Batch Name : PreCAT OM24
Subject Name : Operating System Concepts

CCAT Section – B : 9 Questions
- Mostly all questions are concept oriented & GK of computer science/OS.

# DS DAY-01:

Q. Why there is need of an OS?

What is a Computer?
- Computer is a **hardware/machine/digital device**, which does different tasks efficiently and accurately for user.
- Computer hardware mainly contains:
1. Processor/CPU
2. Memory Devices
3. IO Devices.

Q. What are the basic functions of computer:
1. data storage
2. data processing
3. data movement
4. control

- there are different users:
1. human beings:
2. other machine
3. one computer may be user of another computer
server-client machines

- under human beings there are diff types of users:
1. end user : e.g. data entry operator

2. admin user: install OS, software, configure neteworks, createas & deleted user accounts on system etc....

3. programmer user/developers

- as any user cannot directly interacts with any of computer hardware components, and hence there is a need of some interface between user and hardware, to provide this interface is the job of an OS.

**Q. What is a software?**
**- collection programs**

**Q. What is a Program?**
Program is a finite set of instructions written in any programming langauge (either low level/high level) given to the machine to do specific task.
- there are 3 types of programs:
**1. system programs** – programs which are the part of an OS
inbuilt programs of an OS are called as system programs
e.g. kernel, cpu scheduler, loader, dispatcher, memory manager, device driver etc.....

**2. application programs:**
e.g. notepad, ms office, google chrome, calculator, games, vs code editor, eclipse, compiler, preprocessor, assembler etc....

C, C++, Java, Python .....

**3. user programs:**
e.g.  hello.c, addition.c, hello.java etc....

**As any user cannot directly interacts with an OS, and hence an OS provides two types of interfaces for user in the form of an application program – this program is referred as shell program.**

**- Shell is an application program through which user can interacts with an OS.**
**- there are two types of Shell programs:**
**1. CUI/CLI Shell: (Command User Interface/Command Line Interface)**
- in this type of interface user can interacts with an OS by means of entering commands in a text format through shell.
e.g.
- In Windows, name of the program which provides CUI => **command prompt => cmd.exe**
cls, cp, cd, mkdir etc....

- In Linux, name of the program which provides CUI => **shell/terminal => bash**
- In MSDOS, name of the program which provides CUI => **command prompt => command.com**

**2. GUI Shell : Graphical User Interface**
- in this type of interface user can interacts with an OS by means of making an events like click on buttons, exit button, min, max, menu bar, menu list, check list etc.....
- In Windows, name of the program which provides GUI => **explorer.exe**
- In Linux, name of the program which provides GUI => **GNOME/KDE**

**IDE ( Integrated Developement Environment)**

- An IDE is an **application software** which is a collection of tools ( i.e. application programs like an editor, preprocessor, compiler, assembler, debugger etc....), required for faster software developement.
e.g. vscode editor, eclipse, netbeans, android studio, turbo c, Visual Studio C/ C++ etc....

**source code** – program written in any programming language
source code editor ->

**#include<stdio.h>**
**- stdio.h is a header file which contains only declarations of standard input output library functions.**

**- preprocessor includes contents of header file into the source file**

**- declarations of lib functions are exists into the header files, whereas definitions of all library functions are exists into lib folder in precompiled object module format, which gets linked by the linker with program and final single executable code gets created.**

**Q. Why RAM is also called as Main Memory?**
- for an execution of any program RAM memory is must, hence RAM is also called as Main Memory.

**Q. What is Primary & Secondary Memory:**
- Memory which can be accessible directly by the CPU referred as primary memory:
e.g. CPU registers, cache memory, RAM, ROM etc....

Whereas memory which cannot  be accessible directly by the CPU referred as secondary memory
e.g. magetic disk, magnetic tape, PD, CD, DVD, optical disk etc...

- If CPU want to access hard disk drive contents first it gets loaded into the main memory and then CPU can access that data from RAM.

**Q. What is Program & Process?**
**Program :** is a passive entity which is into the HDD
**Process :** is an active entity which is into the RAM
- Program in execution is called as a process
- When a program gets loaded into the main memory it is referred as a process.

**Loader :** it is a **system program (i.e. inbuilt program of an OS)**, which loads an executable file from HDD into the Main Memory.

**Dispatcher :** it is a **system program (i.e. inbuilt program of an OS)**, which loads data & instructions of a program from the Main Memory onto the CPU (i.e. into the CPU registers).

**Scenario-1:**
**Machine-1: Windows => program.c**

**Machine-2: Linux => program.c => compiler + execute ==> YES**

**Portability :** program written in C on one machine/platform can compile and execute on any other machine/platform.

**Scenario-2:**
**Machine-1: Windows => program.c => compile + link => program.exe**

**Machine-2: Linux => program.exe => execute ?? ==> NO**

**In linux dot extension of any file doesnt matter.**

**Why ?**
File formats of an executable files in Linux and in Windows are different.

**Q. What is a file format ?**
It is a specific way of an OS to keep/store data and instructions of a program in an executable file in an organized manner.

**- In Linux file format of an executable file => ELF (Executable and Linkable Format).**
**- In Windows file format of an executable file => PE (Portable & Executable).**

**- When we run any program/executable file, loader first verifies file format, if file format matches then only it checks magic number which is inside header and if file format as well magic number both matches then only an execution of that program is started.**

**e.g. ELF**
- ELF file format dinvides an executable file logically into sections, and inside each section specific contents can be kept in an organized manner.
- there are 6 sections in ELF file format of an executable file:
**1. elf header/primary header/exe header:**
- it contains all the information which is required to starts an execution of a program
e.g.
**addr of entry point function** -> main( )
**magic number** => it is a constant number generated by the compiler which is file format specific.
e.g. In Linux, magic number of an executable file starts with **7fELF** in its eq hexa decimal format.

**2. bss section (block started by symbol):**
contains uninitialized static & global variables
e.g.
int g_var;//global variable
static int i;

**3. data section:**
contains initialized static & global variables
e.g.
int g_var=99;//global variable
static int i=100;

**4. rodata section (read only data section)**
it contains constants and string literals (constant strings)
e.g.
100
200L
0x25
020
'A'
"SunBeam"

**5. code/text section:**
it contains executable instructions

**6. symbol table:**
it contains information about functions and its variables in a tabular format.

e.g.
```
int addition( int n1, int n2)//n1 & n2 => format params
{
	int sum = 0;//sum => local variable
	sum = n1 + n2;
	return sum;
}
```

**Q. What is an OS?**
- an OS is a **system software** (i.e. collection of thounsands system programs) which acts as **an interface between user and hardware.**

- an OS also acts as **an interafce between programs (i.e. user & application programs) and hardware.**

- an OS allocates required resources like main memory, CPU time, IO devices access etc... to all running programs, hence it is also called as a **resource allocator.**

- an OS not only controls an execution of all programs, it also controls hardware devices which are connected to the computer system, and it is also called as a **control program.**

- an OS manages limited avaialble resources among all running program, it is also called a **resource manager.**

- an OS is a **Software** (i.e. collection of thousands of system programs and application programs which are in a **binary format**) comes either in a CD/DVD/PD and has main 3 components:
**1. Kernel :** it is **core program / core part of any OS** which runs continuosly into the main memory and does **basic minimal functionalities of an OS.**
- **Kernel is a like a heart of an OS.**
- Kernel is OS = OS is Kernel

- If Kernel program gets crashed => OS gets crashed
=> we need to install that OS again.
**e.g.**
**Windows => ntoskrnl.exe**
**Linux => vmlinuz**

**2. Utility Softwares**

**3. Application Softwares**

breathing => basic minimal functionality
teaching => extra utility functionality

---
speaking => basic minimal functionality

**extra utility functionalities:**
teaching
singing
to give speech

- **To install an OS** onto the machine is nothing but **to store thousands of system programs and application programs which are in a binary format comes either in a CD/DVD/PD into the HDD.**

- If any OS want to becomes active, atleast its core program i.e. kernel must be loaded into the main memory from HDD, and process to load kernel from HDD into the main memory is called as **booting.**

**- Booting is a process in which bootstrap program locates the kernel and load it into the main memory.**

# DS DAY-02:

**Q. What is a bootable device?**
- If any storage device contains one special program called as **bootstrap program** in its first sector i.e. in a boot sector then it is referred as a bootable device.
**e.g. HDD/CD/PD/DVD**

**- There are 2 steps of booting:**
**1. Machine Boot (H/W Booting):**
**Step-1:** When we switch on power supply, current gets passed to the motherboard, and one micro-program named as **BIOS (Basic Input Output System)** which exists inside ROM onto the motherboard gets invoked.

**Step-2:** first step of BIOS is **POST (i.e. Power On Self Test),** under POST, BIOS checks wheather peripheral devices are connnected properly or not and their working status.

**Step-3:** after POST, BIOS invokes **bootstrap loader program**, which searches for available **bootable devices** in a computer system and at a time it selects only one bootable device as per the priority decided in a BIOS settings. (bydefault it selects HDD as bootbale device).

**2. System Boot (OS Booting):**
**Step-4:** upon selection of HDD as a bootbale device, one program named as **bootloader program** gets invokes, which displays list of names of operating systems installed onto that machine, from which user has to select any one OS.
**Step-5:** After selection of any one OS from that list, **bootstrap program** of that OS gets invokes, which first locates the kernel and load it into the main memory.

**UNIX**
**MSDOS**
**Windows**
**Linux : Ubuntu/Fedora/RedHat**
**MAC OS X**
**Android**
**Solaris**
**Windows Server**
**Symbian**
.
.
.
.

Q. Why UNIX?
- OS Concepts UNIX:
UNIX (UNICS): Uniplexed Information & Computing Services/System.
- UNIX was invented at **AT&T Bell Labs** in US, in the deacade of 1970's by **Ken Thompson, Denies Ritchie** and team.
- First time UNIX was run on the machine **DEC-PDP-7** in 1969.
- C Programming Language was invented while developement of UNIX by denies ritchie in 1972.
C = B + BCPL
- In 1973, UNIX was rewritten in C
**UNIX = 10,000 ( 9000 C + 1000 Assembly langauage)**
- UNIX was basically designed for **developers by developer.**

- Windows OS was basically designed for end users
- Windows OS is having user friendly UI user interface.
- System Arch. Design of UNIX is followed in all modern OS's like Windows, Linux, Mac OS X, iOS, Android etc...., and hence UNIX is referred as mother of all modern OS's.

System Arch. Design of UNIX:
- Kernel i.e. core part of an OS acts as an interface between user programs and hardware.

# Operating System:
Process Control Subsystem:
File Subsystem
Hardware Control/Hardware Abstraction Layer
System Call Interface
etc...

- there are 2 major subsystems of an OS:
1. file subsystem
2. process control subsystem

- for any OS, file & process are very important concepts:
- UNIX => "file has space and process has life"
i.e. whatever that can be stored UNIX treats that as a file, wheras whatever is in a active state is considered as a process.
e.g.
UNIX treats all devices as a file
- In UNIX, devices can be catagorised into two catagories:
1. character devices – devices from which data gets transferes char by char i..e byte by byte
e.g. printer, keyboard, monitor etc....
- UNIX treats all character devices as a char special device files.

2. block devices - devices from which data gets transferes block by block i.e. sector by sector (usually size of 1 sector = 512 bytes).

e.g. all storage devices
When we copy data from HDD to PD, in this case an OS copies data from one block special device file to another block special device file
- UNIX treats all block devices as a block special device files.

- buffer cache maintained at an OS level => s/w concept/technique

- buffer cache => it is a portion of the main memory used by an OS, in which most recently accessed disk contents can be kept temporarily to get max throughput in min hardware movement.

Kernel => Core part / core program of an OS.

Program:
calculator functions:
main( ) : client function

services:
addition( )
substraction( )
multiplication( )
division(  )
etc...

- System Calls:
System calls are the functions of Kernel program defined in C, C++ & assembly langauge which provides interface of services made available by the kernel for user.

Kernel => Program
System Salls  => functions defined inside kernel program in C, C++ & assembly language.

- If any programmer user want to use services made available by the kernel in his/her program, then these services can used by means of giving call to system calls either directly in a system call programming or indirectly in C/C++ etc... through set of lib functions.

fopen( ) => open( ) sys call => to open a file or to create a new file
fwrite( )/printf( )/fprintf( )/fputs( )/fputc( ) => write( ) sys call => to write data into the file
fread( )/scanf()/fscanf()/fgetc/fgets( ) => read( ) sys call => to read data from the file
etc...

- In UNIX there are total 64 system calls
- In Linux there are total more than 300 system calls
- In Windows there are total more than 3000 system calls

- In UNIX name of system call to create a new process/child process => fork( )
- In Linux name of system call to create a new process/child process => fork( )/ clone( )
- In Windows name of system call to create a new process/child process => CreateProcess( )

- getpid( ) system call returns pid of calling process
pid - process id – unique identifier of a process.
- getppid( ) system call returns pid of parent process of calling process

- Irrespecitve of any OS, there are 6 catagories of system calls:
1. **file operations system calls:** e.g. open(), write(), read(), close() etc....
2. **device control system calls:** e.g. open(), write(), read(), close(), ioctl() etc....
3. **process control system calls:** e.g. fork(), _exit(), wait( ) etc....
4. **accouting information system calls:** e.g. getpid( ), getppid( ), stat( ) etc...
5. **inter process communication system calls :** e.g. signal(), pipe() etc...
6. **protection and security system calls :** e.g. chmod( ), chown( ) etc...

Q. What is an interrupt?
- an interrupt is a **signal** which is received by the CPU from any IO device due to which it stops an execution of one process/job and starts executing another job/program.
- interrupts sent by an IO devices are referred as **hardware interrupts.**

Q. Why system calls are also called as software interrupts/traps ?
- Whenever system call gets called, the CPU switched from user defined code to system defined code, and hence system calls are also called as software interrupts/traps.

```
//user defined program to do addition of two numbers:
//sum.c => user defined code

#include<stdio.h>

int main(void){
	//local vars definitions
	int n1, n2, sum;

	//executable instructions:
	printf("enter n1 & n2 : ");//write( ) sys call => system defined code
	scanf("%d %d", &n1, &n2);//read( ) sys call =>  system defined code

	sum = n1 + n2;

	printf("sum = %d\n", sum);
	return 0;
}
```

- throughout an execution of any user/application program, the CPU switches in between user defined code and system defined code, and hence we can say system runs in a two modes: **1. user mode 2. kernel mode**, and this mode of operation of system is referred as **dual mode operation.**

**1. user mode –** when the CPU executes user defined code instructions
**2. kernel mode –** when the CPU executes system defined code instructions

- the CPU can differentiate between user defined code instructions and system defined code instructions by means of referreing one bit which is onto the CPU referred as **mode bit** whose value will be maintained by an OS.
**mode bit = 0** => system mode/ kernel mode
**mode bit = 1** => user mode

**# Process Control Subsystem:**
- When we say an OS does **process management**, an OS is resposible for
    - process creation
    - to provide platform/environment for a program to complete its execution
    - to allocate resources
    - cpu scheduling
    - inter process communication
    - process synchronization
    etc...

**Q. What is a Program?**
**user point of view:**
- Program is a finite set of instructions written in programming language given to the machine to do specific task.

**system point of view: (Linux)**
- Program is an **executable file** which has specific file format i.e. for example in Linux executable file has got => **exe header, bss section, data section, rodata section , code/text section and symbol table.**

**Q. What is a Process?**
**user point of view:**
- Program in execution
- Running instance of a program
- When a program gets loaded into the main memory it becomes a process
- Program which is in the main memory is called as a process

**system point of view: (Linux)**
process is a program which has got one structure called as **PCB** into the main meomory inside kernel space and program has got **bss section, data section, rodata section, code section** and two new sections got added by an OS for it during runtime
**1. stack section:** it contains FAR's of called functions
**2. heap section:** it contains dynamically allocated memory (malloc(), calloc() or realloc( ) etc...)

**Kernel =>** it is a core program of an OS which runs continuosly into the main memory and does basic minimal functionalities of it.

- Kernel gets loaded into the main memory while booting and it remains active/prensent into the main memory till we do not shut down the system, and hence few portion of the main memory will always occupied by the kernel will be referred as a kernel space and whichever part is left other than kernel space is referred as user space into which only user programs can be loaded.
- Main memory is divided logically into two parts:
1. kernel space
2. user space

- When we execute any program, loader first verifies file format, if file format matches then it checks magic number and if file format as well as magic number both matches then only an execution of that program can be started i.e. process can be submitted.
- Upon process submission or when an execution of any program is started, very first an OS/kernel creates one **structure** for that process into the main memory inside kernel space, in which all the information which is required to complete an execution of that process can be kept, this structure is referred as **PCB (Process Control Block).**
- Loader copies bss section, data section, rodata section, code section as it is into the main memory and two new sections i.e. stack section and heap section gets added for that process during runtime.

Mainly PCB contains:
**- pid :** process id – unique identifier
**- ppid :** parent's pid
- PC: program counter – contains an addr of next instruction which is to be executed.
- memory management info
- cpu scheduling info
- info about resources allocated for that process
**- execution context** : if the CPU is currently executing any program, then info about instructions and data of that program can be kept temporarily into the CPU registers, collectively this information is referred as an execution context and copy of it also kept inside PCB of that process.
etc....

- An OS maintains **one PCB per process**, upon process submission PCB for a process gets created and when an execution of a program is completed PCB of it gets removed/destroyed from the main memory.

## # DS DAY-03:

- Linux & UNIX are two different OS, Linux is UNIX like OS.
- Linux & android are two different OS, Android is Linux based OS.
- Throughout an execution of any program / process it goes through different states and at a time it may exists / present only in one state. Current state of process gets stored into its PCB.

- There are 5 states of process:

**1. new state :** upon process submission it is considered in a new state.
i.e. when for a process PCB gets created into the main memory inside kernel space state of that process is considered as in a new state.
- if PCB of any process is in job queue then state of that process is considered in a new state.

**2. ready state :** if process is in the main memory and waiting for the CPU time i.e. ready to run, state of that process is considered as in a ready state.
- if PCB of any process is in ready queue then state of that process is considered in a ready state.

**3. running state :** if currently CPU is executing any process, state of that process is considered in a running state.

**4. waiting state :** if process is requesting for any io device, state of that process is considered in a waiting state.
- if PCB of any process is in waiting queue of any device then state of that process is considered in a waiting state.

**5. terminated state :** if process completes its execution i.e. upon exit it is considered in a terminated state.

- to keep track on all running programs i.e. on processes, an OS maintains few data structures referred as **kernel data structures:**
- there are 3 kernel data structures:

**1. job queue :** it contains list of PCB's of all submitted processes.

**2. ready queue :** it contains list of PCB's of all processes which are in the main memory and waiting for the CPU time i.e. ready to run.

**3. waiting queue :** it contains list of PCB's of processes which are waiting for that particular device.
- An OS maintains dedicated waiting queue for each device.

Addmission date : 25th July

Batch start date : 31st august

+ features of an OS:
1. multi-programming : system in which more than one processes can be submitted at a time OR system in which an execution of more than one programs can be started.
- degree of multi-programming => no. of programs that can be submitted into the system at a time.

2. multi-tasking : system in which the CPU can execute multiple processes concurrently/simultaneosly (i.e. one after another).

- the speed at which CPU executes multiple processes concurrently we feels / it seems that the CPU executes multiple processes at a time.
- the CPU can execute only one process at a time.

P1 = 40 MB
Q. What is a thread?
- thread is the smallest indivisibe part of a process
- thread is the smallet execution unit of a process
- if the CPU is executing any thread of any process => the CPU is executing that process.

3. multi-threading : system in which the CPU can execute multiple threads which are of either same process or are of different processes concurrently / simultaneosly (i.e. one after another).

- speed at which the CPU executes multiple threads of either same process or are of diff processes concurrently, we feels / it seems that the CPU executes multiple threads at a time.

- the CPU can execute only one thread of any one process at a time.

processor/CPU
uni-processor system -> system can run on such machine in which only one CPU is there.
e.g. MSDOS

4. multi-processor : system can run on such machine in which more than one CPU's are connected in a closed circuit.
e.g. Linux, Windows

5. multi-user : system in which more than one users can logged in at a time.

**1. job scheduler –** it is a system program which decides/schedules processes from job queue to load them onto the ready queue.
- job scheduler is also called as **long-term scheduler.**

**2. cpu scheduler -** it is a system program which decides/schedules process from ready queue to load it onto the CPU.
- job scheduler is also called as **short-term scheduler (for max cpu utilization cpu scheduler must be gets called frequently).**

**- there are total 5 cpu scheduling algorithms**

**- priority for process can be decided by two ways:**
**1. internally** priority for a process can be decided by an OS depends on no. of resources required for it.

**2. externally** priority for a process can be decided by the user, depends on requirement.

- priority of a process can be kept into its PCB.

- When an interrupt occurs the CPU switches from one process to another process => context switch
- during context-switch, the CPU swicthes from an execution context of one process into an execution context of another process.
- during context switch **state-save** of supended process takes place, and **state-restore** of shceduled process will takes place.
**- context switch = state-save (P1: suspended process) + state-restore (P2 : scheduled process).**

**- there are 4 cases in which CPU scheduler gets called:**
**case-1. running ==> terminated : due to an exit**
**case-2. running ==> waiting : due an i/o request**
**case-3. running ==> ready : due to an interrupt**
**case-4. waiting ==> ready : due an i/o request completion**

**- there are two types of CPU scheduling:**
**1. non-preemptive cpu scheduling :** in this type of cpu scheduling control of the CPU released by the process by its own i.e. voluntarily.
e.g. case-1 & case-2

**2. preemptive cpu scheduling :** in this type of cpu scheduling control of the CPU taken away forcefully from a process (i.e. process gets suspended)
e.g. case-3 & case-4

- there are basic 4 cpu scheduling algorithms:
**1. fcfs (first come first served) cpu scheduling algorithm**
**2. sjf (shortest job first) cpu scheduling algorithm**
**3. round robin cpu scheduling**
**4. priority cpu scheduling**

- as there are multiple cpu scheduling algorithms, so there is a need to decide which algo is best suited at which situation and which one is efficient, and to decide this there certain criterias referred as cpu scheduling criterias:
- there are 5 cpu scheduling criterias:
**1. cpu utilization :** one need to select such an algo in which utilization of the cpu must be as max as possible.

**2. throughput :** total work done per unit time
- one need to select such an algo in which throughput must be as max as possible.

**3. waiting time :** it is the total amount of time spent by the process into the ready queue for waiting to get control of the CPU from its time of submission.
- one need to select such an algo in which waiting time must be as min as possible.

**4. response time :** it is the time required for the process to get first response from the CPU from its time of submission.
- one need to select such an algo in which response time must be as min as possible.

**5. turn-around-time :** it is the total amount of time required for the process to complete its execution from its time of submission.
- one need to select such an algo in which turn-around-time must be as min as possible.

**turn-around-time = waiting time + execution time**

**execution time / running time :** it is the total amount of time spent by the process onto the CPU to complete its execution.

**- execution time is also called as cpu burst time :** total no. of cpu cycles required for the process to complete its execution.

**1. fcfs (first come first served) cpu scheduling algorithm:**
- in this algo, whichever process arrived first into the ready queue gets control of the CPU first i.e. control of the CPU gets allocated for processes as per their order of an arrival.

**gant chart** – it  is a bar chart representation of cpu allocation for processes in terms of cpu cycle numbers.

**Convoy effect :** in fcfs cpu scheduling, due to an arrival of longer process before shorter processes, shorter processes has to wait for longer duration due to which their waiting time gets increases and average waiting time also gets increases which results into increase in average turn-around-time, and overall system performance gets down.

- to ovecome problem of **convoy effect** in fcfs, sjf cpu scheduling has been designed.

**2. sjf (shortest job first) cpu scheduling algorithm**
- in this algo, process which is having smallest/shortest cpu burst time gets control of the CPU first.
- in sjf tie can be resolved by using fcfs

- sjf algo fails if arrival time of processes are different

**# DS DAY-04:**

- In SJF – process which is having min CPU burst time gets control of the CPU first, and hence longer process is considered as having very low priority i.e. shorter processes are having always highest priority

Ready Queue => out of 100 processes , P5 -> max CPU burst time

in a multi-programming system at a time multiple processes can be submitted and can keep on submitting during runtime as well

- starvation problem may occures in sjf cpu scheduling algo.

**starvation :** in sjf algo, due to having larger CPU burst time and processes are getting submitted into the ready queue during runtime, process which is having larger cpu burst time may gets blocked into the ready queue i.e. control of the CPU will never gets allocated for that process, this situation is called as **starvation / indifinite blocking.**

- to avoid probolem of starvation in sjf, **round robin algo** has been designed.

**3. round robin cpu scheduling algorithm:**
- in this algo, before allocating CPU for processes in advanced some fixed time quantum / time slice gets decided, and at a given time control of the CPU may remains allocated with any process max for that decided time slice, and once given time slice of any process is finished then it will be suspended and

control of the CPU will be given to the next process again max for decided time slice and so on.....
and if any process completes its execution before given time slice, then it will release the CPU for the next process.

- there is no starvation

**4. priority cpu scheduling algorithm**
- in this algo, process which is having highest priority gets control of the CPU first.
- priority of a process is stored into its PCB.
- priority for a process can be decided by two ways:
**1. internally –** priority of a process can be decided by an OS depends on no. of resources required for it

**2. externally -** priority of a process can be decided by user as per the user requirement
- min priority value indicates highest priority
e.g.

| **Processes/Jobs** | **Priority Value** |
|---|---|
| P1 | - 3 |
| P2 | - 2 |
| P3 | - 1 |

- in priority scheduling algo, **due to very low priority process may gets blocked** into the ready queue it leads to starvation/indefinite blocking.
- in priority scheduling algo, starvation can be resolved by using **ageing** technique.

**ageing :** it is a technique in which **priority of blocked process gets increases gradually** by an OS i.e. after some fixed time interval an OS keeps on incrementing priority of blocked process, so that moment will come priority of blocked process becomes suff enough to get control of the CPU.

- in all modern operating system's only one **cpu scheduler** program is there and this program has been implemented with combined logic of all basic 4 cpu sched algo's.

1. **indepependent processes** – if any process do not shared data (i.e. resources) with any other process.

2. **co-operative processes** - if any process shares data (i.e. resources) with any other process.

2. **Message passing model:**
**i. pipe mechanism:** by using pipe (it a buffer in the main memory which is provided by an OS) one process can send message to another process, it is **unidirectional communication pathway.**

- pipe has 2 ends:
write end : writer process
read end : reader process

processes which are running in the system can be catagorised into two catagories:
1. **related processes** – processes which are of same parent
2. **non-related processes** - processes which are of diff parents

- there are two types of pipe:
1. **unnamed pipe –** only related processes can commincates

2. **named pipe -** related as well as non-related processes can commincates

way-1: text message => text data
way-2: voice call => voice data
way-3: video call => video + voice data
way-4: missed calls => signals

1 missed call – meaning is predefined
2 missed calls – meaning is predefined

ii. message queue:
iii. **signal** – processes can communicates with each other by means of sending signal which is having some predefined meaning.

1. **background processes :** processes with which user cannot interacts
2. **foreground processes :** processes with which user can interacts

- An OS can send signal to any other process, but any other process cannot sned signal to an OS.
- When we shutdown the system, an OS sends **SIGTERM** signal to all processes due to which processes gets terminated normally, but there are few processes who can handle SIGTERM signal i.e. such processes do not gets terminated even after receiving SIGTERM signal from an OS, to such processes an OS sends **SIGKILL** signal due to which processes gets terminated forcefully.

**SIGTERM** – normal termination
**SIGKILL** – forcefull termination

**pipe**
**message queue**
**signal**
**socket**

- by using all above 3 message passing ipc mechnisms, only processes which are running in the same system can communicates.
- if process which is running on one system want to communicatess with process running on another system => socket ipc mechanism

# DS DAY-05:
- As an OS is responsible to starts an execution of any program, so when process gets terminated it returns any one value to an OS which indicates exit status:
1. successfull termination : 0
2. erroneous termination  > 0 => 1  : program gets exited due to some error
3. abnormal termination < 0 => -1 : divide-by-zero error

Process Synchronization / Co-ordination:

A
B
C

Line : PG-DBDA

Desk => 1 Common Notebook => On same page => on same line at time

data incosistency

race condition : if two or more processes are trying to access same resource at a time.

to avoid race condition :
1. an OS has to decide their order of allocation

critical section problem => there is a need of synchronization

- to avoid critical section problem, an OS uses 2 synchronization tools:
1. binary semaphore : S => integer var => 0 OR 1
2. mutex object

- there are 2  synchronization tools:
1. semaphore:
i. binary semaphore
2. mutex object

# deadlock:

A => Ch1 => Ch3

B => Ch2 => Ch1

C => Ch3 => Ch2

- there are 4 necc & suff condition to occurs deadlock:
1. mutual exclusion :
2. no-preemption
3. hold & wait
4. circular wait

covid – government (central/state)
1. prevention =>
2. detection & avoidance
3. recovery

- there 3 deadlock handling methods:
1. **deadlock prevention :** deadlock can be prevented by discarding any one condition out of 4 necc & suff condition.

2. **deadlock detection & avoidance :** in this method, before allocating resources for processes in advanced, all the input can be given to deadlock detection algorithm, and it gets checked wheather there are chances to occurs deadlock or not, if there are chances to occurs, then necc changes can be done and it can be avoided.
- there are 2 deadlock detection & avoidance algorithms:
1. resource allocation graph algorithm
2. banker's algorithm

3. **deadlock recovery:** system can be recovered from deadlock by two methods

# Memory Management (Main Memory i.e. RAM)
Q. Why there is a need of (main) memory management?
- For an execution any program RAM is must and hence it is also called as Main Memory.
- RAM/Main Memory is limited
-

Max CPU Utilization => Multi-tasking => Multi-Programming

Q. What is memory management?
Website => 1st Nov to 30th Nov PreCAT Batch – 1000 of duration 1 month

25th Oct – Batch is full
1 – logical seat number
2 -
3
.
.
1000

SunBeam : Krishna Hall => 225

1000 – into 5 equal size batches

| | | |
|---|---|---|
| batch-1 | : 1 to 200 | : 6 TO 8 AM |
| batch-2 | : 201 to 400 | : 8 TO 10 AM |
| batch-3 | : 401 to 600 | : 10 TO 12 |
| batch-4 | : 601 to 800 | : 12 TO 2 |
| batch-5 | : 801 to 1000 | : 2 TO 4 |

- in every batch first 25 seats are reserved for faculties

batch-1: 1 -> 26,  2->27, .....200->225
bacth-2: 201->26, 202->27, .....400->225
.
.
.

PG => Swap Area
third party => Sahyadri =>

Swap Area => It is a portion of the HDD which is used by an OS as an extension of the main memory into which inactive running programs can be kept temporarily.
- In Linux, swap partition is used as a swap area
- conventionally the size of swap area should be doubles the size of main memory
- In Windows, swap files is used as a swap area.

# OS DAY-06:
- **buffer cache –** it is a purely software technique in which an OS/kernel uses few portion of the main memory temporarily in which most recently accessed disk contents can be kept.

- **swap area –** it is a portion of the hdd used by an OS as an extenstion of the main memory in which inactive running programs can be kept temporarily.

- program gets loaded into the main memory:

RAM => 4 GB
Core i5

relocation registers:
base register : 15K
limit register : 5 K

- values of relocation registers will be kept inside PCB of that process.

- when any process is requesting memory, there are two methods by which memory gets allocated for it, these are reffered as memory allocation methods/ there are 2 memory allocation methods:
1. contiguos memory allocation : process can complete its execution only if memory gets allocated for it in a contiguos manner.
- there are two memory allocation schemes under contiguos memory allocation:
i. fixed sized partitioning scheme:
ii. variable/dynamic sized partitioning scheme:

- as segments of one process has been scattered anywhere into the main memory, and 100's of such processes are there, and hence to keep track on all the segments of one process, an OS maintains one table per process referred as segmentation table in which information about all its segments can be kept.

Swap Area = 8 GB
RAM/Main Memory = 4 GB

is it possible for an OS to complete an execution of any process having size bigger than size of main memory itself ??? ==> YES

# Virtual Memory Management
- An OS not only manages physical memory (i.e. main memory), it also manages virtual memory as well (i.e. swap area which is physically not a part of the main memory).

P1 => 6 GB

Virtual Memory = Main Memory + Swap Area => 4 + 8 = 12 GB

- When any process is requesting for the memory, an OS projects 12 GB of memory is available.
- Virtual Memory Management = Paging + Swapping
- If any bigger process is requesting for the main memory, not all its pages will be loaded into the main memory, only active pages can be their into the main memory and inactive pages can be kept temoprarily into the swap area, and as per request by that processes pages of it can be swapped in and swapped out in between main memory and swap area.
page in & page out => swapper process => lazy swapper
- demand paging: any page of a process will be loaded into the main memory only after requesting by that process, page which is never requetsed will be never gets loaded into the main memory -> pure demand paging.

Main Memory => 4 GB (1 GB Kernel + 3 GB User ) => no. of frames => m

P1 => 6 GB => no. of pages > no. of frames

in this case no. of pages of all processess >>>> no. of frames => all frames becomes full

- if any process is requesting for one of its page and that requetsed page is not present into the main memory => page fault
- when page fault occurs, requsted page will be loaded from swap area into the main memory into any free frame only if it is available.

- if all frames becomes full and page fault occurs, in this case there is a need to remove existing page from the main memory so that free frame will becomes available into which requested page can be loaded, i.e. there is a need of page replacement, and to decide which page should gets removed, there are certain algo's referred as page replacement algorithms:

1. fifo : first in first out page replacement algorithm:
- in this algo, page which was inserted first into the main memory will be removed first and requested page will be loaded into that free frame.

2. optimal page replacement algorithm:
- in this algo, page which will not used in near future will be removed from the main memory and requested page will be loaded into that free frame.

3. lru : least recently used page replacement algorithm
- in this algo, least recently used page will be removed from the main memory and requested page will be loaded into that free frame.

4. lfu least frequnetly used page replacement algorith

5. mfu most frequnetly used page replacement algorith

**- thrashing :** if any process spends more time on paging than execution, this high paging activity is called as a thrashing.

**# File Management:**
**Q. What is a file ?**
**user point of view:**
- file is a named collection of logically related data/information
- file is a basic storage unit

**system point of view:**
- file is a stream of bits/bytes i.e. 0101010101001.....

**- file has 2 things:**
**1. data :** actual file contents which are inside the file
**2. metadata :** information about the file which is inside FCB/iNode.

Process => PCB
File => FCB

- When any file gets created an OS/filesystem creates one structure for that file named as **FCB (File control Block )** inside which all the information about that file can be kept:
- In UNIX FCB is called as an iNode.
- FCB/iNode mainly contains:
    - inode number – unique identifier of a file
    - name of the file
    - size of the file
    - type of the file
    - access perms
    - time stamps

    etc....

**# OS DAY-07:**
- no. of iNodes onto HDD = no. of files on it
- all files (i.e. data + metadata of all files ) gets stored onto the HDD.
- 10 k files = 10 k iNodes
- millions of bytes of data of 10 k files and 10 k iNodes can be kept onto the HDD.

**- filesystem** – it is way to store data (i.e. data + metadata of files ) onto the **disk/any storage device/partition** in an organized manner so that it can accessed efficiently and conveniently.
e.g.
**Windows =>** ntfs (new technology filesystem), fat16(file allocation table), fat32 etc.....
**Linux =>** ext, ext2, ext3 (extended filesystem)
UNIX => UFS (UNIX FileSystem)
MAC OS X => HFS => Heirarchical FileSystem

- to format any storage device/disk/partition is actually not to erase data from it always, it is nothing but to create a new filesystem on it, and while creation of a new filesystem on it data on it may / may not gets erased.
- even after formatting disk data can be recovered from it by using some data recovery tool, any data recovery tool do not gives garauntee.
- PD – 4 GB : free space ~= 3.8
4 – 3.8 = filesystem.

**Q. What is a partition?**
- logical division of HDD.
- if we want to install multiple OS's on a single disk, we need to divide HDD logically into partitions and on each partition we can install one OS.
- We can have multiple OS's on one HDD, but on one partition we can have only one OS.

- filesystem divides disk/partition/storage device logically into **sectors/blocks** and inside each sector/block specific information can be kept.
- there are 4 sectors/blocks:
**1. boot block / boot sector :** it contains information about booting the system e.g. bootstrap program, bootloader peogram etc....

**2. super block / volume control block :** it contains all the information which is required to control disk operations, it contains info about other blocks.

**3. iNode list block / master file table :** it contains linked list of iNodes of all the files onto the disk/partition/storage devices.

**4. data block (logical block) :** it contains **physical data blocks** inside which actual file contents i.e. data of all the files can be kept.
usually size of physical data block = 512 bytes.

- When any file is requesting for free data blocks, there are three methods by free data blocks gets allocated for it, these methods are called as **disk space allocation methods:**

**1. contiguos allocation :**
- in this method, when a file is requesting for free data blocks, free data blocks gets allocated for it only in a **contiguos manner**, and **an addr of starting data block** and **count** i.e. no. of data blocks allocated for it can be kept inside its iNode.

**2. linked list allocation:**
- in this method, when a file is requesting for free data blocks, **randomly any free data blocks gets allocated for it,** and linked list of those allocated data blocks for that file will be maintained inside data block only and **an addr of starting data block as well as an addr end data block in that list** can be kept inside iNode of that file.

**linked list** => it is a list of logically related similar type of data elements in which elements are linked with each other

**3. index allocation:**
- in this method, when a file is requesting for free data blocks, **randomly any free data blocks gets allocated for it**, and any one data block out of those allocated free data block for that file will be considered as a **index block** into which an addresses of other data blocks can be kept, and **an addr of index data block** can be kept into an iNode of that file.

**reference book (5000 pages)**
100 pages => index pages => metadata information about pages
4900 pages => data actual contents

100 students are there in a class room

10 row and 10 colms

list of 10 students => 1 row

list of students

faculty => 22 (10) ==> 10 ( 77) ==> 77 (50) ==> .....=> 89( NULL)

- **waiting queue:** as an OS maintains dedicated waiting for each device.

- in a system 100's of processes are running at a time, and it is possible that out of those processes more than one processes/multiple processes are trying to access data from the disk at a time, as an OS maintains waiting queue for hdd, in this case all the requests can be kept into the **waiting queue** and there is a need to send only one request to the **disk controller** and hence there is a need of scheduling of one of the request at a time, and to do this there are certain algo's referred as **disk scheduling algorithms:**

**- each and every device has its own dedicated processor called as controller which controls all its operations.**
**e.g.**
**hdd => disk controller which controls all read/write operations**
**kbd => kbd**

- **disk controlle**r is a **hardware component** of **hdd** which controls all disk operations, and it can accept and complete only one request at a time.
Disk controller writes/read data to and from disk with the help of one **conducting coil** named as **head**, head is responsible for writing and reading data into/from the sector at a time.
- size of 1 sector = 512 bytes.
- if an OS want to access data from disk, it sends request to the disk controller and disk controller controls head movement and by means head movements data can be read/written into the disk at specific location.

- there are 5 disk scheduling algorithms:
**1. fcfs (first come first served) disk scheduling algorithm:**
in this algo, whichever request arrived first into the waiting queue, will be accepted and completed first by the disk controller.

**seek time :** it it the time required for the disk controller to move head from its current position to desired track.

**2. sstf (shortest seek time first) disk scheduling algorithm:**
- in this algo, whichever request in a waiting queue is closed to the current position of the head will be accepted and completed first by the disk controller.

**3. scan (elevator) disk scheduling algorithm:**
- in this algo, head keeps scanning disk from starting cylinder to end cylinder and again in a backward dir from end cylinder to starting cylinder, and while scanning whichever requests came accross will be accepted and completed by the disk controller.

**4. c-scan (circular scan) disk scheduling algorithm:**

- in this algo, head keeps scanning disk from starting cylinder to end cylinder and from end cylinder it directly jumps to starting cylinder, it means in this algo, scanning takes place only in a one direction, while scanning in one dir whichever requests came accross will be accepted and completed by the disk controller.

**5. look disk scheduling algorithm:**
- this algo can be used either with scan or c-scan in which if there is no requests in a waiting queue, then movement of the head will be stopped, and **unneccessary movement of the head is avoided** by using look policy.

- Min => OS => Lecture + Notes + PPTs + Black Book + Galvin first 3 chapters + GK of an OS.
- Google for practice of MCQ's

- Max => Whole Galvin Book + Google

Educational
Gradution – electronics
Masters – Maths & CA
Work exp:
3+ yrs of exp in software developement : c, c++, java
8+ yrs – teaching
12+ yrs