

Process	max			allocation			need			available		
	A	B	C	A	B	C	A	B	C	A	B	C
P_1	8	6	3	2	1	0	6	5	3	4	3	2
P_2	9	4	3	1	2	2	8	2	1	7	3	3
P_3	5	3	3	0	2	0	5	1	3	7	5	3
P_4	4	2	3	3	0	1	1	2	2	9	6	3

Need = Max - allocation

need <= available

For process P_1:

$(6,5,3) <= (4,3,2)$ (false)

For process P_2:

$(8,2,1) <= (4,3,2)$ (false)

For process P_3:

$(5,1,3) <= (4,3,2)$ (false)

For process P_4:

$(1,2,2) <= (4,3,2)$ (true)

safe sequence: P_4

available = available + allocation of P_4

$= (4,3,2) + (3,0,1)$

$= (7,3,3)$

For process P_1:

$(6,5,3) <= (7,3,3)$ (false)

For process P_2:

$(8,2,1) <= (7,3,3)$ (false)

For process P_3:

$(5,1,3) <= (7,3,3)$ (true)

safe sequence: $P_4 \rightarrow P_3$

available = $(7,3,3) + (0,2,0)$

$= (7,5,3)$

For process P_1:

$(6,5,3) <= (7,5,3)$ (true)

safe sequence: $P_4 \rightarrow P_3 \rightarrow P_1$

available = available + allocation of P_1

$= (7,5,3) + (2,1,0)$

$= (9,6,3)$

For process P_2

$(8,2,1) <= (9,6,3)$ (true)

safe sequence: $P_4 \rightarrow P_3 \rightarrow P_1 \rightarrow P_2$

available = available + allocation of P_2

$= (9,6,3) + (1,2,2)$

$= (10,8,5)$

∴ It is in safe state.

∴ The safe sequence is $P_4 \rightarrow P_3 \rightarrow P_1 \rightarrow P_2$

2.6.3 Security & Protection

Goals of Protection

1. Prevent misuse
2. Ensure correct usage
3. Reliability & safety

Principles of Protection:

1. Least privilege
2. Separation of privilege
3. Fail safe Defaults
4. Economy of mechanism
5. complete mediation
6. open design
7. Separation of duties.

Domain of protection:

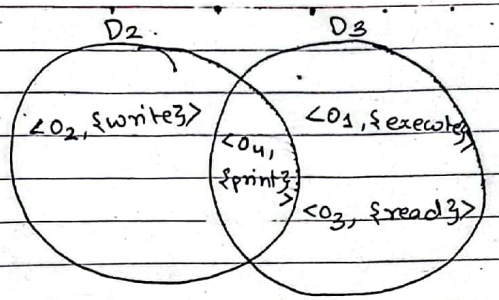
<object_name, rights_set>

ordered collection

Eg: <O1, {print}>

D1

<O3, {read, no write}>
<O2, {read, write}>
<O1, {execute}>



vimp

Access Matrix:

object / Domain	F1	F2	F3	Printer
D1	read		read write	
D2				Print
D3		read	execute	
D4	read write		read write	

(a) Access Matrix

any process running in one domain can access only the resource ~~instruction~~ provided by the domain

Eg: If Pi is the process running in domain D3, the process can read the resource F2 and execute F3 but can't use F1 and Printer

object / domain	F_1	F_2	F_3	Printer	D_1	D_2	D_3	D_4
D_1	read		read			switch	switch	
D_2				Print			switch	switch
D_3		read	execute					
D_4	read write		read write			switch		

(b) Access matrix with domain object

Switch - switch from one domain to another

Eg: If a process P_1 is running in domain D_1 and it needs to be executed then, the process in domain D_1 is switch to domain D_3.

object / domain	F_1	F_2	F_3
D_1	execute		write *
D_2	execute	read *	execute
D_3	execute		

object / Domain	F_1	F_2	F_3
D_1	execute		write *
D_2	execute	read *	execute
D_3	execute	read	

(c) Access matrix with copy matrix

if * is written in any instruction of any domain, then it can be copied in any domain in same column (resource)

obj / Domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		read * owner	read * owner write *
D_3	execute		

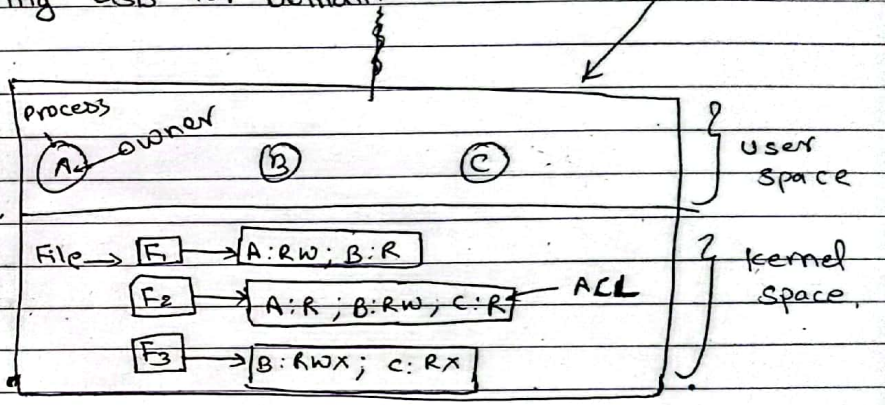
obj / Domain	F_1	F_2	F_3
D_1	owner execute		
D_2		owner read * write *	read * owner write *
D_3		write	write

(d): Access matrix with owner rights.

Here, D_1 is owner of F_1 which can perform all operations.

Implementation of Access matrix:-

1. Global Table
   <domain, object, access_rights>
2. Access Lists for objects (access control List) → (column view of access matrix)
3. Capability Lists for Domain (row view of access matrix)



HW

Access matrix
Access control list
Capability List.

3/13

Access Matrix:-

Access matrix is a security model of protection state in computer system. It is represented as a matrix. Access matrix is used to define the rights of each process executing in the domain with respect to each object. The rows of a matrix represent domains and columns represent object. Different types of rights are:

1. Read: This is a right given to a process in a domain to read the file.
2. Write: Process in domain can write into the file
3. Execute: Process in domain can execute the file
4. Print: Process in domain only has access to printer.

Some times, domain can have more than one right.

Object / domain	F_1	F_2	F_3	Printer
D_1	read		read	
D_2				Print
D_3		Read	execute	
D_4	read write		read write	

Fig: Access Matrix

- There are 4 domains and three object with one ~~por~~ printer.
- The process executing in D_1 can read F_1 and F_2, D_2 can print the file, D_3 can read to file F_2 and execute F_3 and finally D_4 can read and write to file F_1 and F_3

3/13

Object / domain	F_1	F_2	F_3	Printer	D_1	D_2	D_3	D_4
D_1	read		read				switch	
D_2				print				switch
D_3		read	execute			switch		
D_4	read write		execute read write		switch			

(b) Access with domain object.

When we use switch ~~opera~~ a process from one domain is switched to another. We execute switch operation on an object. According to above matrix (b), a process executing in domain D_2 can be switched to domain D_4 and can read and write the process.

Object / domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

Object / Domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(c) Access matrix with copy matrix.

When * is introduced with the rights than it can be copied in any domain in same column (resource) object According to above matrix (c) write ~~can~~ in F_3 D_3 can be copied to any place in F_3.

Obj / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write*
D_3	execute		

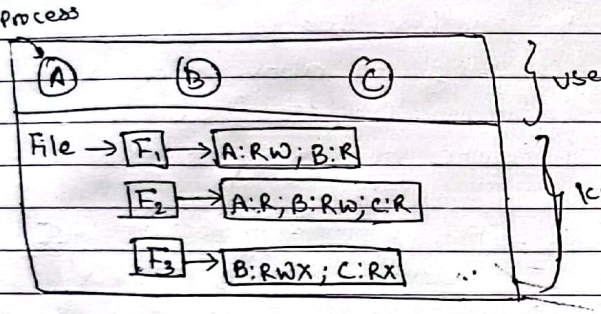
(d) Access matrix with owner rights

Here, when owner is introduced in any domain of any object then the domain is able to change the rights in particular object / resources.

3/13

Access Control List:

Access Control List is essentially a table or list associated with resources (such as files, directory, or network device) that specifies which subject (user, groups or systems) are allowed or denied access to the resources and what operations they can perform.



From fig we can conclude that the process A can read from file, F1 & F2; write to file F1. & B can Read from file F1, F2 & F3, write from to F2 and F3, and execute F3. And similarly, C can read from F2 and F3 and execute F3.

Capability List:

A capability list is a table maintained for each subject (user, process, etc) that specifies the object (resources) the subject can access and the operations it can perform on those objects.

Binding of instructions:
1. compile time
2. load time
3. execution tiom

3/13

Unit: III

Storage Management

3.1 Memory Management:

MMU - memory Management unit.

logical Address: CPU ~~registered~~ generated, temporay

physical Address: Actual address., permanent

3.1.1: Logical and physical address space:-

An address generated by the CPU is commonly reffered to as a logical address, whereas an address seen by the memory unit i.e the one loaded into the memory address register of the memory is commonly reffered to as physical addres

Bind

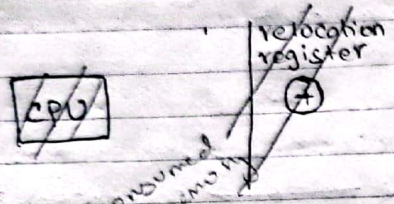
The compile time and load time address binding methods result in an environment where the logical and physical address are the same. However, the execution time address binding scheme results in an environment where the logical and physical address differ. In this case, we usually refer to the logical address as a virtual address.

The set of all logical addresses generated by a program is a logical address space. The set of all physical addresses corresponding to these logical address is a physical address space.

The runtime mapping from virtual to physical address is done by the memory Management unit (MMU), which is a hardware device.

3/13

CPU → MMU → Mery
↳ logical add — convert to physical address



memory consumed in memory by OS.

Station → 14000 to ∞

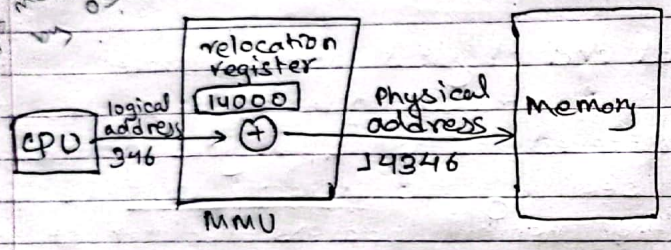
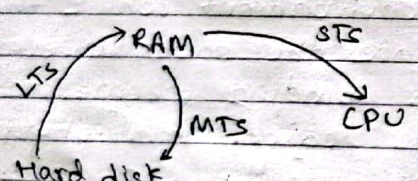


Fig: Dynamic relocation using a relocation register.

3.1.2 Swapping:



scheduler.
1. Long Term scheduler
2. Mid Term scheduler
3. Short Term scheduler

3/13

backing store → hard disk

A process needs to be in ~~memory~~ memory to be executed. A process however can be swapped out of memory to a backing store, and then brought back into memory for continued execution.

Eg: Assume a multiprogramming environment with a round Robin se scheduling algorithm. When a quantum expires, the memory manager will start to swap out the process that just finished and to swap in another process to the memory space that has been freed.

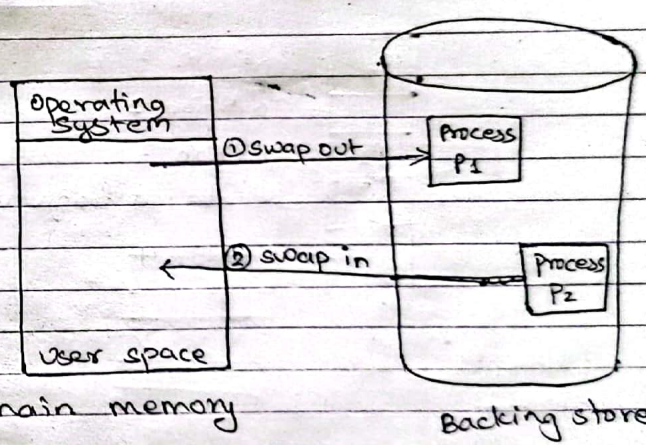


Fig: Swapping of two process using is a disk as a backing store

Similarly for priority scheduling algorithm, memory manger will start to roll out the process and to roll in another process to the memory space

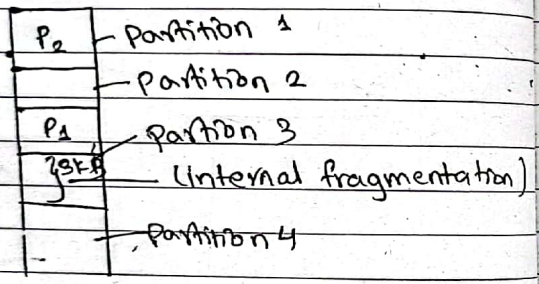
8.1.3 ~~Continuous~~ Contiguous Memory Allocation:-

It means that each logical object is placed in a set of memory locations with strictly ~~cons~~ consecutive address.

Memory partitioning:-

1. Fixed Partitioning:-

Main Memory is divided into a number of static partitions at system generation time. A process may be loaded into a partition of equal or greater size.



Adv.
- Simple to implement
- little OS overhead

Disad.
- Inefficient use of memory due to internal fragmentation.

Internal fragmentation:

The ~~phenn~~ phenomenon in which there is wasted space internal to a partition due to the fact that the block of data loaded is smaller than the partition. is ~~reff~~referred to as internal fragmentation.

2. Dynamic / Variable partitioning:-

To overcome some of the difficulties with ~~f~~ fixed partitioning, an approach known as dynamic partitioning was developed. The partitions are of variable length and the number. When the process is ~~bough~~ brought into main memory, it is allocated exactly as much memory as it requires and no more.

Eventually it leads to a situation in which there are a lot of small holes in memory. As time goes on, memory becomes more and more fragmented and memory utilization declines. This phenomenon is ~~referred~~ referred to as external fragmentation, indicating that the memory that is external to all partition becomes increasingly fragmented.

An example using 64MB of main memory is shown in figure:

3/14

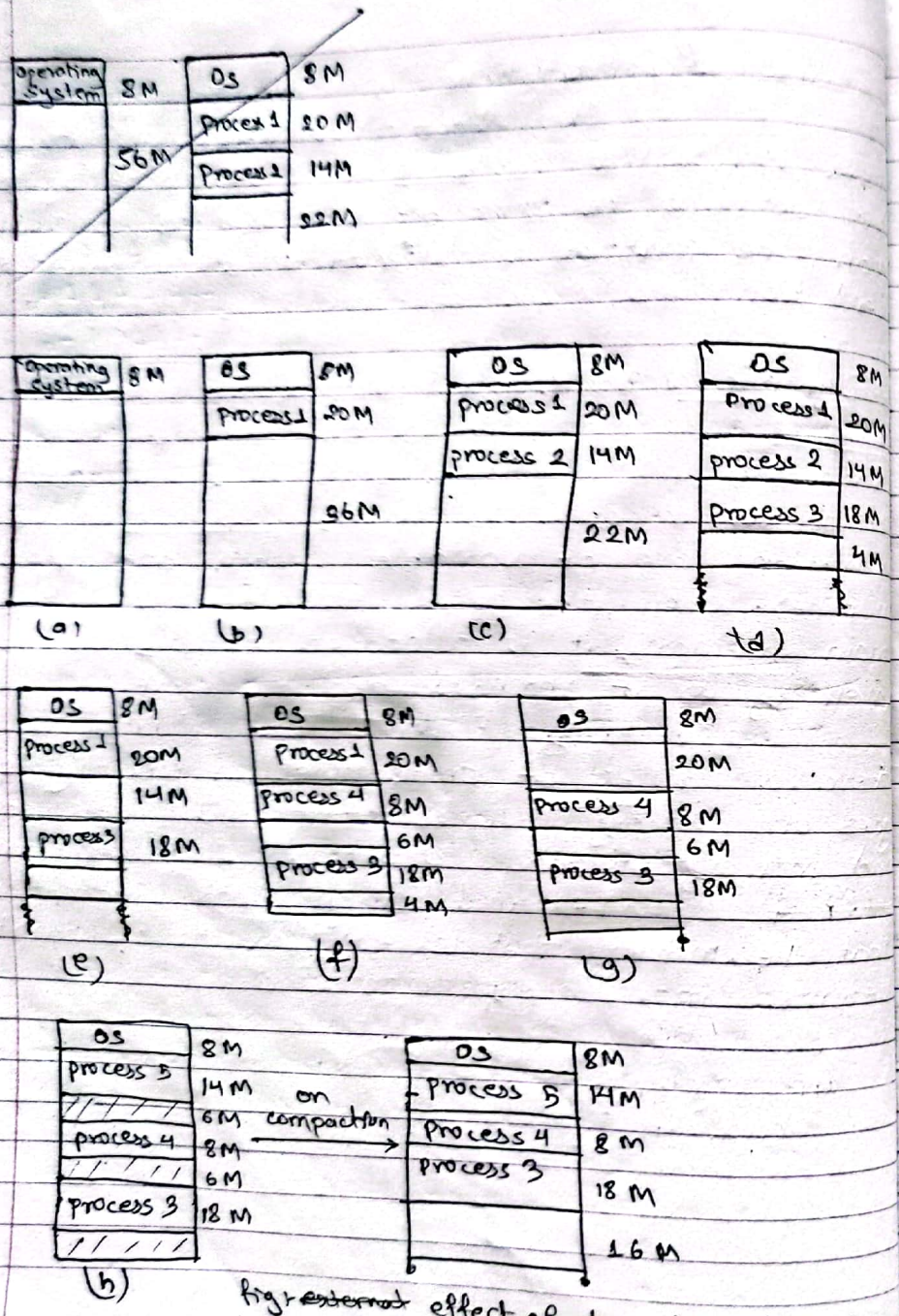


fig: ~~external~~ effect of dynamic partitioning

3/14

One technique for overcomming external fragmentation is compaction. From time to time, the OS shifts the processes so that they are contiguous and all of the free memory is together in one block.

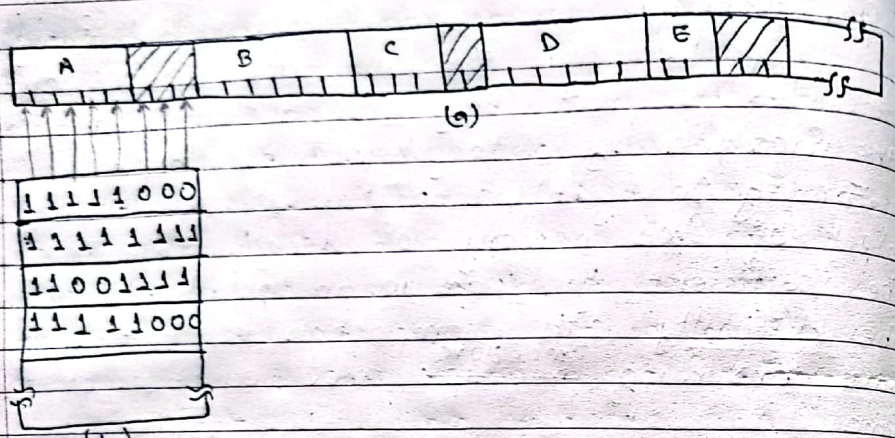
For eg:- In fig (h) compaction will result in a block of free memory of length 16M. This may be sufficient to load in an additional process. The difficulty with compaction is that ~~is~~ it is a time consuming procedure and wasteful of processor time.

Memory Management with Bitmaps:

When memory is assigned dynamically, the OS must manage it. With a Bitmap, memory is divided up into allocation units, perhaps as small as a few words and perhaps as large as several kilo Bytes (KB). Corresponding to each allocation unit is a bit in the bitmap, which is '0' if the unit is free and '1' if it is occupied or vice versa.

Fig below shows part of memory and the corresponding bitmap.

compaction vs coalescing



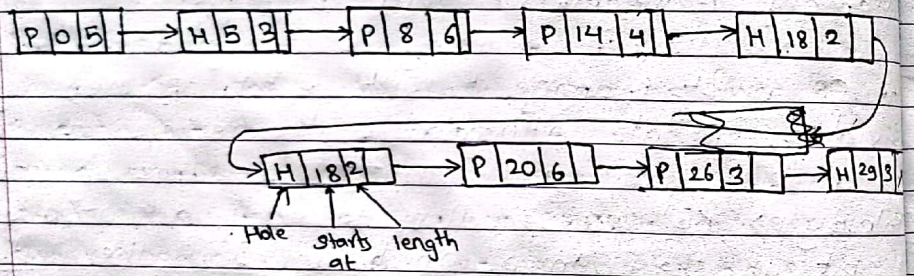


fig (a): a part of memory with 5 process and 3 holes. The tick marks shows memory allocation units. The shaded regions are free.

fig (b): The corresponding bitmap

fig (c): The same information as a list.

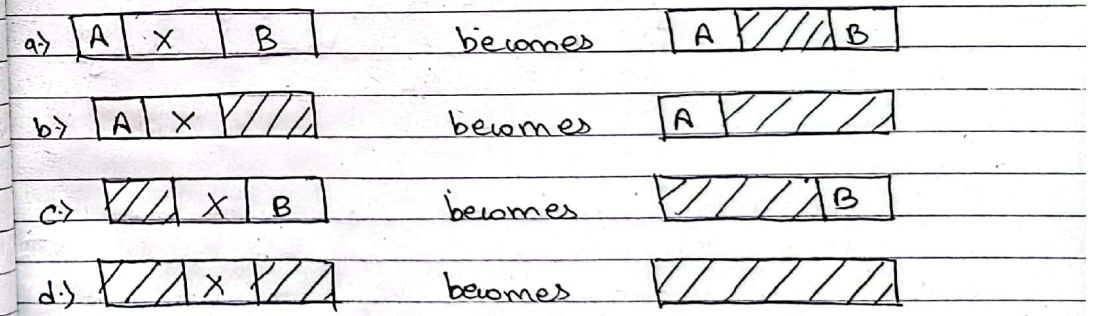
3/14

The size of the allocation unit is an important design issue. The smaller the allocation unit, the larger the bit map. The main problem with is is that when it has been decided to bring a k unit process into the memory, the memory manager must search the bitmap to find a run of k consecutive '0' bits in the map. Searching a bit map for a run of a given length is a slow operation.

3/17

Memory Management with Linked Lists:-

Another way of keeping track of memory is to maintain a linked list of allocated and free segments where a segment is either a process or a hole between two processes.



3/17

Memory Allocation Algorithms:

1. First Fit :-
   - The simplest algorithm is first fit.
   - The process manager scan along the list of segments until it finds a hole that is big enough. The hole is then broken up into two pieces, one for the process and one for the unused memory, except in the statistically unlikely case of an exact fit.
   - First fit is the fast algorithm because it searches as little as possible.

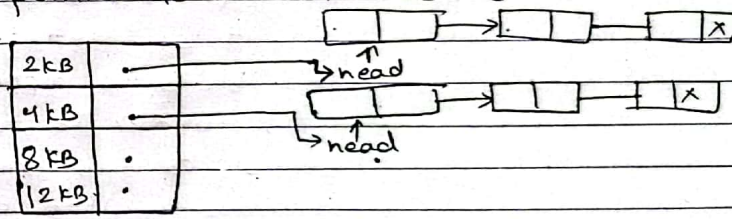
2. Next Fit:-
   - It works the same way as first fit, except that it keeps tracks of where it is whenever it finds the suitable hole. The next time it is called to find a hole, it starts searching the list from the place where it left off last time, instead of always at the beginning, as first fit does.

3. Best fit :-
   - Best fit searches the entire list and takes the smallest hole that is adequate. Rather than breaking up a big hole that might be needed later, best fit tries to find the hole that is close to the actual size needed.

3/17

4. Worst Fit :-
   - Always take the largest available hole, so that the hole broken off will be big enough to be useful. Simulation has shown that the worst is not a very good it idea either.

5. Quick Fit :-
   - Quick fit maintains separate list for some of the more common size requested.
   - For eg :- It might have a table with 'n' entries, in which the first entry is a pointer to the head of a list of 4KB holes, the second entry is a pointer to a list of 8KB holes, the third entry is a pointer to a 12KB holes and so on.



Buddy System :-
- Both fixed and dynamic partitioning scheme has draw back. A fixed partioning scheme limits the number of active processes and may use space inefficiently if there is a poor match between available partition sizes and process sizes.
- A dynamic partitioning scheme is more complex to maintain and includes the overhead of compaction. An interesting compromise is the buddy system.

• In a buddy system, the entire memory space available for allocation is initially treated as a single block whose size is a power of 2. When the first request is made, if its size is greater than half of the initial block then the entire block is allocated. Otherwise, the block is split in two equal companion buddies. If the size of the request is greater than half of one of the buddies, then allocate one to it. Otherwise, one of the buddies is split in half again. This method continues until the smallest block greater than or equal to the size of the request is found and allocated to it. In this method, when a process terminates the buddy block that was allocated to it is freed. Whenever possible, an unallocated buddy is merged with a companion buddy in order to form a larger free block. Two blocks are said to be companion buddies if they resulted from the split of the same direct parent block.

The following fig. illustrates the buddy system at work, considering a 1024K (1-MB) initial block and the process requests as shown at the left of the table.

3/17

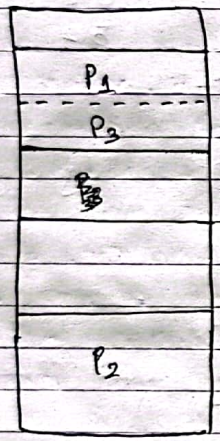
	0	128K	256K		512K		1024K
Start	1024K						
A=70K	A	128		256		512	
B=35K	A	B	64	256		512	
C=80K	A	B	64	C	128	512	
A ends	128	B	64	C	128	512	
D=60K	128	B	64	C	128	512	
B ends	128	64	D	C	128	512	
D ends	256			C	128	512	
C ends	512					512	
end	1024K						

Fig: Example of buddy system

3/20

Q. Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB and 600 KB (in order), how would each of the first fit, best fit and worst fit algorithms place processes of 212 KB, 417 KB, 112 KB and 426 KB (in order)? Which algorithm makes the most efficient use of memory?

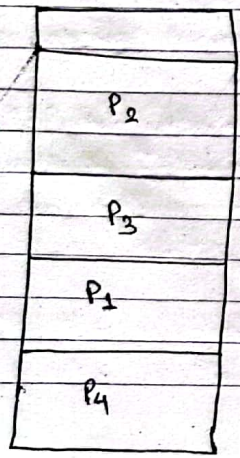
$P_1 = 212$ KB
$P_2 = 417$ KB
$P_3 = 112$ KB
$P_4 = 426$ KB



100 KB
500 KB
200 KB
300 KB
600 KB

212 KB process is put in 500 KB partition
417 KB process is put in 600 KB
112 KB process is put in 288 KB [500-212 = 288] partition
426 KB must wait

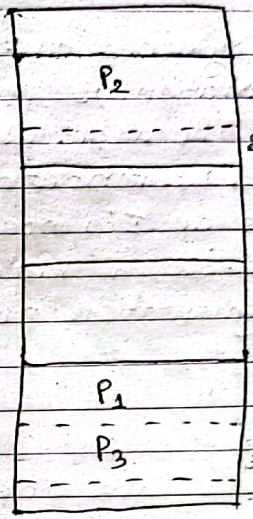
First Fit



100 KB
500 KB
200 KB
300 KB
600 KB.

P_1 (212 KB) process is put in 300 KB partition
P_2 (417 KB) process is put in 500 KB partition
P_3 (112 KB) process is put in 200 KB partition
P_4 (426 KB) process is put in 600 KB partition.

best fit.



100 KB
500 KB
200 KB
300 KB
600 KB

worst fit

212 KB process is put in 600 KB partition
417 KB is put in 500 KB partition
112 KB is put in 388 KB (600 - 212 KB) partition
426 KB must wait.

Non-Contiguous Memory Allocation:-

Fragmentation is main problem in contigious memory allocation. We have seen a method called compaction to resolve this problem. Since it is an I/O operation, system efficiency gets reduced. So, a better method to over come the fragmentation problem is to make our logical address space non-contiguous.

Consider a system in which before applying compaction, there are holes of size 1K and 2K. If a new process of size 3K wants to be executed then its execution is not possible without compaction. An alternative approach is to divide the size of new process 'P' into two chunks of 1K and 2K to be able to load them into two holes at different places.

1. If the chunks have to be of same size for all processes ready for the execution then the memory management scheme is called paging.

2. If the chunks have to be of different size in which process image is divided into logical segments of different sizes then this method is called segmentation.

3. If the method can work with only some chunks in the main memory and the remaining on the disk which can be brought into main memory only when it is required, then the system is called virtual memory management system.

Virtual Memory:-

The basic idea behind virtual memory is that the combined size of the program, data and stack may exceed the amount of physical memory available for it. The OS keeps those parts of the program currently in use in main memory and the rest on the disk.

For eg: A 512 MB program can run on a 256MB machine by carefully choosing which 256 MB to keep in memory at each instant, with pieces of the program being swapped between disk and memory as needed.

Virtual memory can also work in a multiprogramming system, with bits and pieces of many programs in memory at once. While a program is waiting for part of itself to be brought in, it is waiting for input output (I/O) and cannot run, so the CPU can be given to another process, the same way as in any other multiprogramming system.

Virtual memory system separate the

memory addresses used by a process from actual physical addresses, allowing separation of processes and increasing the effectively available amount of ~~ram~~ RAM using disk swapping.

3/21

Paging:-

External fragmentation is avoided by using paging technique. Paging is a technique in which physical memory is broken into blocks of same size called pages. Size is power of two, 512 bytes and 8192 bytes. When a process is to be executed its corresponding pages are loaded into any available memory frames. Logical address space of a process can be non-contiguous and a process is allocated physical memory whenever the free memory frame is available. OS keeps track of all free frames. OS needs 'n' free frames to run a program of size 'n' pages. Address generated by CPU is divided into:

1. Page number (p):-

Page number is used as an index into a page table which contains base address of each page in physical memory.

2. Page offset (d):-

Page upset is combined with base address to define the physical memory address

demand paging
Translation buffer
Thrashing

3/21

Structure of page table

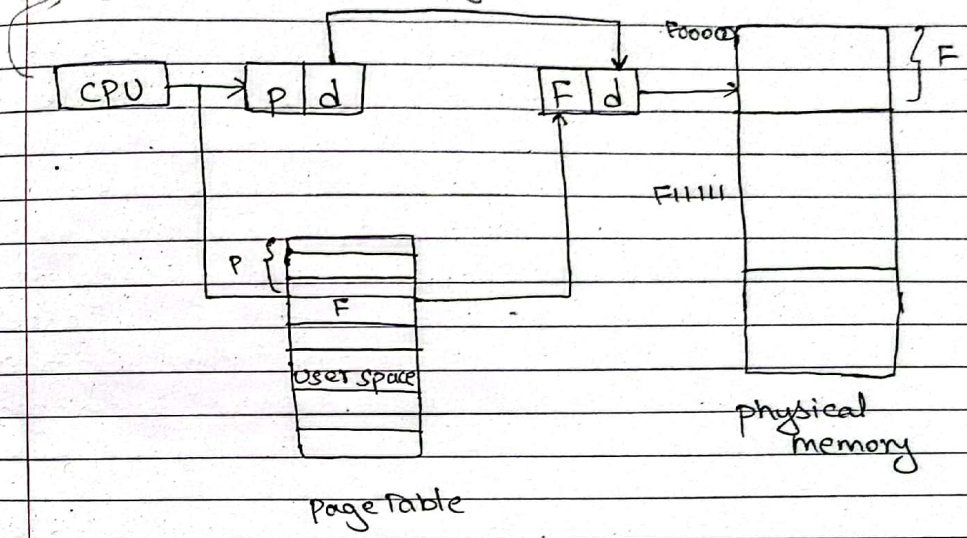
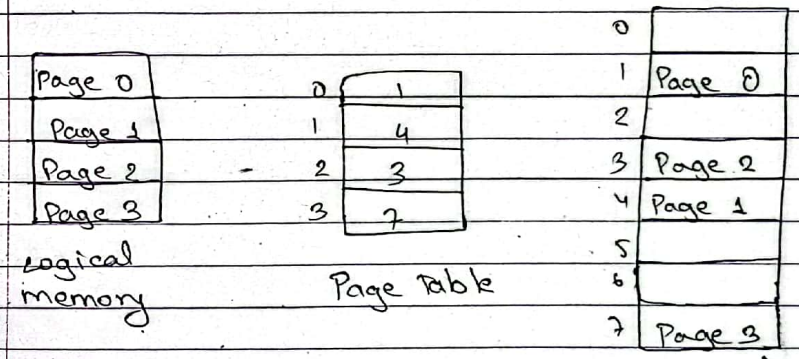


Fig:- Paging Architecture / hardware



Segmentation:

- Segmentation is a technique to break memory into logical pieces where each piece represents a group of related information. For eg: data segments, code segment, stack
- Segmentation can be implemented with or without using paging. Unlike paging, segment is having varying sizes and ~~do~~ thus eliminates internal fragmentation. External fragmentation still exists but to lesser extent. Address generated by CPU is divided into:

1. Segment number(s):

Segment number is used as an index into a segment table which contain base address of each segment in physical memory and a limit of segment.

2. Segment offset(d):

Segment offset is first checked against limit and then is combined with base address to define the physical memory address.

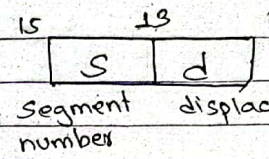
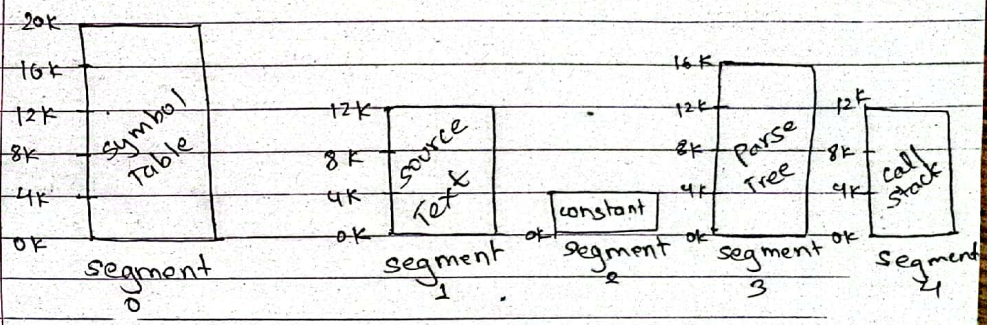
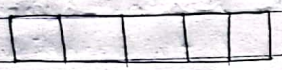


Fig: Virtual/Logical Address Space (16 bit)



Segmentation maintains multiple separate virtual address space per process. Allows each table to ~~broke a s~~ grow or ~~sh~~ shrink independently.

3/2/21

Paging	Segmentation
1. Block replacement easy fixed-length block 	Block replacement hard variable-length blocks. Need to find contiguous, variable-sized, unused part of main memory. 
2. Invisible to application programmer.	Visible to application programmer.
3. No external fragmentation, But there is Internal Fragmentation unused portion of page.	No Internal Fragmentation, But there is external Fragmentation unused portion of main memory.
4. Units of code and data are kept broken into separate pages as a single units.	Units of code and data are kept blocks of code or data as a single units.
5. Segmentation is a logical unit visible to the user's program and is of arbitrary size.	Paging is a physical unit invisible to the user's view and id of fixed size.
6. Segmentation maintains multiple address spaces per process.	Paging maintains one address space
7. No sharing of procedures between users is faciliated	Sharing of procedures between users is faciliated.

Point change value

3/2/21

Page fault is not an error

Page Fault:

A page fault is a trap to a S/w raised by the hardware, ~~when~~ when a program access a page that is mapped in a virtual address space, but not loaded in physical memory. In the typical case, the OS tries to handle the page fault by making the required page accessible at a location in physical memory or kills the program in the case of an illegal access. The hardware that it takes a page fault is MMU (memory management unit) in a processor. The exception handling software that handles the page fault is generally the part of operating system.

When does a page fault occur?

⇒ In demand paging memory management technique, if a page demanded for execution is not present in main memory, then a page fault occurs. To load the page in demand into main memory, a free page frame is searched in main memory and allocated. If no page frame is free, memory manager has to free a frame by swapping its content to secondary storage and thus make room for the required page. To swap pages, many scheme or strategies are used.

3/24

Page replacement algorithms:

- When a page fault occurs, the operating system has to choose a page to remove from memory to make room for the page that has to be brought in. The page replacement is done by swapping the required pages from backup storage to main memory and vice versa.
- If the page to be removed has been modified while in memory, it must be rewritten to the disk to bring the disk copy upto date. If however, the page has not been changed, the disk copy is already up to date, so, no re-write is needed.
- A page replacement algorithm is evaluated by running the particular algorithm on a ~~screen~~ string of memory references and compute the page faults. Referenced string is a sequence of pages being referenced.

3/24

Balady's Anomaly → replace the page that will not be reference for longest time.
↳ caused in FIFO

① Optimal page replacement algorithm:

- The algorithm has lowest page fault rate of all algorithm.
- This algorithm state that, "replace the page which will not be used for longest period of time (i.e future knowledge of reference string is required)."
- Often called Balady's ~~Mean~~ Min Basic Idea
- Impossible to implement.

Q Consider the following reference string:
0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1
How many page fault will occur if the program has 4 page frames for optimal page replacement algorithm?

rows is page frame.

M = Miss
H = Hit

	0	2	1	6	4	0	1	0	3	1	2	1
				6	4	4	4	4	4	4	4	4
			1	1	1	1	1	1	1	1	1	1
		2	2	2	2	2	2	2	2	2	2	2
	0	0	0	0	0	0	0	0	3	3	3	3
	M	M	M	M	M	H	H	H	M	H	H	H

algo with less fault rate is the best

page fault = 6

fault rate = $\frac{6}{12} = \frac{1}{2} = 0.5$

3/24

Q. Consider the following page reference string:
1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.
frame = 3.

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
		3	4	4	4	5	6	6	6	6	6	6	6	6	6	1	1	1	1
	2	2	2	2	2	2	2	2	2	2	2	7	7	7	2	2	2	2	2
1	1	1	1	1	1	1	1	1	1	1	3	3	3	3	3	3	3	3	6
m	m	m	m	H	H	m	m	H	H	H	m	m	H	H	m	m	H	H	m

page fault = 11
fault ratio = $\frac{11}{20}$ = 0.55

Belady's Anomaly (For FIFO)

3/24

page frame ↑, page fault ↑
↳ Belady's Anomaly.

② FIFO replacement algorithm:
- The oldest page in the physical memory is the one selected for replacement.
- very simple to implement.
- keep a list: on a page fault, the page at the head is removed and the new page is added at the tail.

① Eg :- 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

0	2	1	6	4	0	1	0	3	1	2	1
			6	6	6	6	6	6	1	1	1
		1	1	1	1	1	1	3	3	3	3
	2	2	2	2	0	0	0	0	0	0	0
0	0	0	0	4	4	4	4	4	4	2	2
m	m	m	m	m	m	H	H	m	m	m	H

page = 9
ratio = $\frac{9}{12}$

②

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
		3	3	3	3	5	5	5	1	1	1	1	6	6	6	6	6	3	3
	2	2	2	2	1	1	1	2	2	2	2	7	7	7	2	1	1	1	1
1	1	1	4	4	4	4	6	6	6	6	3	3	3	3	2	2	2	2	6
m	m	m	m	H	m	m	m	m	m	H	m	m	m	H	m	m	H	m	m

page fault = 16
ratio = $\frac{16}{20} = \frac{4}{5}$

disad:
- Poor replacement strategy because FIFO doesn't consider the page usage.

3/29

③ Least Recently Used (LRU):-

→ The page that has not been used for longest period of time is selected for ~~ret~~ replacement

→ Theoretically realizable but not cheap (linked list)

→ The difficulty is that the list must be updated on every memory reference.

Q/ Consider the following reference string:

0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

page frame = 3

←

	0	2	1	6	4	0	1	0	3	1	2	1
F3			1	1	1	0	0	0	0	0	2	2
F2		2	2	2	4	4	4	4	3	3	3	3
F1	0	0	0	6	6	6	1	1	1	1	1	1
	m	m	m	m	m	m	m	H	m	H	m	H

Page fault = 9

fault ratio = 9/12 = 3/4

3/29

④ Second chance page replacement:

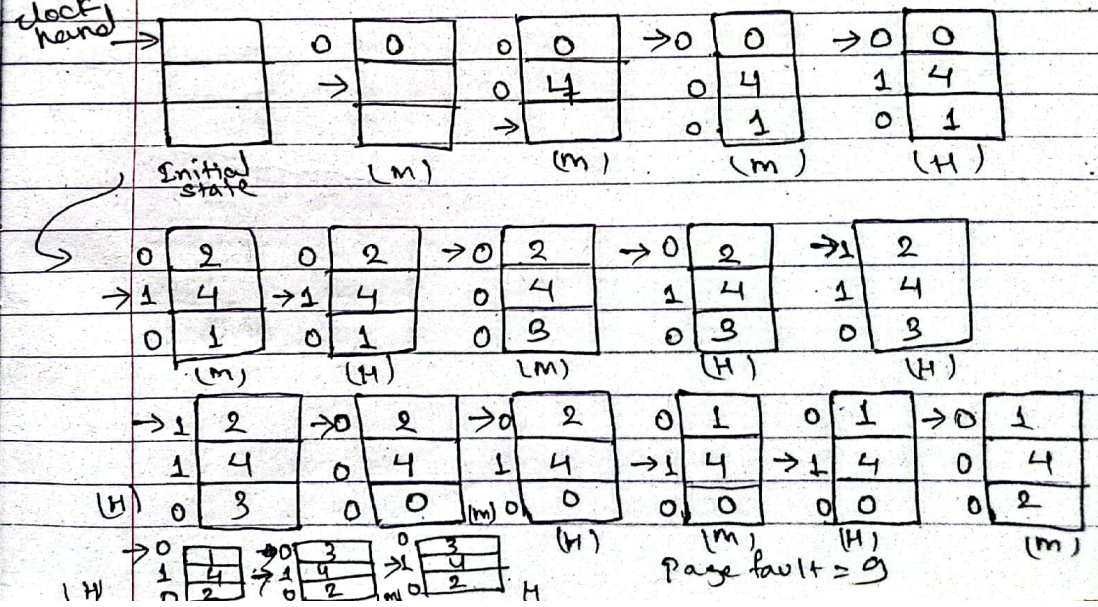
↳ Also called as clock page replacement.

↳ Designed to approximate the behaviour of LRU algorithm without explicitly keeping track of each pages history.

↳ It uses a circular queue (or circular list) and a reference bit associated with each page frame. When a page needs to be replaced, the algorithm checks the reference bit. If the bit is set (indicating the page has been accessed ~~If the bit is set~~ recently), it clears the bit and gives the page a "second chance" to avoid replacement. If the bit is not set, the page is a candidate for replacement.

Q.

0, 4, 1, 4, 2, 4, 3, 4, 2, 4, 0, 4, 1, 4, 2, 4, 3, 4



⑥ Working set clock (WS clock) P. R.
⑦ Enhanced second chance / clock algorithm (NRU: Not recently used)
⑤ Working set (WS) Page Replacement.

3.2.5 Thrashing:-

File :
operation of file :

3/28

3.3 File system Interface and implementation:

File attribute:

Every file has a name and its data. In addition all operating system associated with other information with the file. For eg:

The file date and time the file was created and the file size.

These extra information are called meta data or file attribute.

The list of attribute varies considerably from system to system.

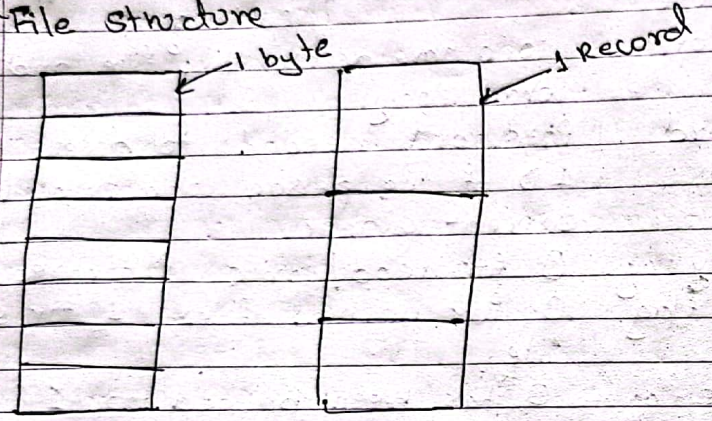
3/28

Attribute	Meaning
Protection	Who can access the file and in what way.
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal files; 1 for system file.
Archive flag	0 for has been backed up; 1 for needs to be backed up.
ASCII/binary flag	0 for ASCII file; 1 for binary flag.
Random access file flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; non-zero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record.
Key length	Number of bytes in key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date & time the file has last changed
Current size	Number of bytes in the file

3/28

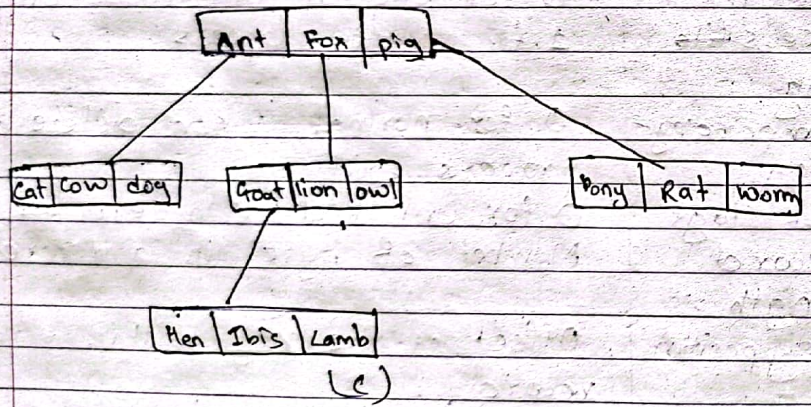
IMP

File structure



(a) Byte sequence

(b) Record sequence.



(c) Tree

3/28

i Byte sequence:

The file in fig(a) is just an unstructured sequence of bytes. In effect, the OS doesn't know or care what is in the file. All it sees are bytes. Any meaning must be imposed by user level programs. Both UNIX and windows use this approach.

ii Record sequence:

In this model, a file is a sequence of fixed length records, each with some internal structure. Central to the idea of a file being a sequence of records is the idea that the read operation returns one record and the write operation over writes or appends one record.

As a historical note, when the 80 column punch card was king many main frame OS based their file system on files consisting of 80 character records.

iii Tree sequence:-

In this organization, a file consists of a tree of records, not necessarily all the same length, each containing a key field in a fixed position in the record. The tree is sorted on the key field to allow rapid searching for a particular key.

3/28
SMG

File organization and access Mechanism:

When a file is used then the stored information in the file must be accessed and read into the memory of a computer system.

Various mechanism are provided to access a file from OS:

1. Sequential access
2. direct access
3. Index access

1. Sequential access:

It is the simplest access mechanism in which information stored in a file are accessed in an order such that one record is processed after the other.

For eg:- ~~Editors~~ Editors and compilers usually access the file in this manner.

2. Direct access:-

It is an alternative method for accessing a file, which is based on the disk model of a file, since this allows random access to any block or record of a file. For this method, a file is viewed as a numbered sequence of blocks or records which are read/written in an arbitary manner i.e there is no restriction of the order of reading or writing. It is well suited for a DBMS.

3/28
IMS

3. Index Access:-

In this method, an index is created which contains a key field and pointers to a variable block. To find an entry in the file for a key value, we first search the index and then use the pointer to directly access the file and find the desired entry.

With large files, the index file itself may become too large to be kept in memory. One solution is to create an index for ~~a~~ the ~~file~~ index file. The primary index file would contain pointers to secondary index file, which would point to the actual data items.

File Allocation Method:

1. Contiguous Allocation
2. Linked List Allocation
3. I nodes

1. Contiguous Allocation

It requires each file to occupy a set of contiguous address on a disk. It stores each file as a contiguous run of disk blocks. Thus on a disk with 1-KB blocks, a 50-KB file would be allocated 50 consecutive blocks. Both sequential and direct access is supported by the contiguous allocation method.

Contiguous disk space allocation has two significant advantages.

1. First, it is simple to implement because keeping track of where a file's blocks are reduced to remembering two numbers: the disk address of the first block and the number of blocks in the file. Given the number of the first block, the number of any other block can be found by a simple addition.

2. Second, the read performance is excellent because the entire file can be read from the disk in a single operation. Only one seek is needed (to the first block). After that, no more seeks or rotational delays are needed so data come in at the full

bandwidth of the disk.

Thus contiguous allocation is simple to implement and has high performance.

Unfortunately, contiguous allocation also has a major drawback: in time, the disk de becomes fragmented, consisting of files and holes. It needs compaction to avoid this.

Eg of contiguous allocation: CD and DVD ROMs

2. Linked List Allocation:

Keep each file as a linked list of disk blocks as shown in the figure. The first word of each block is used as a pointer to the next one. The rest of the block is for data

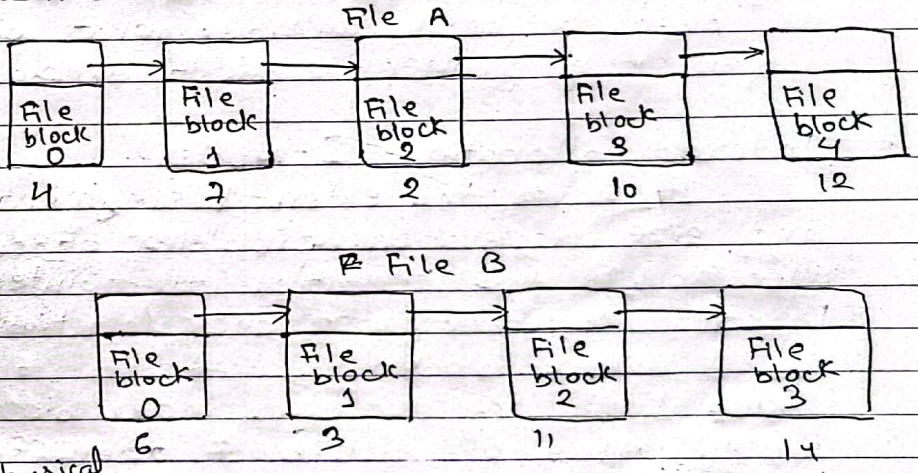


Fig: Storing a file as a linked list of disk blocks.

3/31

Unlike contiguous allocation, every disk block can be used in this method. No space is lost to disk fragmentation. The major problem with linke allocation is that it can be used only for sequential access files. To find the i^{th} block of a file, we must start at the beginning of that file, and follow the pointers until we get the i^{th} block. It is inefficient to support direct access capability for linked allocation of files.

Another problem of linked list allocation is reliability. Since the files are linked together with the pointer scattered all over the disk. Consider what will happen if a pointer is lost or damaged.

3 Indexed allocation (I-Nodes):

It solves the external fragmentation and size declaration problems of contiguous allocation. In this allocation all pointers are brought together into one location called Index block.

Each files has its own index block, which is an array of disk-block addresses. The i^{th} entry is the index block points to the i^{th} block of the file. The directory contains the address of the index block.

3/31

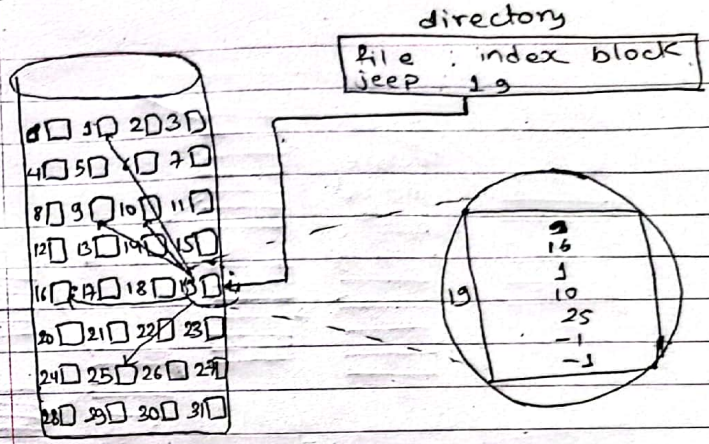


Fig: Index allocation of disk space

To read the i^{th} block, we use the pointer in the i^{th} index block entry to find and rea the desired block.

This scheme is similar to the paging scheme.

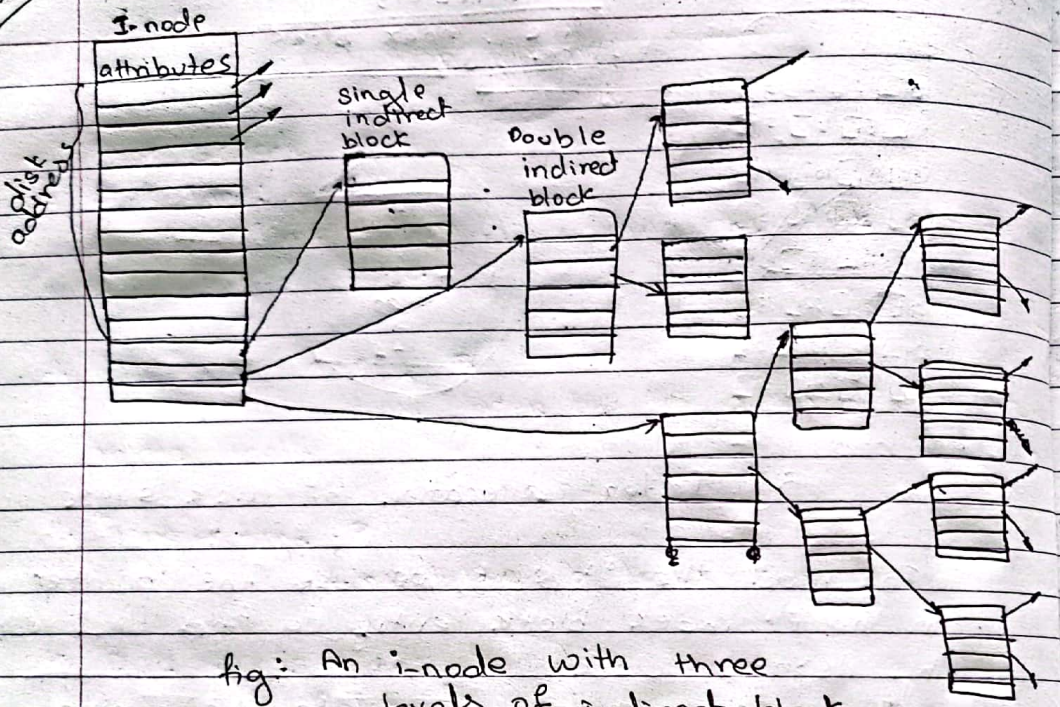
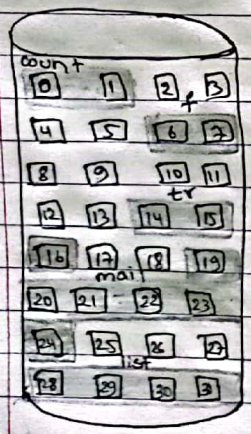


fig: An i-node with three levels of indirect block.



Directory

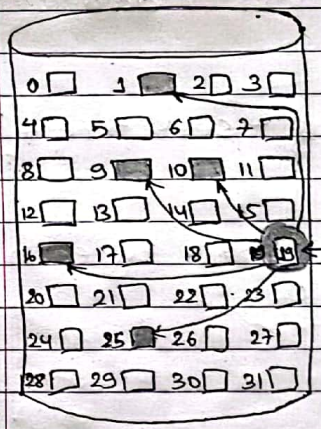
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Fig:- Contiguous ~~re~~allocation method

Directory

file	start	end
jeep	9	25

Fig:- Linked list allocation method



Directory

file	index block
jeep	19

Fig:- I-node allocation mode.

04/01

Unit: IV

I/O Management

Disk scheduling

1. Seek Time:
   Time taken in locating the disk arm to a specified track.

2. Rotational Latency:
   time taken by the desired sector to rotate itself.

3. Transfer time:
   time taken to transfer data.

4. Disk Access time:
   Rotational latency + seek time + transfer time.

5. Disk Response Time:
   average of time spent by each request waiting for the I/O.

04/01

Imp

Disk scheduling Algorithms:

1. FCFS
2. SSTF
3. SCAN
4. Circular SCAN (C-SCAN)
5. LOOK
6. Circular LOOK (C-LOOK)

direction doesn't matter

(1.) FCFS (First Come First Serve)

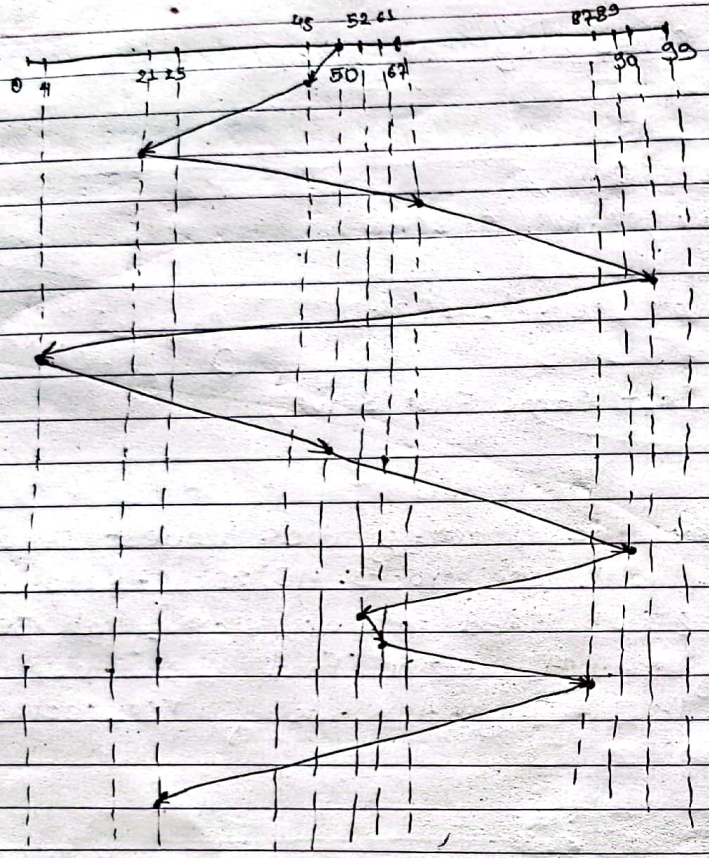
Q. Consider the following disk request sequence for a disk with 100 tracks: 45, 21, 67, 90, 4, 50, 89, 52, 61, 87, 25. Head pointer starting at 50 and moving in left direction. Find the number of head movements in cylinders using FCFS scheduling.

Soln.

P.T.O →

Obj: • head movement (high)

4/01



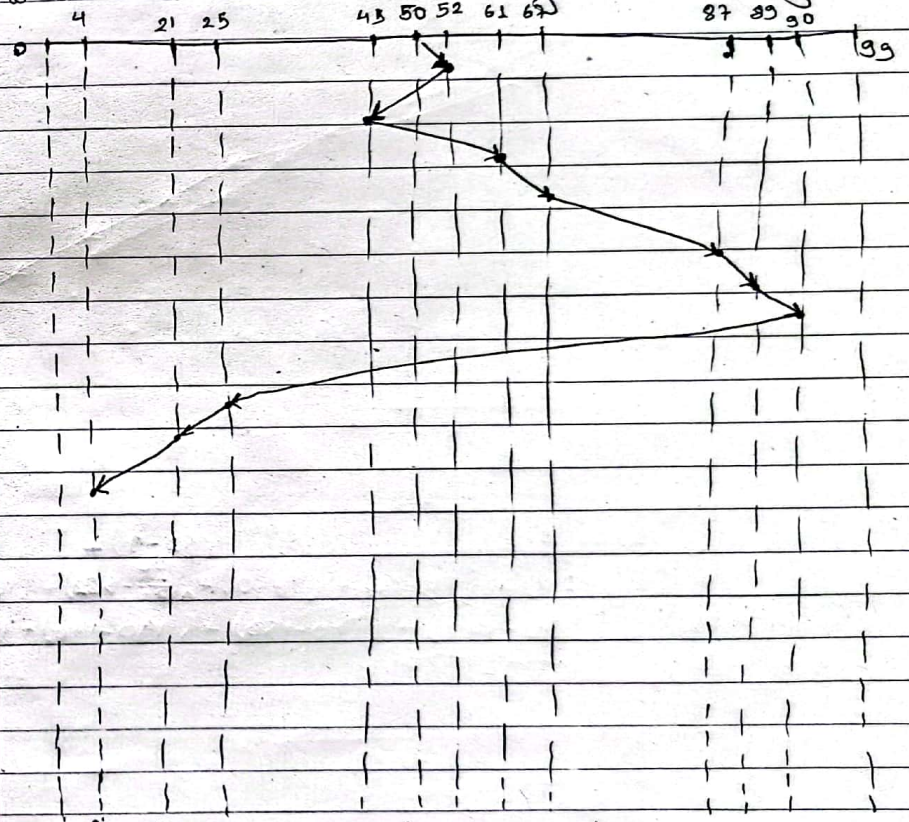
No. of cylinders moved by the head
= (50-45) + (45-21) + (67-21) + (90-67) + (90-4) + (50-4) + (89-50) + (89-52) + (52-50) + (~~85~~ 87 - 52) + (87-25)

= 376,

4/1

(2.) SSTF (Shortest Seek Time First) → direction matters (but ignore)

8. Consider the following disk request sequence for a disk with 100 tracks: 45, 21, 67, 90, 4, 89, 52, 61, 87, 25. Head pointer starting at 50. Find the no. of head movement in cylinder using SSTF.



no. of cylinders moded by head
= (52-50) + (52-45) + (61-45) + (67-61) + (87-67) + (89-87) + (90-89) + (90-25) + (25-21) + (21-4)

= 140,

4/1

Last samay pughai partha either 0 or 199 depends on which direction we choose

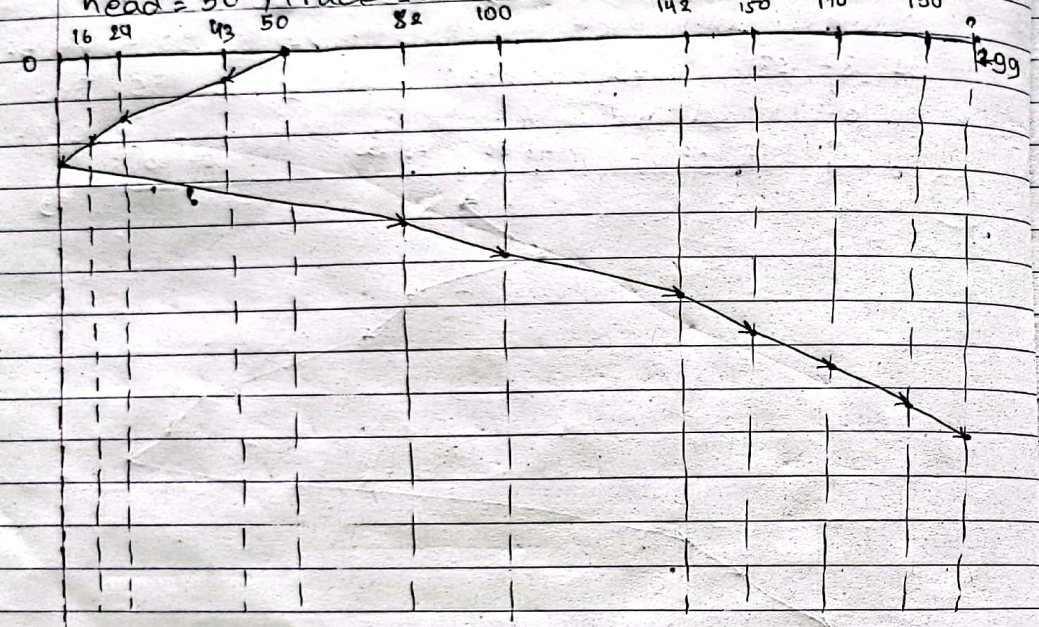
direction matter

outward =

③ SCAN (elevator) algorithm.

Q. 0, 16, 24, 43, 50, 82, 100, 142, 150, 170, 190, 199

head = 50, track = 200



No. of head :

$= (50-43) + (43-24) + (24-16) + (16-0) + (82-0) + (100-82) + (142-100) + (150-142) + (170-150) + (190-170) + (199-190)$

$= 249$

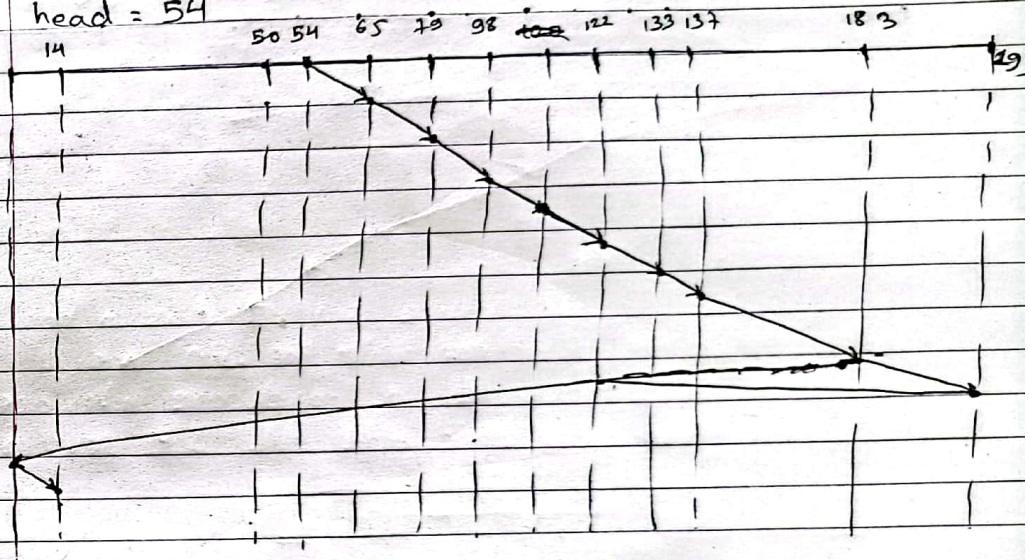
4/1

④ C-SCAN.

Consider the following disk for 200 tracks.

14, 54, 65, 78, 98, 122, 133, 137, 183.

head = 54



No. of head.

$= (65-54) + (78-65) + (98-78) + (122-98) + (133-122) + (137-133) + (183-137) + (199-183) + (199-0) + (14-0)$

$= 357$

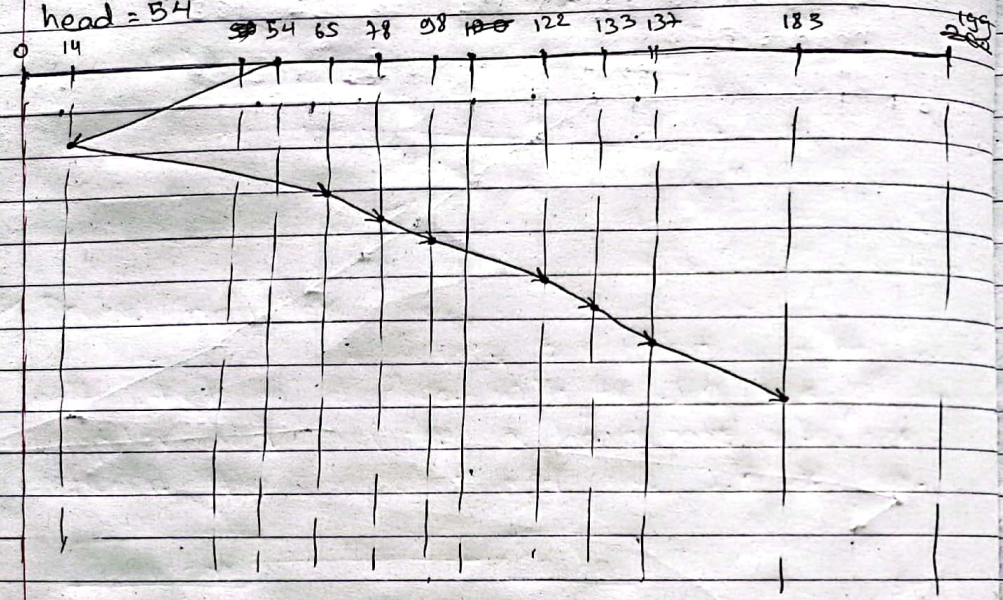
4/1

(5) LOOK

Consider the following disk of 200 tracks:

14, 54, 65, 78, 98, 122, 133, 137, 183,

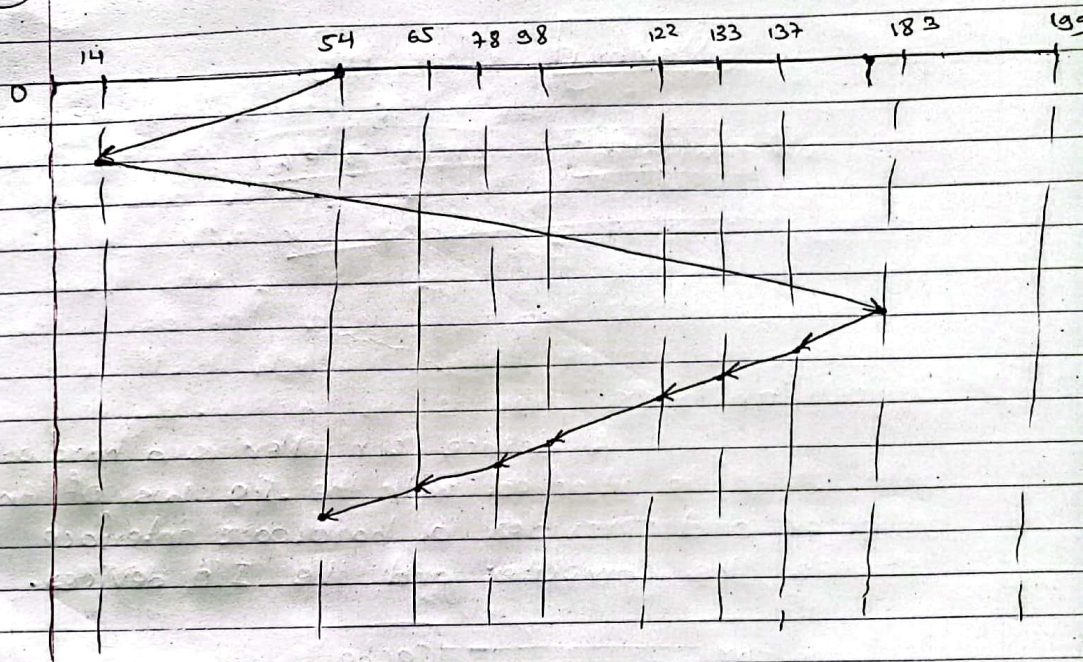
head = 54



no. of heads =

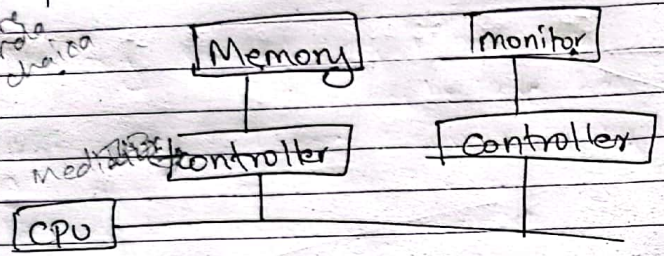
4/1

(6) C-LOOK :



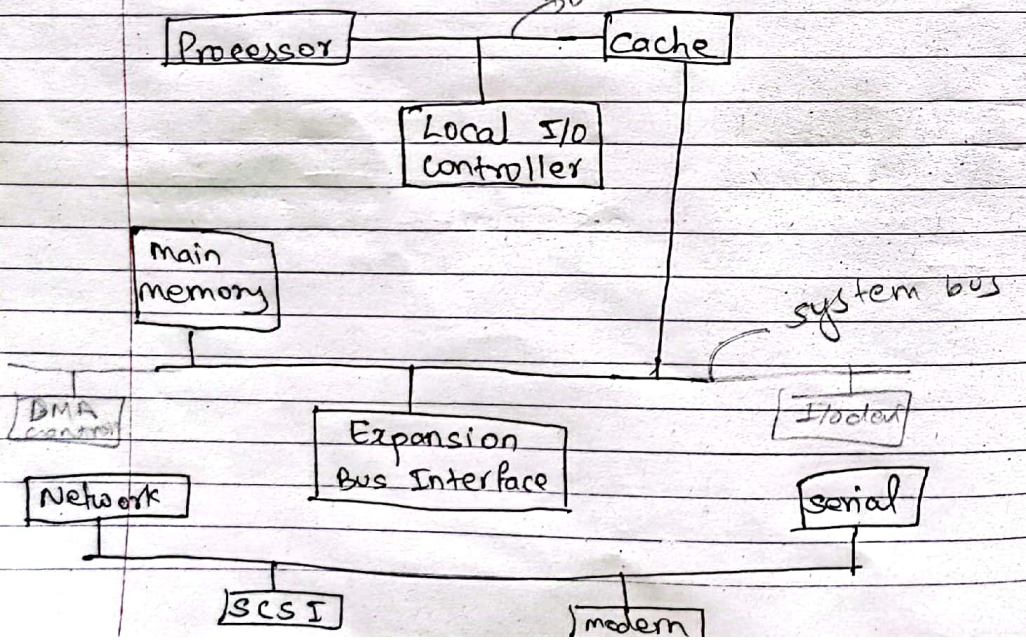
no. of head :-

Input/Output Management:



Imp Device Controller:

A device controller is a hardware unit which is attached with the I/O bus of the computer and provides a hardware interface between the computer and the I/O devices.



On one side it knows how to communicate with I/O devices and on the other side it knows how to communicate with computer system through I/O bus. A device controller usually can control several I/O devices. Typically, the controller is on a card (Eg LAN card, USB card). Device controller plays an important role in order to operate that device. It is just like a bridge between device and OS.

Process ⟷ I/O device (communication)

Q. Process ra I/O device ma 2 way communication kasari garne

↳ 3 ways (i) Polling (Programed I/O)

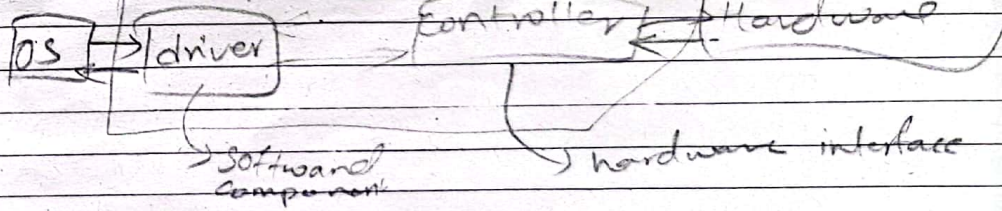
(ii) Interrupt driven I/O

(iii) Direct memory Access (DMA)

↳ no involvement of CPU

↳ when I/O device is very slow

Interdependency (not dependent with one another)

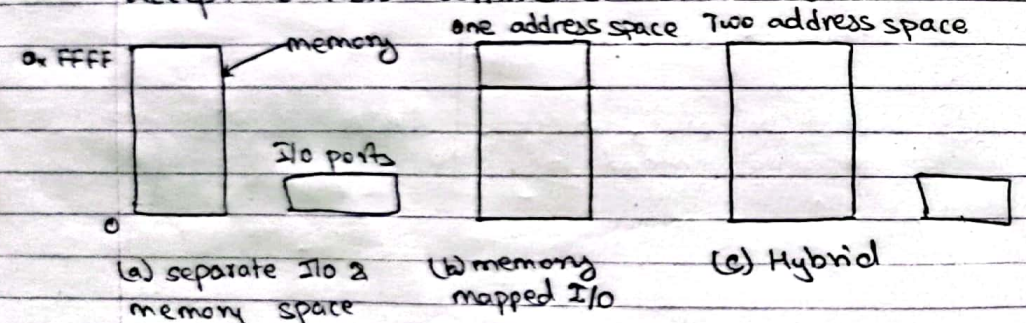


4/8/4

~~Device Driver~~:

Memory Mapped Input/Output:-

Each controller has a few registers that are used for communicating with the CPU. On some computers, these registers are part of the regular memory address space. This is called memory mapped I/O. By writing into these registers, the OS can command the device to deliver data, accept data, switch itself on or off, or otherwise perform some action. By reading from these registers, the OS can learn what the device state is, whether it is prepared to accept a new command and so on.



(a) separate I/O & memory space (b) memory mapped I/O (c) Hybrid

4/4

Port Mapped Input/Output (I/O mapped I/O):-

Each controlled register is assigned an I/O port number, an 8-bit or 16-bit integer. Using a special I/O instruction such as IN REG, PORT. The CPU can read in control register port and store the result in CPU register REG. Similarly, using OUT PORT, REG, the CPU can write the content of REG to a control register. Most early computers, including all main frames such as IBM 360 and all of its successors work this way.

Date
Page

Compaction Vs Coalescing

- Process scheduling (Gantt Chart)
- Page replacement
- DISK ||
- Bankers Algorithm ⊗
- Petestion Algo.
- Deadlock
- system calls
- file structure
- Threads, why light weight process
- process state
- Reader writers problems.