UNIT 1
AN INTRODUCTION TO OPERATING SYSTEMS

Application software performs specific task for the user.
System software operates and controls the computer system and provides a platform to run application software.

An **operating system** is a piece of software that manages all the resources of a computer system, both hardware and software, and provides an environment in which the user can execute his/her programs in a convenient and efficient manner by hiding underlying complexity of the hardware and acting as a resource manager.

Why OS?

1. What if there is no OS?
    a. Bulky and complex app. (Hardware interaction code must be in app's code base)
    b. Resource exploitation by 1 App.
    c. No memory protection.
2. What is an OS made up of?
    a. Collection of system software.

An operating system function -

- Access to the computer hardware.
- interface between the user and the computer hardware
- **Resource management (Aka, Arbitration) (memory, device, file, security, process etc)**
- **Hides the underlying complexity of the hardware. (Aka, Abstraction)**
- facilitates execution of application programs by providing isolation and protection.

User

Application programs
Operating system
Computer hardware

The operating system provides the means for proper use of the resources in the operation of the computer system.

LEC-2: Types of OS

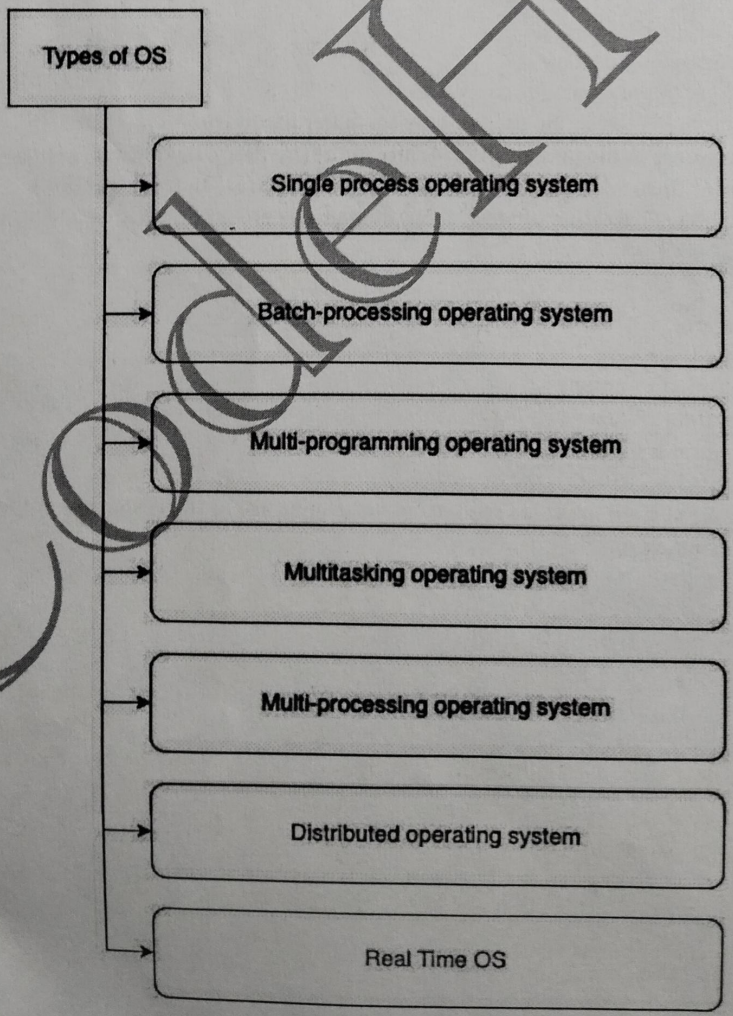
OS goals –

- Maximum CPU utilization → we don't want cpu to be idle.
- Less process starvation → In case of long time taking processes, other processes will starve.
- Higher priority job execution

Ex. → Antivirus protection process execute first having higher priority than other process.

Types of operating systems –

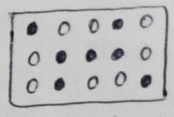
- Single process operating system [MS DOS, 1981]
- Batch-processing operating system [ATLAS, Manchester Univ., late 1950s – early 1960s]
- Multiprogramming operating system [THE, Dijkstra, early 1960s]
- Multitasking operating system [CTSS, MIT, early 1960s]
- Multi-processing operating system [Windows NT]
- Distributed system [LOCUS]
- Real time OS [ATCS]



Single process OS, only 1 process executes at a time from the ready queue. [Oldest]

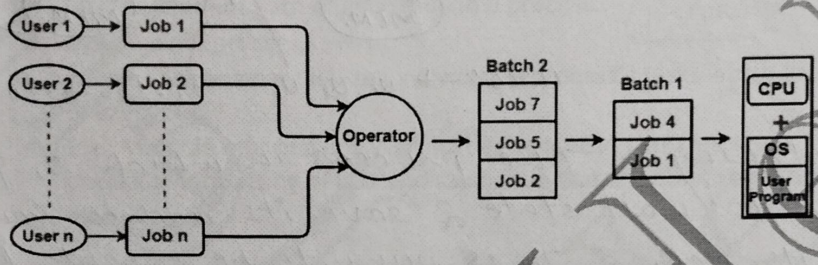
Batch-processing OS,

1. Firstly, user prepares his job using punch cards.
2. Then, he submits the job to the computer operator.
3. Operator collects the jobs from different users and sort the jobs into batches with similar needs.
4. Then, operator submits the batches to the processor one by one.
5. All the jobs of one batch are executed together.



punch card

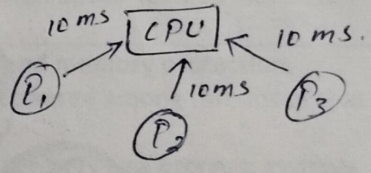
- Priorities cannot be set, if a job comes with some higher priority.
- May lead to starvation. (A batch may take more time to complete)
- CPU may become idle in case of I/O operations.



Multiprogramming increases CPU utilization by keeping multiple jobs (code and data) in the**memory** so that the CPU always has one to execute in case some job gets busy with I/O.

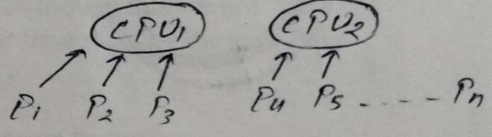
- Single CPU
- Context switching for processes.
- Switch happens when current process goes to wait state.
- CPU idle time reduced.

Multitasking is a logical extension of multiprogramming.

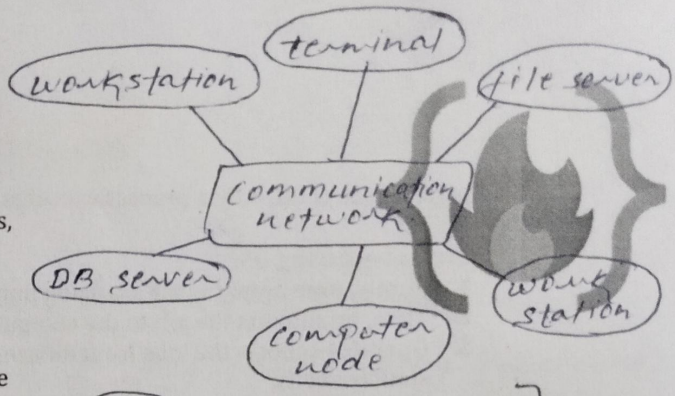


- Single CPU
- Able to run more than one task simultaneously.
- Context switching and time sharing used. (time quantum) → 10 ms.
- Increases responsiveness.
- CPU idle time is further reduced.

Multi-processing OS, more than 1 CPU in a single computer.



- Increases reliability, 1 CPU fails, other can work
- Better throughput.
- Lesser process starvation, (if 1 CPU is working on some process, other can be executed on other CPU.



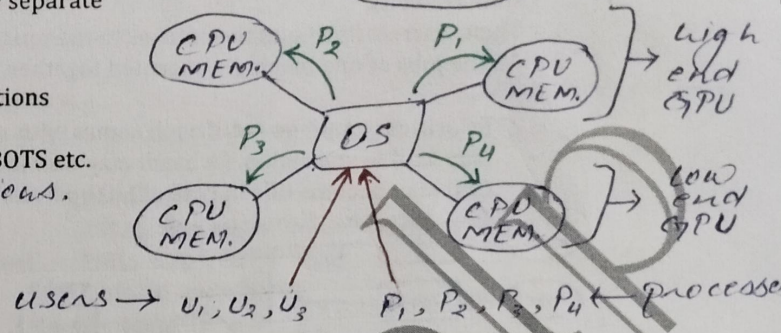
Distributed OS,

- OS manages many bunches of resources, >=1 CPUs, >=1 memory, >=1 GPUs, etc
- **Loosely connected autonomous,** interconnected computer nodes.
- collection of independent, networked, communicating, and physically separate computational nodes.

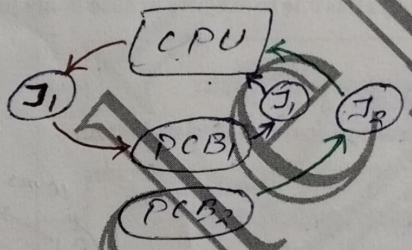
RTOS

- **Real time** error free, computations within tight-time boundaries.
- Air Traffic control system, ROBOTS etc.

→ industrial applications.



context switching → The process in which P1 process goes to wait state & save its address (context) in its PCB1 & J2 is going to be execute by cpu by taking context of J2 from its PCB2.



LEC-3: Multi-Tasking vs Multi-Threading

Program: A Program is an executable file which contains a certain set of instructions written to complete the specific job or operation on your computer.

- It's a compiled code. Ready to be executed.
- Stored in Disk

→ In c++, .exe file is compiled & executable file.

Process: Program under execution. Resides in Computer's primary memory (RAM).

→ when we run or execute .exe file, process happens and we find an output.

Thread:

- Single sequence stream within a process.
- An independent path of execution in a process.
- Light-weight process.
- Used to achieve parallelism by dividing a process's tasks which are independent path of execution.
- E.g., Multiple tabs in a browser, text editor (When you are typing in an editor, spell-checking, formatting of text and saving the text are done concurrently by multiple threads.)

process → main
→ sub-tasks in cloud

Multi-Tasking	Multi-Threading
The execution of more than one task simultaneously is called as multitasking.	A process is divided into several different sub-tasks called as threads, which has its own path of execution. This concept is called as multithreading.
Concept of more than 1 processes being context switched.	Concept of more than 1 thread. Threads are context switched.
No. of CPU 1.	No. of CPU >= 1. (Better to have more than 1)
Isolation and memory protection exists. OS must allocate separate memory and resources to each program that CPU is executing.	**No isolation and memory protection,** resources are shared among threads of that process. OS allocates memory to a process; multiple threads of that process share the same memory and resources allocated to the process.

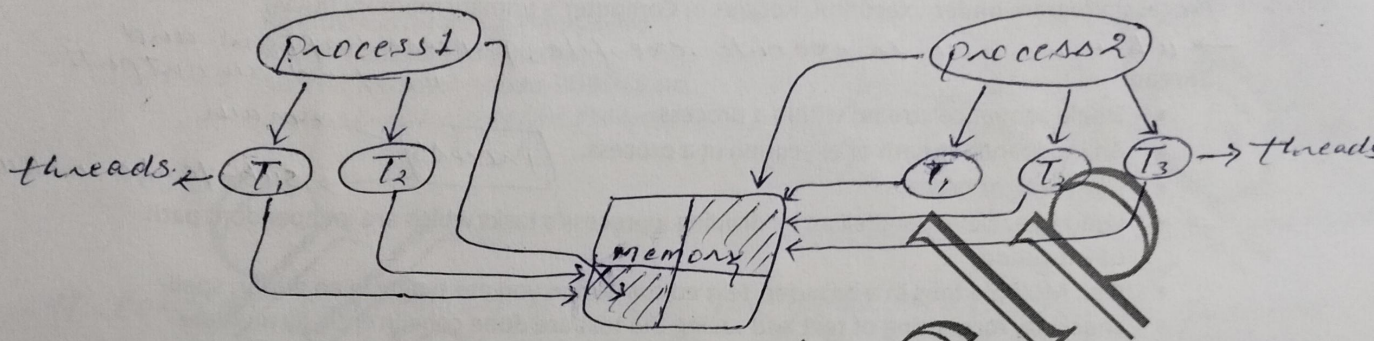
Thread Scheduling:

Threads are scheduled for execution based on their priority. Even though threads are executing within the runtime, all threads are assigned processor time slices by the operating system.

Difference between Thread Context Switching and Process Context Switching:

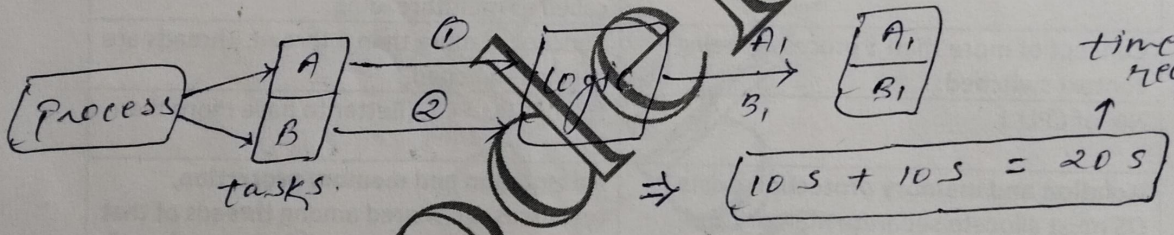
Thread Context switching	Process context switching
OS saves current state of thread & switches to another thread of same process.	OS saves current state of process & switches to another process by restoring its state.

Doesn't includes switching of memory address space. (But Program counter, registers & stack are included.)	Includes switching of memory address space.
Fast switching.	Slow switching.
CPU's cache state is preserved.	CPU's cache state is flushed.

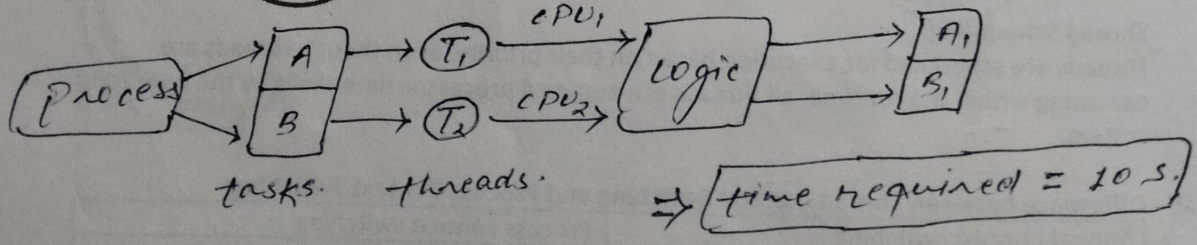


Multithreading

(1) sequential Execution → in multitasking - concatenate.



(2) parallel execution → happens in multiprocessing environment ⇒ ≥ 1 CPU.
→ in multithreading - concatenate.

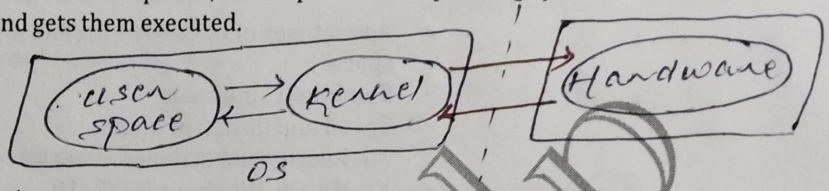


LEC-4: Components of OS

1. **Kernel:** A **kernel** is that part of the operating system which interacts directly with the hardware andperforms the most crucial tasks.
   a. Heart of OS/Core component
   b. Very first part of OS to load on start-up.
2. **User space:** Where application software runs, apps don't have privileged access to the underlying hardware. It interacts with kernel.
   a. GUI → Graphical user interface → terminal
   b. CLI → command line interface → powershell

A **shell**, also known as a command interpreter, is that part of the operating system that receives commands from the users and gets them executed.

Functions of Kernel:



1. Process management:
   a. Scheduling processes and threads on the CPUs.
   b. Creating & deleting both user and system process.
   c. Suspending and resuming processes
   d. Providing mechanisms for process synchronization or process communication.
2. Memory management:
   a. Allocating and deallocating memory space as per need.
   b. Keeping track of which part of memory are currently being used and by which process.
3. File management:
   a. Creating and deleting files.
   b. Creating and deleting directories to organize files.
   c. Mapping files into secondary storage.
   d. Backup support onto a stable storage media.
4. I/O management: to manage and control I/O operations and I/O devices
   a. Buffering (data copy between two devices), caching and spooling.
      i. Spooling → It allows multiple process to queue their data without waiting for the device to become available
         1. Within differing speed two jobs.
         2. Eg. Print spooling and mail spooling.
      ii. Buffering → process of storing data in a temp. memory area called buffer while transferring it b/w two devices or processes
         1. Within one job.
         2. Eg. Youtube video buffering
      iii. Caching → used to store frequently accessed data in a high-speed storage area called cache to reduce access time & improve perfor
         1. Memory caching, Web caching etc.

Types of Kernels:

1. Monolithic kernel
   a. All functions are in kernel itself.
   b. Bulky in size.
   c. Memory required to run is high.
   d. Less reliable, one module crashes -> whole kernel is down.
   e. High performance as communication is fast. (Less user mode, kernel mode overheads)
   f. Eg. Linux, Unix, MS-DOS.

2. Micro Kernel
    a. Only major functions are in kernel.
        i. Memory mgmt.
        ii. Process mgmt.
    b. File mgmt. and IO mgmt. are in User-space.
    c. smaller in size.
    d. More Reliable
    e. More stable
    f. Performance is slow.
    g. Overhead switching b/w user mode and kernel mode.
    h. Eg. L4 Linux, Symbian OS, MINIX etc.

→ cpu works with user-space of OS.

↳ cpu work with kernel & interact with hardware

3. Hybrid Kernel:
    a. Advantages of both worlds. (File mgmt. in User space and rest in Kernel space.)
    b. Combined approach.
    c. Speed and design of mono.
    d. Modularity and stability of micro.
    e. Eg. MacOS, Windows NT/7/10
    f. IPC also happens but lesser overheads
4. Nano/Exo kernels...

Q. How will communication happen between user mode and kernel mode?
Ans. Inter process communication (**IPC**).

1. Two processes executing independently, having independent memory space (Memory protection), But some may need to communicate to work.
2. Done by shared memory and message passing.

* Software Interrupt → Interrupt generated by some software means used for transition b/w user space and kernel space.

LEC-5: System Calls

How do apps interact with Kernel? -> using system calls.

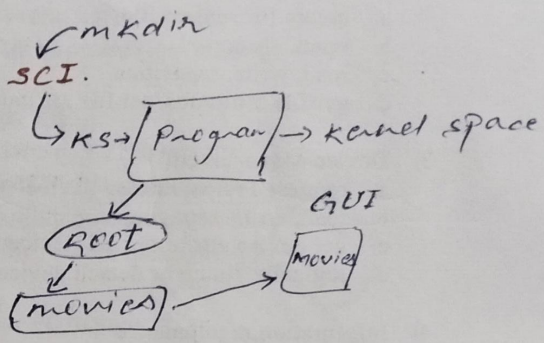
Eg. Mkdir laks

- Mkdir indirectly calls kernel and asked the file mgmt. module to create a new directory. → movies
- Mkdir is just a wrapper of actual system calls.
- Mkdir interacts with kernel using system calls.

GUI → new folder button
CLI → mkdir

Eg. Creating a process.

- User executes a process. (User space)
- Gets system call. (US)
- Exec system call to create a process. (KS)
- Return to US.

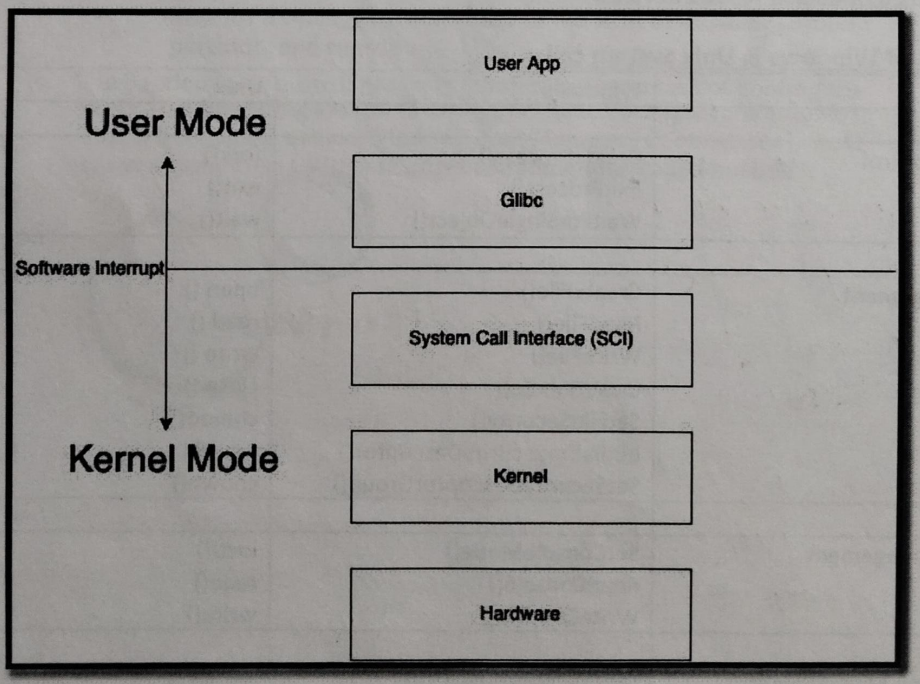


Transitions from US to KS done by software interrupts.

★ **System calls** are implemented in C.

A system call is a mechanism using which a user program can request a service from the kernel for which it does not have the permission to perform.
User programs typically do not have permission to perform operations like accessing I/O devices and communicating other programs.

System Calls are the only way through which a process can go into **kernel mode from user mode.**



★ How does switch from UM to KM ?
(1) Execution of process
(2) After execution we call software interrupt.

Types of System Calls:

1) Process Control
   a. end, abort
   b. load, execute
   c. create process, terminate process
   d. get process attributes, set process attributes
   e. wait for time
   f. wait event, signal event
   g. allocate and free memory

2) File Management
   a. create file, delete file
   b. open, close
   c. read, write, reposition
   d. get file attributes, set file attributes

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

4) Information maintenance
   a. get time or date, set time or date
   b. get system data, set system data
   c. get process, file, or device attributes
   d. set process, file, or device attributes

5) Communication Management
   a. create, delete communication connection
   b. send, receive messages
   c. transfer status information
   d. attach or detach remote devices

Examples of Windows & Unix System calls:

Category	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Management	CreateFile() ReadFile() WriteFile() CloseHandle() SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	open () read () write () close () chmod() umask(chown()
Device Management	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Management	GetCurrentProcessID() SetTimer() Sleep()	getpid () alarm () sleep ()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe () shmget () mmap()

LEC-6: What happens when you turn on your computer?

ON → power supply → motherboard
→ Harddisk
→ storage etc.

i. PC On
ii. CPU initializes itself and looks for a firmware program (BIOS) stored in BIOS Chip (Basic input-output system chip is a ROM chip found on mother board that allows to access & setup computer system at most basic level.)
    1. In modern PCs, CPU loads UEFI (Unified extensible firmware interface)
iii. **CPU** runs the BIOS which tests and initializes system hardware. Bios loads configuration settings. If something is not appropriate (like missing RAM) error is thrown and boot process is stopped.
This is called **POST** (Power on self-test) process.
(UEFI can do a lot more than just initialize hardware; it's really a tiny operating system. For example, Intel CPUs have the Intel Management Engine. This provides a variety of features, including powering Intel's Active Management Technology, which allows for remote management of business PCs.)
iv. **BIOS** will handoff responsibility for booting your PC to your OS's bootloader.
    1. BIOS looked at the MBR (master boot record), a special boot sector at the beginning of a disk. The MBR contains code that loads the rest of the operating system, known as a "bootloader." The BIOS executes the bootloader, which takes it from there and begins booting the actual operating system—Windows or Linux, for example.
    In other words,
    the BIOS or UEFI examines a storage device on your system to look for a small program, either in the MBR or on an EFI system partition, and runs it.
v. The bootloader is a small program that has the large task of booting the rest of the operating system (Boots Kernel then, User Space). Windows uses a bootloader named Windows Boot Manager (Bootmgr.exe), most Linux systems use GRUB, and Macs use something called boot.efi

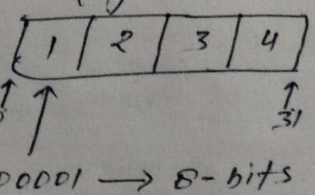
Lec-7: 32-Bit vs 64-Bit OS

1. A 32-bit OS has 32-bit registers, and it can access 2^32 unique memory addresses. i.e., 4GB of physical memory.
2. A 64-bit OS has 64-bit registers, and it can access 2^64 unique memory addresses. i.e., 17,179,869,184 GB of physical memory.
3. 32-bit CPU architecture can process 32 bits of data & information.
4. 64-bit CPU architecture can process 64 bits of data & information.
5. Advantages of 64-bit over the 32-bit operating system:
   a. **Addressable Memory**: 32-bit CPU -> 2^32 memory addresses, 64-bit CPU -> 2^64 memory addresses.
   b. **Resource usage**: Installing more RAM on a system with a 32-bit OS doesn't impact performance. However, upgrade that system with excess RAM to the 64-bit version of Windows, and you'll notice a difference.
   c. **Performance**: All calculations take place in the registers. When you're performing math in your code, operands are loaded from memory into registers. So, having larger registers allow you to perform larger calculations at the same time.
   32-bit processor can execute 4 bytes of data in 1 instruction cycle while 64-bit means that processor can execute 8 bytes of data in 1 instruction cycle.
   (In 1 sec, there could be thousands to billons of instruction cycles depending upon a processor design)
   d. **Compatibility**: 64-bit CPU can run both 32-bit and 64-bit OS. While 32-bit CPU can only run 32-bit OS.
   e. **Better Graphics performance**: 8-bytes graphics calculations make graphics-intensive apps run faster.

समझो → ① we have cpu-chip which is embeded in motherboard.
② cpu-chip contains millions of transistors.
③ transistors together forms circuits (gates)
④ Now these different circuits together forms Registers.

Registers → circuit system or location inside the cpu-chip, where actual computations occurs & it also holds data & stores it.

Register → 4-bytes => 32-bits



00000001 → 8-bits

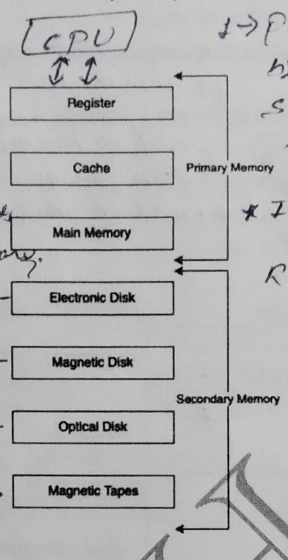
3→ tertiary memory → used for backup and archival storage.
① magnetic taps ② cloud storage
cloud storage → remote data storage on internet servers.
4→ virtual memory → A section of the hard drive that acts as temporary RAM when actual RAM is full.

Lec-8: Storage Devices Basics

What are the different memory present in the computer system?

2→ secondary → used for long term data storage & it is non-volatile.

SSD → solid-state Drive
Flash Drives → USB, SD card
→ It uses NAND flash memory.
HDD → Hard-disk Drive
FLOPPY DISK → slower & cheaper than SSDs
CD, DVD, Blu-ray Disc
→ uses laser beams to read & write data.
Data back-up tapes.



1→ primary → directly accessible by the CPU & used for storing data & instructions that are currently being processed.
* It is volatile.

RAM → temporary storage area where CPU stores data and instructions needed for running programs.
① static RAM → faster & expensive
② dynamic RAM → slower & cheaper.

1. **Register**: Smallest unit of storage. It is a part of CPU itself.
   A register may hold an instruction, a storage address, or any data (such as bit sequence or individual characters).
   Registers are a type of computer memory used to quickly accept, store, and transfer data and instructions that are being used immediately by the CPU.
2. **Cache**: Additional memory system that temporarily stores frequently used instructions and data for quicker processing by the CPU. → It has multiple levels :- L1, L2 & L3 cache.
3. **Main Memory**: RAM.
4. **Secondary Memory**: Storage media, on which computer can store data & programs.

Comparison

1. **Cost**:
   a. Primary storages are costly.
   b. Registers are most expensive due to expensive semiconductors & labour.
   c. Secondary storages are cheaper than primary.
2. **Access Speed**:
   a. Primary has higher access speed than secondary memory.
   b. Registers has highest access speed, then comes cache, then main memory.
3. **Storage size**:
   a. Secondary has more space.
4. **Volatility**:
   a. Primary memory is volatile.
   b. Secondary is non-volatile.

5→ ROM → type of non-volatile memory that stores firmware & system boot instructions.
1- EPROM → can be programmed once.
2- EPROM → can be erased using UV light & reprogrammed.
3- EEPROM → can be erased & reprogrammed electrically.

① → when we use recursion, if there is no base condition, then "stack overflow" error comes.

② → when there is a garbage collection, "out of memory" error comes.

Lec-9: Introduction to Process

1. What is a program? Compiled code, that is ready to execute.
2. What is a process? Program under execution.
3. How OS creates a process? Converting program into a process.

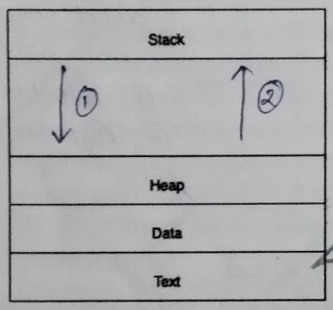
   STEPS:

   a. → Load the program & static data into memory. → used for initialisation. → from disk to RAM.
   b. → Allocate runtime stack → part of memory used for local variables, fxn arguments & return values.
   c. → Heap memory allocation. → part of memory used for dynamic allocation.
   d. → IO tasks.
   e. → OS handoffs control to main ().

   d → initialize process control block.

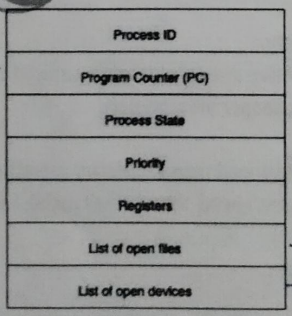
   f → Adding process to the ready queue, waiting for CPU time. CPU-scheduler selects it for execution & executes it.

4. **Architecture** of process:



5. **Attributes** of process:
   a. Feature that allows identifying a process uniquely.
   b. Process table → OS के Kernel space में store होता है।
      i. All processes are being tracked by OS using a table like data structure.
      ii. Each entry in that table is process control block (PCB).
   c. PCB: Stores info/attributes of a process.
      i. Data structure used for each process, that stores information of a process such as process id, program counter, process state, priority etc.
6. **PCB structure**:

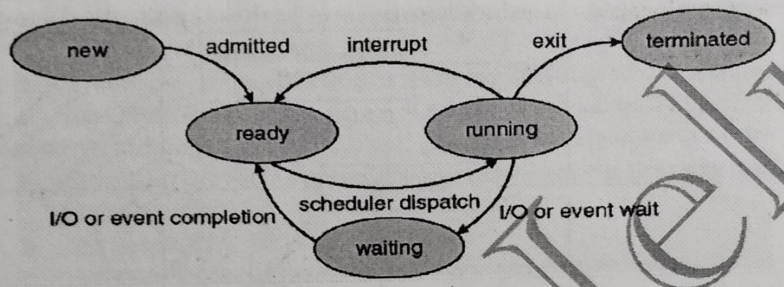
It is a pointer which fetch the instruction & then increases and so on.



✓ **Registers in the PCB**, it is a data structure. When a processes is running and it's time slice expires, the current value of process specific registers would be stored in the PCB and the process would be swapped out. When the process is scheduled to be run, the register values is read from the PCB and written to the CPU registers. This is the main purpose of the registers in the PCB.

Lec-10: Process States | Process Queues

1. **Process States**: As process executes, it changes state. Each process may be in one of the following states.
   a. **New**: OS is about to pick the program & convert it into process. OR the process is being created.
   b. **Run**: Instructions are being executed; CPU is allocated.
   c. **Waiting**: Waiting for IO.
   d. **Ready**: The process is in memory, waiting to be assigned to a processor.
   e. **Terminated**: The process has finished execution. PCB entry removed from process table.

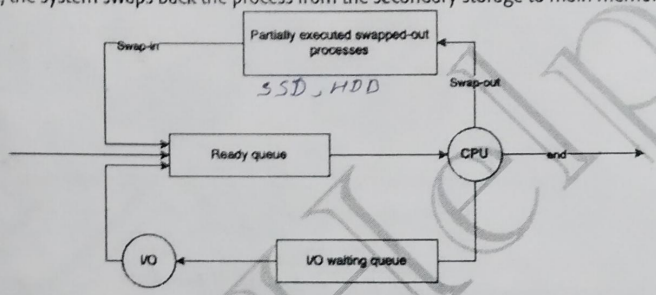


2. **Process Queues:** → data structure
   a. Job Queue:
      i. Processes in new state.
      ii. Present in secondary memory.
      iii. **Job Schedular (Long term schedular (LTS))** picks process from the pool and loads them into memory for execution.
   b. Ready Queue:
      i. Processes in Ready state.
      ii. Present in main memory.
      iii. **CPU Schedular (Short-term schedular)** picks process from ready queue and dispatch it to CPU. ↳ working on high frequency.
   c. Waiting Queue: & Device Queue.
      i. Processes in Wait state.
3. **Degree of multi-programming**: The number of processes in the memory. (ready queue).
   a. LTS controls degree of multi-programming.
4. **Dispatcher:** The module of OS that gives control of CPU to a process selected by **STS**.

why we use process Queues?
→ Multitasking environment → fairness maintain.
→ CPU scheduling
→ Resource allocation.

LEC-11: Swapping | Context-Switching | Orphan process | Zombie process

1. **Swapping**
   a. Time-sharing system may have medium term schedular (MTS).
   b. Remove processes from memory to reduce degree of multi-programming.
   c. These removed processes can be reintroduced into memory, and its execution can be continued where it left off. This is called **Swapping**.
   d. Swap-out and swap-in is done by MTS.
   e. Swapping is necessary to improve process mix or because a change in memory requirements has overcommitted available memory, requiring memory to be freed up.
   f. Swapping is a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.



2. **Context-Switching**
   a. Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process.
   b. When this occurs, the kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.
   c. It is pure overhead, because the system does no useful work while switching.
   d. Speed varies from machine to machine, depending on the memory speed, the number of registers that must be copied etc.
3. **Orphan process**
   a. The process whose parent process has been terminated and it is still running.
   b. Orphan processes are adopted by init process.
   c. Init is the first process of OS.
4. **Zombie process / Defunct process**
   a. A zombie process is a process whose execution is completed but it still has an entry in the process table.
   b. Zombie processes usually occur for child processes, as the parent process still needs to read its child's exit status. Once this is done using the wait system call, the zombie process is eliminated from the process table. This is known as **reaping** the zombie process.
   c. It is because parent process may call wait () on child process for a longer time duration and child process got terminated much earlier.
   d. As entry in the process table can only be removed, after the parent process reads the exit status of child process. Hence, the child process remains a zombie till it is removed from the process table.

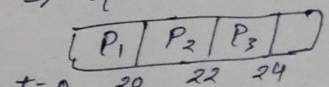
cmd commands for orphane process :-
(i) List all running process -
→ wmic process get PID, PPID, name
(ii) check orphane process → to verify if a PPID exists -
→ tasklist | findstr PPID
(iii) How to check if its parents exists ?
→ wmic process where PID = 15XX get PPID
(iv) Kill the orphan process -
→ taskkill / PID 15XX / F

LEC-12: Intro to Process Scheduling | FCFS | Convoy Effect

1. **Process Scheduling**
   a. Basis of Multi-programming OS.
   b. By switching the CPU among processes, the OS can make the computer more productive.
   c. Many processes are kept in memory at a time, when a process must wait or time quantum expires, the OS takes the CPU away from that process & gives the CPU to another process & this pattern continues.
2. **CPU Scheduler** → process scheduling.
   a. Whenever the CPU become ideal, OS must select one process from the ready queue to be executed.
   b. Done by STS.
3. **Non-Preemptive scheduling** → Lack of time quantum.
   a. Once CPU has been allocated to a process, the process keeps the CPU until it releases CPU either by terminating or by switching to wait-state. → it will leave the cpu when it terminates or goes to wait state.
   b. Starvation, as a process with long burst time may starve less burst time process.
   c. Low CPU utilization. & overhead decreases.
4. **Preemptive scheduling** → It includes time quantum.
   a. CPU is taken away from a process after time quantum expires along with terminating or switching to wait-state.
   b. Less Starvation
   c. High CPU utilization. & overhead increases.
5. **Goals of CPU scheduling**
   a. Maximum CPU utilization
   b. Minimum Turnaround time (TAT).
   c. Min. Wait-time
   d. Min. response time.
   e. Max. throughput of system.
6. **Throughput**: No. of processes completed per unit time.
7. **Arrival time (AT)**: Time when process is arrived at the ready queue.
8. **Burst** time (BT): The time required by the process for its execution.
9. **Turnaround** time (TAT): Time taken from first time process enters ready queue state till it terminates. (CT - AT)
10. **Wait** time (WT): Time process spends waiting for CPU. (WT = TAT – BT)
11. **Response** time: Time duration between process getting into ready queue and process getting CPU for the first time.
12. **Completion** Time (CT): Time taken till process gets terminated.
13. **FCFS** (First come-first serve):
    a. Whichever process comes first in the ready queue will be given CPU first.
    b. In this, if one process has longer BT. It will have major effect on average WT of diff processes, called **Convoy effect.**
    c. Convoy Effect is a situation where many processes, who need to use a resource for a short time, are blocked by one process holding that resource for a long time.
       i. This cause poor resource management.

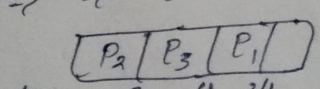
① ⇒ Gantt chart

Process	AT	BT	CT	TAT	WT
P_1	0	20	20	20	0
P_2	1	2	22	21	19
P_3	2	2	24	22	20
					Avg. WT_1 = 13 units.



② ⇒ Gantt chart

P_2	0	2	2	2	0
P_3	1	2	4	3	1
P_1	2	20	24	22	2
					Avg WT_2 = 1 units.



Process	AT	BT	CT	TAT	WT
P_1	0	8	8	8	0
P_2	1	4	12	11	7
P_3	2	9	26	24	15
P_4	3	5	17	14	9

Avg. WT = 7.75 units

Gantt chart ⇒

P_1	P_2	P_4	P_3
t=0	8	12	17	26

LEC-13: CPU Scheduling | SJF | Priority | RR

1. Shortest Job First (SJF) [Non-preemptive]
   a. Process with least BT will be dispatched to CPU first.
   b. Must do estimation for BT for each process in ready queue beforehand, Correct estimation of BT is an impossible task (ideally.)
   c. Run lowest time process for all time then, choose job having lowest BT at that instance.
   d. This will suffer from convoy effect as if the very first process which came is Ready state is having a large BT.
   e. Process starvation might happen.
   f. Criteria for SJF algos, AT + BT.

Pre-emption ⇒ $P_1(15) \rightarrow P_2(4s)$ ↓ pre-empt हो जाएगी, bcz ($BT_{P_1} > BT_{P_2}$)

2. SJF [Preemptive] ⟶

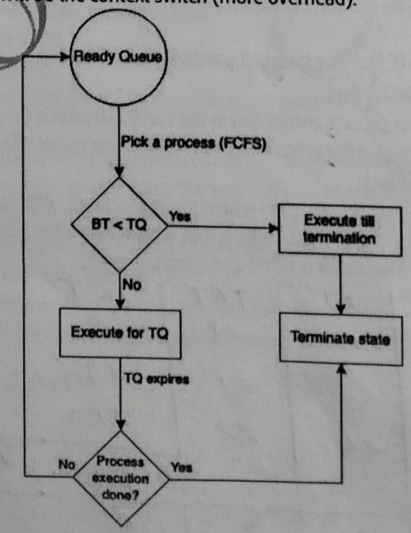
Job	AT	BT	CT	TAT	WT
P_1	0	~~8~~ 7	17	17	9
P_2	1	4	5	4	0
P_3	2	9	26	24	15
P_4	3	5	10	7	2

Avg. WT = 6.5 units

Gantt chart

P_1	P_2	P_4	P_1	P_3
t=0	1	5	10	17	26

   a. Less starvation.
   b. No convoy effect.
   c. **Gives average WT less for a given set of processes as scheduling short job before a long one decreases the WT of short job more than it increases the WT of the long process.**
3. Priority Scheduling [Non-preemptive]
   a. Priority is assigned to a process when it is created.
   b. SJF is a special case of general priority scheduling with priority inversely proportional to BT.
4. Priority Scheduling [Preemptive]
   a. Current RUN state job will be preempted if next job has higher priority.
   b. May cause indefinite waiting (Starvation) for lower priority jobs. (Possibility is they won't get executed ever). (True for both preemptive and non-preemptive version)
      i. Solution: Ageing is the solution.
      ii. Gradually increase priority of process that wait so long. E.g., increase priority by 1 every 15 minutes.
5. Round robin scheduling (RR)
   a. Most popular
   b. Like FCFS but preemptive.
   c. Designed for time sharing systems.
   d. Criteria: AT + time quantum (TQ), Doesn't depend on BT.
   e. No process is going to wait forever, hence very low starvation. [No convoy effect]
   f. Easy to implement.
   g. If TQ is small, more will be the context switch (more overhead).

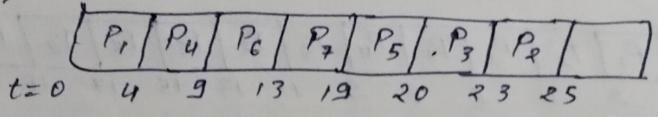


Non-preemptive priority scheduling :-

process	priority	AT	BT	CT	TAT	WT
P_1	2	0	4	4	4	0
P_2	4	1	2	25	24	22
P_3	6	2	3	23	21	18
P_4	10	3	5	9	6	1
P_5	8	4	1	20	16	15
P_6	12	5	4	13	8	4
P_7	9	6	6	19	13	7

Avg. WT = 9.714 units

Gantt chart =>

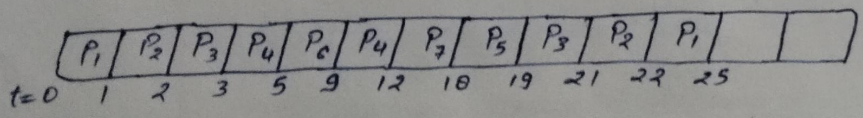


preemptive priority scheduling :-

process	priority	AT	BT	CT	TAT	WT
P_1	2	0	~~4~~ 3	25	25	21
P_2	4	1	~~2~~ 1	22	21	19
P_3	6	2	~~3~~ 2	21	19	16
P_4	10	3	~~5~~ 3	12	9	4
P_5	8	4	1	19	15	14
P_6	12	5	4	9	4	0
P_7	9	6	6	18	12	6

Avg. WT. = 11.4 units

Gantt chart =>



Round-Robin scheduling :- Assume TQ = 2s.

process	AT	BT	CT	TAT	WT
P_1	0	4 → 2 → 0	8	8	4
P_2	1	5 → 3 → 1 → 0	18	17	12
P_3	2	2 → 0	6	4	2
P_4	3	1 → 0	9	6	5
P_5	4	6 → 4 → 2 → 0	21	17	11
P_6	5	3 → 1 → 0	19	14	11

Avg. WT = 7.5 units.

Ready-Queue => P_1 P_2 P_3 P_1 P_4 P_5 P_2 P_6 P_5 P_2 P_6 P_5

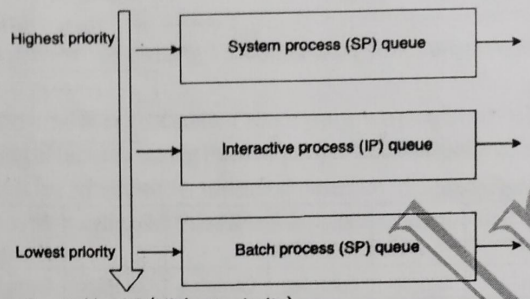
Gantt chart =>

P_1	P_2	P_3	P_1	P_4	P_5	P_2	P_6	P_5	P_2	P_6	P_5		

t=0 2 4 6 8 9 11 13 15 17 18 19 21

LEC-14: MLQ | MLFQ

1. Multi-level queue scheduling (**MLQ**)
   a. Ready queue is divided into multiple queues depending upon priority.
   b. A process is permanently assigned to one of the queues (inflexible) based on some property of process, memory, size, process priority or process type.
   c. Each queue has its own scheduling algorithm. E.g., SP -> RR, IP -> RR & BP -> FCFS.

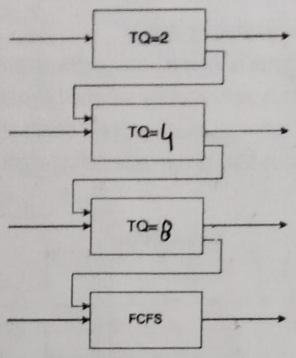


   d. System process: Created by OS (Highest priority)
   Interactive process (Foreground process): Needs user input (I/O).
   Batch process (Background process): Runs silently, no user input required.
   e. Scheduling among different sub-queues is implemented as **fixed priority preemptive** scheduling. E.g., foreground queue has absolute priority over background queue.
   f. If an interactive process comes & batch process is currently executing. Then, batch process will be preempted.
   g. Problem: Only after completion of all the processes from the top-level ready queue, the further level ready queues will be scheduled.
   This came starvation for lower priority process.
   h. Convoy effect is present.

2. Multi-level feedback queue scheduling (**MLFQ**)
   a. Multiple sub-queues are present.
   b. Allows the process to move between queues. The idea is to separate processes according to the characteristics of their BT. If a process uses too much CPU time, it will be moved to lower priority queue. This scheme leaves I/O bound and interactive processes in the higher-priority queue.
   In addition, a process that waits too much in a lower-priority queue may be moved to a higher priority queue. This form of ageing prevents starvation.
   c. Less starvation then MLQ.
   d. It is flexible.
   e. Can be configured to match a specific system design requirement.
   Sample MLFQ design:

How to design MLFQ =>
(1) no. of Queues
(2) Scheduling Algorithm in each queue.
(3) Method to demote a process to lower priority queue.
(4) Method to upgrade a process to higher priority que.
(5) Method to push a process at the starting on which queue.

sample of a MLFQ design:-



3. **Comparison:**

	FCFS	SJF	PSJF	Priority	P-Priority	RR	MLQ	MLFQ
Design	Simple	Complex	Complex	Complex	Complex	Simple	Complex	Complex
Preemption	No	No	Yes	No	Yes	Yes	Yes	Yes
Convoy effect	Yes	Yes	No	Yes	Yes	No	Yes	Yes
Overhead	No	No	Yes	No	Yes	Yes	Yes	Yes

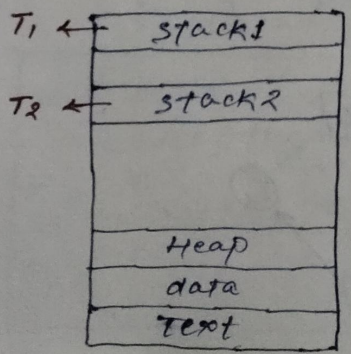
LEC-15: Introduction to Concurrency

1. **Concurrency** is the execution of the multiple instruction sequences at the same time. It happens in the operating system when there are several process threads running in parallel.
2. **Thread**:
   - Single sequence stream within a process.
   - An independent path of execution in a process.
   - Light-weight process.
   - Used to achieve parallelism by dividing a process's tasks which are independent path of execution.
   - E.g., Multiple tabs in a browser, text editor (When you are typing in an editor, spell checking, formatting of text and saving the text are done concurrently by multiple threads.)

1 - we have increased responsiveness
2 - No delay

3. **Thread Scheduling:** Threads are scheduled for execution based on their priority. Even though threads are executing within the runtime, all threads are assigned processor time slices by the operating system.
4. **Threads context switching**
   - OS saves current state of thread & switches to another thread of same process.
   - Doesn't includes switching of memory address space. (But Program counter, registers & stack are included.)
   - Fast switching as compared to process switching
   - CPU's cache state is preserved.
5. **How each thread get access to the CPU?**
   - Each thread has its own program counter.
   - Depending upon the thread scheduling algorithm, OS schedule these threads.
   - OS will fetch instructions corresponding to PC of that thread and execute instruction.
6. **I/O or TQ, based context switching is done here as well**
   - We have TCB (Thread control block) like PCB for state storage management while performing context switching.
7. **Will single CPU system would gain by multi-threading technique?**
   - Never.
   - As two threads have to context switch for that single CPU.
   - This won't give any gain.
8. **Benefits of Multi-threading.**
   - Responsiveness → App (I/O tasks, internet access, back-up)
   - Resource sharing: Efficient resource sharing. → shared memory (no communication needed)
   - Economy: It is more economical to create and context switch threads.
     1. Also, allocating memory and resources for process creation is costly, so better to divide tasks into threads of same process.
   - Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.

Architecture of threads.



⇒ count ++ → count = count + 1

But actually it happens in CPU registers like this-

⇒ temp = count + 1
⇒ count = temp

} one extra temp variable comes into role play-

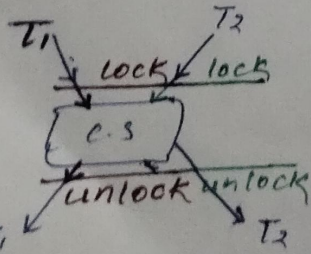
LEC-16: Critical Section Problem and How to address it

1. Process synchronization techniques play a key role in maintaining the consistency of shared data
2. **Critical Section (C.S)**
   a. The critical section refers to the segment of code where processes/threads access shared resources, such as common variables and files, and perform write operations on them. Since processes/threads execute concurrently, any process can be interrupted mid-execution.
3. **Major Thread scheduling issue**
   a. **Race Condition**
      i. A race condition occurs when two or more threads can access shared data and they try to change it at the same time. Because the thread scheduling algorithm can swap between threads at any time, you don't know the order in which the threads will attempt to access the shared data. Therefore, the result of the change in data is dependent on the thread scheduling algorithm, i.e., both threads are "racing" to access/change the data.

* first it will execute the operation & then context switched.

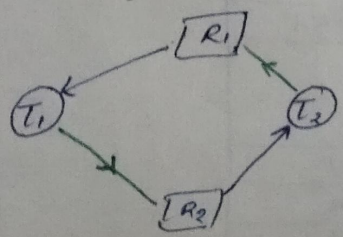
4. **Solution to Race Condition**
   a. Atomic operations: Make Critical code section an atomic operation, i.e., Executed in one CPU cycle. → c++ → atomic<int> →
   b. Mutual Exclusion using locks → sequential execution rather than parallel execution (T_1 & T_2)
   c. Semaphores
5. Can we use a simple flag variable to solve the problem of race condition?
   a. No. Bcz a fixed order comes bcz of initial value of turn.
6. **Peterson's solution** can be used to avoid race condition but holds good for only 2 process/threads.
7. **Mutex/Locks**
   a. Locks can be used to implement mutual exclusion and avoid race condition by allowing only one thread/process to access critical section.
   b. **Disadvantages**:
      i. **Contention**: one thread has acquired the lock, other threads will be busy waiting, what if thread that had acquired the lock dies, then all other threads will be in infinite waiting.
      ii. **Deadlocks**
      iii. Debugging → issues.
      iv. Starvation of high priority threads. → low priority thread has acquired c.s. & locked it & now if a high priority thread comes then it will wait until c.s. will not be unclocked by low priority T.

Locks →

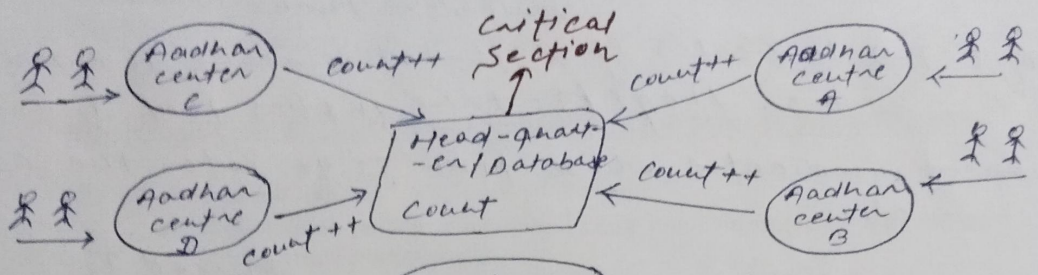


→ used to achieve mutual Exclusion by sequential execution.

Deadlock →



critical section & race condition :-



count = 11 → at time t = T

let say A → count++ at t = T + T_1

↳ temp = count + 1 } ⇒ count = temp

⇒ temp = 12

→ context switch to B →

at the same time t = T + T_1 ⇒ B → count ++

⇒ temp = count + 1

⇒ temp = 11 + 1 = 12. } ⇒ count = temp

finally ⇒ count = 12, but actually should be 13 bcz we had 2 requests from A & B.

so, one request is lost leading to data-inconsistency which we call Race condition.

conditions for having a solution of C.S.

(i) mutual exclusion → sequential execution

(ii) progess → if C.S. is idle then there will be equal opportunity for all threads, to get into critical section (C.S).

(iii) Bounded waiting → (i) should not have indefinite waiting for any thread

(ii) limited waiting.

can we use a single flag variable to solve the problem of race condition ?

T_1 thread

```
while(1){
    while(turn != 0);
    C.S.;
    turn = 1;
    R.S.;
}
```

turn = flag = 0/1

- if turn = 0 → T_1 → T_2
- if turn = 1 → T_2 → T_1

T_2 thread

```
while(1){
    while(turn != 1);
    C.S.
    turn = 0;
    R.S;
}
```

so, mutual exclusion achieved but not progress.

Peterson's solution → Array → flag[2]
variable → turn

flag[2] → indicates if a thread is ready to enter the c.s. flag[i] = true implies that T_i is ready.

turn → indicates whose turn is to enter the c.s.
↳ 0/1

thread T_1

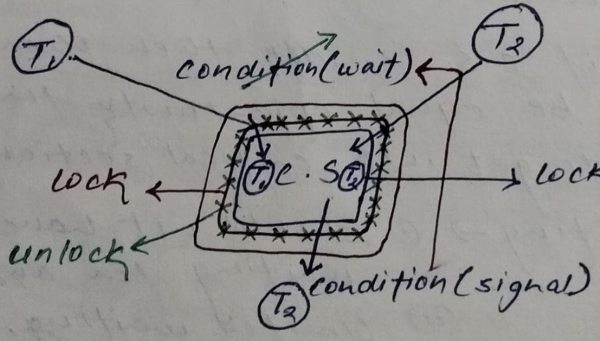
```
while(1){
    flag[0] = True;
    turn = 1;
    while (turn == 1 && flag[1] == true);
    c.s.
    flag[0] = false;
}
```

thread T_2

```
while(1){
    flag[1] = T;
    turn = 0;
    while (turn = 0 & flg[0] = T)
    c.s.
    flag[1] = false;
}
```

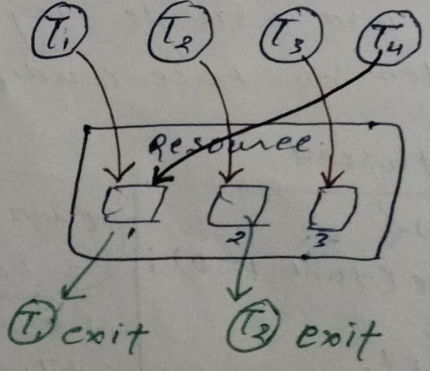
Here → ① mutual exclusion ✓
② progress ✓

conditional variables :- variable => condition ( )
wait signal



semaphores :-

semaphore = 3 2 1 0 0 1
≠ 0



NOTE → ① Threads parallelly execute होते हैं।
② But C.S. में entry sequential होती है due to mutual exclusion.

LEC-17: Conditional Variable and Semaphores for Threads synchronization

1. **Conditional variable**
   a. The condition variable is a synchronization primitive that lets the thread wait until a certain condition occurs.
   b. Works with a lock
   c. Thread can enter a wait state only when it has acquired a lock. When a thread enters the wait state, it will release the lock and wait until another thread notifies that the event has occurred. Once the waiting thread enters the running state, it again acquires the lock immediately and starts executing.
   d. Why to use conditional variable?
      i. To avoid busy waiting.
   e. **Contention** is not here.
2. **Semaphores**
   a. Synchronization method.
   b. An integer that is equal to number of resources (no. of instances of a resource) variable.
   c. Multiple threads can go and execute C.S concurrently.
   d. Allows multiple program threads to access the finite instance of resources whereas mutex allows multiple threads to access a single shared resource one at a time.
   e. Binary semaphore: value can be 0 or 1.
      i. Aka, mutex locks
   f. Counting semaphore
      i. Can range over an unrestricted domain.
      ii. Can be used to control access to a given resource consisting of a finite number of instances.
   g. To overcome the need for busy waiting, we can modify the definition of the wait () and signal () semaphore operations. When a process executes the wait () operation and finds that the semaphore value is not positive, it must wait. However, rather than engaging in busy waiting, the process car block itself. The block- operation places a process into a waiting queue associated with the semaphore, and the state of the process is switched to the Waiting state. Then control is transferred to the CPU scheduler, which selects another process to execute.
   h. A process that is blocked, waiting on a semaphore S, should be restarted when some other process executes a signal () operation. The process is restarted by a wakeup () operation, which changes the process from the waiting state to the ready state. The process is then placed in the ready queue.

main code

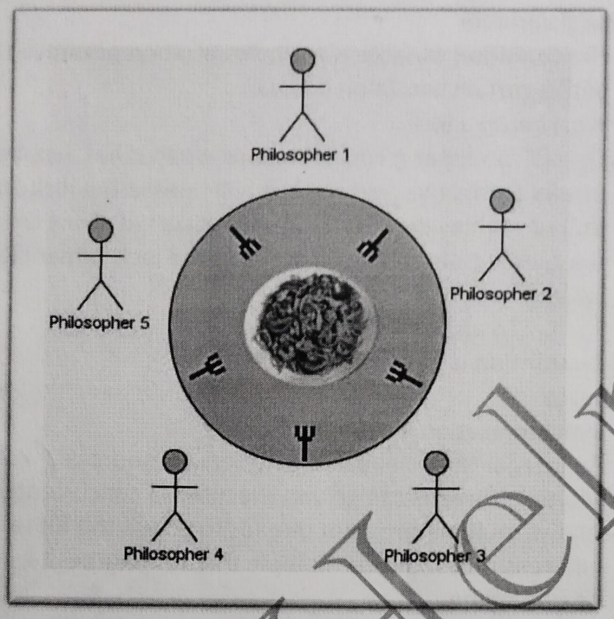
```
wait()
C.S.()
signal()
```

```
wait(s){
    s.value--;
    if(s.value<0){
        add to s.blocklist;
        block();
    }
}
```

```
signal(s){
    s.value++;
    if(s.value<=0){
        remove P from s.blocklist;
        wakeup(P);
    }
}
```

given :→ Semaphore ⇒ S = ?;
T_1 → wait() → s.val = 1 → After C.S. → signal
T_2 → wait() → s.val = 0 → C.S.
T_3 → wait() → s.val = -1 → block() → wakeup() → gets CPU.

Lec-20: The Dining Philosophers problem



problem statement →

1. We have **5 philosophers.**
2. They spend their life just being in **two states:**
   a. Thinking
   b. Eating
3. They sit on a circular table surrounded by 5 chairs (1 each), in the center of table is a bowl of noodles, and the table is laid with 5 single forks.
4. **Thinking state**: When a ph. Thinks, he doesn't interact with others.
5. **Eating state:** When a ph. Gets hungry, he tries to pick up the 2 forks adjacent to him (Left and Right). He can pick one fork at a time.
6. One can't pick up a fork if it is already taken.

brute-force solution. →

7. When ph. Has both forks at the same time, he eats without releasing forks.
8. Solution can be given using semaphores.
   a. Each fork is a binary semaphore. → semaphore fork[5]{1}.
   b. A ph. Calls wait() operation to acquire a fork.
   c. Release fork by calling signal().
   d. **Semaphore fork[5]{1};**
9. Although the semaphore solution makes sure that no two neighbors are eating simultaneously but it could still create **Deadlock.**

problem → deadlock

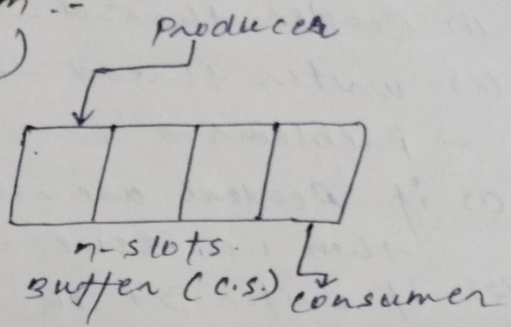
10. Suppose that all 5 ph. Become hungry at the same time and each picks up their left fork, then All fork semaphores would be 0.
11. When each ph. Tries to grab his right fork, he will be waiting for ever (Deadlock)

Solution →

12. We must use **some methods to avoid Deadlock and make the solution work**
    a. Allow at most 4 ph. To be sitting simultaneously.
    b. Allow a ph. To pick up his fork only if both forks are available and to do this, he must pick them up in a critical section (atomically).

Lecture - 18

producer - consumer problem :- (Bounded - Buffer problem)



(1) producer thread
(2) consumer thread.

→ problem ???

(1) C.S. (buffer) must avoid race condition.
→ synchronisation b/w producer & consumer
(2) producer must not insert data when buffer is full.
(3) consumer must not pick/remove data when the buffer is empty.

Solution :- using Semaphores.

(1) m, mutex → binary semaphore used to acquire lock on buffer.
(2) Empty → counting semaphore used to track empty slots having initial value n.
(3) full → tracks filled slots having initial value 0.

producer soln

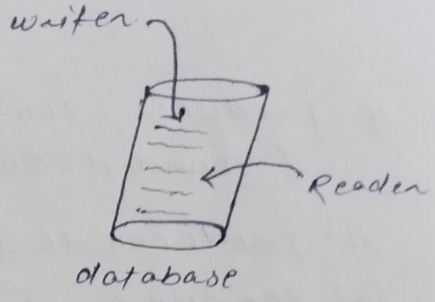
```
do {
    wait (empty); // wait until empty > 0 then empty.val--
    wait (mutex);
    // c.s., add data to buffer
    signal (mutex);
    signal (full); // increment full.val++
} while (1);
```

consumer soln

```
do {
    wait (full); // wait until full > 0, then full.val--;
    wait (mutex);
    // remove data from buffer
    signal (mutex);
    signal (empty); // increment empty.val++
} while (1);
```

Lecture - 19

Reader-writer problem :-



(1) Reader thread → Read
(2) writer thread → write / update

→ Problems →

(1) if Readers are reading > 1 then no issue.
(2) if writer > 1 OR 1 writer && some other thread (R/W then Race condition occurs & data inconsistency happ

→ Solution :- using semaphores.

(1) mutex → Binary semaphore used to ensure mutual exclusion, when readcount (RC) is updated. → no. two threads modify RC at the same time.
(2) wrt → Binary semaphore, common for both reader & writer & used for synchronisation.
(3) readcount (RC) → integer variable used to track how many readers are reading in

writer soln.

```
do{
   wait(wrt);
   // do write operation
   signal(wrt);
} while(true);
```

Reader soln.

```
do{
   wait(mutex); // to mutex RC variable
   rc++;
   if(rc==1){
      wait(wrt); // ensures no writer
      // can enter if there is even 1 re
   }
   signal(mutex);
   // C.S. → reader is reading.
   wait(mutex)
   rc--; // a reader leaves
   if(rc==0){  // no reader is left in C.
      signal(wrt); // writer can enter.
   }
   signal(mutex); // reader leaves
} while(1);
```

5. **Odd-even rule.**
   an odd ph. Picks up first his left fork and then his right fork, whereas an even ph. Picks up his right fork then his left fork.

13. Hence, only semaphores are not enough to solve this problem.
    We must add some enhancement rules to make deadlock free solution.

soln → ph(i)

```
do{
    wait(fork[i]);
    wait(fork[(i+1)%5]);
    // eat
    signal(fork[i]);
    signal(fork[(i+1)%5]);
    // think
}while();
```

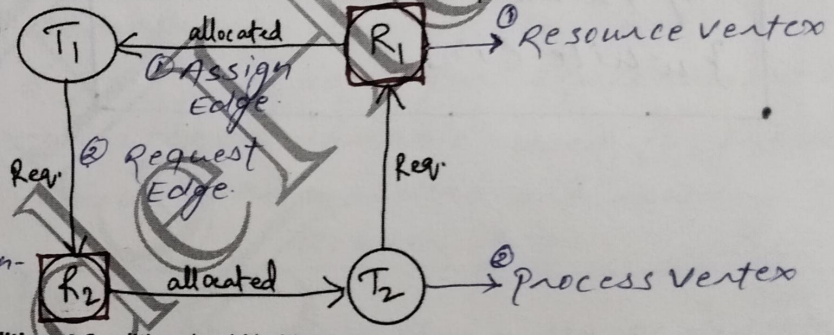
→ this soln creates problem of deadlock.
Hence we need to apply above methods to overcome with this problem.

LEC-21: Deadlock Part-1

1. In Multi-programming environment, we have several processes competing for finite number of resources
2. Process requests a **resource (R)**, if R is not available (taken by other process), process enters in a waiting state. Sometimes that waiting process is never able to change its state because the resource, it has requested is busy (forever), called **DEADLOCK (DL)**
3. Two or more processes are waiting on some resource's availability, which will never be available as it is also busy with some other process. The Processes are said to be in **Deadlock.**
4. DL is a bug present in the process/thread synchronization method.
5. In DL, processes never finish executing, and the system resources are tied up, preventing other jobs from starting.
6. **Example of resources**: Memory space, CPU cycles, files, locks, sockets, IO devices etc.
7. Single resource can have multiple instances of that. E.g., CPU is a resource, and a system can have 2 CPUs.
8. How a process/thread utilize a resource?
   a. Request: Request the R, if R is free Lock it, else wait till it is available.
   b. Use
   c. Release: Release resource instance and make it available for other processes

*used for system Representation

Resource Allocation Graph
① vertex → ① Resource vertex
② process vertex
② Edges → ① Assign Edge
② Request Edge



9. **Deadlock Necessary Condition: 4** Condition should hold simultaneously.
   a. **Mutual Exclusion**
      i. Only 1 process at a time can use the resource, if another process requests that resource, the requesting process must wait until the resource has been released.
   b. **Hold & Wait**
      i. A process must be holding at least one resource & waiting to acquire additional resources that are currently being held by other processes.
   c. **No-preemption**
      i. Resource must be voluntarily released by the process after completion of execution. (No resource preemption)
   d. **Circular wait**
      i. A set {P0, P1, ... ,Pn} of waiting processes must exist such that P0 is waiting for a resource held by P1, P1 is waiting for a resource held by P2, and so on.
10. **Methods for handling Deadlocks:**
    a. Use a protocol to **prevent** or **avoid** deadlocks, ensuring that the system will never enter a deadlocked state.
    b. Allow the system to enter a deadlocked state, **detect it, and recover.**

c. Ignore the problem altogether and pretend that deadlocks never occur in system. (Ostrich algorithm) aka, **Deadlock ignorance.**

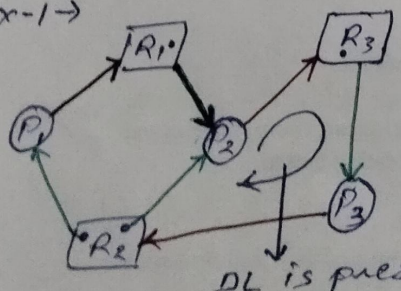
11. To ensure that deadlocks never occur, the system can use either a **deadlock prevention or deadlock avoidance scheme**.
12. **Deadlock Prevention**: by ensuring at least one of the necessary conditions cannot hold.
    a. **Mutual exclusion**
        i. Use locks (mutual exclusion) only for non-sharable resource.
        ii. Sharable resources like Read-Only files can be accessed by multiple processes/threads.
        iii. However, we can't prevent DLs by denying the mutual-exclusion condition, because some resources are intrinsically non-sharable.
    b. **Hold & Wait**
        i. To ensure H&W condition never occurs in the system, we must guarantee that, whenever a process requests a resource, it doesn't hold any other resource.
        ii. Protocol (A) can be, each process has to request and be allocated all its resources before its execution.
        iii. Protocol (B) can be, allow a process to request resources only when it has none. It can request any additional resources after it must have released all the resources that it is currently allocated.
    c. **No preemption**
        i. If a process is holding some resources and request another resource that cannot be immediately allocated to it, then all the resources the process is currently holding are preempted. The process will restart only when it can regain its old resources, as well as the new one that it is requesting. (Live Lock may occur).
        ii. If a process requests some resources, we first check whether they are available. If yes, we allocate them. If not, we check whether they are allocated to some other process that is waiting for additional resources. If so, preempt the desired resource from waiting process and allocate them to the requesting process.

to prevent Live Lock we can provide a delay.

    d. **Circular wait**
        i. To ensure that this condition never holds is to impose a proper ordering of resource allocation.
        ii. P1 and P2 both require R1 and R3, locking on these resources should be like, both try to lock R1 then R2. By this way which ever process first locks R1 will get R2.

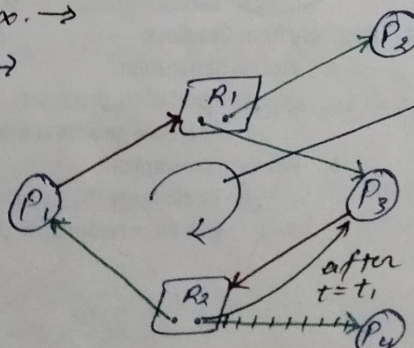
Resource Allocation Graph Ex. →

Ex-1 →



DL is present

Ex-2 →



cycle is present but deadlock is not present.

* By Definition of RAG
  ① if RAG has no cycle → no DL
  ② if RAG has cycles → may be DL

⇒ after time t_1, P_4 will release R_2's 2nd instance & that instance will get assign to P_3 & execution starts for P_3.

current state → ① no. of processes
② max. need of resources for each process
③ no. of currently allocated resources to each process
④ max. no. of instances of each resource & no. of resources available.

LEC-22: Deadlock Part-2

1. **Deadlock Avoidance**: Idea is, the kernel be given in advance info concerning which resources will use in its lifetime.
   By this, system can decide for each request whether the process should wait.
   To decide whether the current request can be satisfied or delayed, the system must consider the resources currently available, resources currently allocated to each process in the system and the future requests and releases of each process.
   a. Schedule process and its resources allocation in such a way that the DL never occur.
   b. Safe state: A state is safe if the system can allocate resources to each process (up to its max.) in some order and still avoid DL.
      A system is in safe state only if there exists a safe sequence.
   c. In an Unsafe state, the operating system cannot prevent processes from requesting resources in such a way that any deadlock occurs. It is not necessary that all unsafe states are deadlocks; an unsafe state may lead to a deadlock.
   d. The main key of the deadlock avoidance method is whenever the request is made for resources then the request must only be approved only in the case if the resulting state is a safe state.
   e. In a case, if the system is unable to fulfill the request of all processes, then the state of the system is called unsafe.
   f. Scheduling algorithm using which DL can be avoided by finding safe state. (Banker Algorithm)
2. **Banker Algorithm**
   a. When a process requests a set of resources, the system must determine whether allocating these resources will leave the system in a safe state. If yes, then the resources may be allocated to the process. If not, then the process must wait till other processes release enough resources.
3. **Deadlock Detection:** Systems haven't implemented deadlock-prevention or a deadlock avoidance technique, then they may employ DL detection then, recovery technique.
   a. Single Instance of Each resource type **(wait-for graph method)**
      i. A deadlock exists in the system if and only if there is a cycle in the wait-for graph. In order to detect the deadlock, the system needs to maintain the wait-for graph and periodically system invokes an algorithm that searches for the cycle in the wait-for graph.
   b. Multiple instances for each resource type.
      i. Banker Algorithm → ① if no safe sequence → Deadlock
      ② if safe sequence → no deadlock.
4. **Recovery from Deadlock**
   a. Process termination
      i. Abort all DL processes
      ii. Abort one process at a time until DL cycle is eliminated.
   b. Resource preemption
      i. To eliminate DL, we successively preempt some resources from processes and give these resources to other processes until DL cycle is broken.

Bankers Algorithm to find safe state →

Process	Allocated Res.			Max. Need			Available			Remaining Need.		
P	A	B	C	A	B	C	A	B	C	A	B	C
P_1	0	1	0	7	5	3	3	3	2	7	4	3 → ④
P_2	2	0	0	3	2	2	5	3	2	1	2	2 → ①
P_3	3	0	2	9	0	2	7	4	3	6	0	0 → ⑤
P_4	2	1	1	4	2	2	7	4	5	2	1	1 → ②
P_5	0	0	2	5	3	3	7	5	5	5	3	1 → ③
total Already	7	2	5				10	5	7			

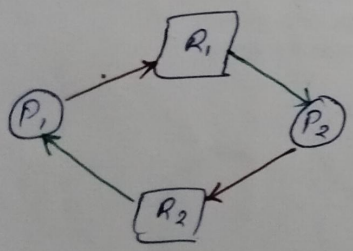
given total ⇒ A = 10
B = 5
C = 7

scheduling ⇒

P_2	P_4	P_5	P_1	P_3	→ safe state.

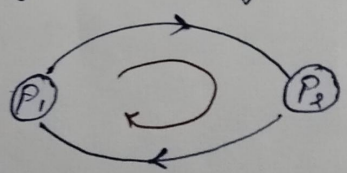
wait-for Graph Method :→ for deadlock Detection.

① Remove the resource vertices
② Merge the Edges from RAG



RAG

⇒

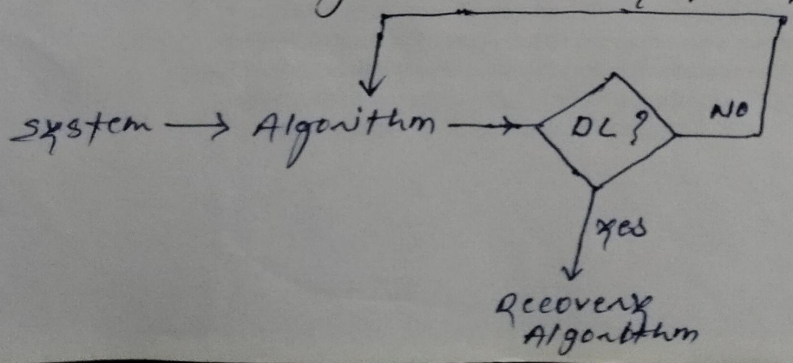


WGM

→ Deadlock Detection.

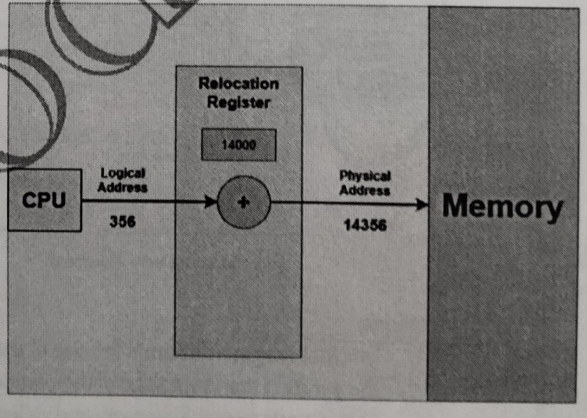
conclusion :→ in WGM ① if cycle → Deadlock
② if no cycle → no deadlock.

* How Detection Algorithms works ?



LEC-24: Memory Management Techniques | Contiguous Memory Allocation

1. In Multi-programming environment, we have multiple processes in the main memory (Ready Queue) to keep the CPU utilization high and to make computer responsive to the users.
2. To realize this increase in performance, however, we must keep several processes in the memory; that is, we must **share** the main memory. As a result, we must **manage** main memory for all the different processes.
3. **Logical versus Physical Address Space**
   a. Logical Address
      i. An address generated by the **CPU**.
      ii. The logical address is basically the address of an instruction or data used by a process.
      iii. User can access logical address of the process.
      iv. User has indirect access to the physical address through logical address.
      v. Logical address does not exist physically. Hence, aka, **Virtual address.**
      vi. The set of all logical addresses that are generated by any program is referred to as Logical Address Space.
      vii. **Range: 0 to *max*.**
   b. Physical Address
      i. An address loaded into the memory-address register of the physical memory.
      ii. User can never access the physical address of the Program.
      iii. The physical address is in the memory unit. It's a location in the main memory physically.
      iv. A physical address can be accessed by a user indirectly but not directly.
      v. The set of all physical addresses corresponding to the Logical addresses is commonly known as Physical Address Space.
      vi. It is computed by the **Memory Management Unit (MMU).**
      vii. **Range: (R + 0) to (R + max), for a base value R.**
   c. **The runtime mapping from virtual to physical address is done by a hardware device called the memory-management unit (MMU).**
   d. The user's program mainly generates the logical address, and the user thinks that the program is running in this logical address, but the program mainly needs physical memory in order to complete its execution.
   e.



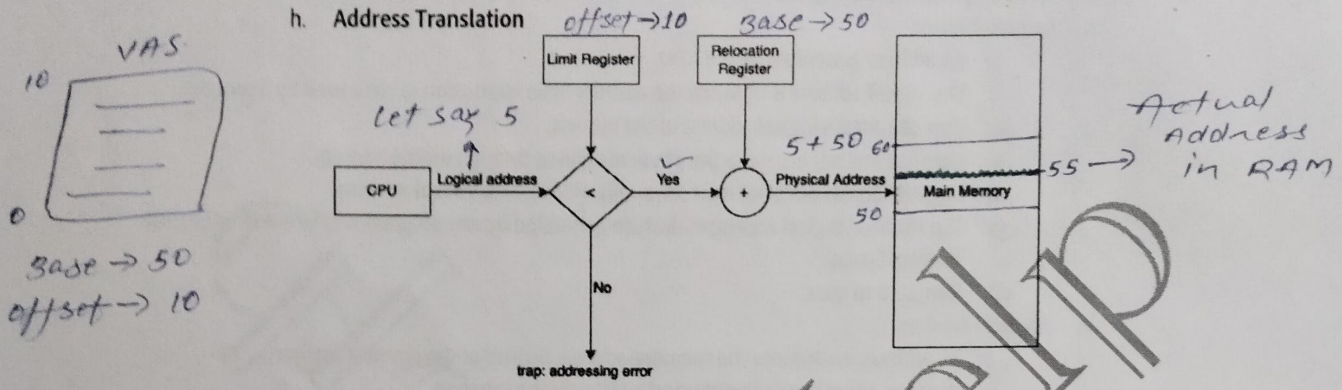
4. How OS manages the isolation and protect? **(Memory Mapping and Protection)**
   a. OS provides this Virtual Address Space (VAS) concept.
   b. To separate memory space, we need the ability to determine the range of legal addresses that the process may access and to ensure that the process can access only these legal addresses.
   c. The relocation register contains value of smallest physical address (Base address [R]); the limit register contains the range of logical addresses (e.g., relocation = 100040 & limit = 74600).
   d. Each logical address must be less than the limit register.

e. MMU maps the logical address dynamically by adding the value in the relocation register.

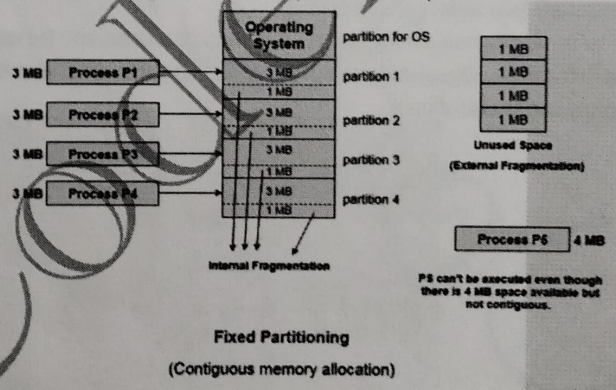
f. When CPU scheduler selects a process for execution, the dispatcher loads the relocation and limit registers with the correct values as part of the context switch. Since every address generated by the CPU (Logical address) is checked against these registers, we can protect both OS and other users' programs and data from being modified by running process.

g. Any attempt by a program executing in user mode to access the OS memory or other uses' memory results in a trap in the OS, which treat the attempt as a fatal error.

h. **Address Translation**



5. **Allocation Method on Physical Memory**
   a. Contiguous Allocation
   b. Non-contiguous Allocation
6. **Contiguous Memory Allocation**
   a. In this scheme, each process is contained in a single contiguous block of memory.
   b. **Fixed Partitioning**
      i. The main memory is divided into partitions of equal or different sizes.

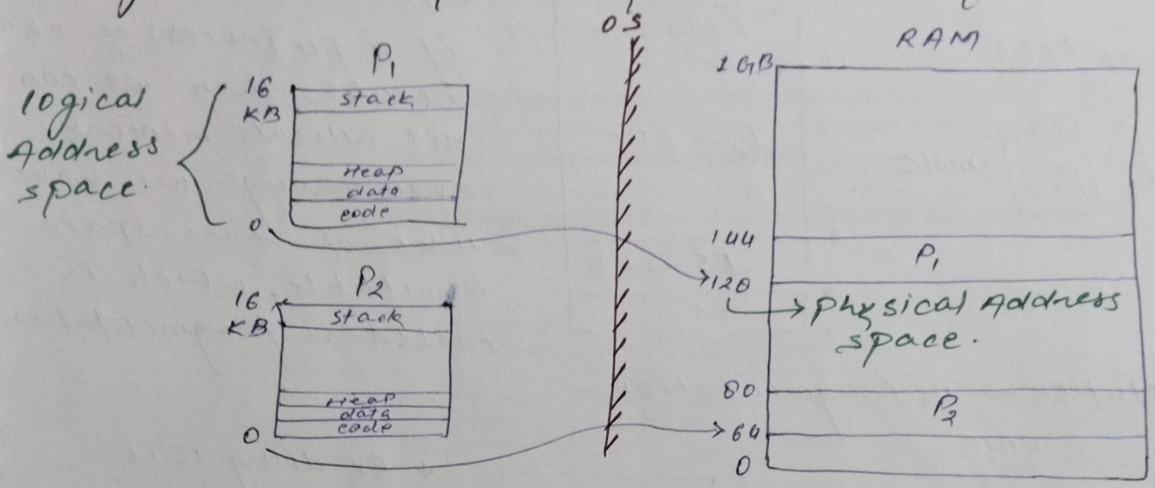


Fixed Partitioning

(Contiguous memory allocation)

      ii.
      iii. Limitations:
         1. **Internal Fragmentation:** if the size of the process is lesser then the total size of the partition then some size of the partition gets wasted and remain unused. This is wastage of the memory and called internal fragmentation.
         2. **External Fragmentation:** The total unused space of various partitions cannot be used to load the processes even though there is space available but not in the contiguous form.
         3. Limitation on process size: If the process size is larger than the size of maximum sized partition then that process cannot be loaded into the memory. Therefore, a limitation can be imposed on the process size that is it cannot be larger than the size of the largest partition.

Logical Address space / Physical Address space.



⇒ OS allocates 2 things → ① Base at RAM → physical Address space.
② offset

① P_1 → process
Base → 120
offset → 16

② P_2 → process.
Base → 64
offset → 16.

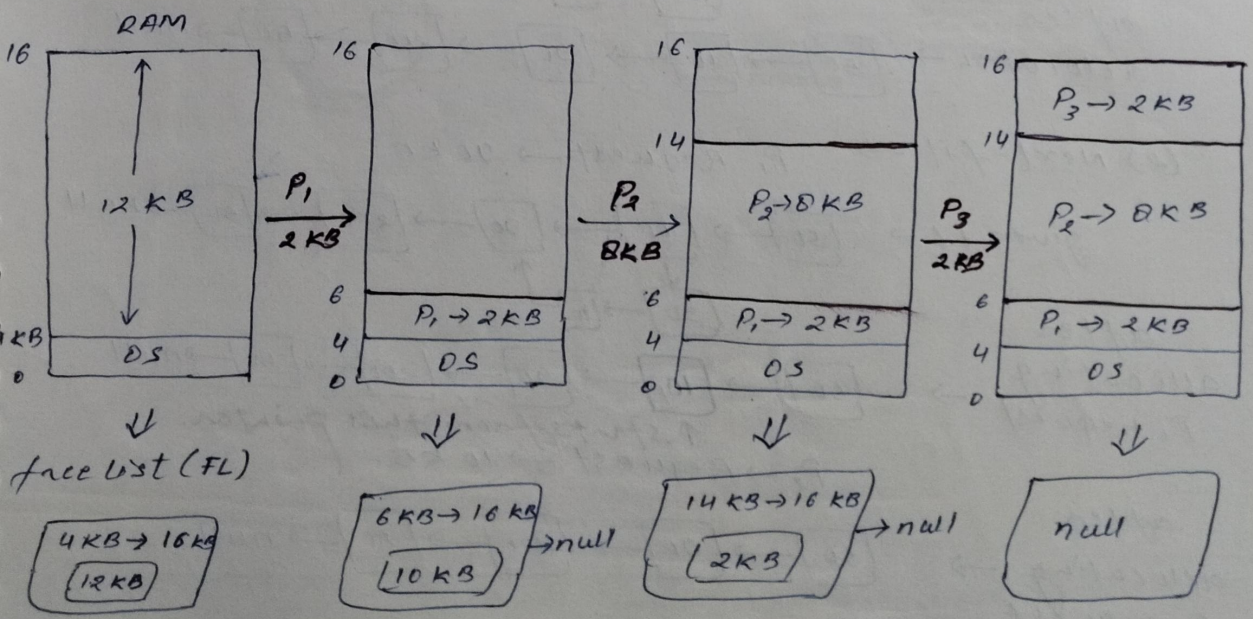
⇒ Let say P_2 wants to allocate memory –

Address → 0 + Random value.
0 + 129

Adress → 129
Actuall physical Address → 64 + 129 = 193

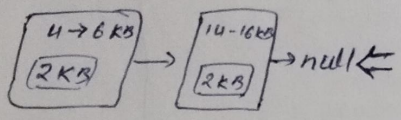
it throws Error.

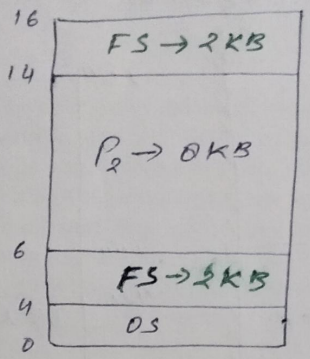
Free Space Management –



→ After P_1 & P_3 exit

Freelist (FL)

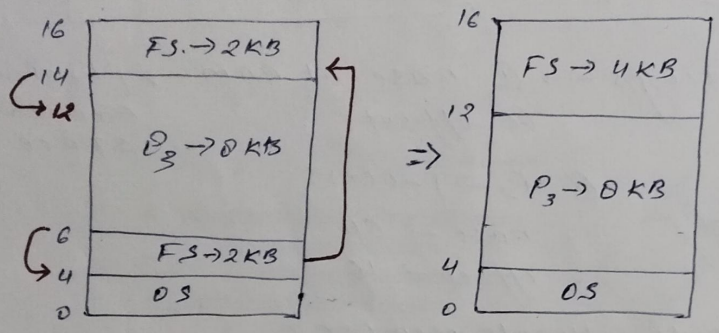




* Problem →
if a P4 process of 3KB occurs, then we can not allocate memory even though we have 4KB of free-space available, which is called as fragmentation

Solution → Defragmentation -

RAM



* By doing this the value of Relocation register will get change from 6 to 4. But limit Register will stay 0.

Algorithms → to find out the holes in LL and allocate them to process -

(1) First-Fit → P_1 Request → 90KB

given FL → 50 → 100 → 90 → 200 → 50 → null

90 → 10

after allocation → 50 → 10 → 90 → 200 → 50 → null

(2) Next-fit → P_1 Request → 90KB

given FL → 50 → 100 → 90 → 200 → 50 → null

90 → 10

after allocating P_1 request → 50 → 10 → 90 → 200 → 50 → null

↑ starts from this pointer.

P_2 → Request → 10KB.

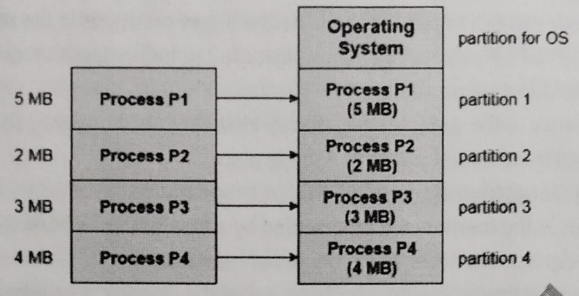
after allocating P_2 request → 50 → 90 → 200 → 50 → null

no. of process that can be in RAM.

4. Low degree of multi-programming: In fixed partitioning, the degree of multiprogramming is fixed and very less because the size of the partition cannot be varied according to the size of processes.

c. **Dynamic Partitioning**

i. In this technique, the partition size is not declared initially. It is declared at the time of process loading.



Dynamic Partitioning

(Process Size = Partition Size)

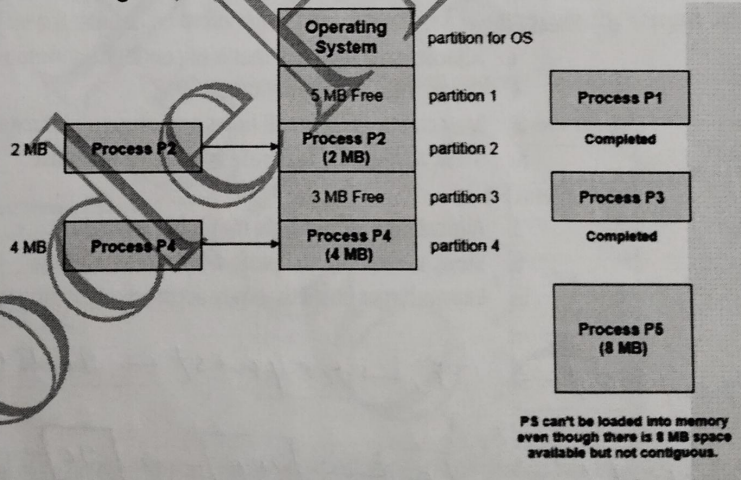
ii.

iii. Advantages over fixed partitioning

1. No internal fragmentation
2. No limit on size of process
3. Better degree of multi-programming

iv. Limitation

1. External fragmentation



External Fragmentation in Dynamic Partitioning

X

LEC-25: Free Space Management

1. **Defragmentation/Compaction**
   a. Dynamic partitioning suffers from external fragmentation.
   b. Compaction to minimize the probability of external fragmentation.
   c. All the free partitions are made contiguous, and all the loaded partitions are brought together.
   d. By applying this technique, we can store the bigger processes in the memory. The free partitions are merged which can now be allocated according to the needs of new processes. This technique is also called **defragmentation**.
   e. The efficiency of the system is decreased in the case of compaction since all the free spaces will be transferred from several places to a single place.
2. **How free space is stored/represented in OS?**
   a. Free holes in the memory are represented by a free list (Linked-List data structure).
3. **How to satisfy a request of a of n size from a list of free holes?**
   a. Various algorithms which are implemented by the Operating System in order to find out the holes in the linked list and allocate them to the processes.
   b. **First Fit**
      i. Allocate the first hole that is big enough.
      ii. Simple and easy to implement.
      iii. Fast/Less time complexity
   c. **Next Fit**
      i. Enhancement on First fit but starts search always from last allocated hole.
      ii. Same advantages of First Fit
   d. **Best Fit**
      i. Allocate smallest hole that is big enough.
      ii. Lesser internal fragmentation.
      iii. May create many small holes and cause major external fragmentation.
      iv. Slow, as required to iterate whole free holes list.
   e. **Worst Fit**
      i. Allocate the largest hole that is big enough.
      ii. Slow, as required to iterate whole free holes list.
      iii. Leaves larger holes that may accommodate other processes.

(3) Best-Fit → P_1 → request → 90 KB.

given FL → 50 → 100 → 90 → 200 → 50 → null

↑ smallest hole that is big enough to allocate.

after allocating P_1 → request. → 50 → 100 → 200 → 50 → null

(4) Worst-Fit → P_1 → request → 90 KB.

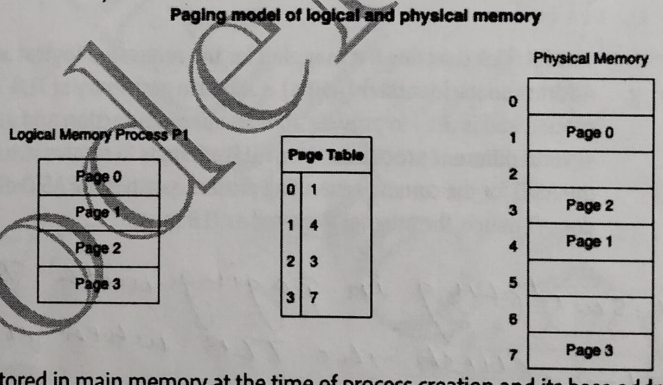
given FL → 50 → 100 → 90 → 200 → 50 → null

↑ largest hole that is big enough to allocate.

after allocating P_1 → request → 50 → 100 → 90 → 110 → 50 → null

LEC-26: Paging | Non-Contiguous Memory Allocation

1. The main **disadvantage of Dynamic partitioning is External Fragmentation.**
   a. Can be removed by Compaction, but with overhead.
   b. **We need more dynamic/flexible/optimal mechanism, to load processes in the partitions.**
2. **Idea behind Paging**
   a. If we have only two small non-contiguous free holes in the memory, say 1KB each.
   b. If OS wants to allocate RAM to a process of 2KB, in contiguous allocation, it is not possible, as we must have contiguous memory space available of 2KB. (External Fragmentation)
   c. **What if we divide the process into 1KB-1KB blocks?**
3. Paging
   a. **Paging is a memory-management scheme that permits the physical address space of a process to be non-contiguous.**
   b. It avoids external fragmentation and the need of compaction.
   c. Idea is to divide the physical memory into fixed-sized blocks called **Frames**, along with divide logical memory into blocks of same size called **Pages**. (# Page size = Frame size)
   d. **Page size** is usually determined by the processor architecture. Traditionally, pages in a system had uniform size, such as 4,096 bytes. However, processor designs often allow two or more, sometimes simultaneous, page sizes due to its benefits.
   e. **Page Table**
      i. A Data structure stores which page is mapped to which frame.
      ii. **The page table contains the base address of each page in the physical memory.**
   f. Every address generated by CPU (logical address) is divided into two parts: a page number (p) and a page offset (d). The p is used as an index into a page table to get base address the corresponding frame in physical memory.

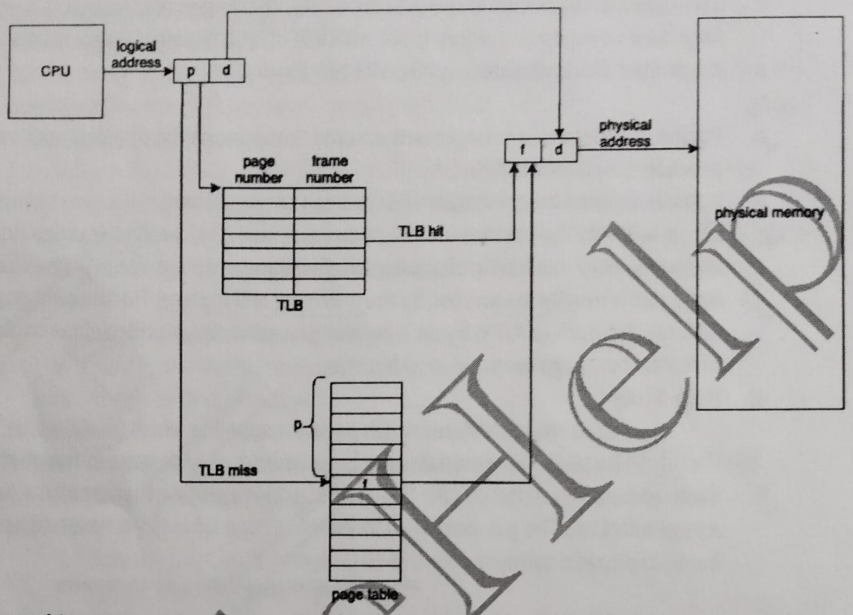


   g. Page table is stored in main memory at the time of process creation and its base address is stored in process control block (**PCB**).
   h. A page table base register (**PTBR**) is present in the system that points to the current page table. Changing page tables requires only this one register, at the time of context-switching.
4. **How Paging avoids external fragmentation?**
   a. Non-contiguous allocation of the pages of the process is allowed in the random free frames of the physical memory.
5. **Why paging is slow and how do we make it fast?**
   a. There are too many memory references to access the desired location in physical memory.
6. **Translation Look-aside buffer (TLB)**
   a. A Hardware support to speed-up paging process.
   b. It's a hardware cache, high speed memory.
   c. TBL has key and value.

d. Page table is stores in main memory & because of this when the memory references is made the translation is slow.

e. When we are retrieving physical address using page table, after getting frame address corresponding to the page number, we put an entry of the into the TLB. So that next time, we can get the values from TLB directly without referencing actual page table. Hence, make paging process faster.

Paging hardware with TLB



f. **TLB hit, TLB contains the mapping for the requested logical address.**

M2→ g. **Address space identifier (ASIDs)** is stored in each entry of TLB. ASID uniquely identifies each process and is used to **provide address space protection and allow to TLB to contain entries for several different processes.** When TLB attempts to resolve virtual page numbers, it ensures that the ASID for the currently executing process matches the ASID associated with virtual page. If it doesn't match, the attempt is treated as TLB miss.

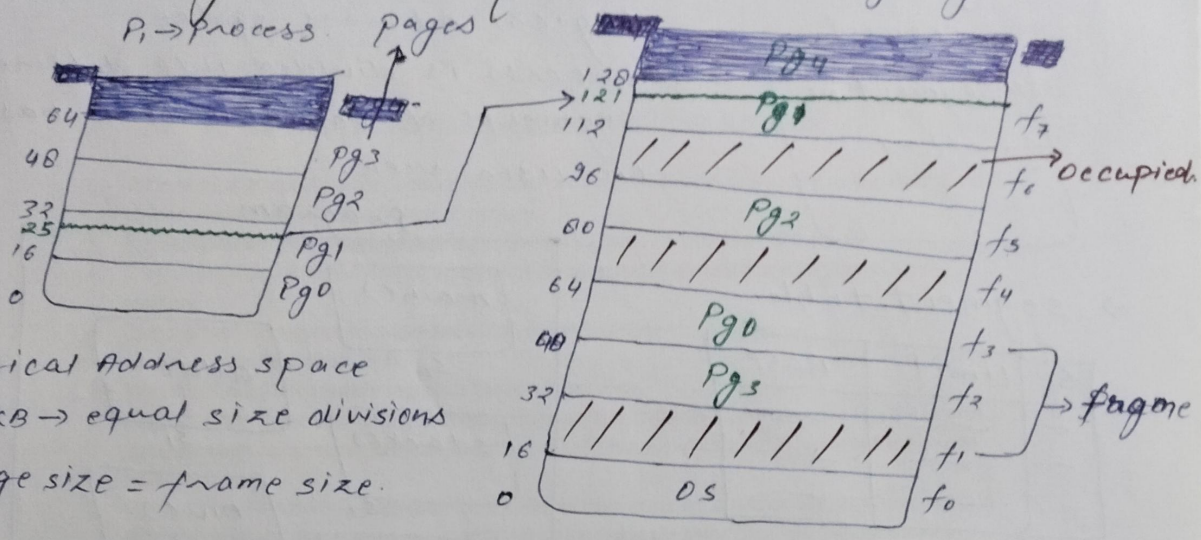
context switching in paging with TLB →

(i) M1→ flush the TLB when context switching occurs.

↳ It will be costly and un-reliable as context switching is a fast process.

NOTE:→ Internal fragmentation is present in paging.

Non-contiguous Memory Allocation - Paging -



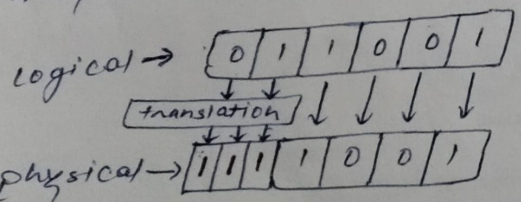
=> logical Address space
16 KB → equal size divisions

* Page size = frame size

page-table

logical page no.	physical frame no.
Pg0	f3
Pg1	f7
Pg2	f5
Pg3	f2

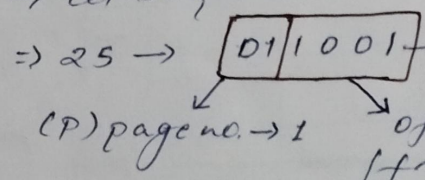
* Address translation →



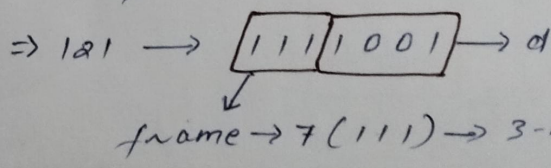
translation → It is alone in context with page table.

① → logical address → P_1 → 64 Bytes
=> $2^6 = 64$
=> 6 bits → required to represent address

→ let say we have taken 25



② → physical address



~~* Address translation →~~

segmentation → variable partitioning of logical address space.

→ process is divided into different segments of variable size based on user view.

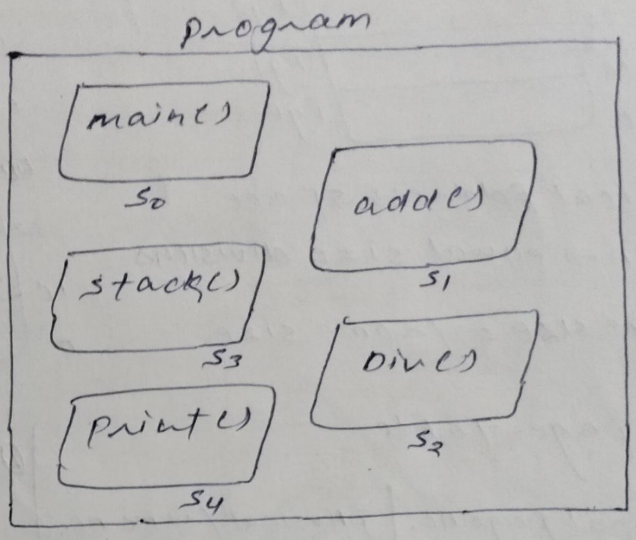
segment.

1→ segment no.

2→ offset.

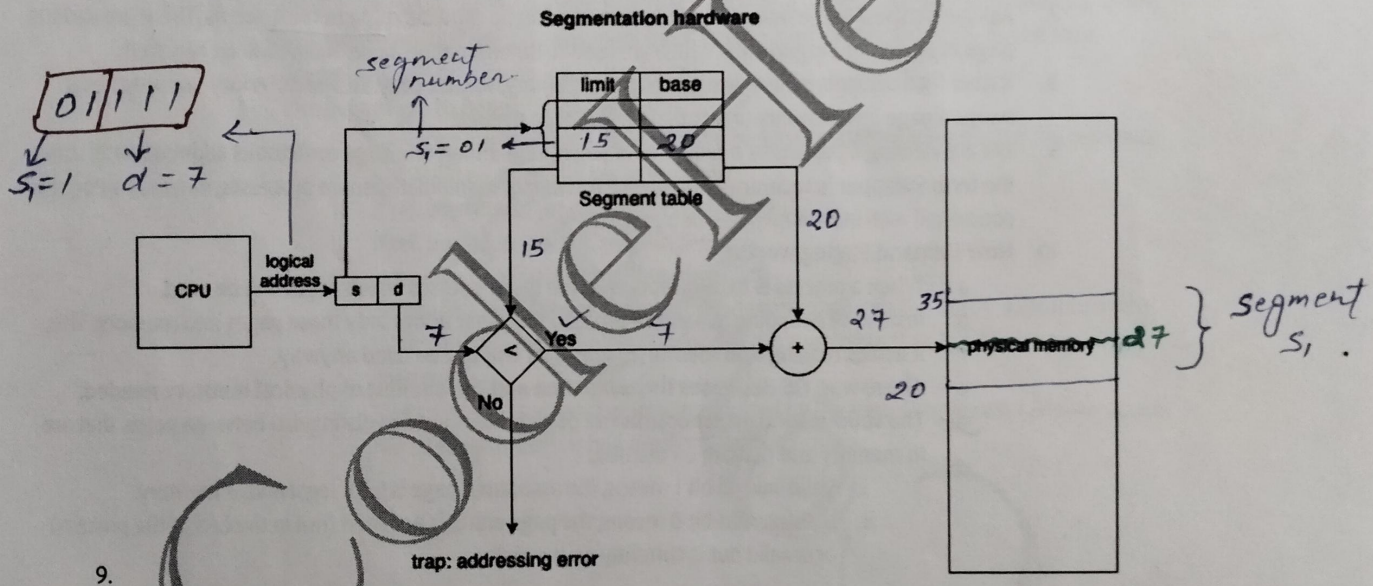
→ segment table.

S	limit	Base
—	~	~
~	~	~
—	~	~



LEC-27: Segmentation | Non-Contiguous Memory Allocation

1. An important aspect of memory management that become unavoidable with paging is separation of user's view of memory from the actual physical memory.
2. Segmentation is memory management technique that supports the **user view of memory.**
3. A logical address space is a collection of segments, these segments are based on **user view** of logical memory.
4. Each segment has **segment number and offset,** defining a segment.
   <segment-number, offset> {s,d}
5. Process is divided into **variable segments based on user view.**
6. **Paging** is closer to the Operating system rather than the **User**. It divides all the processes into the form of pages although a process can have some relative parts of functions which need to be loaded in the same page.
7. Operating system doesn't care about the **User's view** of the process. It may **divide the same function into different pages** and those **pages may or may not be loaded at the same time into the memory.** It decreases the efficiency of the system.
8. It is better to have segmentation which divides the process into the segments. Each segment contains the same type of functions such as the main function can be included in one segment and the library functions can be included in the other segment.

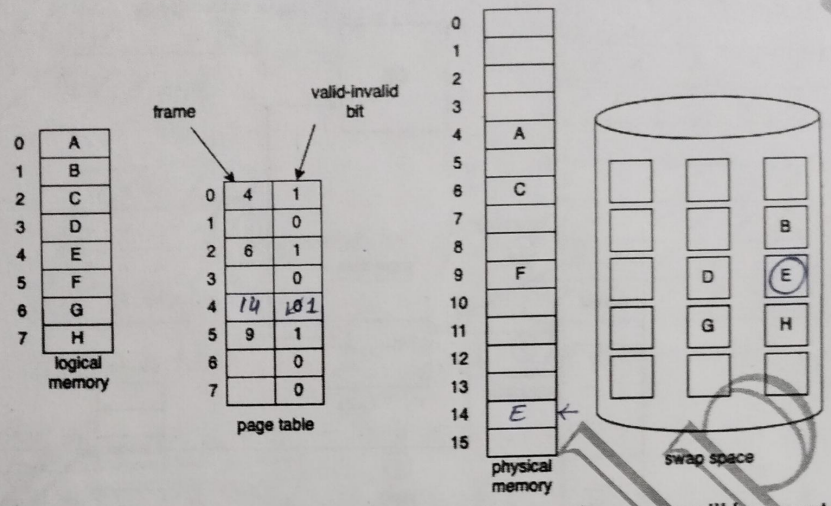


9.
10. **Advantages:**
    a. No internal fragmentation.
    b. One segment has a contiguous allocation, hence efficient working within segment.
    c. The size of segment table is generally less than the size of page table.
    d. It results in a more efficient system because the compiler keeps the same type of functions in one segment.
11. **Disadvantages:**
    a. External fragmentation.
    b. The different size of segment is not good that the time of swapping.
12. Modern System architecture provides both segmentation and paging implemented in some hybrid approach.

LEC-28: What is Virtual Memory? || Demand Paging || Page Faults

1. **Virtual memory** is a technique that allows the execution of processes that are not completely in the memory. It provides user an illusion of having a very big main memory. This is done by treating a part of secondary memory as the main memory. (Swap-space)
2. **Advantage** of this is, programs can be larger than physical memory.
3. It is required that instructions must be in physical memory to be executed. But it limits the size of a program to the size of physical memory. In fact, in many cases, the entire program is not needed at the same time. So, we want an ability to execute a program that is only partially in memory would give many benefits:
    a. A program would no longer be constrained by the amount of physical memory that is available.
    b. Because each user program could take less physical memory, more programs could be run at the same time, with a corresponding **increase in CPU utilization and throughput.**
    c. Running a program that is not entirely in memory would benefit **both the system and the user.**
4. Programmer is provided very large virtual memory when only a smaller physical memory is available.
5. **Demand Paging** is a popular method of **virtual memory management.**
6. In demand paging, the pages of a process which are least used, get stored in the secondary memory.
7. A page is copied to the main memory when its demand is made, or **page fault** occurs. There are various **page replacement algorithms** which are used to determine the pages which will be replaced.
8. Rather than swapping the entire process into memory, we use **Lazy Swapper.** A lazy swapper never swaps a page into memory unless that page will be needed.
9. We are viewing a process as a sequence of pages, rather than one large contiguous address space, using the term **Swapper is technically incorrect.** A swapper manipulates entire processes, whereas a **Pager** is concerned with individual pages of a process.
10. **How Demand Paging works?**
    a. When a process is to be swapped-in, the pager guesses which pages will be used.
    b. Instead of swapping in a whole process, the pager brings only those pages into memory. This, it avoids reading **into memory pages that will not be used anyway.**
    c. Above way, **OS decreases the swap time and the amount of physical memory needed.**
    d. The **valid-invalid bit scheme in the page table** is used to distinguish between pages that are in memory and that are on the disk.
        i. Valid-invalid bit **1** means, the associated page is both legal and in memory.
        ii. Valid-invalid bit **0** means, the page either is not valid (not in the LAS of the process) or is valid but is currently on the disk.

Page table when some pages are not in memory



e.

f. If a process never attempts **to access some invalid bit page, the process will be executed successfully without even the need pages present in the swap space.**

g. What happens if the process tries to access a page that was not brought into memory, access to a page marked invalid causes **page fault.** Paging hardware noticing invalid bit for a demanded page will cause **a trap to the OS.**

h. **The procedure to handle the page fault:**

i. Check an internal table (in PCB of the process) to determine whether the reference was valid or an invalid memory access.

ii. If ref. was invalid process throws exception.
If ref. is valid, pager will swap-in the page.

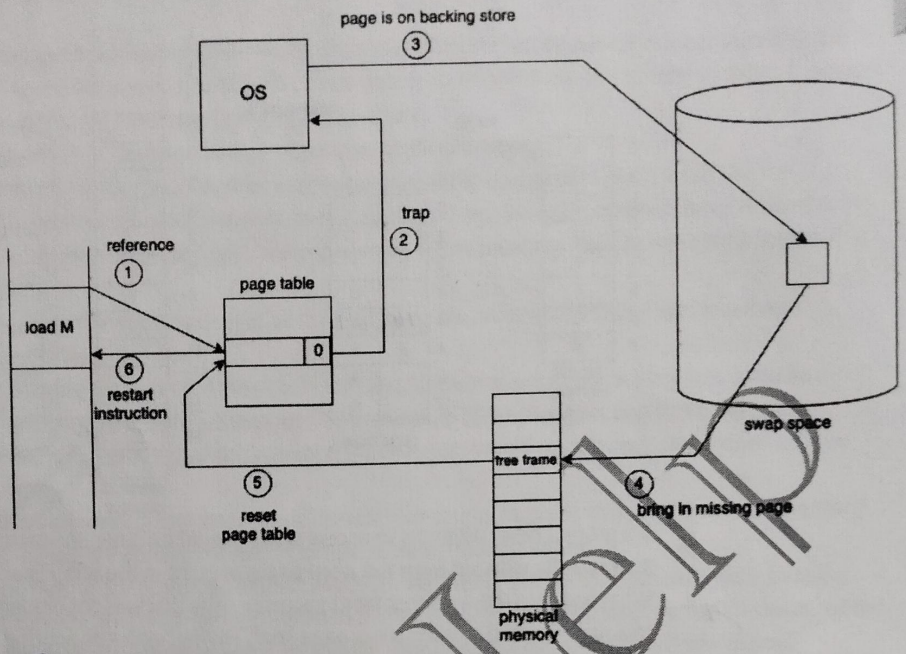
iii. We find a free frame (from free-frame list)

iv. Schedule a disk operation to read the desired page into the newly allocated frame.

v. When disk read is complete, we modify the page table that, the page is now in memory.

vi. Restart the instruction that was interrupted by the trap. The process can now access the page as through it had always been in memory.

Steps in handling a page fault



i.

j. **Pure Demand Paging**

   i. In extreme case, we can start executing a process with no pages in memory. When OS sets the instruction pointer to the first instruction of the process, which is not in the memory. The process immediately faults for the page and page is brought in the memory.

   ii. Never bring a page into memory until it is required.

k. We use **locality of reference** to bring out reasonable performance from demand paging.

11. **Advantages of Virtual memory**

   a. The degree of multi-programming will be increased.

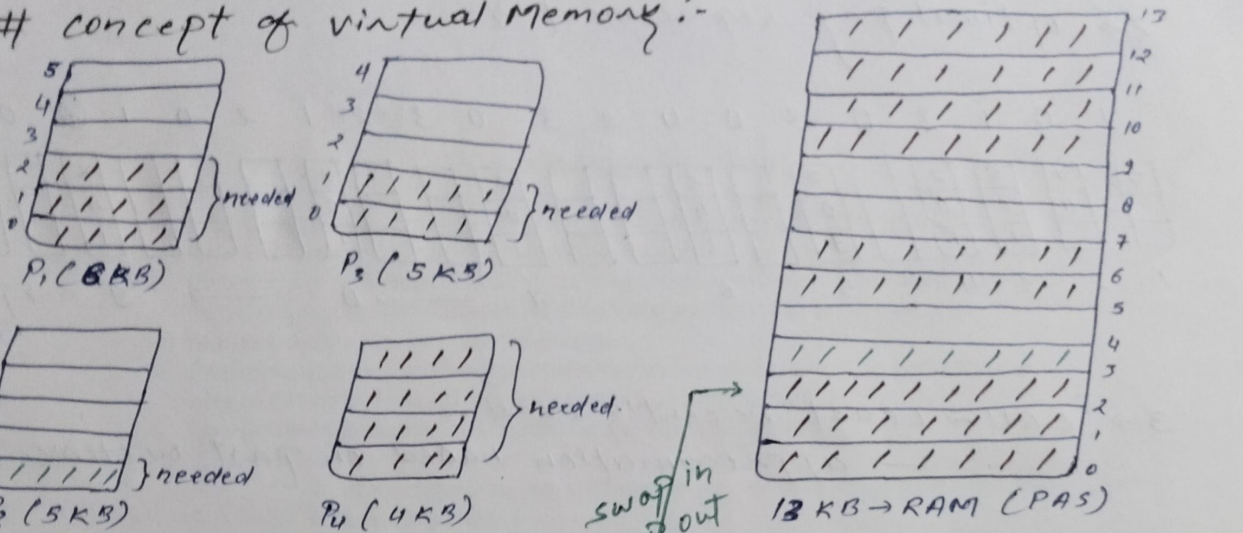
   b. User can run large apps with less real physical memory.

12. **Disadvantages of Virtual Memory**

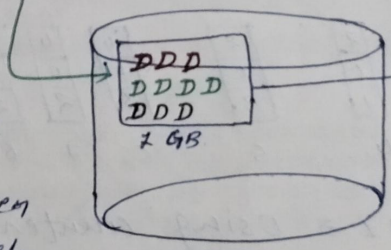
   a. The system can become slower as swapping takes time.

   b. **Thrashing** may occur.

concept of virtual Memory :-



* virtual Memory
  ↳ RAM + swap space



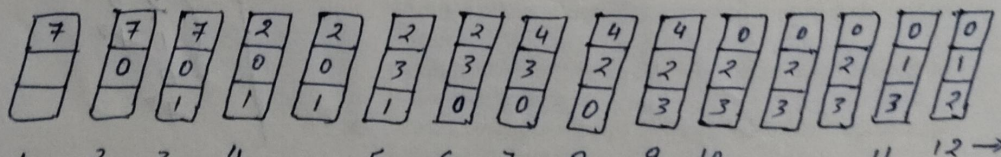
→ Page fault → when we require a new page but it is not available in RAM then OS generates an interrupt which is known as page fault.

→ The time require for swap out & swap in b/w RAM and swap space is called as page fault service time.

Page Replacement Algorithms →

(1) FIFO → given → RAM → 3KB → 3 frames
Reference string.

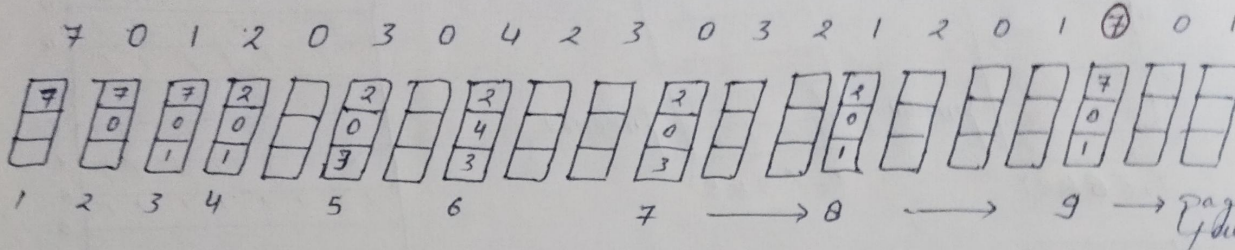
Ref. str. → 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2



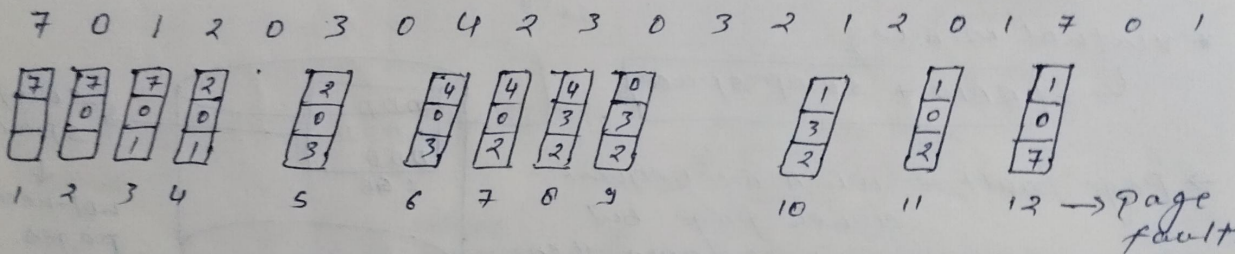
1 2 3 4 5 6 7 8 9 10 11 12 → Page faults.

→ Replacing the oldest page
→ easy to implement
→ performance not always good.

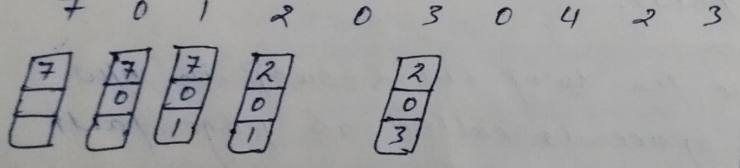
2→ optimal page Replacement →



3→ LRU → Least Recently Used →
→ approximation based on past history.



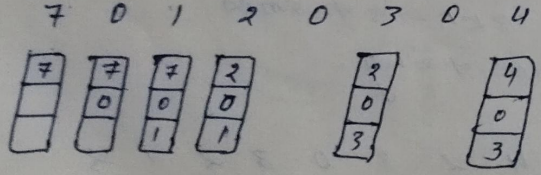
Method 1 → Using counters → Global shared count variable.

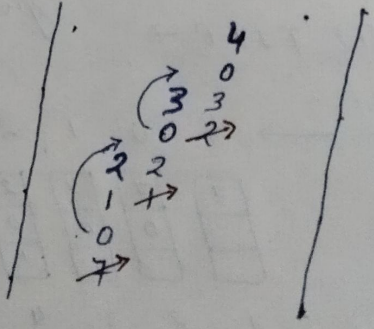


count → universal

0 → ~~2~~ 5
1 → ~~3~~
2 → 4
3 → 6
4 →
7 → ~~7~~

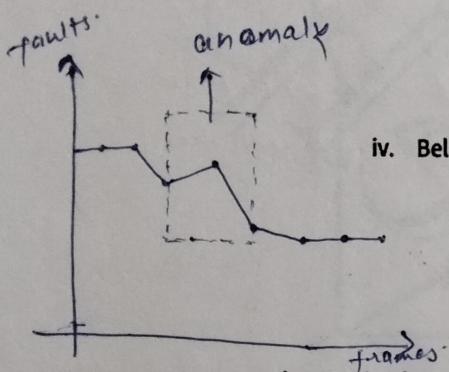
Method 2 → Using stack -





LEC-29: Page Replacement Algorithms

1. Whenever **Page Fault** occurs, that is, a process tries to access a page which is not currently present in a frame and OS must bring the page from swap-space to a frame.
2. OS must do page replacement to accommodate new page into a free frame, but there might be a possibility the system is working in high utilization and all the frames are busy, in that case OS must replace one of the pages allocated into some frame with the new page.
3. The **page replacement algorithm** decides which memory page is to be replaced. Some allocated page is swapped out from the frame and new page is swapped into the freed frame.
4. **Types** of Page Replacement Algorithm: (**AIM** is to have minimum page faults)
    a. **FIFO**
        i. Allocate frame to the page as it comes into the memory by **replacing the oldest page.**
        ii. Easy to implement.
        iii. Performance is **not** always good
            1. The page replaced may be an initialization module that was used long time ago **(Good replacement candidate)**
            2. The page may contain a heavily used variable that was initialized early and is in content use. **(Will again cause page fault)**
        iv. **Belady's** anomaly is present.
            1. **In the case of LRU and optimal page replacement algorithms, it is seen that the number of page faults will be reduced if we increase the number of frames.** However, Balady found that, In FIFO page replacement algorithm, the number of page faults will get increased with the increment in number of frames.
            2. This is the strange behavior shown by FIFO algorithm **in some of the cases.**



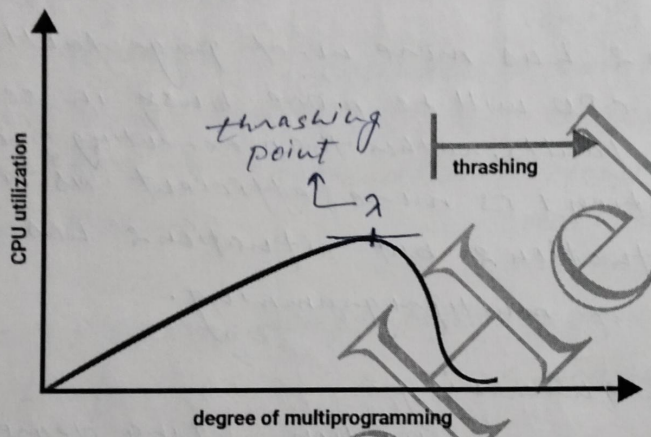
    b. **Optimal** page replacement
        i. Find if a page that is never referenced in future. If such a page exists, replace this page with new page.
        If no such page exists, find a page that is **referenced farthest in future**. Replace this page with new page.
        ii. **Lowest** page fault rate among any algorithm.
        iii. Difficult to implement as **OS requires future knowledge of reference string** which is kind of impossible. (Similar to SJF scheduling)
    c. Least-recently used (**LRU**)
        i. We can use recent past as an approximation of the near future then we can replace the page that has not been used for the longest period.
        ii. Can be implemented by two ways
            1. **Counters**
                a. Associate time field with each page table entry.
                b. Replace the page with smallest time value.
            2. **Stack**
                a. Keep a stack of page number.
                b. Whenever page is referenced, it is removed from the stack & put on the top.
                c. By this, most recently used is always on the top, & least recently used is always on the bottom.
                d. As entries might be removed from the middle of the stack, so Doubly linked list can be used.
    d. **Counting**-based page replacement – Keep a counter of the number of references that have been made to **each** page. (Reference counting)

i. Least frequently used (**LFU**)
    1. Actively used pages should have a large reference count.
    2. Replace page with the smallest count.
ii. Most frequently used (**MFU**)
    1. Based on the argument that the page with the smallest count was probably just brought in and has yet to be used.
iii. Neither MFU nor LFU replacement is common.

Topics → ① storage mgmt
② files & directories

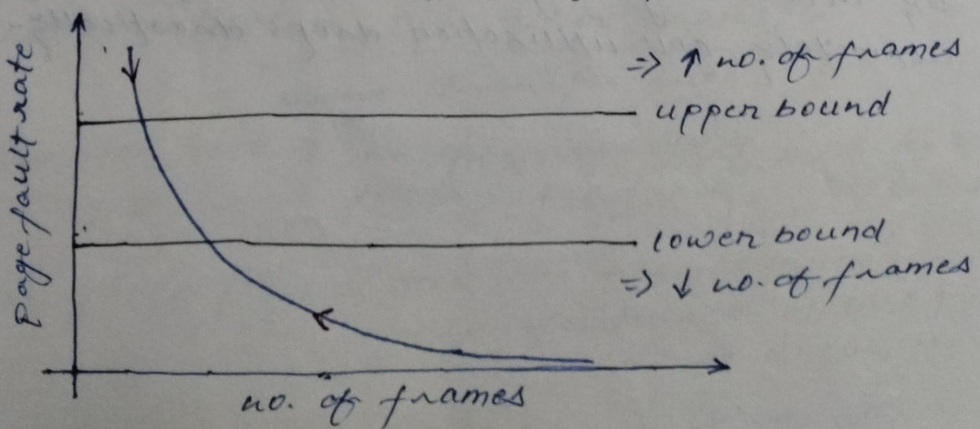
LEC-30: Thrashing

1. **Thrashing**
   a. If the process doesn't have the number of frames it needs to support pages in active use, it will quickly page-fault. At this point, it must replace some page. However, since all its pages are in active use, it must replace a page that will be needed again right away. Consequently, it quickly faults again, and again, and again, replacing pages that it must bring back in immediately.
   b. This **high paging activity is called Thrashing.**
   c. A system is Thrashing when it **spends more time servicing the page faults than executing processes.**



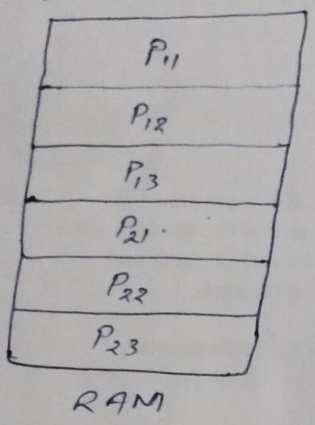
   d. **Technique to Handle Thrashing**
      i. **Working set model**
         1. This model is based on the concept of the **Locality** Model.
         2. The basic principle states that if we allocate enough frames to a process to accommodate its current locality, it will only fault whenever **it moves to some new locality.** But if the allocated frames are lesser than the size of the current locality, **the process is bound to thrash.**

      ii. **Page Fault frequency**
         1. **Thrashing** has a high page-fault rate.
         2. We want to **control** the page-fault rate.
         3. When it is too high, the process needs more frames. Conversely, if the page-fault rate is too low, then the process may have too many frames.
         4. We establish upper and lower bounds on the desired page fault rate.
         5. If pf-rate exceeds the upper limit, allocate the process another frame, if pf-rate fails falls below the lower limit, remove a frame from the process.
         6. By controlling pf-rate, thrashing can be prevented.



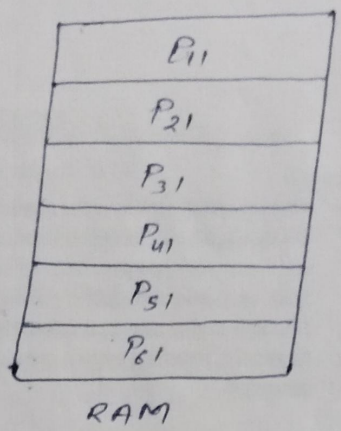
Thrashing :-

situation 1



$P_1 \rightarrow$ 3 pages
$P_2 \rightarrow$ 3 pages

situation 2



$P_1 - P_6 \rightarrow$ each has 1 page.

⇒ situation 2 has more no. of page faults.
↳ Hence CPU will be more busy in serving page faults, rather than executing processes.

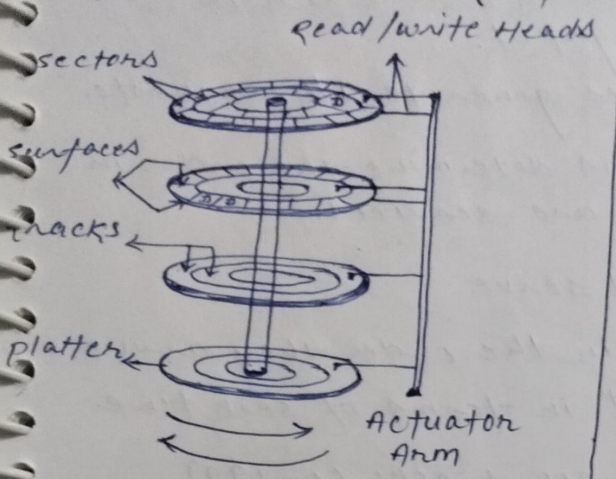
SO → situation 1 is more efficient as compared to situation 2 but situation 2 has more degree of multiprogramming.

→ causes of thrashing :-

(1) initial low CPU utilisation... ↑ing degree of multipro
(2) A global page replacement algorithm, replaces pages w/o regard to the process
(3) A process may need more frames... causes faults ago
(4) Other process's frames are replaced & they may need those soon.
(5) As a result CPU utilisation ↓es.
(6) CPU scheduler now, may increase CPU utilisation by increasing degree of multiprogramming.
(7) Ultimately, CPU utilization drops drastically.

Module → Disc structure and scheduling

#1 → Hard Disk Architecture →



platter → circular disk coated with magnetic material.

tracks → concentric circles on the surface of the platter.

sectors → subdivisions of tracks; smallest addressable units.

cylinders → same track number across all platters.

Read/write Head → Mounted on an actuator arm that moves across the platter.

Rotational speed :→ platters spin (measured in RPM)
↓
Rotation per minute

* platter → surface → tracks → sector → data

=> Disk size = P x S x T x S x D

Example → 8 x 2 x 256 x 512 x 512 KB

=> Disk size → 2^{30} KB or 1 TB

* 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 to desired sector → Half of rotation time.

(4) Transfer time → $\frac{\text{Data to be transfer}}{\text{transfer rate}}$

→ transfer Rate → (no. of surfaces / Heads) x capacity of one track x no. of rotations in one second

Disk Access Time = seek time + Rotation time + transfer time + control time + Queue time.

Disk Scheduling Algorithms :-

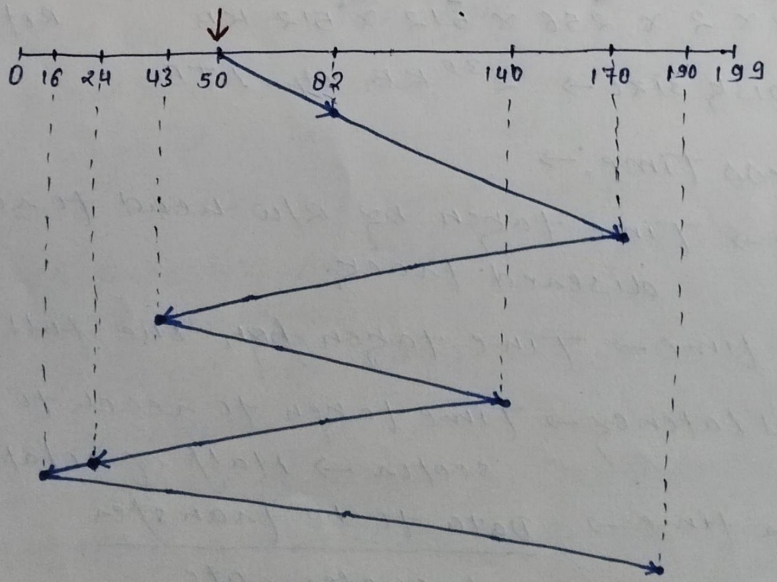
Goal → • To minimise the seek time (ST) to improve throughput.
• Multiple processes generate I/O requests.

→ Disk scheduling algorithms determine the order in which disk I/O requests are serviced.

(1) FCFS → First come First serve

↳ requests are served in the order they arrive.
↳ simple but inefficient in terms of seek time.

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



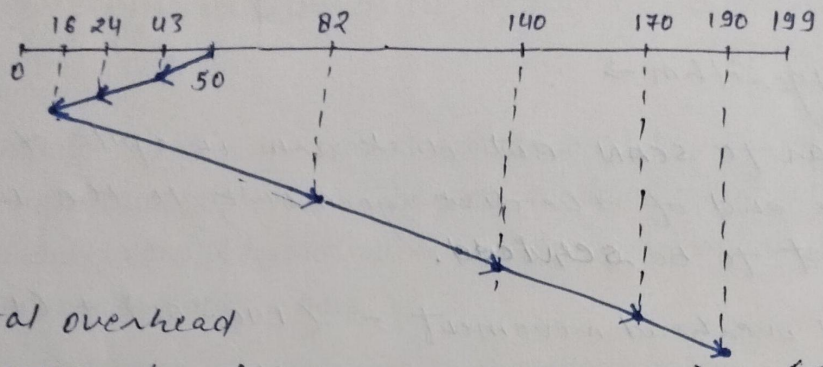
Total no. of tracks movement ⇒ (82-50) + (170-82) + (170-43) + (140-43) + (140-24) + (24-16) + (190-16)

= 642

• Advantage → no request starvation
• Disadvantage → Does not try to optimize seek time.

(2) SSTF → Shortest seek Time First

→ The seek time of every request is calculated in advance in the queue and then they are scheduled according to their calculated seek time.



total overhead movement ⇒ (50-43) + (43-24) + (24-16) + (82-16) + (140-82) + (170-140) + (190-170)

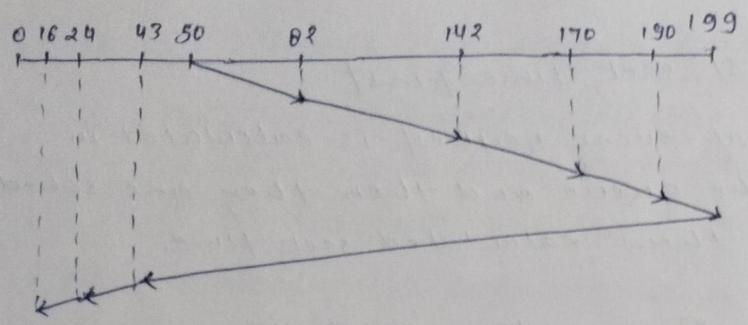
= 208

- Advantages → 1- Average response time decreases
  2- Throughput increases.
- Disadvantage → 1→ overhead to calculate seek time in advance.
  2- starvation

(3) SCAN → Elevator Algorithm

→ In this the disk arm moves in a particular direction & services the requests coming in its path & after reaching the end of the disk, it reverses its direction & again services the request arriving in its path.

- Advantages → ① High throughput ✓
  ② Average response time decreases ✓
  ③ low variance of response time.
- Disadvantage → ① long waiting time for requests for locations just visited by disk arm.



total overhead movement =
(199 - 50) +
(199 - 16)
= 332

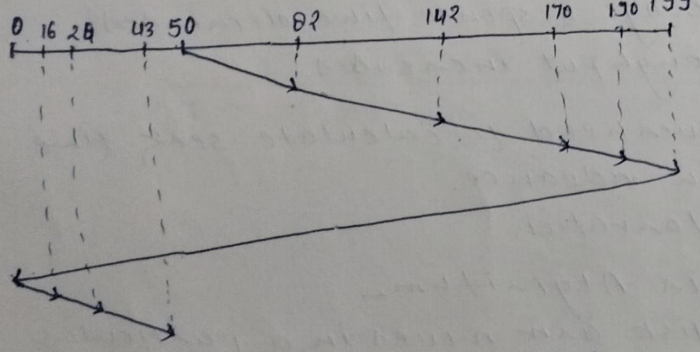
(4) LOOK Algorithm →

→ similar to SCAN but disk arm in spite of going to the end of the disk goes only to the last request to be serviced.

⇒ total overhead movement = (190 - 50) + (190 - 16)
= 314

(5) C-SCAN → circular SCAN

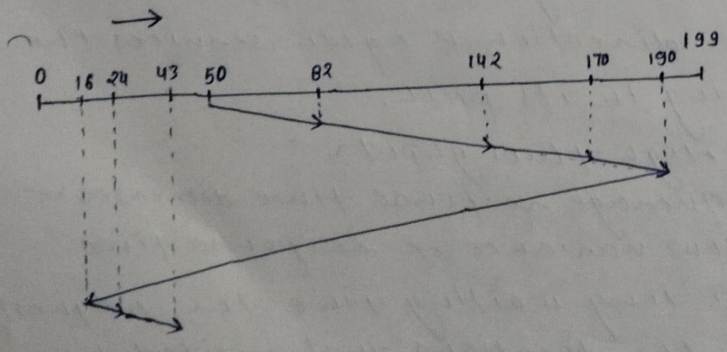
→ In this the disk arm instead of reversing its direction goes to the other end of the disk and starts servicing the requests from there.



total overhead movement =
(199 - 50) + (199 - 0)
+ (43 - 0) = 391

• Advantage →
Provides more uniform wait time compared to SCAN.

(6.) C-LOOK → circular LOOK



movement →
(190 - 50) + (190 - 16)
+ (43 - 16)
= 341

File Systems in Operating System

Last Updated : 14 Jan, 2025

A computer file is defined as a medium used for saving and managing data in the computer system. The data stored in the computer system is completely in digital format, although there can be various types of files that help us to store the data.

File systems are a crucial part of any operating system, providing a structured way to store, organize, and manage data on storage devices such as hard drives, SSDs, and USB drives. Essentially, a file system acts as a bridge between the operating system and the physical storage hardware, allowing users and applications to create, read, update, and delete files in an organized and efficient manner.

What is a File System?

A file system is a method an operating system uses to store, organize, and manage files and directories on a storage device. Some common types of file systems include:

- **FAT (File Allocation Table):** An older file system used by older versions of Windows and other operating systems.
- **NTFS (New Technology File System):** A modern file system used by Windows. It supports features such as file and folder permissions, compression, and encryption.
- **ext (Extended File System):** A file system commonly used on Linux and Unix-based operating systems.
- **HFS (Hierarchical File System):** A file system used by macOS.
- **APFS (Apple File System):** A new file system introduced by Apple for their Macs and iOS devices.

A file is a collection of related information that is recorded on secondary storage. Or file is a collection of logically related entities. From the user's perspective, a file is the smallest allotment of logical secondary storage.

The name of the file is divided into two parts as shown below:

- Name
- Extension, separated by a period.

Issues Handled By File System

We've seen a variety of data structures where the file could be kept. The file system's job is to keep the files organized in the best way possible.

A free space is created on the hard drive whenever a file is deleted from it. To reallocate them to other files, many of these spaces may need to be recovered. Choosing where to store the files on the hard disc is the main issue with files.one block may or may not be used to store a file. It may be kept in the disk's non-contiguous blocks. We must keep track of all the blocks where the files are partially located.

Files Attributes And Their Operations

Attributes	Types	Operations
Name	Doc	Create
Type	Exe	Open
Size	Jpg	Read
Creation Data	Xis	Write
Author	C	Append
Last Modified	Java	Truncate
protection	class	Delete

File Types and Their Content

File Type	Usual extension	Function
Executable	exe, com, bin	Read to run machine language program
Object	obj, o	Compiled, machine language not linked
Source Code	C, java, pas, asm, a	Source code in various languages
Batch	bat, sh	Commands to the command interpreter
Text	txt, doc	Textual data, documents
Word Processor	wp, tex, rrf, doc	Various word processor formats
Archive	arc, zip, tar	Related files grouped into one compressed file
Multimedia	mpeg, mov, rm	For containing audio/video information
Markup	xml, html, tex	It is the textual data and documents
Library	lib, a, so, dll	It contains libraries of routines for programmers
Print or View	gif, pdf, jpg	It is a format for printing or viewing an ASCII or binary file.

File Directories

The collection of files is a file directory. The directory contains information about the files, including attributes, location, and ownership. Much of this information, especially that is concerned with storage, is managed by the operating system. The directory is itself a file, accessible by various file management routines.

Below are information contained in a device directory.

- Name
- Type
- Address
- Current length
- Maximum length
- Date last accessed
- Date last updated
- Owner id
- Protection information

The operation performed on the directory are:

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

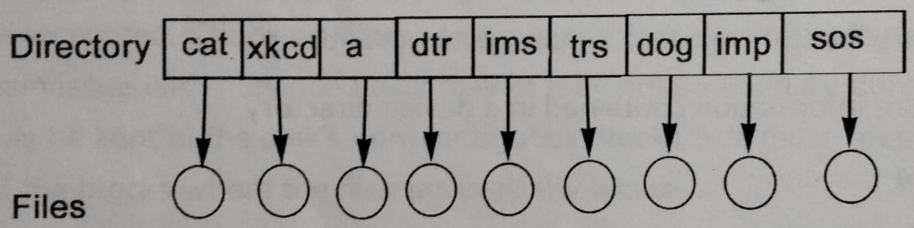
Advantages of Maintaining Directories

- **Efficiency:** A file can be located more quickly.
- **Naming:** It becomes convenient for users as two users can have same name for different files or may have different name for same file.
- **Grouping:** Logical grouping of files can be done by properties e.g. all java programs, all games etc.

Single-Level Directory

In this, a single directory is maintained for all the users.

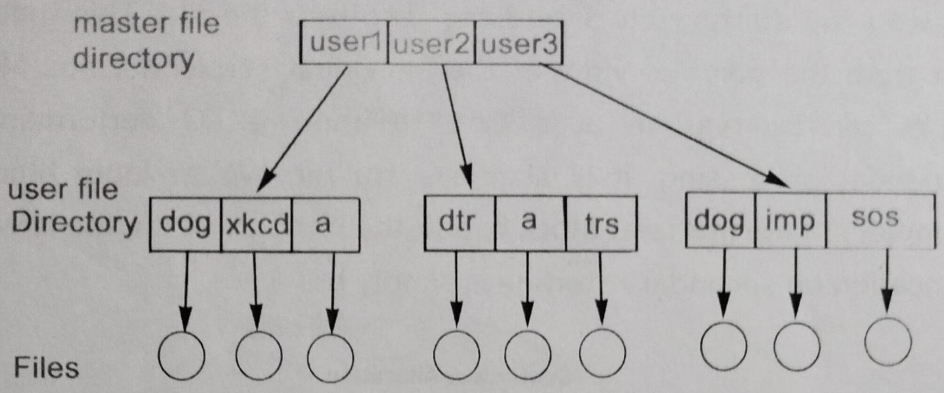
- **Naming Problem:** Users cannot have the same name for two files.
- **Grouping Problem:** Users cannot group files according to their needs.



Two-Level Directory

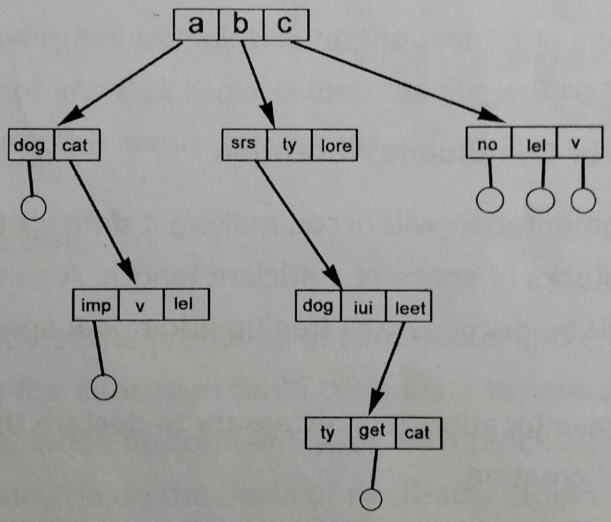
In this separate directories for each user is maintained.

- **Path Name**: Due to two levels there is a path name for every file to locate that file.
- Now, we can have the same file name for different users.
- Searching is efficient in this method.



Tree-Structured Directory

The directory is maintained in the form of a tree. Searching is efficient and also there is grouping capability. We have absolute or relative path name for a file.



File Allocation Methods

There are several types of file allocation methods. These are mentioned below.

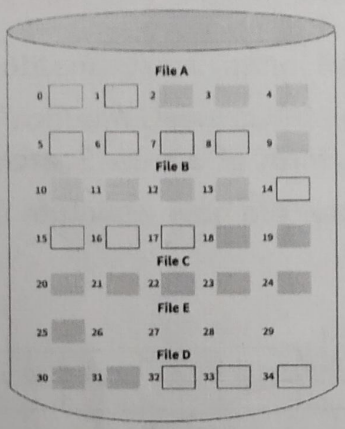
- Continuous Allocation
- Linked Allocation(Non-contiguous allocation)
- Indexed Allocation

Continuous Allocation

A single continuous set of blocks is allocated to a file at the time of file creation. Thus, this is a pre-allocation strategy, using variable size

portions. The file allocation table needs just a single entry for each file, showing the starting block and the length of the file. This method is best from the point of view of the individual sequential file. Multiple blocks can be read in at a time to improve I/O performance for sequential processing. It is also easy to retrieve a single block. For example, if a file starts at block b, and the i^{th} block of the file is wanted, its location on secondary storage is simply b+i-1.

Continuous Allocation



File Name	Star block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

Disadvantages of Continuous Allocation

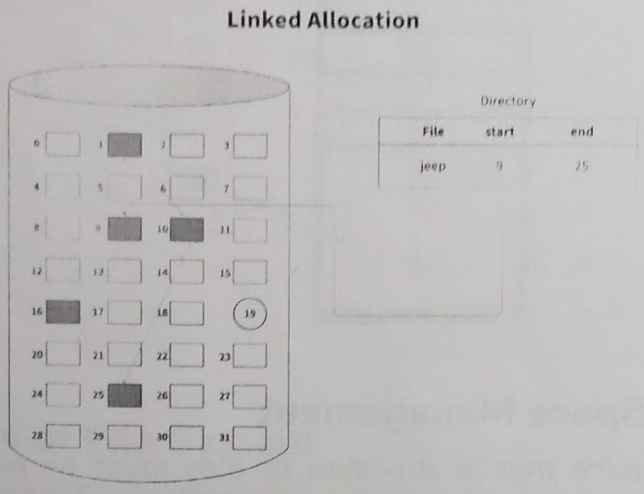
- External fragmentation will occur, making it difficult to find contiguous blocks of space of sufficient length. A compaction algorithm will be necessary to free up additional space on the disk.

- Also, with pre-allocation, it is necessary to declare the size of the file at the time of creation.

Linked Allocation(Non-Contiguous Allocation)

Allocation is on an individual block basis. Each block contains a pointer to the next block in the chain. Again the file table needs just a single entry for each file, showing the starting block and the length of the file. Although pre-allocation is possible, it is more common simply to allocate blocks as needed. Any free block can be added to the chain. The blocks need not be continuous. An increase in file size is always possible if a free disk block is available. There is no external fragmentation because only one block at a time is needed but there can

be internal fragmentation but it exists only in the last disk block of the file.



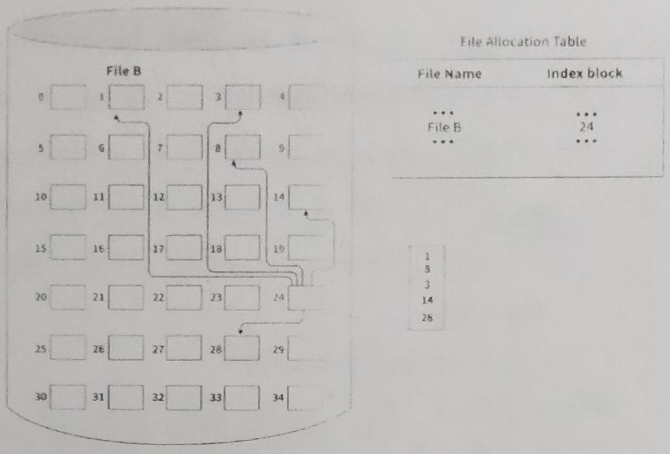
Disadvantage Linked Allocation(Non-contiguous allocation)

- Internal fragmentation exists in the last disk block of the file.
- There is an overhead of maintaining the pointer in every disk block.
- If the pointer of any disk block is lost, the file will be truncated.
- It supports only the sequential access of files.

Indexed Allocation

It addresses many of the problems of contiguous and chained allocation. In this case, the file allocation table contains a separate one-level index for each file: The index has one entry for each block allocated to the file. The allocation may be on the basis of fixed-size blocks or variable-sized blocks. Allocation by blocks eliminates external fragmentation, whereas allocation by variable-size blocks improves locality. This allocation technique supports both sequential and direct access to the file and thus is the most popular form of file allocation.

Indexed Allocation

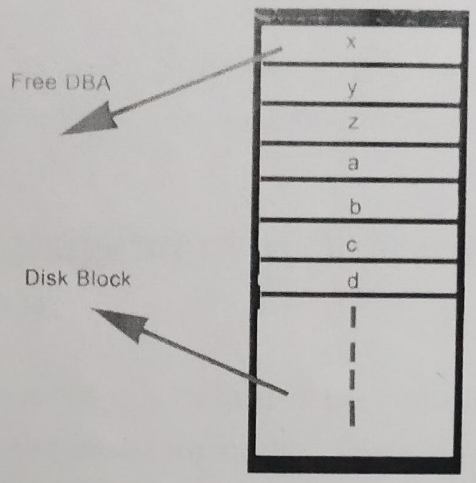


Disk Free Space Management

Just as the space that is allocated to files must be managed, so the space that is not currently allocated to any file must be managed. To perform any of the file allocation techniques, it is necessary to know what blocks on the disk are available. Thus we need a disk allocation table in addition to a file allocation table. The following are the approaches used for free space management.

- **Bit Tables**: This method uses a vector containing one bit for each block on the disk. Each entry for a 0 corresponds to a free block and each 1 corresponds to a block in use.
  For example 00011010111100110001
  In this vector every bit corresponds to a particular block and 0 implies that that particular block is free and 1 implies that the block is already occupied. A bit table has the advantage that it is relatively easy to find one or a contiguous group of free blocks. Thus, a bit table works well with any of the file allocation methods. Another advantage is that it is as small as possible.

- **Free Block List**: In this method, each block is assigned a number sequentially and the list of the numbers of all free blocks is maintained in a reserved block of the disk.



Advantages of File System

- **Organization:** A file system allows files to be organized into directories and subdirectories, making it easier to manage and locate files.

- **Data Protection:** File systems often include features such as file and folder permissions, backup and restore, and error detection and correction, to protect data from loss or corruption.

- **Improved Performance:** A well-designed file system can improve the performance of reading and writing data by organizing it efficiently on disk.

Disadvantages of File System

- **Compatibility Issues:** Different file systems may not be compatible with each other, making it difficult to transfer data between different operating systems.

- **Disk Space Overhead:** File systems may use some disk space to store metadata and other overhead information, reducing the amount of space available for user data.

- **Vulnerability:** File systems can be vulnerable to data corruption, malware, and other security threats, which can compromise the stability and security of the system.

Free Space Management in Operating System

Last Updated : 16 Aug, 2024

Free space management is a critical aspect of operating systems as it involves managing the available storage space on the hard disk or other secondary storage devices. The operating system uses various techniques to manage free space and optimize the use of storage devices. Here are some of the commonly used free space management techniques:

Free Space Management Techniques

- **Linked Allocation:** In this technique, each file is represented by a linked list of disk blocks. When a file is created, the operating system finds enough free space on the disk and links the blocks of the file to form a chain. This method is simple to implement but can lead to fragmentation and waste of space.
- **Contiguous Allocation:** In this technique, each file is stored as a contiguous block of disk space. When a file is created, the operating system finds a contiguous block of free space and assigns it to the file. This method is efficient as it minimizes fragmentation but suffers from the problem of external fragmentation.
- **Indexed Allocation:** In this technique, a separate index block is used to store the addresses of all the disk blocks that make up a file. When a file is created, the operating system creates an index block and stores the addresses of all the blocks in the file. This method is efficient in terms of storage space and minimizes fragmentation.
- **File Allocation Table (FAT):** In this technique, the operating system uses a file allocation table to keep track of the location of each file on the disk. When a file is created, the operating system updates the file allocation table with the address of the disk blocks that make up the file. This method is widely used in Microsoft Windows operating systems.

- **Volume Shadow Copy:** This is a technology used in Microsoft Windows operating systems to create backup copies of files or entire volumes. When a file is modified, the operating system creates a shadow copy of the file and stores it in a separate location. This method is useful for data recovery and protection against accidental file deletion.

Overall, free space management is a crucial function of operating systems, as it ensures that storage devices are utilized efficiently and effectively.

The system keeps tracks of the free disk blocks for allocating space to files when they are created. Also, to reuse the space released from deleting the files, free space management becomes crucial. The system maintains a free space list which keeps track of the disk blocks that are not allocated to some file or directory. The free space list can be implemented mainly as:

1. Bitmap or Bit vector

A Bitmap or Bit Vector is series or collection of bits where each bit corresponds to a disk block. The bit can take two values: 0 and 1: *0 indicates that the block is free* and 1 indicates an allocated block. The given instance of disk blocks on the disk in *Figure 1* (where green blocks are allocated) can be represented by a bitmap of 16 bits as: **1111000111111001**.

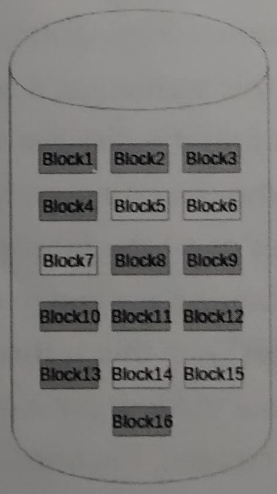


Figure - 1

- Simple to understand.
- Finding the first free block is efficient. It requires scanning the words (a group of 8 bits) in a bitmap for a non-zero word. (A 0-valued word has all bits 0). The first free block is then found by scanning for the first 1 bit in the non-zero word.

Disadvantages:

- For finding a free block, Operating System needs to iterate all the blocks which is time consuming.
- The efficiency of this method reduces as the disk size increases.

2. Linked List

In this approach, the free disk blocks are linked together i.e. a free block contains a pointer to the next free block. The block number of the very first disk block is stored at a separate location on disk and is also cached in memory.

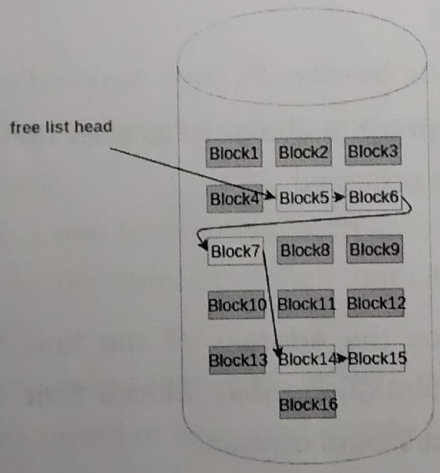


Figure - 2

In *Figure-2*, the free space list head points to Block 5 which points to Block 6, the next free block and so on. The last free block would contain a null pointer indicating the end of free list. A drawback of this method is the I/O required for free space list traversal.

Advantages:

- The total available space is used efficiently using this method.
- Dynamic allocation in Linked List is easy, thus can add the space as per the requirement dynamically.

Disadvantages:

- When the size of Linked List increases, the headache of miniating pointers is also increases.
- This method is not efficient during iteration of each block of memory.

Grouping

This approach stores the address of the free blocks in the first free block. The first free block stores the address of some, say n free blocks. Out of these n blocks, the first n-1 blocks are actually free and the last block contains the address of next free n blocks. An **advantage** of this approach is that the addresses of a group of free disk blocks can be found easily.

Advantage:

- Finding free blocks in massive amount can be done easily using this method.

Disadvantage:

- The only disadvantage is, we need to alter the entire list, if any of the block of the list is occupied.

Counting

This approach stores the address of the first free disk block and a number n of free contiguous disk blocks that follow the first block. Every entry in the list would contain:

- Address of first free disk block.
- A number n.

Advantages:

- Using this method, a group of entire free blocks can take place easily and Fastly.
- The list formed in this method is especially smaller in size.

Disadvantage:

- The first free block in this method, keeps account of other free blocks. Thus, due to that one block the space requirement is more.

Advantages of Free Space Management Techniques

- **Efficient Use of Storage Space:** Free space management techniques help to optimize the use of storage space on the hard disk or other secondary storage devices.
- **Easy to Implement:** Some techniques, such as linked allocation, are simple to implement and require less overhead in terms of processing and memory resources.
- **Faster Access to Files**: Techniques such as contiguous allocation can help to reduce disk fragmentation and improve access time to files.

Disadvantages of Free Space Management Techniques

- **Fragmentation:** Techniques such as linked allocation can lead to fragmentation of disk space, which can decrease the efficiency of storage devices.
- **Overhead**: Some techniques, such as indexed allocation, require additional overhead in terms of memory and processing resources to maintain index blocks.
- **Limited scalability**: Some techniques, such as FAT, have limited scalability in terms of the number of files that can be stored on the disk.
- **Risk of data loss**: In some cases, such as with contiguous allocation, if a file becomes corrupted or damaged, it may be difficult to recover the data.
- Overall, the choice of free space management technique depends on the specific requirements of the operating system and the storage devices being used. While some techniques may offer advantages in terms of efficiency and speed, they may also have limitations and drawbacks that need to be considered.

Conclusion

Free space management in an operating system ensures that storage is efficiently utilized and accessible for new data. It involves tracking

Path Name in File Directory

Last Updated : 05 May, 2023

Prerequisite – File Systems Hierarchical Directory Systems – Directory is maintained in the form of a tree. Each user can have as many directories as are needed so, that files can be grouped together in a natural way. Advantages of this structure:

- Searching is efficient
- Groping capability of files increase

When the file system is organized as a directory tree, some way is needed for specifying file names. Two different methods are commonly used:

1. **Absolute Path name** – In this method, each file is given an **absolute path** name consisting of the path from the root directory to the file. As an example, the path **/usr/ast/mailbox** means that the root directory contains a subdirectory usr, which in turn contains a subdirectory ast, which contains the file mailbox. Absolute path names always start at the root directory and are unique. In UNIX the components of the path are separated by '/'. In Windows, the separator is '\'. **Windows** \usr\ast\mailbox **UNIX** /usr/ast/mailbox
2. **Relative Path name** – This is used in conjunction with the concept of the **working directory** (also called the **current directory**).A user can designate one directory as the current working directory, in which case all path names not beginning at the root directory are taken relative to the working directory. For **example**, if the current working directory is /usr/ast, then the file whose absolute path is /usr/ast/mailbox can be referenced simply as mailbox. In other words, the UNIX command : **cp /usr/ast/mailbox /usr/ast/mailbox.bak** and the command : **cp mailbox mailbox.bak** do exactly the same thing if the working directory is /usr/ast.

When to use which approach? Some programs need to access a specific file without regard to what the working directory is. In that case, they should always use absolute path names. For example, a spelling checker might need to read /usr/lib/dictionary to do its work. It should use the full, absolute path name in this case because it does not know what the working directory will be when it is called. The absolute path name will always work, no matter what the working directory is. Of course, if the spelling checker needs a large number of files from /usr/lib, an alternative approach is for it to issue a system call to change its working directory to /usr/lib, and then use just dictionary as the first parameter to open. By explicitly changing the working directory, it knows for sure where it is in the directory tree, so it can then use relative paths.

Why would you use an absolute path name instead of a relative path name?

Absolute path names provide an unambiguous way to refer to a file or directory regardless of the current working directory. This is useful when you need to specify a file or directory location in a command or script that will be run in different working directories. It is also useful when referring to files or directories that are located in the same location on different machines.

Next Article >
Java | Print root directories

Similar Reads

Java | Print root directories

In Java we can use listRoots() of File class to find all root directories. import java.io.*; public class GeeksforGeeks { public static void...

What exactly Spooling is all about?

Last Updated : 20 Apr, 2023

Introduction

SPOOL is an acronym for **simultaneous peripheral operations on-line**. It is a kind of buffering mechanism or a process in which data is temporarily held to be used and executed by a device, program or the system. Data is sent to and stored in memory or other volatile storage until the program or computer requests it for execution.

Need of Spooling

Peripheral equipment such as printers and punch card readers are often slow relative to the performance of the rest of the system, creating a bottleneck in the I/O process. This is where spooling comes in. Spooling resolves this by accumulating data, instructions, and processes from multiple sources in a request queue, which is then processed in a first-in, first-out (FIFO) manner.

How Does Spooling Work:

Spooling can be implemented using the computer's physical memory, buffers, or device-specific interrupts. The most common applications of spooling are found in I/O devices like keyboards, printers, and mice. For example, when a document is sent to a printer, it is first stored in the printer's spooler or memory. When the printer is ready, it fetches the data from the spool and prints it.

Batch processing systems also use spooling to maintain a queue of ready-to-run jobs, which can be started as soon as the system has the resources to process them. Additionally, spooling is capable of overlapping I/O operation for one job with processor operations for another job, allowing multiple processes to write documents to a print queue without waiting and resume with their work.

Applications/Implementations of Spool:

1) The most common can be found in I/O devices like keyboard printers and mouse. For example, In printer, the documents/files that are sent to the printer are first stored in the memory or the printer spooler. Once the printer is ready, it fetches the data from the spool and prints it.

Ever experienced a situation when suddenly for some seconds your mouse or keyboard stops working? Meanwhile, we usually click again and again here and there on the screen to check if its working or not. When it actually starts working, what and wherever we pressed during its hang state gets executed very fast because all the instructions got stored in the respective device's spool.

2) A batch processing system uses spooling to maintain a queue of ready-to-run jobs which can be started as soon as the system has the resources to process them.

3) Spooling is capable of overlapping I/O operation for one job with processor operations for another job. i.e. multiple processes can write documents to a print queue without waiting and resume with their work.

4) E-mail: an email is delivered by a MTA (Mail Transfer Agent) to a temporary storage area where it waits to be picked up by the MA (Mail User Agent)

5) Can also be used for generating Banner pages (these are the pages used in computerized printing in order to separate documents from each other and to identify e.g. the originator of the print request by username, an account number or a bin for pickup. Such pages are used in office environments where many people share the small number of available resources).

About the Author:

Ekta is a very active contributor on Geeksforgeeks. Currently studying at Delhi Technological University. She has also made a Chrome extension for www.geeksquiz.com to practice MCQs randomly. She can be reached at github.com/Ekta1994

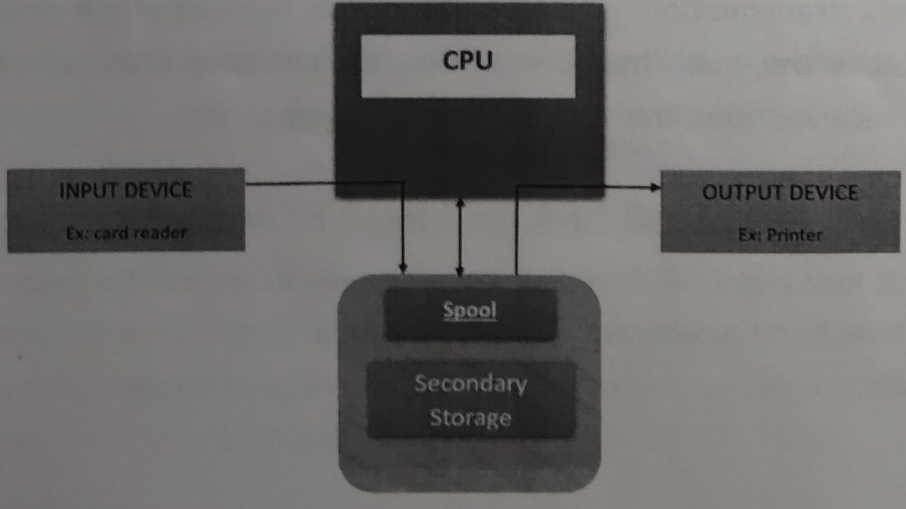
Difference between Spooling and Buffering

Last Updated : 04 Apr, 2025

In computing, efficient management is crucial when processing data or transmitting it. Two ways by which Input/output subsystems can improve the computer's performance and efficiency by using memory space in the main memory or on the disk are *spooling and buffering*. These are two techniques widely used to handle data between devices, processes, and memory. To optimize system performance, we need to understand their differences is essential.

Spooling

Spooling stands for **Simultaneous peripheral operation online**. It is a special process in a special area on disk where data is temporarily stored and queued for execution. A spool is similar to a buffer as it holds the jobs for a device until the device is ready to accept the job. It considers the disk as a huge buffer that can store as many jobs for the device till the output devices are ready to accept them. Multiple tasks are handled simultaneously by using this technique. It is commonly used in a scenario like printing, where documents are arranged or stored in a queue and sent to the printer sequentially.



Advantages of Spooling

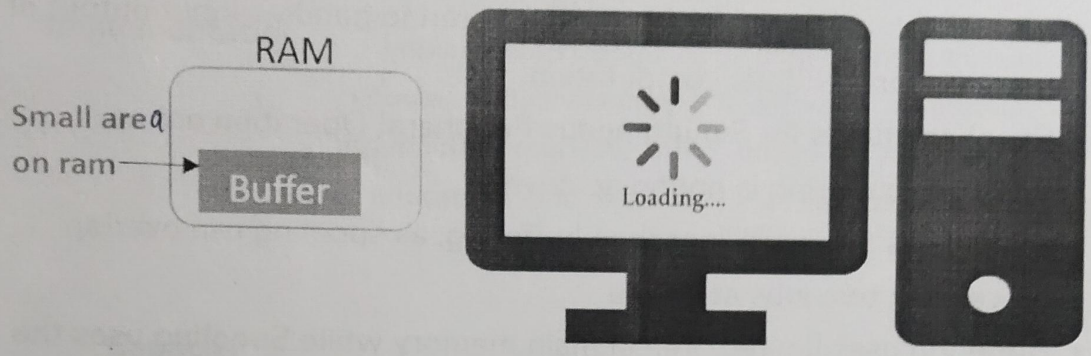
- **Management of Resources:** means when there are tasks in queue then the resources must be utilized fully without idle time.
- **Improved Efficiency:** Spooling helps to increase overall system throughput by allowing multiple jobs to be processed concurrently.
- **Data Integrity:** by queuing the tasks spooling helps the data to be processed in the correct sequence, reducing chances of errors.

Disadvantages of Spooling

- **Disk Space Usage:** In spooling we need to queue the data and for this disk space is required to store the queued data, and which can cause resource constraints in limited environments.
- **Delay in Processing:** If the number of jobs increases, and the system may not able to handle these jobs and the load efficiently then the tasks might experience delays.

Buffering

The main memory has an area called buffer that is used to store or hold the data temporarily that is being transmitted either between two devices or between a device or an application. Buffering is an act of storing data temporarily in the buffer. It helps in matching the speed of the data stream between the sender and the receiver. If the speed of the sender's transmission is slower than the receiver, then a buffer is created in the main memory of the receiver, and it accumulates the bytes received from the sender and vice versa.



buffering

Advantages of Buffering

- **Matching the Speed:** It accommodates speed differences between devices, also reduce the chances of bottlenecks by allowing smoother data transfer.
- **Minimized Latency:** In buffering we don't need to wait for the source and destination to catch up for the data to be processed or transmitted, it reduces latency.
- **Better User Experience:** In media streaming the data is preloaded, so that video is consistent while video is playing.

Disadvantage of Buffering

- **Memory Consumption:** It needs memory allocation and this can be a limitation for different systems as some systems are with limited resources as well.
- **Potential Data Loss:** In real-time application there is a chance of loosing data or corrupting of data if the buffer overflows.

Differences Between Spooling and Buffering

- The basic difference between Spooling and Buffering is that Spooling overlaps the input/output of one job with the execution of another job while the buffering overlaps the input/output of one job with the execution of the same job.

- The key difference between spooling and buffering is that Spooling can handle the input/output of one job along with the computation of another job at the same time while buffering handles input/output of one job along with its computation.
- Spooling stands for Simultaneous Peripheral Operation online. Whereas buffering is not an acronym.
- Spooling is more efficient than buffering, as spooling can overlap processing two jobs at a time.
- Buffering uses limited area in main memory while Spooling uses the disk as a huge buffer.

	Spooling	Buffering
Basic Difference	It overlap the input/output of one job with the execution of another job.	It overlaps the input/output of one job with the execution of the same job.
Full form (stands for)	Simultaneous peripheral operation online	No full form
Efficiency	Spooling is more efficient than buffering	buffering is less efficient than spooling.
Consider Size	It consider disk as a huge spool or buffer.	Buffer is limited area in main memory.
remote processing	It can process data at remote places.	It does not support remote processing.

	Spooling	Buffering
Implementation	Implemented using spoolers which manage input/output requests and allocate resources as needed	Implemented through software or hardware-based mechanisms such as circular buffers or FIFO queues
Capacity	Can handle large amounts of data since spooled data is stored on disk or other external storage	Limited by the size of memory available for buffering.
Error handling	Since data is stored on external storage, spooling can help recover from system crashes or other errors	Error can occur if buffer overflow happens, which can cause data loss or corruption.
Complexity	More complex than buffering since spooling requires additional software to manage input/output requests.	Less complex than spooling since buffering is a simpler technique for managing data transfer.

Next Article >
Difference between Buffer and Cache

Similar Reads

Operating System - Security

Security refers to providing a protection system to computer system resources such as CPU, memory, disk, software programs and most importantly data/information stored in the computer system. If a computer program is run by an unauthorized user, then he/she may cause severe damage to computer or data stored in it. So a computer system must be protected against unauthorized access, malicious access to system memory, viruses, worms etc. We're going to discuss following topics in this chapter.

- Authentication
- One Time passwords
- Program Threats
- System Threats
- Computer Security Classifications

Authentication

Authentication refers to identifying each user of the system and associating the executing programs with those users. It is the responsibility of the Operating System to create a protection system which ensures that a user who is running a particular program is authentic. Operating Systems generally identifies/authenticates users using following three ways −

- **Username / Password** – User need to enter a registered username and password with Operating system to login into the system.
- **User card/key** – User need to punch card in card slot, or enter key generated by key generator in option provided by operating system to login into the system.
- **User attribute - fingerprint/ eye retina pattern/ signature** – User need to pass his/her attribute via designated input device used by operating system to login into the system.

One Time passwords

One-time passwords provide additional security along with normal authentication. In One-Time Password system, a unique password is required every time user tries to login into the system. Once a one-time password is used, then it cannot be used again. One-time password are implemented in various ways.

- **Random numbers** – Users are provided cards having numbers printed along with corresponding alphabets. System asks for numbers corresponding to few alphabets randomly chosen.
- **Secret key** – User are provided a hardware device which can create a secret id mapped with user id. System asks for such secret id which is to be generated every time prior to login.
- **Network password** – Some commercial applications send one-time passwords to user on registered mobile/ email which is required to be entered prior to login.

Program Threats

Operating system's processes and kernel do the designated task as instructed. If a user program made these process do malicious tasks, then it is known as **Program Threats**. One of the common example of program threat is a program installed in a computer which can store and send user credentials via network to some hacker. Following is the list of some well-known program threats.

- **Trojan Horse** – Such program traps user login credentials and stores them to send to malicious user who can later on login to computer and can access system resources.
- **Trap Door** – If a program which is designed to work as required, have a security hole in its code and perform illegal action without knowledge of user then it is called to have a trap door.
- **Logic Bomb** – Logic bomb is a situation when a program misbehaves only when certain conditions met otherwise it works as a genuine program. It is harder to

detect.

- **Virus** – Virus as name suggest can replicate themselves on computer system. They are highly dangerous and can modify/delete user files, crash systems. A virus is generatlly a small code embedded in a program. As user accesses the program, the virus starts getting embedded in other files/ programs and can make system unusable for user

System Threats

System threats refers to misuse of system services and network connections to put user in trouble. System threats can be used to launch program threats on a complete network called as program attack. System threats creates such an environment that operating system resources/ user files are misused. Following is the list of some well-known system threats.

- **Worm** – Worm is a process which can choked down a system performance by using system resources to extreme levels. A Worm process generates its multiple copies where each copy uses system resources, prevents all other processes to get required resources. Worms processes can even shut down an entire network.
- **Port Scanning** – Port scanning is a mechanism or means by which a hacker can detects system vulnerabilities to make an attack on the system.
- **Denial of Service** – Denial of service attacks normally prevents user to make legitimate use of the system. For example, a user may not be able to use internet if denial of service attacks browser's content settings.

Computer Security Classifications

As per the U.S. Department of Defense Trusted Computer System's Evaluation Criteria there are four security classifications in computer systems: A, B, C, and D. This is widely used specifications to determine and model the security of systems and of security solutions. Following is the brief description of each classification.

S.N.	Classification Type & Description
1	**Type A** Highest Level. Uses formal design specifications and verification techniques. Grants a high degree of assurance of process security.
2	**Type B** Provides mandatory protection system. Have all the properties of a class C2 system. Attaches a sensitivity label to each object. It is of three types.

- **B1** – Maintains the security label of each object in the system. Label is used for making decisions to access control.
- **B2** – Extends the sensitivity labels to each system resource, such as storage objects, supports covert channels and auditing of events.
- **B3** – Allows creating lists or user groups for access-control to grant access or revoke access to a given named object.

Type C

Provides protection and user accountability using audit capabilities. It is of two types.

3

- **C1** – Incorporates controls so that users can protect their private information and keep other users from accidentally reading / deleting their data. UNIX versions are mostly CI class.
- **C2** – Adds an individual-level access control to the capabilities of a CI level system.

4

Type D

Lowest level. Minimum protection. MS-DOS, Window 3.1 fall in this category.