Eg:-

Operating System

OS is an intermediatery interface b/w user & hardware. It is basically brain of computer without which it can't work. It helps to peform such actions without knowing how things actually work at computer level. It peforms all basic tasks like file management, memory management, process management, etc.

GMO was the first full fledged OS built by General motors for IBM 701. It was Batch OS.

Convinence – Windows (82%)
Throughput – Linux (18%)

OS Structure

UI or Interface
↓
Shell (User level commands converted to low level commands. Comes under utilities like editors)
↓
Kernel (Assigning of resources & other things to the commands. Part of OS.
↓
Hardware

Functions of OS

- Process Mgmt
- Memory Mgmt
- File Mgmt
- Device Mgmt
- Security
- Job Accounting
- Secondary Storage mgmt
- Networking
- Procss Mgmt
- Coordination b/w other software & users.

Types of OS

1) Batch OS 2) Multiprogramming Batch OS 3) Time Sharing OS
4) Multiprocessor OS 5) Distributed OS 6) Real Time OS.

Extra:- Network OS, Mobile OS.

Simple Batch OS

* Similar category of jobs are batched together & to CPU for processing.
* No direct communication b/w user & computer.
* Batch of jobs created on basis of type of language & nee
* Eg - Bank Statements, Payroll systems. (IBM's z/OS)

Advantages

1) No mechanism to prioritize processes.
2) Ideal time very less for Batch OS.

Disadvantages

1) Debugging is very hard.
2) These are costly.

(2) Multiprogramming Batch OS

* Have multiple programs in queue for operation. When this job requires I/O operation, OS switches to another job which keeps OS & CPU always busy.
* Jobs present in memory are always less than jobs in prol

Eg - Windows, Excel, IOS etc.

Real life Examples -

- Writing an email & opening a notification in between, OS keeps both programs loaded in memory.
- Automatic backups
- Music player playing in background.

(3) Time Sharing Operating System

* We assign some time to each job so that all jobs work efficiently & smoothingly.
* Time taken by each job is called as quantum. It allows various numbers of users to be placed at various terminals so that they can use particular system at same time. Shares processor's time with multiple users simultaneously.

Eg- Windows, macOS, Linux.

Real life examples:
- Multitasking on computers
- Online collaboration tools like editing google docs simultaneously.
- Streaming services manage downloading data for next part of video while simultaneously playing the portion already downloaded.

Adv - CPU is idle for less time, quick responses.
Disadv - Problem of data communication, not reliable.

* Time sharing - minimize response time
Multiprogramming - increase use of processor.

(4) Multiprocessor OS

* Use 2 or more processors within single computer system. Processors are in close communication & share memory & computer buses etc.
* Offers high speed & computer power & is also known as tightly coupled system.

Adv - Improved performance, throughput increased, increased reliability
Disadv - Complex architecture, bigger memory required.

Real life examples -
* ML & AI use multiprocessing by managing workloads on systems with multiple GPUs.
- Video editing workstations.
- Web browsers with this architectures run different tabs in separate processors.

(5) Distributed OS

* Loosely coupled systems where can multiple central processors are used to serves multiple real time applications & multiple users.
* Applications run on multiple interconnected computers offering enhanced communication & integration capabilities compared to a network OS.

Real Life Examples:-

- AWS, Microsoft Azure offers a vast array of servi... from compute power to storage distributed across multip... data centers.
- supercomputers & Internet itself.

Adv - speed is increased, offers better services to compu... Offers reliability, reduces load on host computer.

Disadv - More expensive, failure of central network stops whole communication.

(6) Real Time OS

* Used in real time applications where data processing must be done in fixed time interval (Stingent time constraint)
* Give very fast response & used where large no. of events are processed in short interval of time.

Adv - Error free, Offer memory allocation mgmt, more focus on running applications than in queue.

Disadv - Expensive, task of algo writing is complex.

Real Life Examples -

- Online bank payments
- Anti locking Braking systems in modern cars.
- Image equipments like MRI Scanners or ultrasound machines
- Financial trading systems for rapid order exchange.
- Automatic doors.
- In daily life, microwave ovens for timers require RTOS.

* In Time Sharing - Quantum
  In Real Time - Response Time.

Buses

Electronic pathway through which data can be transfered.

Three types of Bus lines — Data Bus (carry data)
Address Bus (manage addresses)
Control Bus (specifies whether to read/write data to location)

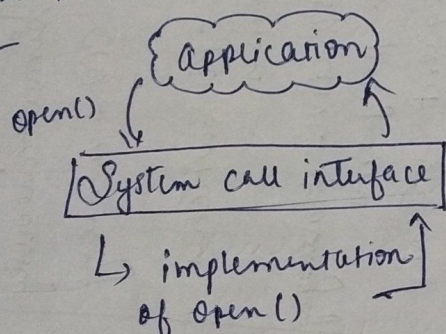
Broadly categorized into I/O & memory Buses

Multi-Tasking	Multi-Programming	Multi-Threading	Multi-Processing
• Executing multiple tasks concurrently • Shares CPU & memory • Single CPU switches b/w tasks. • User perspective of simultaneous tasks. • Frequent context switches. • Improves CPU utilization • Moderate complexity	• Managing multiple programs simultaneously. • Shares CPU & memory • Single CPU switches b/w programs. • OS perspective of managing multiple programs. • Less frequent context switching. • Improves overall system throughput. • Higher than multitasking	• Creates multiple threads within a process. • Shares code, data & OS resources. • Multiple threads share CPU time. • Within process parallelism. • Faster context switches than processes • Improve responsiveness within program/process • Lower than multiprocessing	• Using multiple CPUs to execute tasks. • Shares memory but all processes have their own memory. • Multiple CPUs execute diff. tasks concurrently. • Hardware level parallelism • Context switching b/w processes on diff CPUs. • Improves overall system performance. • Highest complexity.
Job/Task is a program that is in job queue of CPU. (excludes I/O operations) (What is to be done) • Job/Task are synonyms.	Program is passive entity which is set of instructions. Hierarchy – Program ↓ Task ↔ Process ↔ Job ↓ Thread	Thread is the smallest unit of execution.	Process is a program in execution. (Includes I/O operations) (How is to be done)

System Calls

- It is a call or request made by a program to the OS's to perform a specific task. It is a way for processes to with kernel & access hardware resources & perform ope that they are not allowed to do directly.
- It provides interface b/w a program and OS to allow user level processes to request services of OS.
- System calls languages – C & C++.
- There is a number associated with each system call & system call interface maintains a table accordingly.
- System call interface invokes intended system call in OS kernel & return status of system call & any return values.

Eg:-



- Parameters in System call are passed in 3 ways:-
  - In registers
  - Stored in tables, blocks, memory and address of block passed as parameters
  - Push the parameters on stack & popped off by OS.

Services by System calls – * Process management
* Memory management
* File System management
* Device handling
* Protection & Network etc

Types of System Calls –

(1) Process Calls – Create process, terminate process, end, abort, load, execute etc

(2) File Management – Create file, delete file, open, close file, read, write etc

(3) Device Management – request or release device, read, write, reposition etc

(4) Information Maintainance – get time or date, set time or date, get system data, set system data

(5) Communication – create/delete connections, send/receive messages, shared – memory model, transfer status info.

6) Protection – Control access to resources, get and set permissions, allow / deny user access.

* Check names of system calls in Windows – Unix } Pdf 2

Kernel

- Computer program that has control over everything in system.
- When a system starts, kernel is first program that is loaded after bootloader because kernel has to handle rest of things.
- Responsible for low level tasks (hardware level).
- Os has a layered approach from user interface to hardware which provides a systematic approach, modularity, isolation & security.

Functions of Kernel – * Access computer resources like CPU, I/O devices & others.
* Resource management i.e share the resources b/w various processes.
* Memory management i.e allocation & deallocation to various processes.
* Device management i.e connection of peripheral devices used by processes.

Types of Kernels

1) Monolithic – User services & kernel services are in same memory space. Contains all essential Os components within a single executable.
Eg – Unix, Linux, OpenVMS.

2) Microkernel – User services & kernel services are in different spaces. Include only essential functions in kernel. More secure in case of failure in one component. Communication via message passing.
Eg – Mach, L4. [more reliable & secure – adv
performance overhead – disadv]

3) Hybrid – Combination of Monolithic & Microkernel. Offers balance b/w performance & modularity.
Eg – Windows, Os/2.

4) Nano kernel - The code executing in privileged mode of hardware is very small. Used in embedded systems with limited resources.
Eg - IOT devices.

5) Exokernel - Resource protection is separated from management & provides minimal interface for application interaction with hardware resources. Offers high customization & control over resource allocation.
Eg - Research & ~~example~~ experimental systems.

Difference b/w OS, Shell, Kernel & System Calls

* OS - Master controller that manages computer hardware & software resources & acts as an intermediatery b/w user & hardware.
* Shell - Interacts with user providing GUI or CLI & translates user commands to system calls.
* Kernel - Manage hardware resources & provide essential services
* System Calls - Provides a way for applications to request services from kernel.

• Kernel is interface b/w application & hardware & OS is interface b/w user & hardware.

Difference b/w GUI & API

GUI is communication b/w user & interface whereas API is communication b/w software & other software.
GUI is like what's happening and API is like how's it implemented at backend.
GUI is imp for naive users & API is imp for developer & programmer concerned with background working of visuals.

DIFFERENCE BETWEEN MULTICORE AND MULTIPROCESSOR

- **Multiprocessor** – When more than one CPU works together to carry out computer instructions or programs. Such multiple processing units share memory & peripheral devices.
  - **Assymmetric** – Here, not all of the multiple CPUs are treated equally. Only the master processor runs all tasks of OS. CPUs are in master-slave relationship.
    - Disadvantage – Unequal load placed on processors. While other processors may be idle, one CPU might have huge job queue.
  - **Symmetric** – Two or more identical processors are connected to a single, shared main memory and have full access to all input & output devices. Here each processor is self-scheduling.

Assymmetric	Symmetric
• Processors are not treated equally	• Processors are treated equally.
• Tasks of OS are done by master processor.	• Tasks of OS are done by individually processor.
• No communication b/w processors	• All processors communicate by a shared memory.
• Process scheduling is master-slave.	• Process is taken from the ready queue.
• Assymetric multiprocessing system are cheaper.	• Symmetric multiprocessing systems are costlier.
• Easier to design.	• Complex to design.
• In case, master processor malfunctions, slave processor continues the execution which turns out to be master processor.	• In case of processor failures, there is reduction in system's computing capacity.

- **Multicore** - It is an integrated circuit with two or more processors connected to it for faster simultaneous processing of several tasks, reduced power consumption and for greater performance. It comprises numerous processing units each of which has potential of doing distinct tasks.
- A high performance computer can have six to eight cores. A 64 bits core CPU is exceptionally powerful. Recently, 128-core CPU have entered market. These cores are what read computer program instructions and execute commands.
- IBM was the first to integrate a multicore processor into a computer successfully. In 2001, it launched the Power4, world's first multicore processor.

Multicore and Multiprocessor bring increased speed and performance to a computer system.
Multicore - Single CPU and Multiprocessors - multiple CPUs.
Multiprocessor is increase in performance but have little problem running multiple applications.
Multicore CPUs are more common cos these provide excellent performance.
Most famous example of Multicore - IBM's Watson.

{User Level Thread VS Kernel Level Thread}

→ {Difference b/w Process & Thread} -

- Process is an independent program that is executing in its memory space. It includes everything needed to run: the code, data, resources & process control info. Processes are isolated from each other.
- Thread is smallest unit of execution of process which share same resource & memory. This makes them lightweight and faster to create.

→ we usually work in multitasking or multiprogramming environment because of presence of single CPU. To improve the efficiency, we create multiple threads which can execute similar kinds of ~~prog~~ processes concurrently.

Eg - if multiple requests for printing process occurs, non-existance of Threads will lead to have multiple processes with same job in a queue.

Therefore, a thread is basically a copy of process sharing a common data and code but with individual stack and registers to hold memory elements & faster content switching.

- 2 types of threads:-

1) User level thread - Managed by user library or applications, without any intervention of OS's kernel.
Eg - Web server using user-level threads to handle multiple client requests concurrently. However, if one request involves a slow I/O operation, all other requests might get blocked.
* Creation, scheduling & management are done in user space, making them fast & efficient.

2) Kernel level Thread:- Threads managed by OS & involve system calls.
Eg - A web server handling multiple client requests concurrently using kernel-level threads can efficiently utilize multiple CPU cores.
* Creation, scheduling and management involves system calls leading to higher overhead compared to ULTs.

USER LEVEL THREADS AND KERNEL LEVEL THREADS.

* Managed by user level libraries	* Managed by OS directly.
* Typically faster	* Slower than user level threads because these involve system calls.
* Content switching is faster	* Content switching is slower.
* If one user level thread is blocked, whole process is blocked.	* If one kernel thread is blocked, no affect to other threads.
* Less reliable.	* More reliable.
* Eg - POSIX threads (Pthreads) and Java Threads etc.	* Eg:- Windows Threads, Linux Kernel Threads.
* A Thread has 3 states :- Running, Ready and Blocking.	

Process's Context Switch Time > Kernel Level Thread's CT > User Level Thread's CT

Important Terms Related to System Calls

Q→ File Descriptor:-

- An integer that uniquely identifies an open file of the process.

Q→ File Descriptor Table:-

Collection of integer array indices that are file descriptors in which elements are pointers to file table entries. One unique FD table is provided for each process.

Q→ File Table Entry:-

A structure in-memory surrogate for an open file which is created when processing a request to open file & these entries maintain file position.

* Whenever we write any character from keyboard, it reads from stdin through fd 0, whenever we see any output to video screen, its from fd 1 and any error to video screen is from fd 3.

Different System Calls.

1) create():- It is used to create new empty file in C. It is defined inside <unistd.h> & flags that are passed as arguments are defined inside <fcntl.h>

Syntax - int create (char *filename, mode_t mode);

- Returns first unused file descriptor & -1 when error.
- Working - • Creates new empty file on disk • Create file table entry • Set first unused FD else -1 upon failure.

2) open():- It is used to open a file for reading, writing or both. Also capable for creating file if does not exist. Defined inside <unistd.h> & flags that are passed are defined in <fcntl.h> header file.

Syntax - int open (const char * Path, int flags);

- Flags - O_RDONLY, O_WRONLY, O_RDWR, O_CREAT, O_EXCL, O_APPEND, O_ASYNC, O_CLOEXEC, O_NONBLOCK, O_TMPFILE
- Working - • Finds existing file on disk, create file table entry, set first unused FD to point to file table entry.

3) close():- It tells the OS that you are done with a file descriptor & closes file pointer by fd., defined under <unistd.h> header file.

- Syntax - int close(int fd);
- Working - Destroys file table entry reference by fd. Set element fd of file to NULL in table.

4) Read():- It reads the specified amount of bytes out of input into memory area indicated by buf. It is also defined inside the <unistd.h> header file.

- Syntax:- size_t read (int fd, void * buf, size_t cnt);
- buf needs to point to a valid memory location with a length not smaller than specified size because of overflow.
- cnt is requested number of bytes read while return is actual number of bytes read. Sometimes read system call read fewer bytes than cnt

(5) Write():- Writes cnt bytes from buf to the file or socket associated with fd. cnt should not be greater than INT_MAX It is also defined in <unistd.h> file.

- Syntax - size_t write (int fd, void * buf, size_t cnt);
- buf needs to be at least as long as specified by cnt
- If write() is interrupted by a signal:-
  (a) If write() has not written data, returns -1 & sets errno to EINTR
  (b) If write() has written data, returns no. of byte it wrote.

PROCESS MANAGEMENT

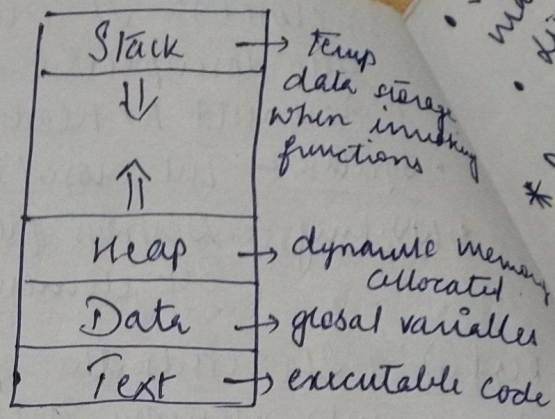
- A process is a program in execution. The original as well as binary code (compiled file) both are programs and when we actually run a binary code, it becomes process.
- Process management is how processes are carried out which includes, stopping processes, setting which process should get more attention etc. Efficient resource allocation, conflict-free process execution & optimal system performance.

Characteristics of Process:-

- Process ID
- Process State
- CPU registers
- I/O status info
- CPU scheduling info
- Accounts info

These are also known as content of process. Every process has its own process control Block (PCB).

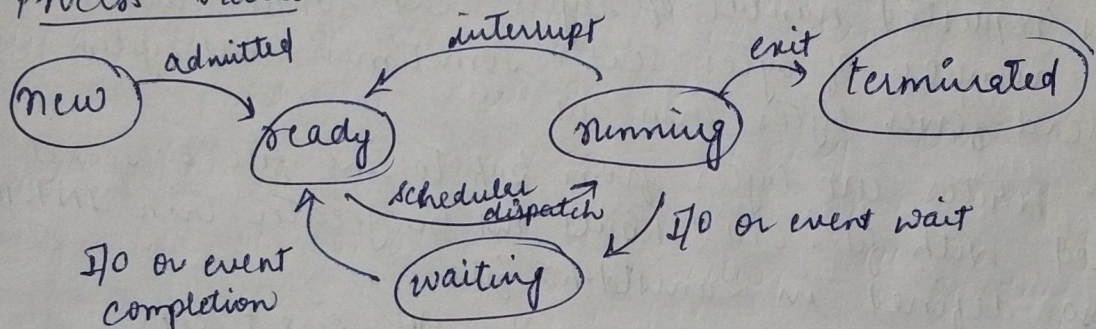
Process in a memory



- Text Section – A process also includes the current activity represented by value of PC.
- Stack – Contains Temporary data such as function parameters, returns addresses and local variables.
- Data Section – Contains global variable
- Heap Section – Dynamic memory allocated to process during run time.

* Size of text and data → fixed and stack & heap → shrink/grow.

Process States



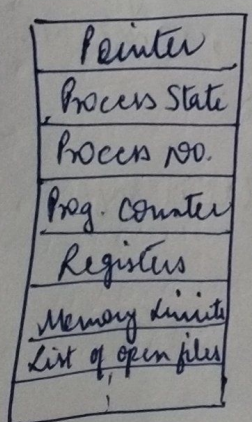
Backing store serves as a backup to the primary memory holding data that can be loaded into RAM when needed

New → Newly created process (OS is setting up PCB)
Ready → After creation, process moves to ready state (ready for execution)
Running → Currently running process in CPU.
Wait → When process requests I/O access.
Complete → Process completed its execution

Backing store – A secondary storage is a non-volatile storage medium used to store data that is not currently in active use by computer

PROCESS CONTROL BLOCK (PCB)

A data structure used by OS to manage info about a process. It contains info about process i.e registers, quantum, priority etc.



- Pointer – Stack pointer required to be saved when process is switched from one state to another.
- Process state – Stores respective state of process.
- Process number – Unique ID that stores process identifier.
- Program Counter – Stores address of next instruction to be executed
- Registers – When a process is running & its time slice expires, current value of process specific register would be stored in PCB & process will be swapped out. When process is scheduled to be run, register values is read.

- **Memory limits** - Field contains info about memory management system used by OS.
- **List of Open files** - Includes list of files opened for Process.

* PCB is stored in a special part of memory that normal users can't access. Some OS place the PCB at start of kernel stack for process as this is safe & secure spot.

Advantages of PCB

- Efficient Process Management
- Resource Management
- Process Synchronization
- Process Scheduling.
- Individuality
- State saving
- ACID property

DISADVANTAGES OF PCB

- Overhead
- Complexity
- Scalability
- Security

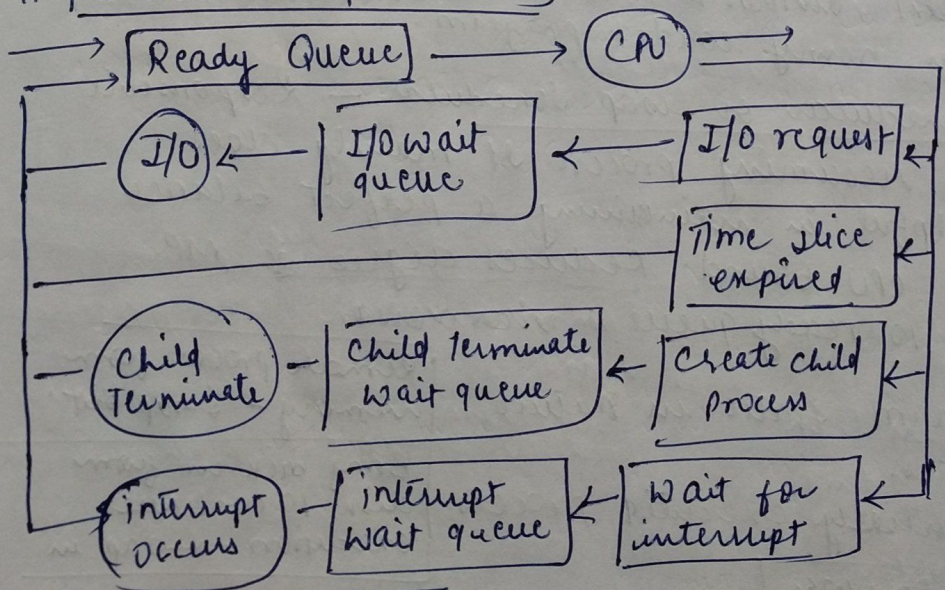
* For n processes, we need n PCBs.

PROCESS SCHEDULING

- Objective of multiprogramming
- Objective of Time sharing

No. of processes currently in memory is known as degree of multiprogramming.

SCHEDULING QUEUE



TYPES OF QUEUES

① **Job Queue** ↓ All processes are placed in work queue at start

② **Ready Queue** ↓ Processes that are ready to execute Stored in primary memory.

③ **Waiting Queue** ↓ PCB saved in waiting queue, which CPU will use after I/O is completed

SCHEDULARS

It is a software module in an Operating system that chooses which processes to run and which jobs to admit into the system.

Process scheduling is an activity of process manager that handles removal of running process of CPU and selection of another process based on particular strategy.

Types of Schedulars

(1) Long Term or Job Scheduler – Brings new processes to Ready state, controls degree of multiprogramming. These make careful selection b/w I/O and CPU bound processes.
I/O bound tasks are which use much of their time in input / output operations, while CPU bound processes are which spend their time on CPU.

(2) Short Term or CPU Scheduler – It only selects process to schedule it doesn't load the process on running. It is responsible for ensuring no starvation due to high burst time process. Dispatcher is responsible for loading the process selected by STS on CPU. ~~scheduler~~

Dispatcher – Switch context, switch to user mode, jump to proper location in newly loaded program.

(3) Medium Term Scheduler or Swap Scheduler – Responsible for suspending and resuming process. It mainly does swapping. It is helpful in maintaining a perfect balance b/w I/O bound and CPU bound. Reduces degree of MP.
It swaps from wait to ready queue & vica versa.

- LTS – Deals with program static in nature.
- STS – Deals with process
- MTS – Deals with partially executed process.

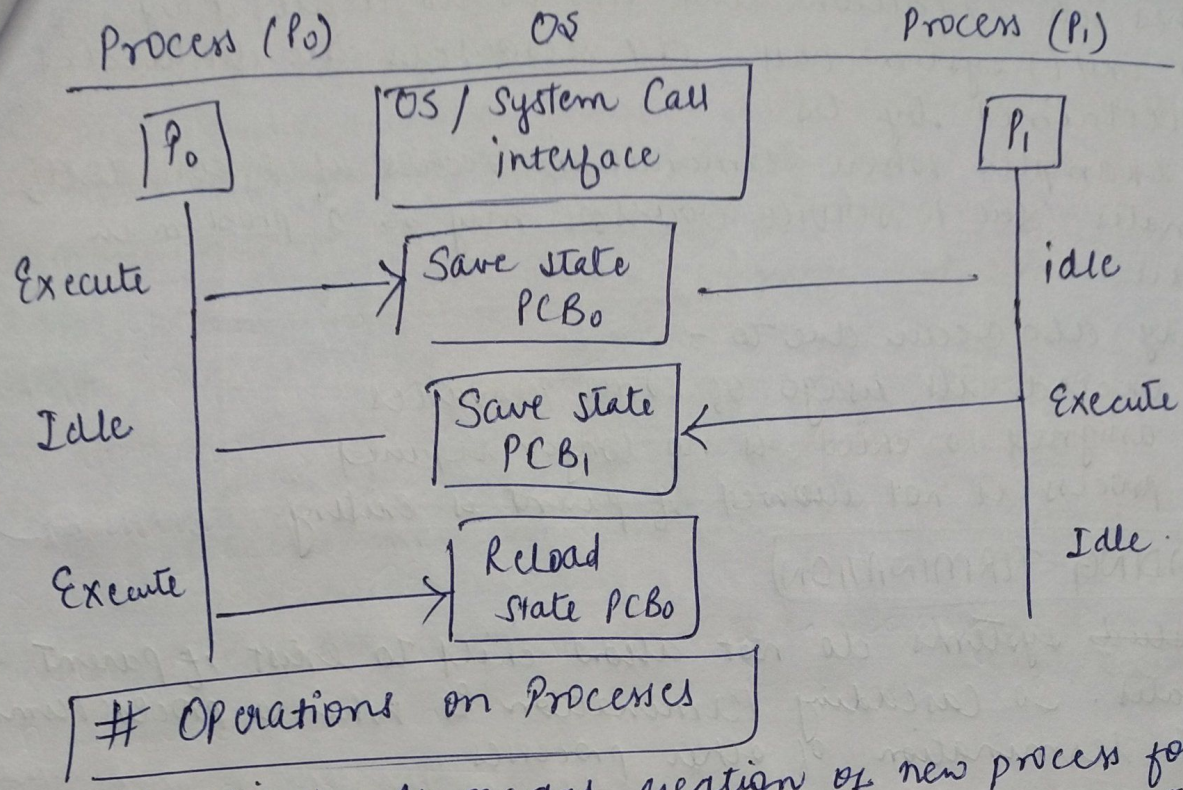
Remove process from memory – swap out

Bring back in from disk to continue execution – swap in

USE OF PROCESS SCHEDULING

- Increase degree of multiprogramming
- Increase efficiency of CPU.
- Avoid collision b/w processes.
- Smooth context switching.

CONTEXT SWITCHING



Operations on Processes

(1) Process Creation - It means creation of new process for execution. During course of execution, process may create several new processes. Creating process is called parent process & new process is called child process.

For new process, two possibilities for execution :-
(a) Parent continues to execute concurrently with children
(b) Parent waits until some/all of its children have terminated.

For new process, 2 address-space possibilities :-
(a) Child process is a duplicate of parent process.
(b) Child process has a new program loaded into it.

(2) Scheduling / Dispatching - State of process is changed from ready to run.
It is done by OS when resources are free or process has higher priority than ongoing process.

(3) Blocking - When a process invokes an input/output system call that blocks process, OS is put in block mode.
In this operation, OS puts process in waiting state.

(4) Preemption - When a timeout occurs, then OS preempts the process. This operation is valid where CPU scheduling supports preemption. Basically this happens in priority scheduling.
OS then puts process in ready state.

(5) Process Termination – It is done when a process finishes its execution and asks OS to delete it by using exit() system call. All resources are deallocated and reclaimed by OS.
Other examples where termination occurs if process itself terminates due to service error or may be a problem in hardware.

→ It may also occur due to –
- Child exceeds its usage of some resources
- Task assigned to child is no longer required.
- Child process is not allowed if parent is exiting.

CASCADING TERMINATION

Some ~~stud~~ systems do not allow child to exist if parent terminates. So cascading termination is when a process termination leads to termination of other processes.

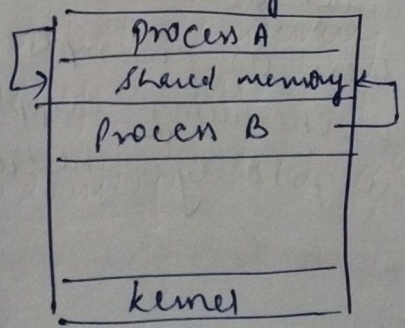
Independent Process – cannot be Affected by other processes executing in system. They do not share data.

Cooperating Process – Can be affected / can affect other processes executing in system. They share data. Info sharing, modularity, computation speed up & convenience are the reasons.

Interprocess Communication (IPC) (for cooperative processes)

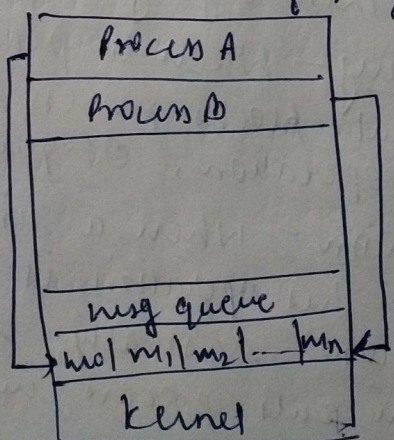
Shared Memory

Region of memory shared by cooperating processes is established. Processes can then share msg / info by reading / writing to shared region.
(Share same memory space)



Message Passing

Communication takes place by means of messages exchanged b/w them. Useful for exchanging smaller amount of msg.



• Shared memory can be faster as system calls are required only to establish shared-memory regions.
• Message passing is useful in a distributed environment where communicating processes may reside on different computers connected with a network.

Message Passing —
→ Processes communicate with each other without resorting to shared variables.
→ 2 operations – send() & receive()
→ Msg size is either fixed or variable.
→ If P & Q processes have to communicate, they need to establish a communication link.
  ↳ Physical – shared memory, hardware bus, network
  ↳ Logical – direct/indirect, sync/async, automatic/explicit buffering

SCHEDULING CRITERIA

* CPU utilization – Keep CPU as busy as possible
* Throughput – No. of processes that complete their execution per unit time
* Turn Around Time – Amount of time to execute a particular process
* Waiting Time – Amt of time a process has been waiting in the ready queue.
* Response Time – Amt of time it takes from when a request was submitted until first response is produced not output
* Arrival Time – Time at which process enters ready queue

Formulae –
(1) TAT = Completion Time – Arrival Time
(2) WT = TAT – Burst Time
(3) RT = CPU allocation Time – AT

Burst Time – Total running duration
Completion – The point of time at which process ended.

Optimisation Criteria
• Max CPU utilization • Max Throughput
• Min turnaround Time • Min waiting time • Min response time

Types of Scheduling
• Preemptive – Means once a process started its execution, the currently running process can be paused for a short period of time to handle some other process of higher priority
Eg – Round Robin, SRTF, Preemptive Priority Scheduling, Longest Remaining Time First

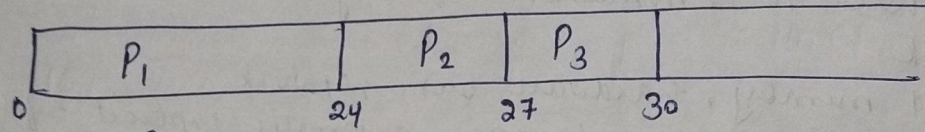
- Non-Preemptive - Means once a process starts its execution, it cannot be halted.

Eg - FCFS, SJF, Non-preemptive Priority Scheduling, Longest Job First (LJF)

① First Come First Serve (FCFS) :- Just Straight first process Then next ---

Process	BT	AT	WT	TAT	RT	CT
P_1	24	0	0	24	0	24
P_2	3	0	24	27	24	27
P_3	3	0	27	30	27	30

TAT = BT + WT - AT



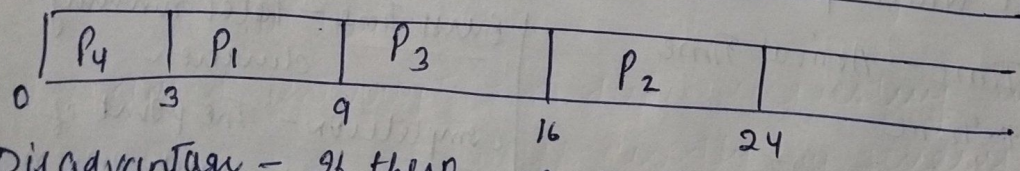
Gantt Chart

Disadvantage - Convoy effect which means process with small Burst Time have to wait for quite long for process with large BT to finish executing.

② Shortest Job first (SJF) :- The one with small BT gets executed.

Process	BT	AT	WT	RT	TAT
P_1	6	0	3	3	9
P_2	8	0	16	16	24
P_3	7	0	9	9	16
P_4	3	0	0	0	3

Has min avg wait Time



Disadvantage - If there are a lot of short processes, long process may never get a chance to run.
Also difficult to know the length of next CPU request.

③ Shortest Remaining Time First (SRTF) :- (Preemption + Arrival Time + SJF)

Advantage - Makes processing of jobs faster than SJN algo.

Disadvantage - Much more context switching is done.

INTERRUPTS

It is a signal emitted by hardware or software when a process or an event needs immediate attention. In I/O devices one of the bus control lines is dedicated to this purpose and is called Interrupt Service Routine

Process-

1) Device raise IRQ
2) Process interrupts prog. currently executing
3) Device is informed, its request is recognized & device deactivates the request signal.
4) Requested action is performed
5) Interrupt is enabled & interrupted is resumed.

How interrupts are handled ?

Handling Multiple Devices IRQ ?

3 methods if more than one device raises interrupt :-

1) Polling :- First encountered device with IRQ set bit is the device that is to be served first. It is easy to implement but a lot of time is wasted by interrogating IRQ bit of all devices.

2) Vectored Interrupts :- A device requesting an interrupt identifies itself directly by sending a special code to the processor over bus. The special code can be starting address of ISR or where ISR is located in memory & is called interrupt vector

3) Interrupt Nesting :- I/O devices are organised on a priority structure. Interrupt request from higher ~~device~~ priority device is recognised but not from lower priority device.

• Process' priority is encoded in few bits of Process Status Register.

Interrupt Latency

Amount of time b/w generation of interrupt and its handling is called interrupt latency.

- It is affected by :-
  - → No. of created interrupts
  - → No. of enabled interrupts
  - → No. of interrupts that may be handled
  - → Time required to handle each interrupt

While processor is handling interrupts, it must inform the device that its request has been recognized so that it stops sending interrupt request signal (IRQ)

How CPU reacts to Interrupts ?

1. Interrupt Detection
2. Interrupt Acknowledgement
3. Interrupt Handling
4. Content Switching
5. Transfer control
6. Interrupt Servicing.

Types of Interrupts

1. Software Interrupts – Produced by software or system. Another names are traps and exceptions. Serve as a signal for Operating system to carry out a certain function or respond to an error condition. Software interrupts often occur when system calls are made. Interrupt instruction is used (INT) to create software interrupts. The interrupt handler routine completes the required work or handles any error.

The (INT) interrupt instruction is used to trigger a software interrupt & is followed by number that specifies which interrupt to trigger.

2. Hardware Interrupts – All devices are connected to the interrupt request line and a single request line is used for all n devices.

- 2 types –

A) Maskable interrupt – A bit in mask register corresponds to each interrupt signal.

B) Spurious interrupt – An interrupt for which there is no source. Also known as phantom / ghost interrupts

DIFFERENCE B/W SOFTWARE AND HARDWARE INTERRUPTS

Hardware Interrupts	Software Interrupts
• Generated from an external device or hardware.	• Generated by internal system of computer.
• Do not increment PC.	• Increments PC.
• Can be invoked by external device or occurance of hardware failure.	• Invoked with help of INT instruction.
• Lowest priority than software interrupts.	• Highest priority among all interrupts.
• Asynchronous event.	• Synchronous event
• Eg- keystroke event & mouse movements.	• Eg- All system calls.

Benefits of Interrupts

- Real Time Responsiveness – Permits a system to reply promptly to outside events.
- Efficient Resource usage – Permits processor to remain idle until an event occurs.
- Multitasking and Concurrency – Allows processor to address multitasking concurrently.
- Improved System Throughput – It allows device to overlap computation with I/O operations thus maximizing throughput of system.

INSTRUCTION CYCLE

It is the cycle that the CPU follows from boot up until the computer has shut down in order to process instructions.

It composes of 3 stages :-

1) Fetch 2) Decode 3) Execute

- Registers involved :-

1) Memory Address Register (MAR) :- Connected to address lines of system bus. Specifies address in memory for read/ write operation

2) Memory Buffer Register (MBR) :- Connected to data lines of system bus. Contains value to be stored in memory

3) Program counter - Holds the address of next instruction to be fetched.

4) Instruction Register - Holds last instruction fetched.

STAGE-1 (FETCH)

1) PC holds memory address of next instruction which is copied into memory address register (MAR) & then PC is incremented.
2) CPU then takes instruction at memory address described by MAR and copies to memory data register (MDR).
3) MDR holds data fetched from memory or data waiting to be stored in memory.
4) Instruction in MDR is copied to current instruction register (CIR or IR) acting as temporary holding ground for instruction just fetched.

STAGE-2 (DECODE) (Optional)

1) Control unit (CU) decodes instruction in IR.
2) If instruction is direct, nothing is done during this clock pulse.

STAGE-3 (EXECUTE)

1) CPU sends signals to Arithmetic Logic Unit (ALU) & floating point unit (FPU).
2) ALU performs add, sub, multiplication (via repeated addition) and division (via repeated substraction).
3) Also performs logic operations (AND, OR, NOT, binary shift)
4) FPU is reserved for performing floating point operations.

Instruction Primer

Instruction contains bits that specify action processor is to take. Processor reads instruction & perform necessary action.

4 Categories of action -

1) Memory - Data transfer from processor to memory / memory to processor
2) I/O - Data transfer to / from peripheral devices (I/O module).
3) Data processing - Perform arithmetic or logical operations on data.
4) Control - An instruction may specify sequence of execution be altered.

Important Leftover Topics

FORK() — New process has its own address space, memory

A system call in Unix based OS which is used to create a new process which is a copy of calling (parent) process. New process is called as child process.

Characteristics :-

(1) Process Duplication — Child process is almost a duplicate of parent process and it gets its own unique process ID but initially it shares same code, data and open file descriptors as parent.

(2) Return values -
- Parent Process - fork() returns PID of child process.
- Child Process — fork() returns 0.
- Error - If fork() fails, it returns -1 & no child created.

(3) Memory Space - Child process gets a copy of parent's memory space. Initially this is a copy-on-write, meaning memory is shared until either process modifies it.

Use Case -

(1) Creating Daemons — Background processes that handle system tasks

(2) Handling multiple clients - Servers can fork a child process to handle each client allowing concurrent handling of multiple connections.

(3) Parallel Processing — Fork can be used to execute tasks in parallel in separate processes.

* Zombie Process — If a parent process doesnot call wait() or waitpid() for a terminated child, child remains in the process table as a zombie.

* The fork system call use the memory management strategy known as copy-on write. Until one of them makes changes to shared memory, it enables parent & child processes to share the same physical memory. To preserve data integrity, a second copy is then made.

- ADVA
  - Improvi
  - code re
  - Memory spa
  - Process isolat
- DISADVANTA
  - Mem

• ADVANTAGES OF FORK SYSTEM CALL

→ Improving system's efficiency & multitasking skills.
→ code reuse.
→ Memory optimisation
→ Process isolation (every fork system call process is given its own memory afterwards)

• DISADVANTAGES OF FORK SYSTEM CALL

→ Memory overhead
→ Duplication of resources
→ Communication Complexity
→ Impact on system performance.

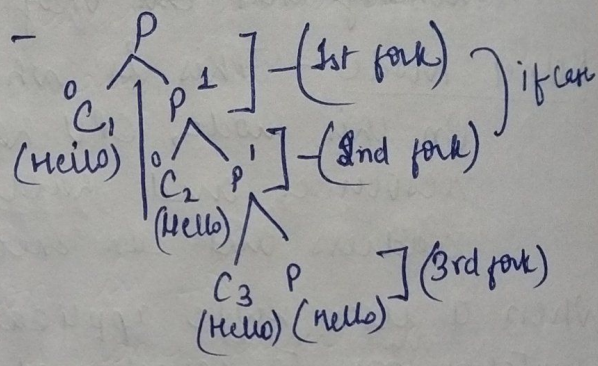
QUESTIONS ON FORK

* In simple fork, 2^n is total no. of times a whole process executes.
* No. of children = $2^n - 1$, No. of parent = 1

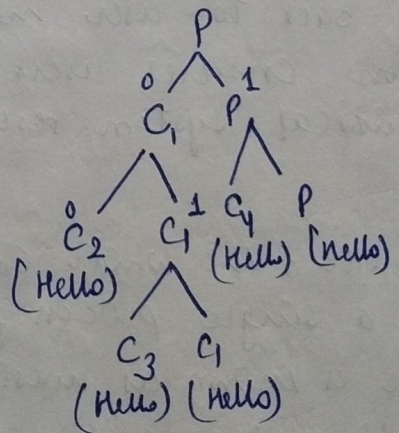
Q1)
```
int main()
{
if (fork() && fork())
  fork();
printf("Hello");
return 0;
}
```

Output: Four times Hello

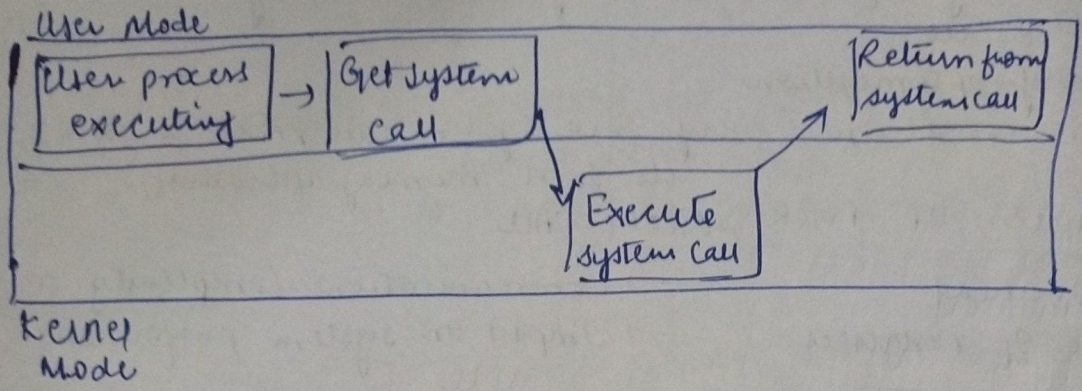
Explanation –



Q2)
```
int main()
{
if (fork() || fork())
  fork();
printf("Hello");
return 0;
}
```



User Vs Kernel modes



User Mode - This is where regular applications run. CPU has restricted access to system resources and hardware. It can only execute a subset of instructions and can only access memory that ~~can only~~ has been allotted to application.

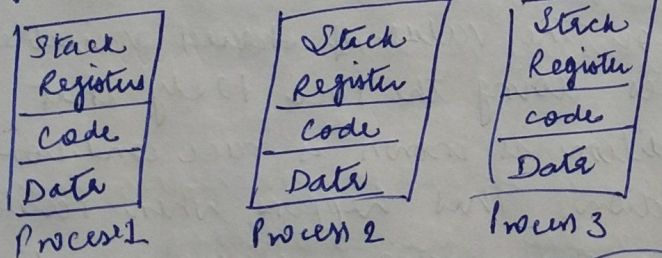
Kernel Mode - This is where OS's core components run. In this mode, CPU has unrestricted access to all system resources and hardware. Kernel can access any memory address and can execute any instruction.

When a user mode application makes a system call, CPU switches ~~back~~ to kernel mode to execute requested operation and then switches back to user mode once operation is completed. This separation ensures users applications can't directly access or modify critical system resources providing a layer of security and stability.

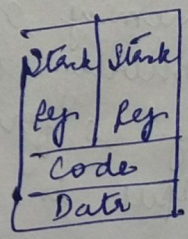
* In kernel mode, the whole OS might go down if an interrupt occurs but a single procss fails in user mode.
* Kernel Mode is known as master mode, priviledged mode or system mode and User Mode is known as unprividedged, restricted or slave mode.
* In kernel mode, processes share single virtual address space. In user mode, processes have separate virtual address space.

#PROCESS VS THREADS

- Any program in execution
- Takes more time to Terminate
- Takes more time for creation
- Takes more time for content switching.
- Less efficient in terms of communication.
- Process is isolated.
- Heavyweight process.
- Process switching use interface in OS.
- If one process is blocked, then it will not effect execution of other processes.
- Process has its own PCB, stack and Address space.
- System call is involved.



- Segment of a process.
- Takes less time to terminate
- Takes less time for creation
- Takes less time for content switching.
- More efficient in terms of communication.
- Threads share memory.
- Light weight.
- Thread switching does not require calling an OS
- If user level thread is blocked then all other threads are blocked.
- Thread has Parent's PCB, its own Thread Control Block, stack and common address space.
- No system call, its created using APIs.



PROCESS SYNCHRONIZATION

It is crucial in OS to ensure that multiple processes can run concurrently without interfering with each other. When multiple processes access shared resources, there's a risk of conflicts and inconsistencies. Process synchronization ensures that these shared resources are accessed in a controlled manner, preventing issues like race conditions, deadlocks & data corruption.

- **CRITICAL SECTION** – Part of a program that accesses shared resources and must not be executed by more than one process at a time. Challenge is to design mechanism that allow only one process to enter critical section at a time.

```
do
{
entry section
    critical section
exit section
    remainder section
}
while(TRUE);
```

* Any solution to the critical section must satisfy 3 requirements:

* Mutual Exclusion - If a process is in critical section, then no other process is allowed to execute in critical section.

* Progress - If no process is executing in critical section & other processes are waiting, then only those processes that are not executing in their remainder section can participate in deciding which will enter critical section next.

* Bounding Wait - A bound must exist on no. of times that other processes are allowed to enter their critical sections after a process has made request to enter its critical section & before request is granted.

RACE CONDITION

It occurs when the output or state of process depends on sequence or timing of other uncontrollable events.

When more than one process is executing same code or accessing same memory or any shared variable in that condition there is a possibility that the output or the value of shared variable is wrong so for all the processes doing the race to say that my output is correct, this condition is known as race condition. This occurs inside a critical section. This happens when result of multiple thread execution in critical section differs according to order in which the thread executes. It can be avoided if critical section is treated as atomic instruction.
Proper synchronization mechanisms can prevent race conditions.

PETERSON'S SOLUTION (Solution to critical section Problem)

Classic software - based solution to critical section problem.
We have 2 shared variables.

- boolean flag[i] => initialised to false (initially no one is in CS)
- int turn => Process whose turn is to enter in CS.

This solution preserves all three conditions:-
Mutual Exclusion, Progress & Bounded Waiting.

```
do
{
flag[i] = TRUE;
turn = j;
while ( flag[j] && turn == j);
        CRITICAL SECTION
flag[i] = false;
        REMAINDER SECTION
} while(true);
```

DISADVANTAGES

→ Involves Busy Waiting
→ Limited to 2 processes
→ Can't be used in modern CPU architectures.

SEMAPHORES – Used to implement critical sections → P means down (-) V means up (+)

It is a signaling mechanism and a thread that is waiting on a semaphore can be signaled by another thread.
Different from mutex which can be signalled only by thread that is called wait function.
Use 2 atomic operations - wait and signal.
semaphore is an integer variable.

If semaphore value is 10, how many processes can perform/ enter CS?
10 (Negative means blocked)

- There are 2 Types of Semaphores:-

(1) Binary Semaphores:- Can either be 0 or 1. Also known as mutex locks (as they provide mutual exclusion). All processes can share same mutex semaphore that is initialized to 1. Process has to wait until lock becomes 0. Then process makes mutex as 1 & start CS & then again set to 0.

(2) Counting semaphores:- Can have any value & are used to access to a resource that has a limitation on no. of simultaneous accesses. It is initialized to no. of instances of resource. Whenever a process wants to use that resource, it checks if no. of remaining instances is more than 0. The process enter its CS and decrease value by 1.

Semaphores are more flexible synchronizing primitives.

Advantages - Ensures data consistency & integrity; avoids race condition, prevents inconsistant data, supports efficient & effective use of shared resources.

Disadvantages - Adds overhead, lead to performance degradation, increases complexity, can cause deadlock.

PRODUCER CONSUMER PROBLEM

It is a classic example of a multi-process synchronization problem. It involves 2 types of processes, the producer and consumer, which share a common fixed size buffer.

Producer – Generates data & puts in buffer

Consumer – Takes data from buffer for processing.

Ensure that producer does not try to add data into full buffer, consumer does not try to remove data from empty buffer.

KEY CONCEPTS

1. Buffer – Shared memory area used by both producer & consumer. Has a fixed size N acting like circular queue.
2. Mutex – Ensures only one process accesses the buffer at a time.
3. Semaphores –
   a. Full – Counts no. of items in buffer. Initialized to 0.
   b. Empty – Counts no. of empty spaces in buffer. Initialized to N.

We use semaphores to keep track of number of full and empty slots in buffer and a mutex to ensure mutual exclusion when accessing the buffer.

This problem demonstrates necessity of process synchronization to prevent race condition and ensure data consistency.

PRODUCER PROCESS

```
void Producer(){
    int item;
    while (true){
        item = produce_item();
        sem_wait(&empty);
        sem_wait(&mutex);
        add_to_buffer(item);
        sem_post(&mutex);
        sem_post(&full);
    }
}
```

CONSUMER PROCESS

```
void consumer(){
    int item;
    while (true){
        sem_wait(&full);
        sem_wait(&mutex);
        item = remove_from_buffer();
        sem_post(&mutex);
        sem_post(&empty);
        consumer_item(item);
    }
}
```

PRINTER SPOOLER PROBLEM

It is like a manager for your print jobs. When you send something to print, spooler puts it in a queue and waits for printer to be ready.

- This problem is a classic example of process synchronization which involves coordinating processes to ensure they execute without conflict especially ~~without~~ when sharing resources.

Imagine multiple processes sending print jobs to single printer which can handle only one job at a time, so spooler is used to manage these jobs.

Key concepts

1. Critical Section - Shared resource here is printer queue. When a process adds or removes a job from queue, it enters a CS.
2. Mutual Exclusion - Ensures that only one process can modify the printer queue at a time, preventing race condition.
3. Deadlock and Starvation - Synchronization must ensure that no process gets stuck wait indefinitely and then every process eventually gets its turn.

SYNCHRONIZE MECHANISMS

1. Mutex - Ensures only one process can modify print queue at any time.
2. Semaphores - To manage number of available spots in queue and number of jobs waiting.
   - Empty semaphores - Tracks available slots in queue.
   - Full semaphores - Tracks number of jobs in queue.

WORKING

1) Producer - When a process submits a print job, it waits if queue is full, locks mutex to safely add the job to queue. Unlocks mutex and signals that a new job is available.
2) Consumer - Printer process continuously checks for jobs, Waits if queue is empty, locks the mutex to safely remove job from queue, unlocks the mutex and signals that there's now space in queue & processes print job.

LOCK VARIABLE

A lock variable (mutex lock) is used to control access to shared resources by multiple processes or threads. The primary purpose of lock is to ensure that only one process / thread can access a critical section of code at a time, preventing race conditions & ensuring data consistency.

HOW IT WORKS

- Lock Acquisition - Before entering CS, a thread attempts to acquire lock. If lock is available, the thread acquire it and proceeds. If lock is already held by another thread, the attempting thread must wait until the lock is released.
- Lock Release - After finished the operations in CS, the thread releases lock, allowing other waiting threads to acquire.

TYPES OF LOCKS

1. Mutex - A binary lock that ensures mutual exclusion.
2. Spinlock - A lock where a thread waits in a loop (spins) repeatedly checking if lock is available. Suitable for short CS.
3. Read-Write Lock - Allows multiple readers but only one writer at a time, optimizing scenarios with frequent read operations and infrequent write operations.

- Lock variables are essential for managing concurrency in OS, preventing race conditions and ensuring safe access to shared resources.
- It is a software mechanism implemented in user mode.
- It is a busy waiting solution.
- Can be used for more than 2 processes.
- No mutual exclusion preset since it is used for more than 2 processes

TEST AND SET INSTRUCTIONS

It is a crucial atomic operation used in OS for process synchronization. Useful for implementing mutual exclusion in concurrent programming.

Test and
Performs
1) Tests the
2) Sets the
Test - Reads
Set - Writes
• When a
lock is
false

Test and Set instruction is an atomic operation that performs 2 actions in a single :-

(1) Tests the current value of a specified memory location.
(2) sets the value to a new value usually 1.

Test - Reads current value of a memory location.
Set - Writes a new value to that location if current value is 0.

- When a process calls lock function, it continuously loops until lock is available. This TAS instruction ensures if the lock was false then it sets to true and returns old value false allowing process to enter the C.S. When process finishes its CS, it calls the unlock function to set lock to false allowing other processes to enter their CS.

ADVANTAGES

- Simple & easy to implement
- Provides strong mutual exclusion guarantees

DISADVANTAGES

- Busy waiting can lead to inefficient CPU usage.
- High priority processes can be delayed by low-priority ones holding lock.

TURN VARIABLE

It is used to control the execution order of processes or threads that need to access a shared resource or critical section. It essentially provides a simple way to alternate access b/w processes.

The turn variable holds a value (0 or 1) that indicates which process has the permission to enter critical section. The processes check this variable and enter CS only if turn value matches their identity / identifier. Once process finishes, it changes turn variable to allow other process to enter.

Advantages - Simplicity.

Disadvantages - Busy waiting and limited to 2 processes.

* Turn variable is implemented only in user mode. This variable is also known as Strict Alternation Approach. It is a synchronization mechanism that is implemented for synchronizing 2 processes.
* Mutual Exclusion is applicable here.

READERS - WRITERS PROBLEM

A classic synchronization problem where a shared database is present that multiple processes need to access. Some processes want to read data while other want to write data. Goal is to allow multiple readers to access data simultaneously but only one writer can modify data at time.

KEY CHALLENGES

- Read - Write conflicts - Ensure no reader reads data while a writer is modifying it.
- Write - Write conflicts - Ensure that no two writers modify data simultaneously.

SOLUTION

① Reader Priority - Prioritizes readers, allowing them to access resource if no writers are currently writing even if writers are waiting.

② Writer Priority - Prioritizes writers, ensuring that once a writer requests access, it gets it as soon as possible even if readers are currently reading.

Using Semaphores

- rw_mutex - Ensures mutual exclusion for writers, block readers if writer has acquired it.
- mutex - Protects read count variable.
- read_count - Keeps track of no. of readers currently accessing data.

→ Reader's Process

- Entry section - Readers increment 'read_count' & if its the first reader, they wait on 'rw_mutex' to block writers.
- Exit section - Readers decrement 'read_count' and if its last reader, they signal 'rw_mutex' to allow writers.

→ Writer's Process

- Writers wait on 'rw_mutex' to ensure exclusive access.
- Once done, they signal 'rw_mutex' to release the lock for others.

DINING PHILOSOPHER PROBLEM

Imagine 5 philosophers around a table. There are 5 forks between them and each philosopher needs two forks to eat.

KEY CHALLENGES

① Deadlock

DINING PHILOSOPHER PROBLEM

Imagine 5 philosophers sitting around a circular dining table. There are 5 plates of noodles & 5 forks. Each philosopher needs 2 forks to eat noodles. They alternate between thinking and eating.

KEY CHALLENGES

(1) Deadlock Prevention - No 2 philosophers should be able to pick up same fork simultaneously, which could cause a deadlock where none of the philosophers can eat.

(2) Starvation Avoidance - No philosopher should starve by being unable to get both forks indefinitely while waiting for others to complete.

(3) Concurrent Access - Multiple philosophers should be able to eat concurrently without conflict when possible.

SEMAPHORE SOLUTION (Similar in number as to no. of forks)

- mutex - ensures mutual exclusion when picking up or putting down forks.
- fork[] - array of semaphores representing forks.

Process - Thinking, Enter CS, Pick forks (left first), Eat, Put down fork (left first)

- Deadlock Prevention is by introducing an asymmetry by having an odd numbered philosopher pick up left fork first & even numbered philosopher pick up right fork first.
- Starvation Avoidance by implementing fair scheduling to ensure each philosopher gets chance to eat.
- Deadlock condition - When all philosophers have one fork to eat but are waiting for other fork which is held by the adjacent one. (Case when a process executes some time & then preempt)

DEADLOCK

It is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource held by another process in set. As a result, none of the processes can proceed.

CONDITIONS FOR DEADLOCK

1. Mutual Exclusion - At least one resource must be held in a non-sharable mode; only one process can use resource at any given time.
2. Hold and Wait - A process holding at least one resource is waiting to acquire additional resources currently held by other processes.
3. No Preemption - Resources cannot be forcibly taken away from process holding them; they must be released voluntarily by holding process.
4. Circular Wait - There exists a set of processes forming a circular chain waiting for each other resources.

DEALING WITH DEADLOCKS

1. Mutual Exclusion :- This condition must hold for non-shareable resources.
2. Hold and Wait - Require processes to request all resources at once or ensure they release all held resources before requesting new ones.
3. No Preemption - Allow preemption of resources.
4. Circular Wait - Impose a total ordering of all resource types and require process to request resources in increasing order of enumeration.

DEADLOCK AVOIDANCE

- Banker's Algorithm allocates resources safely ensuring that the system never enters an unsafe state where deadlocks could occur.

BANKER'S ALGORITHM

It is a resource allocation and deadlock avoidance algorithm. It ensures that a system will allocate resources only if it remains in safe state; thus avoiding deadlock.

KEY CONCEPTS

(1) Safe state – A state is safe if there is a sequence of processes such that each process can request and obtain needed resources & eventually release them.

(2) Resource Allocation Graph – Tracks which resources are currently allocated & which are available.

- The banker's algorithm is named so because it is used in banking system to check whether a loan can be sanctioned to a person or not.

Resource Allocation Graph (RAG) – A visual representation used to model and analyse allocation of resources to process in system. It helps in understanding & detecting potential deadlocks.

- Components –

(1) Processes – Represented as circles.
(2) Resources – Represented as squares.
(3) Edges – (a) Request Edge – Directed from process to resource.
(b) Assignment Edge – Directed from resource to process.

- In a RAG –

(1) If there are no cycles, there is no deadlock.
(2) If there is a cycle, it indicates a potential deadlock but there must be the condition of hold & wait.

ADVANTAGES

(1) Visualization of state of resource allocation & conflicts.
(2) Simplifies process of detecting deadlocks.
(3) Provides clear way to analyse allocation strategies and impacts.

Multi Instance Resource Allocation Graph

In real world, some resources have multiple instance & so Multi instance RAG allows resources to have more than one instance.
Instances are represented by dots inside the square.
- If there is no cycle, there's no deadlock.
- If there is a cycle, check if processes within cycle can get the required resources. If they can't, deadlock exists.

∴ Multi-Instance RAGs are an essential tool for managing complex systems with limited resources ensuring smooth and efficient operation.

DEADLOCK HANDLING METHODS

(1) Deadlock detection and recovery - The system periodically checks for deadlocks using algorithms that analyse RAG for cycles. Once deadlock is detected, system recovers by -
(a) Terminating one or more processes involved.
(b) Preempting resources from some processes & allocating to others.

(2) Deadlock Avoidance - System ensures it never enters a deadlock state by carefully allocating resources. Uses Banker's Algo to evaluate resource requests.

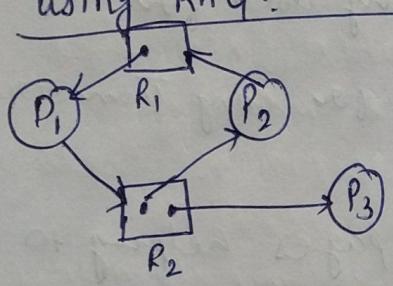
(3) Deadlock Prevention -
(a) Mutual Exclusion - At least one resource must be held in a non-sharable mode.
(b) Hold & wait - A process holding at least one resource is waiting to acquire additional resources held by other processes.
(c) No-preemption - Resources can't be preempted.
(d) Circular Wait - A closed chain of processes exists, where each process holds at least one resource needed by next process in chain.

(4) Deadlock ignorance (Ostrich method) :- Used in some systems where OS assumes that deadlocks rarely occur & therefore does not implement any specific mechanism to prevent

Difference b/w Deadlock and Starvation

Deadlock	Starvation
→ Involves a set of processes where each process waits indefinitely for a resource held by another process in set.	→ Involves one or more processes waiting indefinitely for resources due to continuous preference given to other processes.
→ Requires specific algorithms for detection and strategies for recovery or prevention.	→ Generally resolved through changes in resource allocation and scheduling policies
→ System may come to complete halt	→ Only affected processes are hindered.

Q→ How to detect deadlock using RAG?



	Allocated R_1	Allocated R_2	Requested R_1	Requested R_2
P_1	1	0	0	1
P_2	0	1	1	0
P_3	0	1	0	0

∴ No deadlock

• Current Availability = (0, 0)

Using this avail

	Availability
✓ P_3 ⇒	0 1
✓ P_1 ⇒	1 1
✓ P_2 ⇒	1 1

Q→ Question on Banker's Algorithm (A=10, B=5, C=7) Given

Processes	Allocation A	B	C	Max Need A	B	C	Available A	B	C	Remaining Need A	B	C
P_1	0	1	0	7	5	3	~~3~~ 7	~~3~~ 5	2 ~~5~~	7	4	3
P_2	2	0	0	3	2	2	5	3	2	1	2	2
P_3	3	0	2	9	0	2	10	5	7	6	0	0
P_4	2	1	1	4	2	2	7	4	3	2	1	1
P_5	0	0	2	5	3	3	7	4	5	5	3	1
	7	2	5									

Max − Allocation

Sequence → P_2, P_4, P_5, P_1, P_3

332
+200
532
+211
743
+002
745
+010
755
+302
1057

MEMORY MANAGEMENT AND DEGREE OF MULTIPROGRAMMING

Crucial function of Operating system that manages allocation and deallocation of memory spaces.

(1) Allocation of Memory –

- Static Allocation – Memory is allocated at compile time and remains fixed throughout program execution.
- Dynamic Allocation – Memory is allocated during runtime allowing for flexibility.

(2) Types of Memory –

- Primary Memory (RAM) – Fast & volatile memory used for active processes.
- Secondary Memory – non-volatile memory for long term storage

(3) Memory management Techniques –

- Paging – Divides memory into fixed-size pages helping to manage and allocate memory efficiently.
- Segmentation – Divides memory into variable sized segments based on logical division of programs.
- Virtual memory – Extends physical memory by using disk space to simulate additional RAM, allowing for larger processes than available physical memory.

(4) Address Translation –

- Logical Address – Generated by CPU.
- Physical Address – Actual address in memory unit.
- MMU (Memory Management Unit) – Hardware that translates logical addresses to physical addresses.

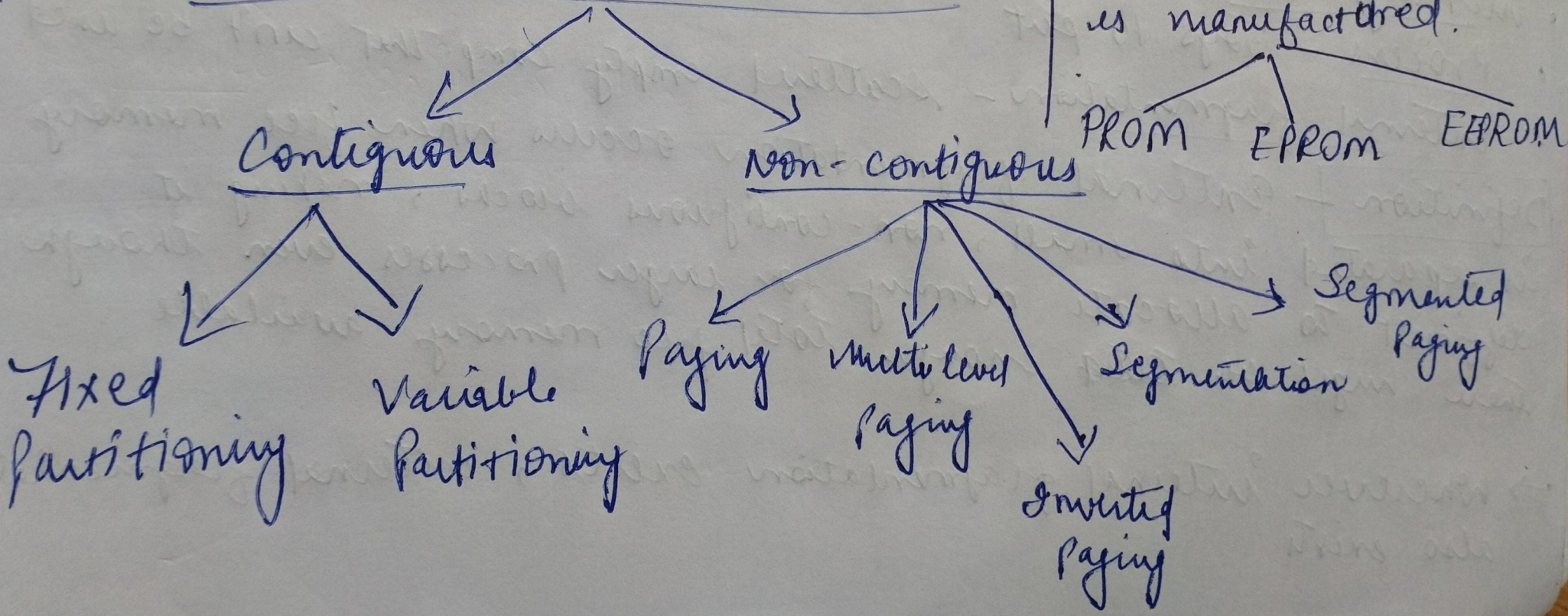
- Degree of multiprogramming refers to number of processes in memory at one time. It's a measure of how much program are executing simultaneously, improving resource utilization and system throughput.

Factors affecting degree of multiprogramming –

→ Memory size → Efficient CPU scheduling algos → Efficient I/O systems

the physical page frames.

MEMORY MANAGEMENT TECHNIQUES



ROM is memory that is defined when computer is manufactured.

PROM EPROM EEPROM

INTERNAL FRAGMENTATION

Imagine you have a bunch of empty boxes that you want to fill with toys & each box can hold exactly 10 toys. Now, suppose you have a toy set with 8 toys only. We put that set in one box & still have 2 empty spaces which we can't put another toy set. This is similar to internal fragmentation.

- Memory Block - Similar to box that can hold toys.
- Process - Toy set you need to fit in box.
- Internal fragmentation - Empty spaces left in box

Definition - Internal fragmentation occurs when fixed sized memory blocks are allocated to processes and memory allocated exceeds actual memory requirement of process leading to unused space within allocated block. This leads to inefficient memory utilization and reduced system performance.

EXTERNAL FRAGMENTATION

Imagine having a toy box divided into smaller compartments and each compartment can hold one toy and you keep putting & taking out toys over time. Now if you get a new larger toy that needs more space & can't fit in single compartment you can't fit it in the box even though total empty space in toy box is enough. These scattered empty compartments cause issue of external fragmentation.

- Memory blocks - Compartment in your box
- Process - Toys to put
- External fragmentation - Scattered empty comp that can't be used.

Definition - External fragmentation occurs when free memory is separated into small, non-contiguous blocks, making it difficult to allocate memory for larger processes even though there might be enough total free memory available.

⇒ Wherever internal fragmentation exists, external fragmentation also exists.

Fixed Partitioning (Static Partitioning)

Giving fixed memory spaces to processes in which they can accomodate.

Problems - Internal fragmentation, Limit in process size, Limitation on degree of multiprogramming, External fragmentation.

Variable Partitioning (Dynamic Partitioning)

Memory partitions are done at runtime.

- No change of internal fragmentation
- No limitation on no. of processes
- No limitation on size of processes.

- External fragmentation can be present.

* External fragmentation can be prevented using compaction.

Memory Management Algorithms

(1) First Fit Allocation - Allocates first available block of memory that is large enough to satisfy request. It is simple and fast but can lead to external fragmentation.

(2) Next-Fit Allocation - Similar to first fit, it starts searching from the location of last allocation rather than the beginning of memory. It reduces search time over first fit and can lead to fragmentation.

(3) Best-Fit Allocation - Allocates the smallest available block of memory that is large enough to satisfy the request. It minimizes leftover space but can lead to external fragmentation.

(4) Worst-Fit Allocation - Allocates largest available block of memory. It reduces fragmentation by leaving larger blocks available but can lead to poor utilization of large blocks.

Questions

Non-Contiguous Memory Allocation (One process can be spanned at 2 locations or more)

PAGING :-

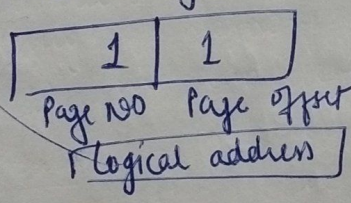
A process present in secondary memory is divided into pages whereas main memory/RAM is divided into frames. Also, size of page will always be equal to size of frame.

Paging is a process that eliminates the need for contiguous allocation of physical memory. (made by memory management unit)
Each process has a Page Table that maps logical pages to physical frames. Page table stores base address of each page in physical memory. When a process requests particular address, CPU uses page table to convert virtual address to physical address.

Benefits - No external fragmentation, efficient memory utilization, simplified allocation.

Disadvantages - Internal fragmentation, page table overhead, address translation overhead.

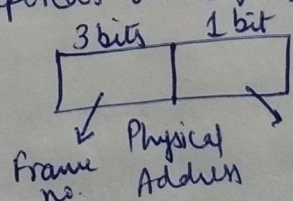
NOW, correct page no. & byte can be represented as



Eg - 3 (11)

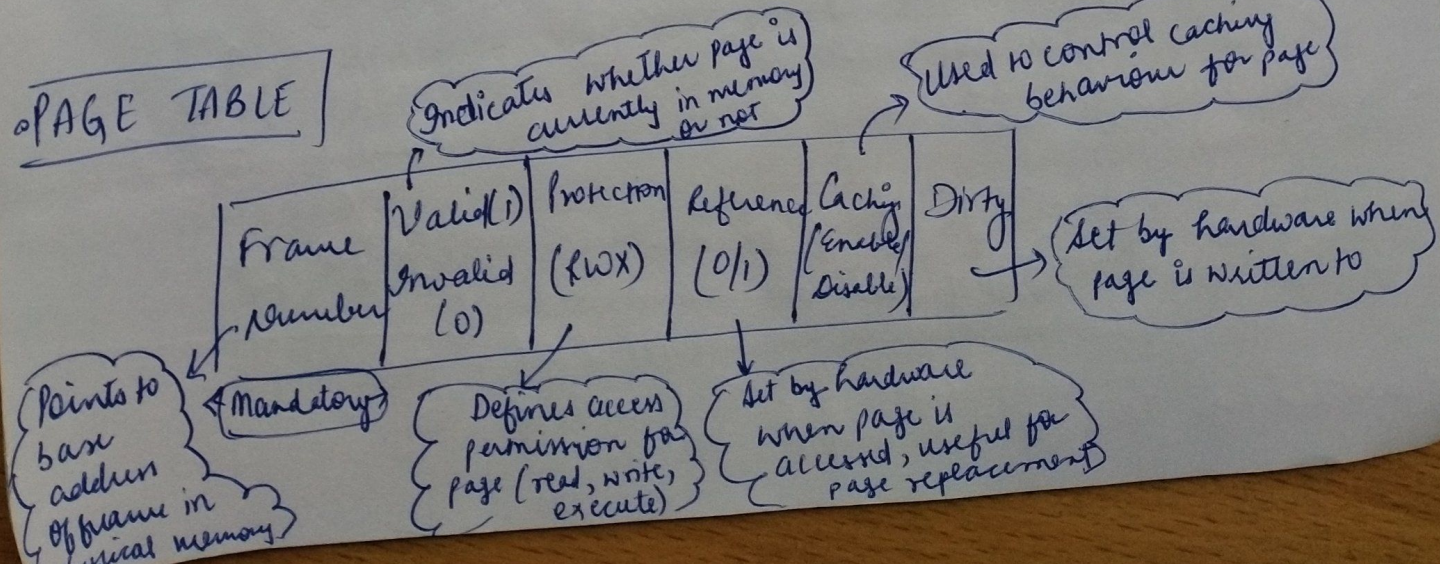
Size of physical address depends on size of main memory :-

Eg - M/M size = 16 (4 bits)



3 bits coz no of frames in eg is 8

PAGE TABLE



Level Paging

Multi level paging also known as hierarchical paging is a paging scheme used to manage large address spaces in a more efficient way by breaking down page table into multiple levels.

- Breakdown of Page Table

→ Instead of having a single, large page table, page table is divided into multiple levels.

- Address Translation Process

→ Virtual address is divided into multiple parts, each part corresponding to different level of page table

Advantages - • Smaller page tables reduce memory overhead.
• More suitable for managing large virtual address space.

Disadvantages - Complexity rises & performance overhead.

Logical address → Outer Page Table → Page of page table → Main memory

Inverted Paging

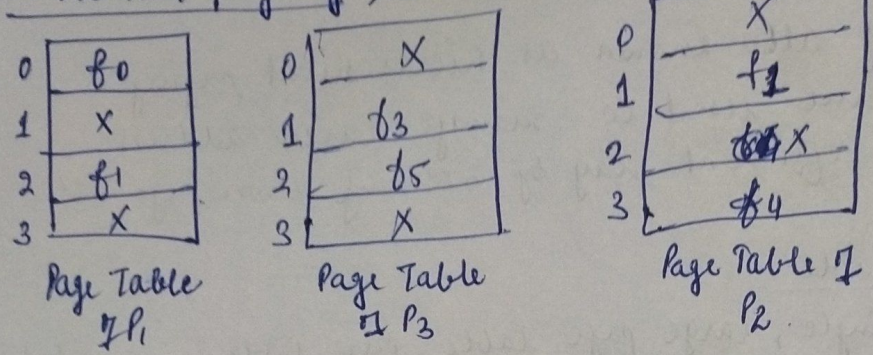
A type of paging mechanism used to manage memory in a more efficient manner specially in a system with large address spaces. Normal paging system has every process with its own page table but this uses a single page table for entire system.

→ When a process references a virtual address, system needs to translate this to a physical address for which system searches inverted page table for an entry. Often a hashmap is used to speed up the process of searching.

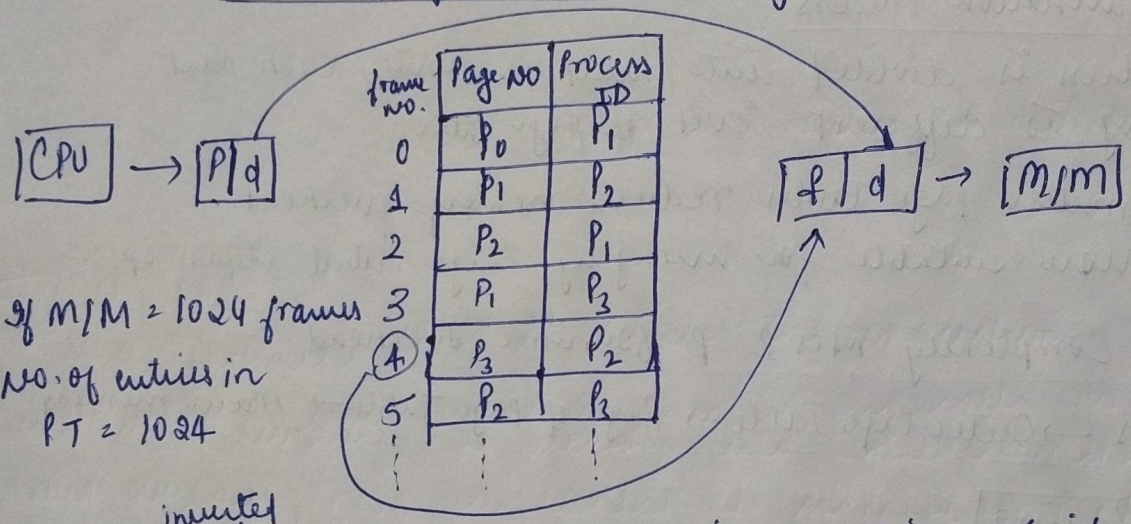
- Advantages - • Reduced memory overhead since only one page table reduces memory needed to store page tables.
• Simplified management.
- Disadvantages - Complex Address Translation, Hash collision.

(Key concept) - • Single Page Table for entire system.
• Each entry in inverted page table corresponds to a page frame in physical memory, not a page in virtual memory.
• Mapping of pages with frame

In normal paging, we have:-



But in inverted paging, we have a single global page table:-



If M/M = 1024 frames
No. of entries in PT = 1024

Here, in inverted paging, the only problem is searching which takes a lot of time since it goes for linear search.

Ques - Consider a virtual address space of 32 bits & page size of 4KB. System is having a RAM of 128 KB. Then what will ratio of page table and inverted page table size if each entry is of size 4B in both?

⇒ ~~Page size = frame size = 4KB~~

MM = 128 KB

Virtual Address space = | 20 (No. of pages) | 12 (Page offset/size) |

← 32 →

If virtual address space = 32 bits,
Virtual address
$= 2^{32}$
$= 2^2 \, 2^{30}$
→ 4 GB

- Page Table size = $2^{20} \times 4B$

4 KB
$2^2 \, 2^{10} = 2^{12}$

| 5 | 12 | → (4KB)

Physical address

2^5 = No. of frames ← 17 →

- Inverted Page Table size = $2^5 \times 4B$.

Ratio = $\frac{2^{20} \times 4}{2^5 \times 4}$

$= 2^{15} : 1$

128 KB
$2^7 \, 2^{10}$
$= 2^{17}$

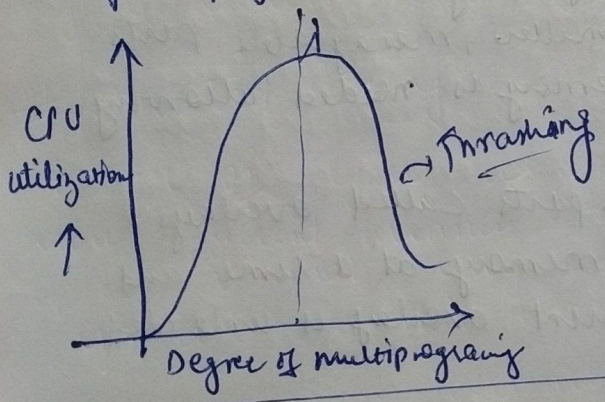
THRASHING

It refers to a scenario where the operating system is spending most of its time swapping pages in and out of memory rather than executing the actual processes. This usually happens when system is overloaded with too many processes and not enough physical memory to handle them efficiently.

Thrashing can occur due to -
- Insufficient RAM
- High degree of multiprogramming
- Inappropriate page replacement algorithm.

Thrashing can lead to -
- Decreased system performance
- Increased Response Time
- High Disk Activity.

Thrashing Prevention -
- Increase Physical memory
- Reduce multiprogramming level
- Adjust page replacement algorithm
- Working set model.



Segmentation Vs Paging

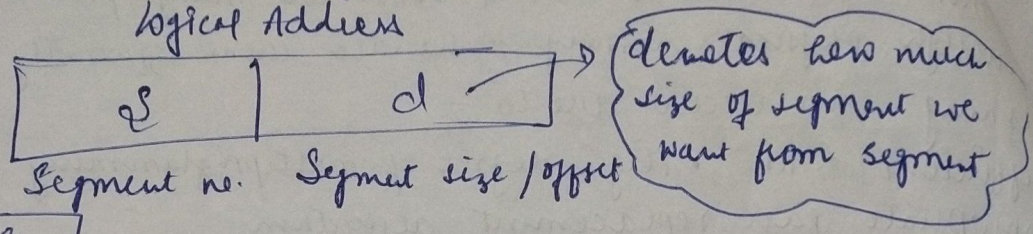
Segmentation is like organising books by topics where each topic/section can be of different sizes depending on how many books it needs to store.

Paging is dividing each shelf into fixed sized boxes called pages. Regardless of how big or small the books are, they must fit into these fixed size boxes.

Technically,

Segmentation is a memory management technique where the memory is divided into variable size segments. Each

Segment corresponds to a logical unit such as a function, array / data structure. OS maintains a segment table for each process which stores base address & length of each segment.



∴ if $d \leq size$

If d > size, it creates a trap or generates an interrupt.

OVERLAY

It is a Technique used to enable the execution of programs that require more memory than is physically available. It involves dividing the memory into smaller, manageable parts called overlay. These are loaded into memory as needed allowing program to run in limited memory.

→ A large prg is divided into small parts called overlays.
→ Only required overlay is loaded into memory at a time and when another part is needed, the current overlay is unloaded and new one is loaded in its place
→ The overlay manager, a part of OS handles loading & unloading of overlays.

Advantages - Efficient memory usage, simplifies programming
Disadvantages - Overhead, complex management.

- Overlays are mostly used in embedded systems.
- Overlays are typically designed to be independent of each other to a certain extent.

Ques - Consider a two pass assembler pass 1 - 80 KB and pass 2 - 90 KB. Symbol Table - 30 KB, Common routine - 20 KB. At a time only one pass is in use, what is min partition size required if overlay driver is 10 KB size?

⇒ Total = 230 kB.

If we use pass 1

80 kB
30 kB
20 kB
10 kB
140 kB

If we use pass 2

90 kB
30 kB
20 kB
10 kB
150 kB

∴ Min size of partition needed is 150 kB.

VIRTUAL MEMORY

A memory management technique that provides an 'idealized abstraction' of storage resources that are actually available on given machine. It creates the illusion of very large memory by both hardware & software mechanisms, allowing a system to execute processes that may not completely fit into RAM.

Virtual memory divides address space into virtual addresses. This is only implemented using Paging

Benefits - Increased memory capacity, Isolation and Security, efficient memory management.

- Effective Memory Access Time

Let 'p' is the probability of Page fault occurance.

∴ EMAT = p(Page fault service Time) + (1-p)(main memory access time)

(milliseconds) (nanoseconds)

TRANSLATION LOOKASIDE BUFFER

It is a crucial component in memory management system that is a specialized cache used to speed up the translation of virtual addresses to physical addresses.

It is a small fast cache that stores a subset of Table entries. It holds most recently or frequently accessed mappings of virtual pages to physical frames.

When a virtual address needs to be translated, TLB is checked first. If the mapping is found in TLB, it's TLB hit & address is obtained quickly. If mapping is not found, it's TLB miss. In this case, OS looks up Page Table to find mapping. Once mapping is found, it's added to TLB for future reference.

EMAT = (TLB + x) + miss (TLB + x + x)
↳ memory access time

Ques -

PAGE REPLACEMENT

It is a process used by OS to manage contents of RAM when it gets full. When page fault occurs and there is no free frame available in physical memory, the OS must decide which existing page to evict to make room for new page. This decision is made using a page replacement algorithm

PAGE REPLACEMENT ALGORITHMS

(1) FIFO (First In, First Out)

- Evicts page that has been in memory the longest.
- Simple to implement but can lead to poor performance due to "Belady's Anomaly" where increasing no. of frames can increase no. of page faults.

		1	1	1	1	0	0	0	3	3	3	3	2	2				
	0	0	0	0	3	3	3	2	2	2	2	1	1	1				
7	7	7	2	2	2	2	4	4	4	0	0	0	0	0				
*	*	*	*	Hit	*	*	*	*	*	*	Hit	*	*					

Q → 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 1, 2, 0

* Denotes page fault occured

Page Hit = 3

Page fault = 12.

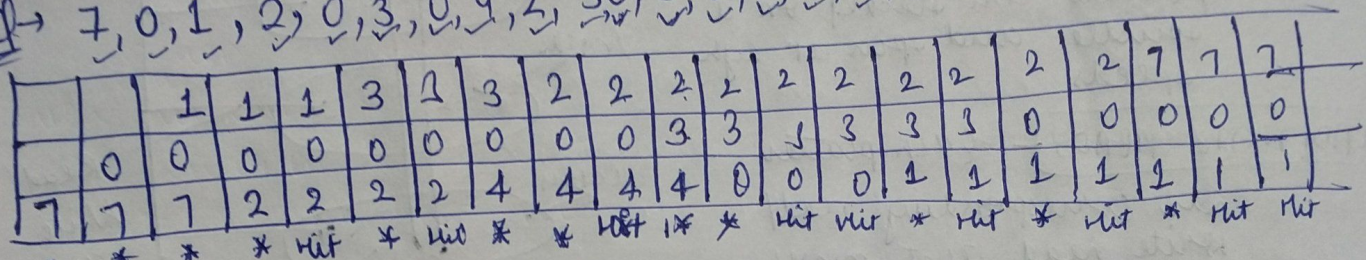
Hit ratio = No. of hits / No. of ref = 3/15 = 1/5

- Belady's anomaly is commonly observed with FIFO page replacement algorithm where increasing no. of frames results in more page faults contrary to expectation. It highlights the importance of choosing an appropriate page replacement algorithm.

(2) LRU (Least Recently Used)

- Evicts page that has not been used for longest time.
- Requires additional hardware support to track usage history of pages.

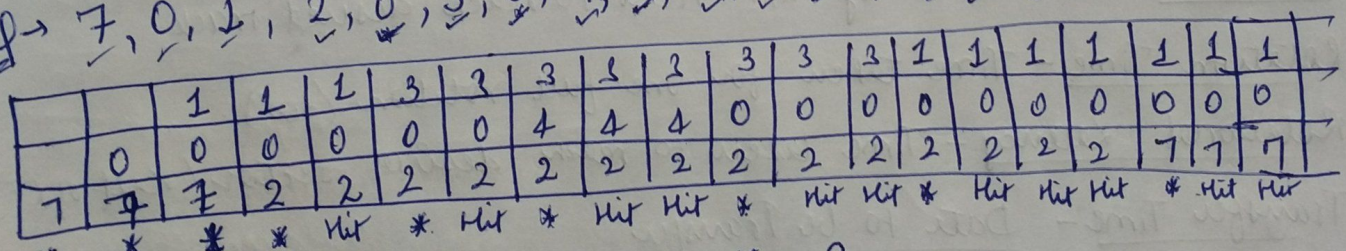
Q→ 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



(3) Optimal Page Replacement

- Evicts page that will not be used for the longest time in future.
- Provides best performance but is impractical to implement because it requires knowledge of future memory accesses.
- Here, while solving, when page fault occurs, check for the pages in the memory if they are going to chosen recently or not. The one which will arrive late should be replaced.

Q→ 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



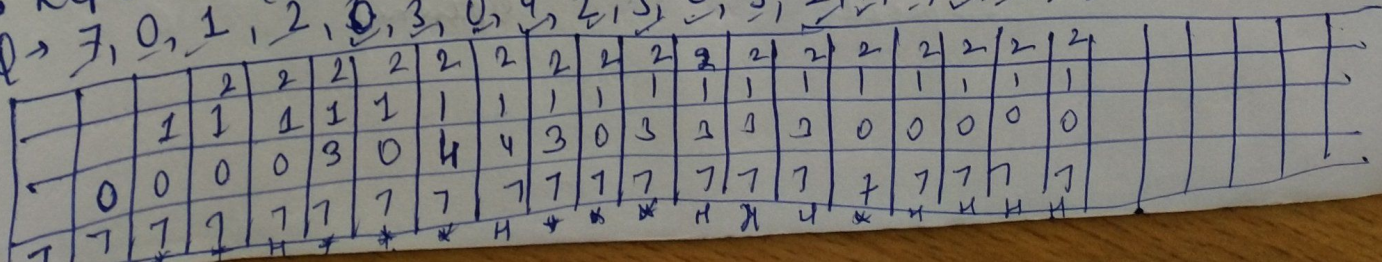
No. of hits = 11 No of page faults = 9

Hit ratio = $\frac{11}{20}$

(4) Most Recently Used (MRU)

- Replace the recently used page.

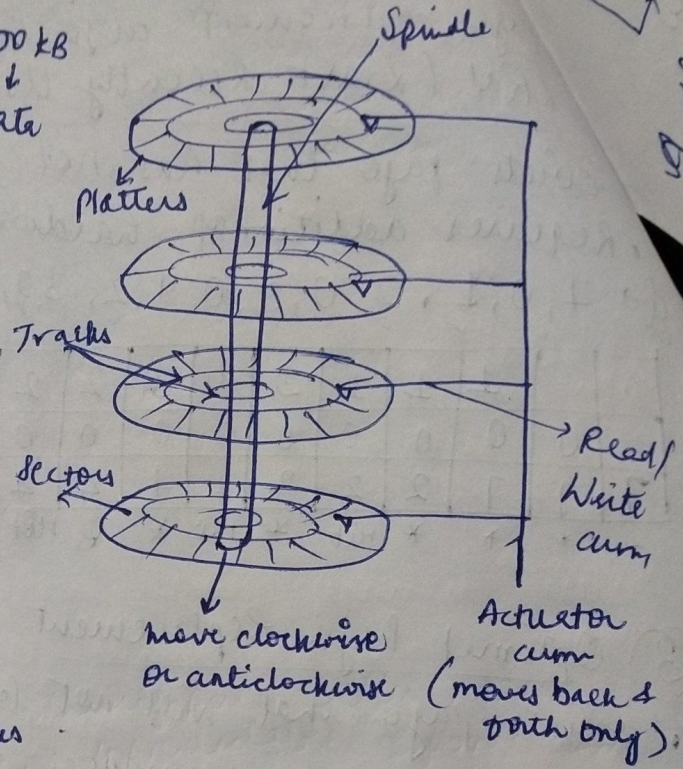
Q→ 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



Hard Disk Architecture

Platters → (along with spindle) → (on platter we have read/write head) → Tracks → Sectors → Data

Eg - Total Disk size = 8 × 2 × 256 × 512 × 500 kB
(8 Platters, 2 Surfaces, 256 Tracks, 512 Sectors, 500 kB data)



- **PLATTERS** - Circular disks coated with magnetic material. These platters are stacked on a spindle and spin at high speeds.
- **READ/WRITE HEADS** - Each platter has corresponding read/write head that moves across its surface to access data.
- **TRACKS** - Each platter is divided into concentric circles called Tracks.
- **SECTORS** - Tracks are further divided into smaller units called sectors which are smallest units of storage on disk.

DISK ACCESS TIME

1. **Seek Time** - Time taken by R/W head to reach desired track.
2. **Rotation Time** - Time taken for one full rotation (360°).
3. **Rotational Latency** - Time taken to reach desired sector (Half of RT)
4. **Transfer Time** - $\frac{\text{Data to be Transfer}}{\text{Transfer rate}}$
5. **Transfer Rate** = (No. of Heads (Surface) × Capacity of One Track × No. of Rotations in one second)
6. **Disk Access Time** = ST + RT + TT + (Controller Time + Queue Time) — Optional

DISK SCHEDULING ALGORITHMS

Goal is to minimize seek time.

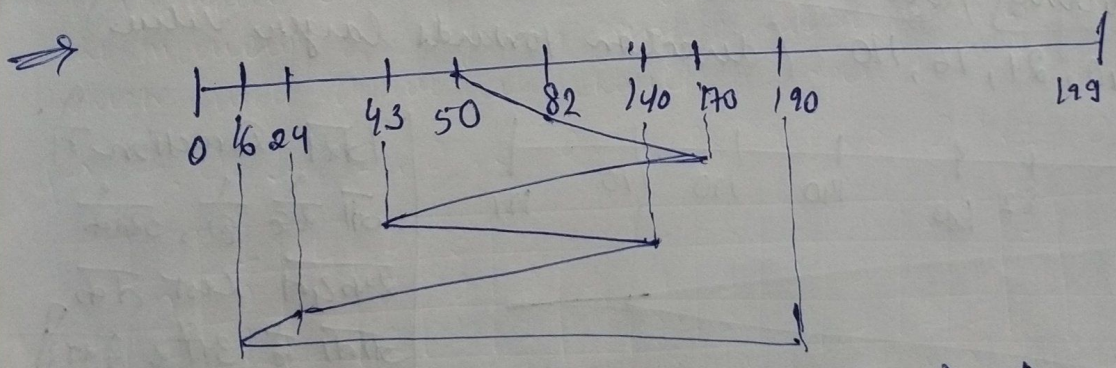
- FCFS (First come First serve)
- SSTF (Shortest Seek Time First)
- SCAN
- LOOK
- CSCAN (Circular Scan)
- CLOOK (Circular Look)

A disk scheduling algorithm decides the sequence of I/O requests that the disk controller will handle. Since disk access speed is significantly slower compared to CPU & RAM, efficient disk scheduling can greatly improve overall system performance.

(1) First Come First Serve (FCFS)

It processes I/O requests in order of their arrival time but can lead to long wait times and insufficient seek movements.

Q→ A disk contains 200 tracks (0-199). Request queue contains track number 82, 170, 43, 140, 24, 16, 190 respectively.
Current position of R/W head = 50
Calculate total no. of track movement by R/W head.



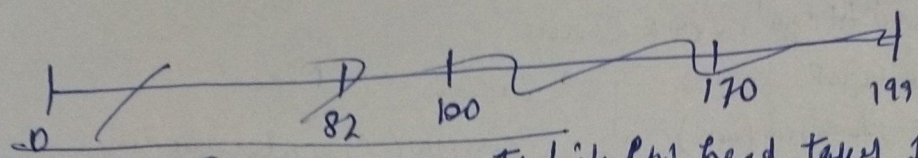
Total track movements = (170-50) + (170-43) + (140-43) + (140-16) + (190-16)

= 120 + 127 + 97 + 124 + 174 = 642

Advantages - Simplicity, no starvation, fairness, low overhead.

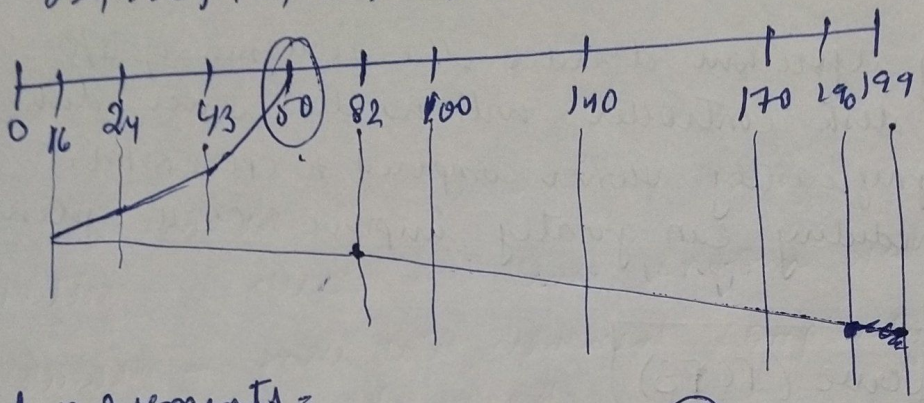
Disadvantages - Insufficient for high seek times, long wait times, poor performance, no prioritization.

② Shortest Seek Time First (SSTF)



Calculate total no. of movements if R/W head takes 1 ms to move from one head to another, what's total time

Q → 82, 170, 43, 140, 24, 16, 190. Starting at 50



Total movements =

(50-16) + (190-16) = 34 + 174 = 208

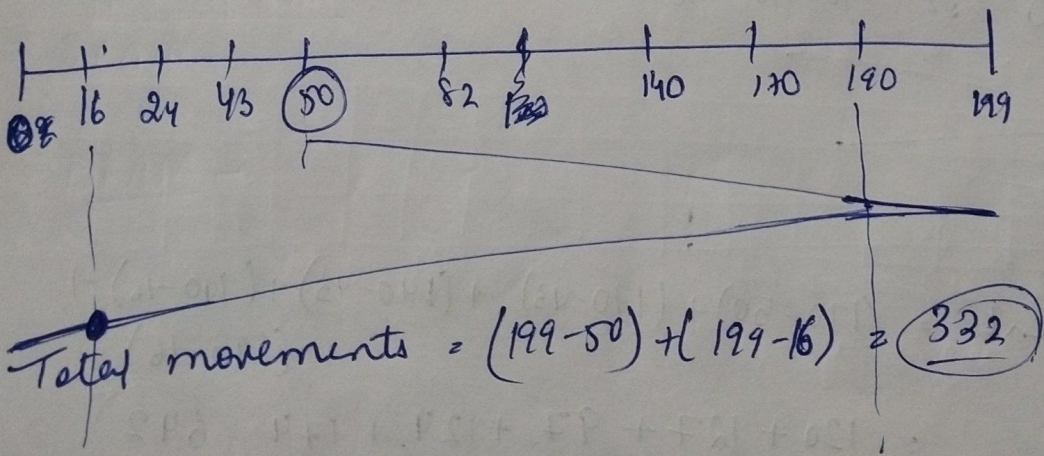
Total time = 208 × 1 ms = 208 ms

Advantages – min seek time, improved throughput, better resource use.

Disadvantages – starvation, complexity, unpredictable response time.

③ SCAN algorithm (Elevator algo)

82, 170, 43, 140, 24, 16, 190. (direction towards larger value)



Total movements = (199-50) + (199-16) = 332

जिस Direction में जा रहे हो, उसके आखिरी Last तक जाना है और दूसरी Direction के given disk no. तक

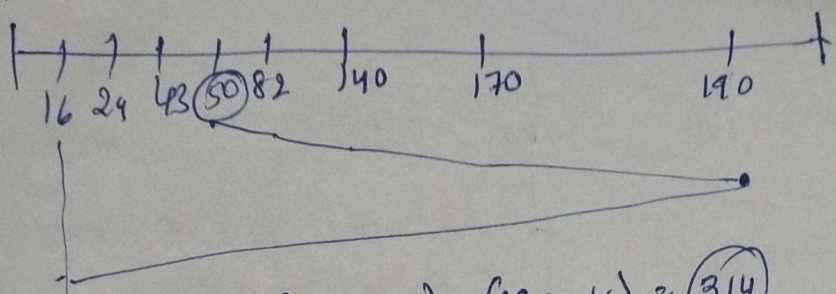
Advantages – Reduced seek time, fairness, predictable performance, balanced workload.

Disadvantages – Long wait time for edge requests, unnecessary movements, complexity.

④ Look

Q→ 82, 170, 43, 140, 24, 16, 190. (Direction is larger)

=



Does not go till last as covered by SCAN

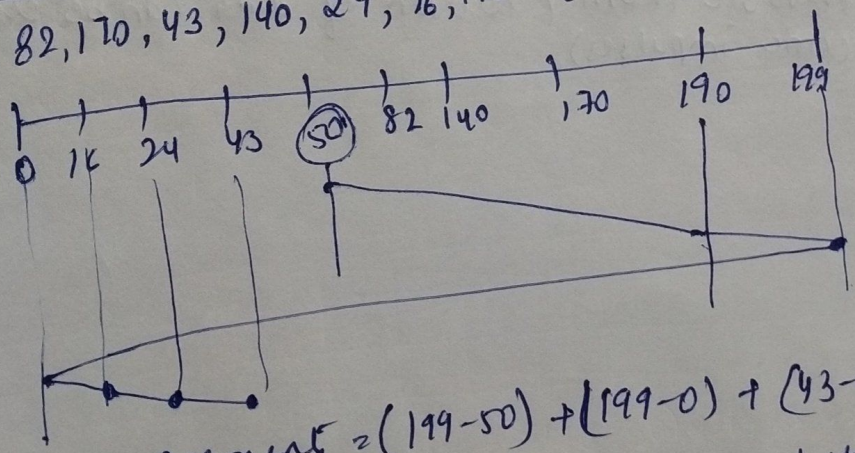
Track movements = (190 - 50) + (190 - 16) = 314

Advantages - Reduced seek Time, fairness, predictable performance

Disadvantages - Long wait for edge cases, complexity

⑤ CSCAN -

Q→ 82, 170, 43, 140, 24, 16, 190. (Towards large)



Always move & calculate in one direction only.

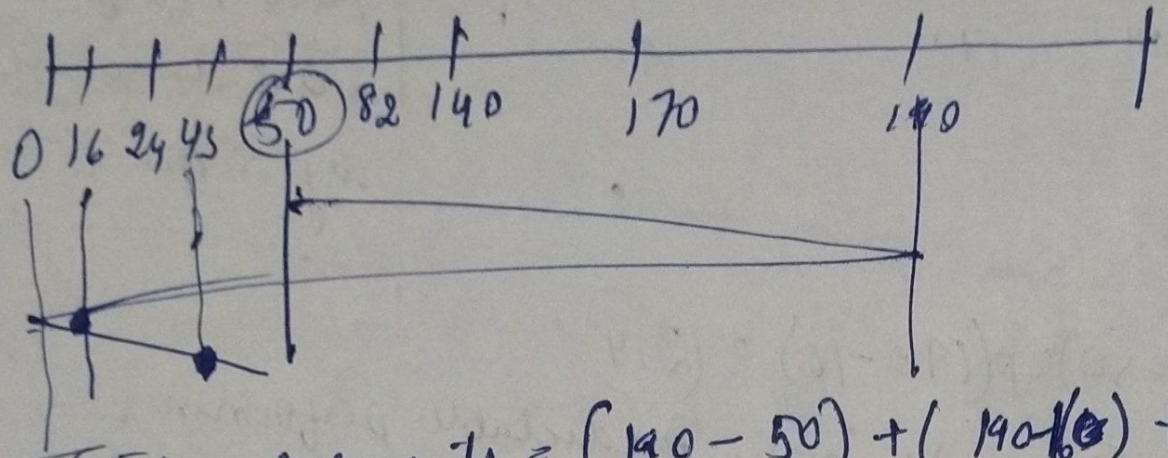
Total movement = (199 - 50) + (199 - 0) + (43 - 0) = 391

Advantages - Uniform Wait Time, reduced starvation, fairness, efficient use of disk movement

Disadvantages - Longer total movements, complexity, increased seek time.

(6) CLOOK -

Q → 82, 170, 43, 140, 24, 16, 190 (Towards large)



Total movements = (190 - 50) + (190 - 16) + (43 - 16) = 341

Advantage - Reduced seek time, uniform wait, efficient use of disk.

Disadvantage - Longer total movement, Complexity, increased seek time for edge requests.