Module Name : OS

- OS is there in Section-B and total 9 questions.
- all the questions are concept based & GK of OS/Computer Sci.

Q. Why there is a need of an OS?

- "Computer" is a hardware/machine/digital device can be used to perform diff tasks efficiently and accurately.
- "Computer Hardware System" mainly contains:
    - Processor/CPU
    - Memory Devices
    - IO Devices: Keyboard & Monitor -> Mouse

- basic functions of computer:
    1. data storage
    2. data processing
    3. data movement
    4. control

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

- there are 3 types of user:
1. "end user": e.g. data entry operator, billing counter

2. "admin user": person who installs os, softwares, configure networks, creates and maintains user accounts etc...

3. "programmer user": who can write programs in any programming languages.

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

Q. What is a Program?
- Program is a finite set of instructions written in any programming language given to the machine to do specific task.

- "Program" is a passive entity

Q. What is a Process?
- Program in execution is called as a process
- Running program is called as a process
- Process is an active entity

- When a program gets loaded into the main memory/RAM it becomes a process

- If any program want to becomes active, it must be loaded from secondary memory into the RAM/main memory.

- there are 3 types of programs:
1. "user programs": e.g. hello.c, addition.cpp, main.java etc...

2. "application programs": programs which either comes with an OS or can be installed later
e.g. notepad, google chrome, mozilla firefox, calculator, MS Office, Compiler IDE , disk cleaner etc...

3. "system programs": inbuilt programs of an OS OR part of an OS
e.g. kernel, scheduler, loader, memory manager, device driver etc...

ALU: Arithmetic Logic Unit - CPU -> H/W

I/O devices

What is an IDE (Integrated Development Environment)
- it is an "application software" which is a collection of tools i.e. application programs like editor, preprocessor, compiler, assembler, linker, debugger etc... required for faster software developement.
e.g. Turbo C, MS Visual Studio, Netbean, Eclispse, Android Studio, vs code editor etc...

"command line args" -> args which can be passed to the main() while executing program from command line.

- as an any user cannot directly interacts with an OS, and hence an OS provided two kinds of interfaces for the user in the form of "programs":
1. "CUI" : Command User Interface/"CLI": Command Line Interface
- in this type of interface user can interacts with an OS by means of entering commands through command line in a text format.
e.g.
$gcc program.c -> command used to compile a program
$./program.out OR $./a.out $.\a.exe --> commands to execute program through command line
cd, cp, ls, etc...
In Linux - name of the program which provides CUI => "shell"/"terminal"

In Windows - name of the program which provides CUI => "cmd.exe"/"powershell"
In MSDOS - name of the program which provides CUI => "command.com"

2. "GUI": Graphical User Interface
- in this type of interface user can interacts with an OS by means of making an events like click on buttons, left click, right click, double click, menu bar, menu list etc....
In Linux - name of the program which provides GUI => "GNOME"/"KDE"
In Windows - name of the program which provides GUI => "explorer.exe"
e.g.
- to execute program in windows - usually we "double click" on an excutable file.

"editor"/"source code editor":
e.g. notepad, vi editor, gedit, vs code editor etc...

#include -> file inclusion preprocessor directive -> preprocessor included contents of header file into the source file
#define -> it replaces macro by their values/definitions

- printf(), scanf() etc.. are the "library functions": whose declarations are there in a header files, whereas their definitions are exists into the lib folder in a "precompiled object module format"

- "loader" - it is a system program (i.e. part of an OS/inbuilt program of an OS) which loads an executable file from hdd int the main memory.
- loader is responsible to starts an execution of any application program as well as user program -> and hence OS is responsible
- usually programs contains: data & instructions

- "dispatcher" - it is a system program (i.e. part of an OS/inbuilt program of an OS) which loads instructions and data of processes from the main memory onto the CPU.

- to starts an execution of a program -> OS (loader)
- to provide environment/to allocate required resources to processes -> OS
- to terminate program -> OS

"Scenario-1":
Machine-1: Linux: "program.c"
Machine-2: Windows : "program.c" -> compile -> execute YES

"portability": program written in c programming language on one machine/platform can be compile and execute on any other machine/platform with very small or no changes.

e.g. "sizeof()" - operator

"Scenario-2":
Machine-1: Linux: "program.c" -> compile -> executable code/"program"
Machine-2: Windows : "executable code/"program.out" -> execute NO

- in Linux OS .extension do not matters
Why ?? -> "file format"

- file format of an executable file in Linux is "ELF: Executable and Linkable Format", whereas file format of an executable file in Windows "PE: Portable Executable".

- file format is a specific way of an OS to keep data & instructions of program in an organized manner, and it varies from one OS to other OS.

+ Structure of an ELF file format in Linux OS:
- ELF file format divides an executable file logically into sections, and in each section specific data (data + instructions) can be kept.
1. "elf header/exe header/primary header"
2. "bss section"
3. "data section"
4. "rodata section"
5. "code/text section"
6. "symbol table"

Q. Why an execution of every C programs starts from main() function?
- as compiler bydefault writes an addr of main() function inside primary header/elf header as an addr of entry point function.

- main() function is "system declared user defined fuction"

- to execute a program -> OS(loader)
- when we execute any program, loader first verifies the file format of an executable file, if file format matches then only it checks magic number and if both file format as well as magic matches then only its starts an execution of it i.e. it loads an executable file from the hdd into the main memory.

- "readelf" is a name of command in linux to check magic number of an executable file which is in ELF file format.

- Primary Memory : Memory which can be accessible directly by the CPU

e.g. Main Memory/RAM

- Secondary Mmeory : Memory which cannot be accessible directly by the CPU
e.g. HDD

```
int g_num1;//uninitialized global var -> "bss section"
int g_num2=999;//initialized global var -> "data section"

static int n;//uninitialized static var -> "bss section"
static int n = 99;//initialized static var -> "data section"
```

```
char str[ ] = "sunbeam";
sizeof(str): 8 bytes

"string variable"
char str[32] = "sunbeam";
sizeof(str): 32 bytes - in 32 bytes "sunbeam" string stored

str is a string variable whose value can be change later

scanf("%s", str);
```

```
"string literals" - "rodata section"
char *str = "sunbeam";
sizeof(str): 4 bytes - str will contains starting addr of string "sunbeam"

*str = 'p';//lvalue required error

10 = 20;//lvalue required error
```

```
#include<stdio.h>

int main(void)
{
	int n1, n2, n3;
	int sum, product;

	//executable instructions
	printf("enter n1, n2 & n3 : ");
	scanf("%d %d %d", &n1, &n2, &n3);

	sum = n1 + n2;
	product = n2 * n3;

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

}

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 computer hardware.
- an OS also acts as an interface between programs (application programs & user programs)and hardware.
- an OS allocates required resources like main memory, CPU time, IO devices access etc... to all running programs, and hence it is also called as a "resource allocator".
- an OS controls an execution of all programs, it also controls hardware devices which are connected to the computer system, and hence an OS is also called as a "control program".
- an OS manages available limited resources among all running programs, hence it is also called as "resource manager".

- an OS is a "software" (i.e. collection of system programs and application programs in a binary format) comes in CD/DVD/PD, and has following three main components:

1. "kernel": it is a core part/program of an OS which runs continuosly into the main memory and does basic minimal functionalities of it.

2. "utility softwares"
3. "application softwares"

"Kernel is an OS OR OS is Kernel"

+ Functions of OS:
basic minimal functionalities of an --> "kernel functionalities"
1. Process Management
2. Memory Management
3. CPU Scheduling
4. Hardware Abstraction - to hide physical prooperties of h/w devices from the user
5. File & IO Management

- extra utility functionalties - "utitlity softwares"
6. Protection & Security
7. User Interfacing
8. Networking

- to install any OS on the machine, it means to store OS software (i.e. thousands system programs and application programs which are in a binary format) onto the hard disk drive.

- if any OS wants to becomes active, then atleast its core program i.e. kernel must be loaded from hdd into the main memory initially, and process to load kernel from hdd into the main memory is called as "booting".

- where motherboard exists?
    - motherboard is inside the cabinet onto the PCB,

+ "bootable devices":
if any storage device contains one special program named as "bootstrap program" in its first sector i.e. in a boot sector (in first 512 bytes), then such a device is referred as a bootable device.

e.g.
- if any PD contains "bootstrap program" in its first 512 bytes -> bootable PD
- if first 512 bytes of any PD are empty -> non-bootable PD
- HDD, CD, DVD ->

+ Booting:
- there are two steps of booting:
1. "Machine Boot":
"step-1": when we switched on power supply current gets passed to the motherboard, on which one ROM memory is there which contains one micro-program called as "BIOS"(Basic Input Ouput System) gets executes first.
"step-2": first step of BIOS is POST(Power On Self Test), under POST, BIOS checks wheather all peripheral devices are connected properly or not and their working status.
"step-3": after POST, BIOS invokes "Bootstrap Loader" program, this program searches for available bootable devices in a computer system, and as per the priority decided in a BIOS settings it selects only one bootable device at a time. (bydefault in every computer system "HDD" is a bootable device).

2. "System Boot":
"step-4": Upon selection of HDD as a bootable device, "bootloader program" in it gets executes, and it displays list of names of an operating systems installed onto the machine, user has to select any one OS at a time from that list.

"step-5": upon selection of any OS, "bootstrap program" of that OS gets invoked, which locates the kernel and load it into the main memory.

- after booting the kernel it continuosly runs into the main memory till we do not shutdown the machine.

OS:
- Windows
- Linux
- MAC OSX
- Android
- iOS
- Symbian

- Chrome OS
- Solaris
- MS DOS
etc... thousands of OS's are available into the market

- OS Concepts of "UNIX": UNICS (Uniplexed Information & Computing Services/System).
Q. Why "UNIX"?
- UNIX basically designed for the developers by developers.
- UNIX was developed at AT&T Bell Labs, in the decade of 1970's by Ken Thompson, "Denies Ritchie" and team.
Ken Thompson - B.E. Electricals from UCB
Denies Ritchei - M.Sc. Physics & Maths

- "if you are expert in c programming language then you can become expert in any programming language".

- C programming language we invented while developement of UNIX

B Programming Lang: Ken Th
BCPL - by martin richards
C = B + BCPL

- in 1972 C was invented by denies ritchie for UNIX, and in 1973 UNIX was rewritten in C on PDP-11 (10,000 = 9000 C + 1000 assembly language).

- Linux was developed by "linus torwarlds" as his academic project in 1990, and its first kernel version 0.01 was released 1991
by getting inspired from UNIX.

- Linux is like UNIX OS.
- Linux open source -> source code of linux kernel "vmlinuz" is freely available on internet

- Windows - "ntoskrnl.exe"

"human body system":
- nervous system
- respiratory system
- digestive system
- reproduction system
- excretory system
etc...

"UNIX OS System":
- Process control subsystem
- File subsystem

- System call iterface block
- Hardware Control Subsystem/Block (HAL: Hardware Abstraction Layer)
etc...

- there are two major subsystems:
1. "file subsystem"
2. "process control subsystem": ipc, scheduler & memory management

- from UNIX system point of view, file and process these two are very imp concepts.
- UNIX phylosopy, "file has space and process has life"
i.e. In UNIX whatever that can store is considered as a file, and whatever is in a active state is considered as a process.

"human body" - physical existence
"soul"

human body without soul - dead body

- "UNIX system treats all devices as a file"

from user point of view keyboard is an i/p device, but from UNIX system point of view keyboard is a file
- UNIX system point of view monitor is also file, hdd is also file
- In UNIX devices can be catagorised into two catagories:
1. "character devices": devices from which data gets transferes char by char
e.g. keyboard, monitor, printer etc...

- UNIX treats all char devices as "char special device files".

2. "block devices": devices from which data gets transferes block by block i.e. sector by sector
e.g. all storage devices

- UNIX treats all block devices as "block special device files".

- when there is a tranfer of data from device to another device, from UNIX point of view there is transfer of data from one file to another file.

"buffer cache": portion of the main memory maintained by the kernel in which most recently accessed disk contents can be kept temporarily to get max throughput in min h/w movement.

- "system calls" are the functions defined in c, c++ & assembly language which provides interface of the 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 (user program), then either it can be called directly by giving

to system calls or indirectly system calls can be called through set of library functions.

system call programming -> desd
application c programming ->

"Kernel": Program/Core part of an OS

"system calls": functions of kernel program defined in C, C++, Assembly language.

Program: main() function + other functions

"slll.c":
- main(): client

server function - "services"
- add_last()
- create_node()
- add_first()
- add_at_pos()
- delete_last()
- delete_first()
- delete_at_pos()
- count_nodes()
etc....

e.g.
"fopen()" - to open a file or to create a new file
fopen() lib function internally makes call to open() sys call

- to write data into the file --> service of the kernel
"fwrite()/fputc()/fputs()/fprintf()/printf()" --> write() sys call
- to read data from the file
"fread()/fgets()/fgetc()/fscanf()/scanf()" --> read() sys call

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

to create a new/child process - OS/Kernel => fork()
to terminate a process - OS/Kernel => exit() --> _exit() sys call
to suspend a process - OS/Kernel => wait() - to stops an execution of a process for some time interval, which can resumed back

In UNIX - fork()

In Linux - fork(), clone()
In Windows - CreateProcess();

- "getpid()" sys call is used to get pid: "process id" - an unique identifier of a process.
- "getpid()" sys call returns pid of calling process/program
getpid() sys call never fails
- getppid() sys call returns pid of parent of calling process
- stat() sys call is used to get info about the file
- chmod() sys call is used to change access perms of file
- chown() sys call is used to change owner of the file

"cd" - it is a command in Linux to change dir

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

"write()" sys call - kernel function - "system defined function"

```
int write( .... )
{
        .........
        .........
        .........
        .........
}
```

```
//user program -> "user defined code" - executable file
#include<stdio.h>

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

        //executable instructions
        printf("enter the values of n1 & n2: ");//write() sys call
        scanf("%d %d", &n1, &n2);//read() sys call

        res = n1 + n2;
        printf("res = %d\n", res);//write() sys call - sys defined code
```

```
        return 0;//successfull termination -> _exit(0) sys call
}
```

CPU - "user defined code"

- when any sys call gets called the CPU switched from user defined code to system defined code and hence it is also called as "software interrupt"/"trap"

Q. What is an interrupt?
- an interrupt is a signal which is recieved by the CPU due to which it stops an execution of one process and starts executing another process.

- interrupts sent by h/w devices are referred as h/w interrupts
- system calls - kernel -> s/w interrupts

- throughout an execution of any program the CPU switches between user defined code & system defined code, and hence we can say system runs in two modes, and this mode of operation is called as "dual mode operation".

1. "user mode": when the CPU executes user defined code instructions system runs in a user mode

2. "kernel mode"/"system mode": when the CPU executes system defined code instructions system runs in a system mode/kernel mode

- CPU : "hardware" - has its own intelligence??

-> how the CPU can differentiate between user defined code instructions and system defined code instructions ?

- CPU can differentiate between user defined code instructions and system defined code instructions by using one bit which onto the CPU itself called as "mode bit", which is maintained by an OS.

user mode --> mode bit = 1
system mode --> mode bit = 0.

- in kernel mode only CPU can accedd hardware devices, whereas in user mode access to h/w is restricted, and hence to achieve hardware protection at an OS level dule mode operation is there.

+ "Process Control Subsystem":

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

"system point of view":
- program is nothing an executable file/code which has got an elf header, bss section, data section, rodata section, code/text section & symbol table.

- loader is a system program (i.e. part of an OS) which
    - it verifies file format
    - if file format matches then it checks magic number
    - if file format as well as magic number both matches then only it loads an exeutable code/program into the main memory i.e. an execution of program is started, it becomes a process

    - when we say loader starts an execution of any programs, it means it does following things stepwise:
        1. one structure gets created for that process by the kernel inside kernel space, in which all the info which is required to complete an execution of that process can be kept, this structure is referred as "PCB": Process Control Block.
        2. loader remove exe header and symbol table from an executable file
        and copies remaining all sections as it is into the main memory for a
        process and two new sections got added for the process by the kernel:
        i. "stack section": function activation records
        ii. "heap section": dynamically allocated memory

- per process an OS creates one structure into the main memory inside kernel space called as "PCB", PCB remains present inside kernel space till its process execution, as soon as any process execution is completed PCB of it gets destroyed from the main memory.
- PCB is also called as PD: Process Descriptor, In Linux PCB is called as Task Control Block.

- "PCB" mainly contains:
    - pid : unique identifier
    - ppid: parent's process id
    - PC Program Counter: an addr of next instruction to be executed
    - CPU sched info: sched algo, priority
    - memory management info

- io devices info allocated for that process
- "execution context":
info about data & istructions of the process which is currently executing by the CPU present onto the CPU registers also can be kept inside PCB.

etc...

- if the CPU is currently executing any process, data & instructions of that process can be kept temporarily onto the CPU registers, and collectively this info is referred as an "execution context" of that process.

"kernel" 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 into the main memory till we do not shut down the machine, and hence few portion of the main memory will be always occupied by the kernel, this space is called as "kernel space"

- main memory is logically divided into two parts:
1. "kernel space": portion of the main memory which is occupied by the kernel is referred as a kernel space.

2. "user space": whatever part of the main memory other than kernel space is referred as user space.

"buffer cache": it is a portion of the main memory inside kernel space in which most recently used disk contents can be kept.

What is a Process?
"user point of view":
- running program is called as a process
- program in execution is called as a process
- program inside the main memory is called as a process
- running instance of a program is called as a process

"system point of view":
- process is nothing but a program which is in the main memory has PCB inside the kernel space and has got stack section, heap section, bss section, data section, rodata section, code section.

"recursion" -> if we do not write termination cond -> program gets terminated due "stack overflow"
"malloc() fail" - due insuff heap memory

no. of PCB's into the main memory inside kernel space = no. of process running currently.

OS DAY-05

- after program execution is started/upon process submission, very first PCB gets created for it into the main memory inside kernel space, and program may be there into the main memory or it may not be there.

- if PCB of any program is there into the main memory inside kernel space and program is also there into the main memory inside user space -> "active running program"

- if PCB of any program is there into the main memory inside kernel space but program is not there into the main memory inside user space (i.e. it can be kept temporarily inot the swap area) -> "inactive running program".

"swap area": it is a portion of HDD kept reserved by an OS to keep inactive running programs temporarily.

```
//user program -> "user defined code" - executable file
#include<stdio.h>

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

	//executable instructions
	printf("enter the values of n1 & n2: ");//write() sys call
	scanf("%d %d", &n1, &n2);//read() sys call

	res = n1 + n2;
	printf("res = %d\n", res);//write() sys call - sys defined code

	return 0;//successfull termination -> _exit(0) sys call
}
```

- to keep track on all running programs/processes, an OS maintains few data structures called as a "kernel data structures":

- upon submission of any process its PCB gets created into the main memory inside kernel space onto the job queue.

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.

- an OS maintains dedicated waiting queue for each device
3. "waiting queue": if any process is waiting for a particular device then PCB of that process can be kept inside its waiting queue.

+ "features of an OS":

1. "multi-programming": sytem in which more than one programs/processes can be submitted at a time OR an execution of multiple programs can be started at a time.
- "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 executes multiple processes simultaneously/concurrently (i.e. one after another).
- the speed at which CPU executes multiple processes concurrently it seems that it executes multiple processes at once.
- "the CPU can execute only one program at a time".
- multi-tasking is also called as time sharing
- "time-sharing": system in which CPU time gets shared among all running programs.

"Process - 40 MB"
Q. What is thread?
- smallest indivisible part of a process is called as a thread
- smallest execution unit of a process is called as a thread

"PreCAT OM18 - "300":

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

- the speed at which CPU executes multiple threads concurrently it seems that it executes multiple threads at once.
- "the CPU can execute only one thread of any one process at a time".

- "uni-processor": system can run on a machine in which only one CPU/processor is there.
e.g. MS DOS

4. "multi-processor": system can run on a machine in which more than one CPU's/processor's are connected in a closed circuit.
e.g. Window32 onwards, Linux

5. "multi-user": system in which more than one users can loggedin at a time
e.g. server systems, solaris by sun microsystem, windows server

+ how ride a bike with gear

"day-01":
step-1: to switch on
step-2: to start bike either by kick or by click
step-3: to press cluch fully
step-4: to change the gear from neutral to first
step-5: slowly we need release cluch & increase acclerator
.
.
.
.

"day-20":
step-1: to switch on
step-2: to start bike either by kick or by click
step-3: to press cluch fully
step-4: to change the gear from neutral to first
step-5: slowly we need release cluch & increase acclerator
.
.
.
.

- "responsiveness to stumuli"

- priority of a process is there inside its PCB
- priority for a process can be decided by 2 ways:
1. "internally" an OS decides priority for a process depends on resources required for it.
2. "externally" priority for process can be decided by the user as per the his/her requirement.

- to do state-save of suspended process - "interrupt handler"
- to do state-restore of scheduled process by the cpu scheduler - "dispatcher"

OS DAY-06
- there are 4 cases in which cpu scheduler must gets called:
case 1: running --> terminated - due an exit
case 2: running --> waiting - due an io request
case 3: running --> ready - due an interrupt
case 4: waiting --> ready - due an io request completion

- there are two types of CPU scheduling:
1. "non-preemptive scheduling": 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 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 basic 4 cpu scheduling algo's:
1. fcfs (first come first served) cpu scheduling
2. sjf (shortest job first) cpu scheduling
3. round robin cpu scheduling
4. priority cpu scheduling

- OS: "cpu scheduler": this program has been implemented in an OS with combination of logic of all 4.

- "gant chart": it is a bar chart representation of cpu allocation for processes in terms of cpu cycles.

- in sjf tie can be resolved by using fcfs

OS DAY-07:
- processes which are running in the system can be catagorised into two catagories:
1. "independent processes"
2. "co-operative processes"

- there are two ipc techniques/ipc models provided by an OS for processes to do communication:
1. "shared memory model":

e.g.
common dir --> "git repo dir" : staff & students
staff - write as well read data into/from it
students - read data from it

e.g.
whatsapp group -> common platform

staff & students can communicates with each other by means of writing and reading messages/data onto whatsapp group.
shmget() - sys call used for requesting shared memory region

2. "message passing model":

i. "pipe":
- pipe has two ends, in this ipc mechanism, from one end i.e. "write end" one process writes data into the it, whereas from another end i.e. from read end another process can read data from it.

- processes which are running in a system can be catagorised into two more catagories:
1. "related processes": processes which are of same parent
"unnamed pipe": only related processes can communicates.

"terminal/shell" - "program" - CUI/CLI program in Linux
shell -> parent process
ls -> child process - command to display contents of dir
wc -> child process - word count

pipe => "buffer": it is a temp memory area of the main memory (kernel space) maintained by the kernel.

2. "non-related processes": processes which are of different parent's
"named pipe": related as well as non-related processes can communicates.

way-1: voice call - "voice data"
way-2: message - "text data"
way-3: video call - "video & voice data"
way-4: missed calls - "signal"

1 missed call - predefined meaning
2 missed calls - predefined meaning
3 missed calls - predefined meaning

3. "signals":

- OS can send signal to any other process, vice-versa is not possible

SIGTERM: normal termination
SIGKILL: forcefull termination

SIGSEGV: termination due to segment violation - "segmentation fault" - illegal memory access

- if any process is trying to access memory which is not allocated for it, then an OS sends SIGSEGV signal to that process, due to which process gets terminated by printing message as "segmentation fault".

SIGSTOP - signal to suspend a process
SIGCONT - to resume suspended process
etc...

- limitation of pipe, message queue & signals --> by using these message passing ipc mechanisms only processes which are running in the same system can communicates.

- if process which is running on one machine want to communicates with another process running on another machine -> "socket"

Q. Why there is a need of Process Synchronization/Co-ordination?
- 3 students seated on one desk, if they are sharing only one "notebook"
- if i ask to all of them to write something into the same notebook on same page and on same line at a time ??? --> race condition

A
B
C

NoteBook: Page: Line ==> eDESD

+ "race condition": if two or more processes are trying access same resource at a time, race condition occurs, i.e. conflictions takes place
- to avoid race condition an OS has to decide their order of allocation, as well as whatever changes did by last accessed process can be considered as final changes.

"data incosistency"

- there are 2 synchronization tools:
1. "semaphore":
    i. "binary semaphore": it is simply an int variable "S" having value either 0 OR 1 and at a time it may has only one value.

    ii. counting semaphore/classis semaphore
2. "mutex":

- Process may return any one return value of type of int to an OS
0 -> successfull termination
>0 -> erroneous termination -> malloc() fail - exit(1); file opening error
<0 -> abnormal termination -> divide-by-zero error

OS_DAY-08

"A" <-- Ch1, A -> Ch2

"B" <-- Ch2, B -> Ch3

"C" <-- Ch3, C -> Ch1

"deadlock":
- there are 4 necc & suff conditions to occur deadlock:
1. "mutual exclusion": at a time resource can be acquired by only process
2. "no preemption": control of the resource cannot taken away forcefully from any process
3. "hold & wait": each process is holding one resource and requesting for another resource which is held by any other process.
4. "circular wait"

"covid" - problem --> "government"
1. "prevention": vaccination, lockdown
    1. use of mask
    2. social distancing
    3. washing hands/sanitization regulary

2. testing & contact tracing, and if +ve patients found isolation ....
3. treatment in hospitals

- deadlocks can be handled by 3 ways: --> "OS"
1. "deadlock prevention": in this method, deadlock can be prevented by discarding any one condition out of 4 necc & suff conditions.

2. "deadlock detection & avoidance": in this deadlock handling 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 occur deadlock or not, if there are chances to occur deadlock then it can be avoided by doing necc changes.
- 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 two ways:
    1. "process termination": in this method, any one process gets selected randomly and it gets terminated forcefully, process which gets terminated
    forcefully is referred as a "victim process".

2. "resource preemption":

- in all modern OS, there is a not a single line of code for deadlock handling.

"Memory Management":
- when we say an OS does memory management, it means it manages main memory.
- why there is a need of main memory management?
    - main memory is limited
    - to achieve max cpu utilization -> multi-tasking -> multi-programming
    - if multiple processes has been submitted into the system, as main memory
    is limited and it is a responsibility of an OS to complete an execution of all submitted processes, and hence there is a need of memory management.
    - effective utilization of available resources for porgrams execution

e.g. RAM -> 1 GB/2 GB/4 GB/8 GB/16 GB/32 GB
HDD -> 1 TB.....

SunBeam, Hinjwadi Center --> "Krishna Hall" : "225"
physical seat numbers: 1-225

SunBeam - announcement on website - PreCAT Batch of 1 Month - 1000 capacity will be starting from Icard: "1st July 2021 - 31st July 2021".

- to pay fees and register yourself from today onwards:
batch becomes full -> logical
1-1000 -> logical seat numbers: 1 to 1000 : 20th June

divide batch of 1000 students into 5 equal size sub batches
batch-1: 1-200 : 6 TO 8
batch-2: 201-400 : 8 TO 10
batch-3: 401-600 : 10 TO 12
batch-4: 601-800 : 12 TO 2
batch-5: 801-1000 : 2 TO 4

- first 25 seats are reserved for the faculty members for each sub batch

Main Memory: Kernel Space + User Space

On 1st July ->
batch-1: 1 -> 26, 2 -> 27, 3 -> 28, ..., 200 -> 225
batch-2: 201->26, 202->27, 203->28, ...,400 -> 225
batch-3: 401->26,.....
batch-4: 601->26,.....
batch-5: 801->26

"PG Hostel" - "Sahyadri Building" - third party service for students : swap area

"swap area": it is a portion of the HDD used by an OS as an extension of the main memory into which inactive running programs can be kept temporarily.

- conventionally size of swap area should be doubles the size of main memory e.g.
Main Memory - 2 GB --> Swap Area = 4 GB
Main Memory - 4 GB --> Swap Area = 8 GB
.
.
.

- "swap partition" -> "swap area" --> In Linux
- "data partition" -> data
- "swap files" -> swap files --> In Windows

- addresses assigned/generated by the compiler for process before loading it into the main memory are called as "logical addresses".
- addresses which can be seen by the process when it is in the main memory called as "physical addresses".
- physical addresses of processes may gets changed throughout its execution, and hence the CPU always executes program in its logical memory space. whichever address is requested by the CPU is always a logical addr of a process.
- the CPU always deals with logical addresses of process, and hence there is a need of conversion of logical addr into its equivalent physical address, this job is done by one hardware unit called as MMU: Memory Management Unit.

- MMU is a hardware unit which converts logical addr requested by the CPU into its equivalent physical address and from that PA, data & instruction of that process can be fetched.

Whatsapp: 1000 : 1-1000

logical seat number: 555

physical seat number: 200

OS DAY-09

- deadlock & deadlock handling methods:
- memory management

Machine: Core i5
RAM: 8 GB

"relocation registers": used for relocation of a process into the main memory
1. base : 15 K
2. limit: 5 K

- memory management info: base & limit etc... can be kept PCB of a process.
- memory space of one process is getting protected from memory space of another process:
- when any process is requesting for the memory, there are two methods by which memory gets allocated for it, referred memory allocation methods:
1. "contiguos memory allocation": under this method, process can completes its execution only if memory gets allocated for it in a contiguos manner.
- there are 2 memory allocation schemes under contiguos:
    1. fixed size partitioning scheme
    2. variable/dynamic size partitioning scheme

- when any process is requesting for free partition, then any one memory allocation method is used out of 3 methods:
1. "first fit": in this method, whichever first free partition in which process can fit gets allocated for a process.
- this is faster allocation mechanism.

2. "best fit": in this method, such a smallest free partition gets allocated for a process in which there is a very less internal fragmentation.
- effective memory utilization

3. "worst fit": in this method, larger free partition gets allocated for a process and there is wastage of memory.

2. "non-contiguos memory allocation": under this method, process can completes its execution even if memory gets allocated for it in a non-contiguos manner.
- there are two memory management techniques by using which programs can completes their execution under non-contiguos memory allocation:
1. "segmentation":

- to keep track on all the segments of one process, an OS maintains one table per process called as "segment table", into which info about all its segments can be kept.

2. paging

physical memory = frames, size of all frames must be same
logical memory = pages

- size of any page <= size of frame
- if size of frame = 4 K then max size of any page of any process = 4 K

- to keep track on all the pages of one process, an OS maintains one table per process called as "page table", into which info about all its pages can be kept in the form of page addr and its corresponding frame addr.

"Modern OS": Segementation + Paging

"Main Memory - 4 GB" [1 GB - Kernel Space + 3 GB - User Space]
"Process - 6 GB"

- is it possible for an OS to complete an execution of such a process having size bigger than size of main memory itself ??? --> "YES"

- OS projects [Main Memory = 4 GB + 8 GB] = 12 GB of memory is availability for processes with it

"Virtual Memory Management": in this memory management technique, an OS not only manages main memory (i.e. physical memory), it also manages swap area (i.e. virtual memory).

"Virtual Memory Management" = Paging + Swapping

- big size process gets divided into pages, and not all pages gets loaded at once into the main memory, only active pages of it will be there into the main memory, and inactive pages can be kept temp into the swap area, and swapping of pages can be done as per request of the process between main memory and swap area, and process can completes its execution part by part.

- no. of pages >>>> no. of frames

- moment will come, all frames becomes full, and if any process is requesting for any of its page and if that requested page is not present into the main memory it is called as a "page fault", and hence there is a need to swap in requested into the main memory from swap area, but as there is no free frame

available there is need to remove any existing page from the main memory and can be swapped out into the swap area, and to decide which page will be removed so that free frame will becomes available into which requested page can be loaded, there are certain algo's called as "page replacement algo's".
- page replacement algorithms:
1. fifo (first in first out) page replacement algorithm
2. optimal page replacement algorithm
3. lru (least recently used) page replacement algorithm
4. mfu (most frequently used) page replacement algorithm
5. lfu (least frequently used) page replacement algorithm

page replacement algo in which min no. of page faults => efficient one

A1: optimal page replacement -> no. of page faults = 9 -> set a bechmark
A2: no. of page faults = 10

A3: no. of page faults = 12

A4: no. of page faults = 7 - wrong algo

OS DAY-10
+ "Memory Management":
 - process loading
 - memory allocation method:
        1. contiguos memory allocation: fixed size & variable size partitioning
        schemes: internal fragmentation, external fragmentation
        solution on external fragmentation: compaction,
        2. non-contiguos memory allocation: there are 2 memory management
        techniques: segmentation & paging
 - "virtual memory management": swapping + paging
 - page replacement algo's:
 1. fifo 2. optimal 3. lru 4. mfu 5. lfu

+ "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
 - file is a like a container which contains logically related data.

"system point of view":
 - file is a stream of bits/bytes
 - file = data + metadata
 1. data = actual file contents (kept inside the file)
 2. metadata = info about the file (kept inside FCB) like
        - "inode number" - unique identifier
        - name of the file

- parent folder location
- size of the file
- access perms
- time stamps
etc...

- on one filesystem two files may has same name, but each file has unique identifier called as inode number.

- metadata i.e. info about the file can be kept inside one structure called as "FCB (File Control Block)", in UNIX FCB is referred as an "iNode".
- iNode/FCB contains information about the file.
- In Linux "stat" command is used to display iNode contents i.e. info about the file, which internally makes call to stat() sys call.
- Per file one structure gets created at the time new file creation into which all its info can be kept.

total no. of iNodes/FCB's = no. of files
- all the files (i.e. data + metadata of all files) gets stored onto the HDD.
- Example: HDD - 10,000 files => HDD contains 10,000 iNodes + millions of bytes of data of those 10 K files.
- "filesystem": it is a way to store data onto the disk in an organized manner so that it can be accessed effieciently.
e.g.
Windows: NTFS/extNTFS (New Technology FileSystem)
Windows: FAT16/FAT32 (File Allocation Table)
Linux : ext2/ext3/ext4
UNIX : UFS (UNIX FileSystem)
MACOSX : HFS (Hierachical FileSystem)
etc...

- Each OS, has its own filesystem
- to format any storage device is not to erase data from it, it means to create a new filesystem on it, and while creating a new filesystem on it as a side effect data from it may gets erased.
- data can be recovered from hdd even after formatting by using some data recovery tools.

Mall: Floor: Stalls - labelling

- filesystem divides disk logically into sectors/blocks, and inside each block/sector specific contents can be kept.
1. "boot sector/boot block": it contains info which is required for booting the system. e.g. bootstrap program, bootloader program etc...

hdd = volume

2. "volume control block/super block": this block contains all info which is required to control disk operations i.e. it contains metadata (data about data).

3. "master file table/iNode list block": this block contains linked list of iNodes/FCB's.

4. "data block": (logical) data block contains physical data blocks having size 512 bytes = size of 1 sector on disk), inside actual file contents i.e. data of all the files can be kept.

PD - 4 GB

4 GB - ~3.80 GB --> filesystem info

- when any file is requesting for free data blocks, there are 3 methods by which free data blocks gets allocated for it called as "disk space allocation methods".
1. "contiguos allocation method": in this method, when any file is requesting for free data blocks then filesystem allocates free data data blocks only in a contiguos manner.
there are chances to occur external fragmentation

2. "linked allocation method": in this method, when any file is requesting for free data blocks, any free (i.e. randomly in a non-contiguos manner) data blocks gets allocated for that file, and linked list of those allocated data blocks can be maintained.

3. "index allocation method": in this method, when any file is requesting for free data blocks, any free (i.e. randomly in a non-contiguos manner) data blocks gets allocated for that file, inside any one out of it info about other data blocks can be kept, such data block is referred as index data block.

- In Linux: index allocation method is used

- "log file" is maintained by the filesystem, while filesystem doing operations like to write file/to copy file/file reading etc..., filesystem keeps stepwise track i.e. before performing any operation all the steps can be written in one log file and operations gets tracked, so that even if due power failure in between then filesystem do not gets corrupted - "journaling" - journeled filesystem

ext3
ext4 = ext3 + journalling support

reference book: 1000 pages

first 20 pages - index pages - info about pages ==>
980 pages contains actual info

- In a system --> 1000's
- out of 1000's of processes, is it possible that 100's of processes are requesting to access data from the disk at a time

- when multiple requests has been submitted for the disk, at that time all requests can be kept inside waiting queue of hdd, as disk controller can accept and complete only one request at a time, there is need of scheduling only one request at a time from the queue, to do this there are certain algo's called "disk scheduling algorithms".

- "controller": dedicated processor
- each and every device in a computer system has got its own dedicated processor called as controller
e.g.
hdd - disk controller controls disk operations
keyboard - controller controls keyboards operations
-
disk controller can accept and complete only one request at a time
- "disk controller" - its a dedicated processor of hdd/hardware part of hdd that controls all disk operations (read/write/movement of head etc...)
"head" - its conducting coil inside hdd which is responsible for actual read/write operation.

"seek time": total amount of time required for the disk controller to move head from its current position to desired sector.

"disk scheduling algorithms":
1. "fcfs (first come first served)": in this algo, whichever request arrived first into the waiting queue gets accepted and completed first by the disk controller.

2. "sstf (shortest seek time first)": in this algo, whichever request is closed to the current position of the head gets accepted and completed first.

3. "scan": in this algo, head keeps scanning disk from one end to another end back and forth (like an elevator), and while scanning whicever requests comes gets accepted and completed.

4. "c-scan (circular scan)": in this algo, head keeps scanning disk from one end to another end and from then directly jumps to the starting, so scanning takes place only in one direction, and while scanning whichever requests comes across gets accepted and completed.

5. "look": even though there are no requests in a waiting queue head keeps moving unneccessarily in scan as well as c-scan, to overcome this limitation look algo has been designed, in which if there is no request in a waiting queue movement of the head will be stopped.

Minimum - Class Notes + Black Colour Book Notes + Galvin First 2 chapters GK of OS Questions

Maximium: Galvin Book + Google

Minimum : Class Contents

"Maximum : Design And Analysis of an Algorithm (By Coreman)" + Google