

Course Name : PreCAT Crash Course – OC07  
Subject Name : Operating System Concepts

---

Section-B => 9 questions are reserved for this subject

- 90%

- all questions are concept oriented – (no programming /no numericals)

# OS DAY-01:

Q. Why there is a need of an OS?

What is a Computer?

- Computer is a machine/hardware/digital device used to do diff tasks efficiently and accurately for user.

- Computer H/W mainly contains: Processor/CPU, Memory Devices & IO Devices etc...

- basic 4 functions of computer:

1. data storage
2. data processing
3. data movement
4. control

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

What is a software?

- collection of programs

**Program:**

- finite set of instructions written in any programming language given to the machine to do specific task.

- there are 3 types pf programs:

**1. system programs:** programs which are integral part of an OS (in built programs of an OS).

e.g. kernel, cpu scheduler, loader, dispatcher, memory manager etc....

**2. application programs:**

e.g. compiler, notepad, ms office, google chrome, calculator, games, IDE etc...

**3. user programs:** programs which are defined by programmer user

e.g. main.c, addition.cpp, program.java etc...

- program a passive entity, and process is an active entity
- if any program wants becomes active it must be loaded into the RAM
- program which is in the RAM is called as a process.
- program in execution is called a process
- running program is called as a process

- set of instructions written => program => HDD

- as any user cannot directly interacts with an OS, and hence an OS provides two types of interface for the user in the form of programs:

### 1. CUI/CLI: Command User Interface/Command Line Interface

- in this type of interface user can interact with an OS by means of entering commands in a text format through command line.

Example:

In Windows: **Command Prompt** => **cmd.exe**

In Linux: **terminal/shell**

`$gcc program.c` => command to compile a program  
`./program.out` OR `$.a.exe` => to execute a program  
`cp, mv, ls, clear, cls` etc.....

### 2. GUI: Graphical User Interface

- in this type of interface user can interact with an OS by means of making an events like single click, double click, click on buttons, exit, min, max, menu bar, menu list etc.....

In Windows : **explorer.exe**

**task manager** => search for a program **explorer.exe** => end task

In Linux: **GNOME/KDE**

**#include<stdio.h>**

- **header files** contains only declarations of library functions, and definitions of library functions are exists inside a lib folder in precompiled object module format.

- **stdio.h** file contains declarations of standard input output functions like `printf( )`, `scanf( )` etc...

- **RAM is also called as Main Memory?**

- for an execution of any program RAM memory is must, and hence RAM is also called as main memory.

- **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 program which is in the main memory onto the CPU.

Sachin => SunBeam ==> SunBeam  
Loader => OS ==> OS

Scenario-1:

Machine-1 : Linux : program.c

Machine-2 : Windows : program.c => compile + execute => YES

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

Scenario-2:

Machine-1 : Linux : program.c => compile => program.out (executable code)

Machine-2 : Windows : program.out (executable code) => execute NO

- file format of an executable file in Linux is ELF (Executable & Linkable Format), whereas file format of an executable file in Windows is PE(Portable Executable).

File format => OS specific

file format is a specific way of an OS to keep/store data & instructions inside an executable file in an organized manner.

- for example: elf file format divides an executable file logically into sections:

1. elf header/primary header/exe header:

2. bss section(block started by symbol):

```
int g_num;//global var  
static int num;
```

3. data section:

```
int g_num=99;//global var  
static int num=999;
```

4. rodata section:

100 => int constant  
1000L => long int constant  
012 => octal constant  
'A' => char constant  
0x12 => hex constant  
"SunBeam" - string literal

5. code section/text section  
6. symbol table

- magic number - it is a constant number generated by the compiler which is file format specific (e.g. In Linux: ELF => magic number in Linux starts with ELF in its hexa decimal eq ) => OS specific.

- when we execute a program, loader first verifies file format, if file format matches then only it checks magic number, and if file format as well as magic number both matches then only it loads an executable file from HDD into the main memory.

**Q. What is an OS?**

- An OS is a **system software** (i.e. collection of system programs) which acts as an interface between user and hardware.
- An OS also acts as an interface between programs (application & user programs ) & hardware.
- An OS allocates required resources like main memory, CPU time & IO devices access to all running programs, it is also called as **resource allocator**.
- to control an execution of all programs => OS
- to control hardware devices which are connected to the computer system => OS, and OS is also called as **control program**.
- an OS manages limited available resources among all running programs, it is also called as a **resource manager**.
- an OS is a software (i.e. collection of system programs & application programs which are in a binary format) comes with either in a CD/DVD/PD mainly has 3 components:
  1. Kernel :

OS is Kernel OR Kernel is OS => basic minimal functionalities of an OS.

breathing =>

teaching =>

Kernel - crash => reinstall an OS

MS Office - crash => repair

speaking => basic minimal functionality

extra utility functionalities:

teaching

singing

to give speech

2. Utility softwares

3. Application softwares:

=> installation of an OS: to install an OS onto the machine is nothing but to store OS software (i.e. collection of thousands of system programs and application programs which are in a binary format) onto the HDD.

=> if any OS want to becomes active, atleast its core program i.e. kernel must be loaded initially from HDD into the main memory, process to load kernel from HDD into the main memory is called as booting, and this job is done by bootstrap program (it is exists into the HDD in first sector i.e. in a boot sector in first 512 bytes).

- while booting, bootstrap program locates the kernel and load it into the main memory, and kernel remains present into the main memory till we do not shutdown the system.

UNIX: basically designed for the developer by the developers

Windows => Desktop/Server => commercial OS => buisness

Linux => Desktop/Server: Open Source OS : source code kernel of that OS is freely available

vmlinux - source code is freely available onto internet

Ubuntu - Linux distro

Redhat - Linux distro

Fedora Ubutu

Centos Linux

MAC OSX => MAC Machine by Apple  
iOS => iPhone by Apple  
UNIX  
Android => mobile phones  
Solaris => Server  
etc...  
thousands of OS's are available in market

Ken Thompson => B.E. Electricals  
Denies Ritchie => M. Sc. Physics & Ind Maths

UNIX = B + BCPL + Assembly Language

C was invented while developement UNIX, in 1972, and in 1973 UNIX was rewritten in C => Portable => and hence UNIX can run from nailtop to super computer.

- Linux OS was invented by Linus Torvalds as his academic project in the University of Helsinki in the decade of 1990's by getting inspired from UNIX.

- Linux & UNIX these are 2 difefrent OS
- Linux is a UNIX like OS.
- Windows => Microsoft

Google =>

GK Question based on OS ==> Google/Wikipedia

Human Body System:

- Nervous System
  - Excretory System
  - Digestive System
  - Respiratory System
- etc...

OS:

- File Subsystem
- Process Control Subsystem: IPC, Memory Management & Scheduling
- Hardware abstraction system
- IO Devices Management Subsystem
- System Call Interface Block

- there are 2 major subsystems:

1. file subsystem
2. process control subsystem

- file & process are two very very imp concept in any OS
- In UNIX : file has space and process has life

- In UNIX whatever that can be stored is considered as a file, and whatever is in active state is considered as a process.

- In UNIX all devices are considered as a file/UNIX treats all devices as file:
- devices can be categorised into 2 categories:

Keyboard => hardware/input device

UNIX point of view => KBD => character special device file

Monitor => hardware/output device

UNIX point of view => Monitor => character special device file

HDD => memory device

UNIX point of view => HDD => block special device file

usually size of 1 sector = 512 bytes

HDD (file) ==> PD (file)

Human Body - Soul => Dead Body => passive => program/files  
Human Body + Soul => living being => active

=> file subsystem  
=> process control subsystem

calculator: program  
main( ): client

services:  
addition( )  
division( )  
subtraction( )  
multiplication( )

Kernel: Program  
main( )  
functions defined in it: system calls =>

- system calls are the functions in kernel program defined in c, c++ & assembly language which provides interface of services made available by the kernel for user.

In other words:  
if any programmer user want to use services made available by the kernel in his/her program (i.e. in a user program), then it can be directly used by means of giving call directly to system calls or indirectly system calls can be called from inside library functions.

system developers: PGDESD

fopen( ) => open( ) : to open a file/to create a new file  
fclose( ) => close( ) : to close a file  
fprintf( )/printf( )/fwrite( )/fputs( )/putc( ) => write( ) : to write data into the file (to print => stdout file which is associated with standard o/p device: monitor).  
fscanf()/scanf()/fread()/fgets()/getc() => read() : read data from file

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



`fork( )` => to create a new process/child process

In UNIX => `fork( )`

In Linux => `fork( )` / `clone( )`

In Windows => `CreateProcess( )`

- irrespective of any OS there 6 categories of system calls:

1. file operations system calls: e.g. `open()`, `close()`, `read()`, `write()` etc...
2. device control system calls: e.g. `open()`, `close()`, `read()`, `write()`, `ioctl()` etc...
3. process control system calls: e.g. `fork( )`, `_exit()`, `wait()` etc...
4. inter process communication system calls: e.g. `pipe( )`, `signal( )` etc...
5. accounting information system calls: e.g. `getpid( )`, `getppid( )` etc...
6. protection & security system calls: e.g. `chmod( )`, `chown( )` etc....

`getpid( )` sys call returns pid of calling process

pid: process id – unique identifier of a process

`getppid( )` sys call returns pid of parent of calling process

# OS DAY-02:

introduction to an OS:

- why there is a need of an OS?
- what is an OS?
- compilation flow: elf, magic number, elf structure
- installation & booting
- kernel
- history of OS (UNIX)
- system arch design of UNIX
- system calls & its categories

**//program to do addition of 2 numbers => user defined code/user program**

```
#include<stdio.h>
```

```
int main( void ){
```

```
    int n1, n2, res;
```

```
    //executable instructions
```

```
    printf("enter the values of n1 & n2 : "); //write( ) sys call => system  
    defined code
```

```
    scanf("%d %d", &n1, &n2); //read( ) sys call => system defined code
```

```
    res = n1 + n2;
```

```
    printf("res = %d\n", res); //write( ) sys call => system defined code
```

```
    return 0; //sucessful termination
```

```
}
```

**sum.c => compile + link => sum.out => execute this program => process**

- Whenever system call gets called the CPU switches from user defined code to system defined code, and hence system calls are also called as **s/w interrupts or trap**.

**Interrupt:** an interrupt is a signal which is recieved by the CPU from any io device due to which it stops an execution of one job/process and starts executing another job/process => interrupt sent by h/w devices => h/w interrupt.

- throughout an execution of any program, the CPU keeps switches in between user defined code and system defined code, and hence we can say system runs in **2 modes**:

**1. user mode:** if currently the CPU executes user defined code instructions system runs in a user mode.

**2. system mode/kernel mode:** if currently the CPU executes system defined code instructions system runs in a system/kernel mode.  
and this mode of operation of an OS is referred as **dual mode operation**.

- to differentiate between user defined code instructions & system defined code instructions for the CPU, one bit is there onto the CPU itself called mode bit which is maintained by an OS.

if **mode bit = 1** ==> user mode

if **mode bit = 0** ==> kernel/ mode

**process control subsystem**  
**file subsystem**

**file & process :**

**process control subsystem:**

**process management:** an OS is responsible for

- process creation
  - to provide environment for an execution of all processes
  - to allocate required resources for processes
  - scheduling
  - inter process communication
  - process synchronization
  - to terminate process
- etc....

**What is a program?**

**User point of view:**

- A program is a finite set of instructions written in any programming language given to the machine to do specific task.

- Linux is UNIX like OS
- UNIX is referred as mother of all modern OS's like Windows, Linux, iOS, MAC OSX, android etc...

**System point of view: (Linux )**

- A program is an executable file which having elf format has got elf header, bss section, data section, rodata section, code section and symbol table.

**What is a Process?**

**User point of view:**

- program in execution
- running program is called as a process
- running instance of a program
- program which is loaded into the main memory

**System point of view:**

Process is a program which is in the main memory has got PCB into the main memory inside kernel space, and has got bss section, data section, rodata section, code section and 2 new sections got added for it:

1. **stack section:** it contains FAR's of called functions
2. **heap section:** dynamically allocated memory

**around size of 1 PCB  $\approx$  1 KB**

**kernel** : it is a core part/program of an OS which runs continuously into the main memory does basic minimal functionalities of it.

- kernel of any OS gets loaded into the main memory while booting an OS, it remains active/present into the main memory till we do not shut down system.

- portion of the main memory will be always occupied by the kernel, hence main memory is logically divided into two parts:

1. **kernel space**: portion of the main memory occupied by the kernel

2. **user space**: portion of the main memory other than kernel space, all user processes can be loaded only into a 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 magic number both matches then only it starts an execution of that program or in other words process gets submitted.

process submission => an execution of a process is started

upon process submission, very first one **structure** gets created for that process into the main memory inside kernel space, which is used by an OS/kernel to control an execution of it, this structure is referred as **PCB(Process Control Block)**.

**no. of PCB's inside kernel space = no. of processes**

- per process an OS maintains one structure called as PCB, also called as **PD(Process Descriptor)**, and PCB of a process will be destroyed from the main memory after exiting it.

- PCB contains all the information which is required to an OS for controlling an execution of that program, mainly it contains:

- **pid: process id** - unique identifier of a process

- **ppid** - parent's pid

- **PC: Program Counter** - it contains an addr of next instruction to be executed

- memory management information

- cpu scheduling information

- information about resources allocated for that process

- **execution context**

etc....

- if currently the CPU is executing any process, data & instructions of that process gets stored temporarily into the CPU registers, collectively this information is called as an **execution context** of that process, and copy of this information also kept inside PCB of that process to keep track.

- hundreds of processes are running into the main memory, out of only few can be active and remaining are inactive  
after submission of a process => program is running but it may be active or may be inactive.

**active running program** => if PCB of a process is there into the main memory inside kernel space and program is also there into the main memory inside user space.

**inactive running program** => if PCB of a process is there into the main memory inside kernel space and program is not there into the main memory inside kernel space (it can be kept temp into the swap area).

if for a process PCB is not there into the main memory inside kernel space  
=> process has been terminated => not running.

**swap area:** it is a portion of HDD used/kept reserved by an OS, so an OS uses this area as an extension of the main memory in which inactive running programs can be kept temporarily.

- throughout an execution of any program/process, it goes through different states, and at a time it may exist in only one state.

- there are 5 states of process:

1. new state
2. ready state
3. running state
4. waiting state
5. terminated state

+ features of as OS:

1. multi-programming
2. multi-tasking
3. multi-threading
4. multi-processor
5. multi-user

**1. multi-programming:** system in which more than one processes can be submitted at a time OR system in which at a time an execution of multiple programs can be started.

**+ degree of multi-programming:** no. of programs that can be submitted into the system at time.

**2. multi-tasking:** system in which the CPU executes multiple processes concurrently / simultaneously (i.e. one after another ).

**i.e. the CPU executes only one process at a time**

the speed at which the CPU executes multiple processes concurrently, we feels that/it seems, it executes multiple processes at once.

P1 = 40 MB

to drive a bike with gear:

day-1:

step-1: switch on

step-2: start bike either by click or by kick

step-3: need to press cluch fully

step-4: change gear from neutral to first

step-5: slowly need to release cluch and increase an accelerator

.  
. .

day-20:

step-1: switch on

step-2: start bike either by click or by kick

step-3: need to press cluch fully

step-4: change gear from neutral to first

step-5: slowly need to release cluch and increase an accelerator

.  
. .

responsiveness to stimuli – reaction time given by the brain to all actions is so quick we cannot differentiate these actions, we feel that we performed all steps at once.

**3. multi-threading:**  
what is a thread?

- thread is the smallest execution unit of a process
- thread is the smallest indivisible part of a process

system in which the CPU executes multiple threads which are of either same process or are of diff processes concurrently / simultaneously (i.e. one after another ).

i.e. the CPU executes only one thread of any one process at a time

the speed at which the CPU executes multiple threads of processes concurrently, we feel that/it seems, it executes multiple threads at once.

**processor/CPU**

- **uni-processor:** single processor

system can run on such a machine in which only one CPU/Processor is there.  
e.g. MSDOS

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

**5. multi-user:** system in which more than one users can logged into at a time  
e.g. Windows Server, Solaris etc...

- Windows OS basically designed for end users
- Linux / UNIX OS basically designed for developers

- to keep track on all running programs an OS maintains few data structures referred as 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 processes which are in the main memory and waiting for the CPU time.

3. **waiting queue:**

- an OS maintains one queue per device called as waiting queue/device queue
- it contains list of PCB's of processes which are waiting for that particular device.

**job scheduler :** it is a system program which schedules/selects processes/jobs from job queue to load them into the ready queue.

- it is also called as **long term scheduler**.

**cpu scheduler :** it is a system program which schedules/selects a process/job from ready queue to load it onto the CPU.

- for max cpu utilization, cpu scheduler must gets called frequently, and hence it is also called as **short term scheduler**.

- there are 4 cases in which cpu scheduler must gets called:

<b>case-1</b>	<b>running =&gt; terminated</b>	<b>: due to exit</b>
<b>case-2</b>	<b>running =&gt; waiting</b>	<b>: due to an io request</b>
<b>case-3</b>	<b>running =&gt; ready</b>	<b>: due to an interrupt</b>
<b>case-4</b>	<b>waiting =&gt; ready</b>	<b>: due to an io request</b>

- there are 2 types of cpu scheduling:

1. **non-preemptive:** in this type of cpu scheduling, control of the CPU released by the process by its own i.e. voluntarily.

e.g. in above case-1 & case-2

2. **preemptive:** in this type of cpu scheduling, control of the CPU taken away forcefully from a process.

e.g. case-3 & case-4



- there are basic 4 cpu scheduling algorithms:

1. fcfs (first come first served) cpu scheduling
2. sjf (shortest job first) cpu scheduling
3. round robin cpu scheduling
4. priority cpu scheduling

- as there are multiple algo's exists for cpu scheduling, there is need to decide which algo is best suited at specific situation, and which is an efficient one, to decide this there are 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 a 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:** total amount of time spent by the onto the CPU to complete its execution.

**CPU burst time:** total no. of cpu cycles required for the process to complete its execution.

### # OS DAY-03:

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

- in SJF tie can be resolved by using FCFS.

- starvation: in SJF process which is having min CPU burst time gets always control of the CPU first (SRTF/SNTF)

ready queue => 100 processes are there  
larger CPU burst time => P

system is a multi-programming system – during runtime as well processes are keep on submitting

- to overcome problem starvation in SJF => round robin scheduling algo has been designed.

- priority for processes can be decided by 2 ways:

1. **internally** => internally an OS decides priority for any process depends on no. of resources required for it.

2. **externally** => externally priority for processes can be decided by the user as per the requirement.

- priority of a process is stored into the PCB of it.

Priority value			
P1	-	3	- 20
P2	-	1	- 25
P3	-	2	- 12

P2 – highest priority

P3 -

Priority Scheduling:

ready queue = 100 processes

P1 = very low priority

system multi-programming - during runtime processes are getting submitted into the ready queue

Inter Process Communication:  
Why there is a need of an IPC?

What is an IPC?

SunBeam:

A = Ground Floor

B = 3<sup>rd</sup> floor

voice call - mobile => network down => medium - SIM Card/Network  
vodafone/airtel/jio

any process cannot communicate directly with any other process, for communication between processes there must exist some medium, and to provide this medium is the job of an OS/Kernel

"whatsapp group"

write => students can post/write doubts  
read => faculty -> answer to ur question

write answer => group  
read

message passing ipc model : there are 4 ipc techniques:

1. pipe
2. message queue
3. signal
4. socket ipc mechanism

way-1: video call - voice data + video data

way-2: voice call - voice data

way-3: messages - text data

way-4: missed calls - signal

1 missed call - predefined

2 missed call - predefined

in a system processes can also communicate with each other by means of sending signals which are having some predefined meaning.

- one process cannot send signal to any other process
- an OS can send signal to any other process, but any other process cannot send signal to an OS.

SIGTERM - normal termination  
SIGKILL - forcefull termination  
SIGSTOP - to suspend a process  
SIGCONT - to resume suspended process  
etc....

Process Co-ordination:

Why there is a need of Process Co-ordination/Synchronization:

A     B     C     => Desk => 1 Notebook

A

B

C

NoteBook : Page : Line => PGDBDA

- if i ask to all of them to write something into the same notebook on same page and same line at a time =>

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

we need to decide their order allocation:

there is a need of co-ordination => OS

"conflictions"

"data inconsistency"

critical section problem => data inconsistency

- in an OS there are 2 synchronization tools:

1. semaphore: binary semaphore

- simply it is a an int var - whose value is either 0 OR 1

- it used signaling mechanism

2. mutex object:

- mutual exclusion

an OS maintains mutex object with resource

mutex object has 2 possible states:

1. locked

2. unlocked

unlocked  $\Rightarrow$  resource is available

locking & unlocking mechanism

ClassRoom:

A  $\leq$  Ch1      A  $\Rightarrow$  Ch3

B  $\leq$  Ch2      B  $\Rightarrow$  Ch1

C  $\leq$  Ch3      C  $\Rightarrow$  Ch2

- at a time only person can seat on one chair

deadlock

- there are 4 necc & suff conditions:

1. **mutual exclusion:** at a time resource can be acquired by only one proces

2. **no preemption:** control of the resource cannot be taken away forcefully from any process

3. **hold & wait:** each process is holding resource and requesting for another resources which is already held by any other process

**# COVID : Government**

- 1. prevention : use of mask, social distancing and washing hands regularly**
- 2. detection & avoidance**
- 3. treatment – hospitalization - recovery**

**Deadlock - OS need to handle this problem**

**- there are 3 deadlock handling methods:**

**1. deadlock prevention:** deadlocks can be prevented by discarding any one condition out of 4 necc & suff conditions.

**2. deadlock detection & avoidance:** in this method before allocating resources for processes in advanced all the input can be given to algorithms for detection, if there are chances to occur deadlock necc changes can be done and deadlock can be avoided.

**- there are 2 deadlock detection & avoidance algo's:**

- 1. resource allocation graph algorithm**
- 2. banker's algorithm**

**3. deadlock recovery:** system can be recovered from deadlock by 2 ways i.e. there are 2 deadlock recovery methods:

- i. process termination:**
- 2. resource preemption**

**Memory Management**

**File Management**

SunBeam