Batch Name : PreCAT OM19 & OM21
Module Name : Operating System Concepts

- CCAT - Section-B, 9 questions are expected
- 95%
- No Programming/No Practicals
- All questions are purely concept/theory based
- No numericals
- OS Concepts – By Galvin

OS DAY-01:
Q. Why there is a need of an OS?

Q. What is a Computer?
- Computer is a machine/hardware/digital device used to do diff tasks efficieciently and accurately for user.

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

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

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.
- there are 3 types of programs:
1. system programs: programs which are the part of an OS/inbuit programs of an OS.
e.g. kernel, cpu scheduler, loader, device driver, interrupt handler, dispatcher etc....

2. application programs:
e.g. notepad, google chrome, MS Office, calculator, games etc....

3. user programs:
e.g. main.c, calculator.java, student.cpp etc...

- As any user cannot directly interacts with any OS, hence an OS provides 2 types of interfaces for 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 in a text format through command line/commond prompt.

e.g.
gcc – command to compile a program
./a.out OR .\a.exe OR ./program/out – command to execute a program
ls, cp etc.....

- In Linux name of the program which provides CUI => shell/terminal
- In Windows name of the program which provides CUI => cmd.exe - can be referred as command prompt/powershell etc...
- 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, menu bar, menu list etc...

- In Linux name of the program which provides GUI => GNOME(GNU Networok Modelling Environment)/KDE(Common Desktop Environment).

- In Windows name of the program which provides GUI => explorer.exe

IDE: Integrated Developement Environment
-

#include<stdio.h>
#include – file inclusion preprocessor directive, which includes contents of header file into the source file
stdio.h -file contains only declarations of standard i/p library functions
e.g. printf(), scanf(), ect....
malloc(), calloc(), free(), fopen() etc....

- header files contains only declarations of library functions
- definitions of all library functions are exists in a lib folder, in a precompiled object module format, which gets linked with object code of your program by the linker

- When a Program gets loaded into the main memory it becomes a process.
Q. What is a Process
- running instance of a program is called as a proces
- program in execution is called as a process

- Program is a passive entity, whereas a process is an active entity.

+ loader: it is a **system program (i.e. inbuilt program of an OS/part of an OS)** which loads an executable program from HDD into the main memory.
- to starts an execution/to load program from HDD into the main memory => loader => OS.

+ dispatcher: it is a **system program (i.e. inbuilt program of an OS/part of an OS)** which loads program (data & instructions of a program) from the main memory onto the CPU.

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 compiled and execute on any other machine/platform.

Scenario-2:
Machine-1 : Linux => program.c => compile + link ==> program(excutable code).

Machine-2 : Windows => program(excutable code) => execute ??? NO

Why ?
- **file format** of an executable file in Linux is **ELF(Executable & Linkable Format).**

- **file format** of an executable file in Windows is **PE(Portable Executable).**

What is a file format of an executable?
- It is a specific way of an OS to store data & instructions of a program in an executable file in an orgnanized manner.
- file format of an executable file is vary from OS to OS.
- elf file format divides an executable file logically into sections, and inside each section specific data (data & instructions) can be kept.
- there are mainly 6 sections:

1. elf header/primary header/exe header:
it contains info which is required to starts an execution of a program.
e.g. in elf header compiler bydefault writes addr of main() function as an entry point function.

2. bss section (block started by symbol): it contains uninitialized global & static variables.
int g_var;//globally defined var
static int i;

3. data section: it contains initialized global & static variables.
int g_var=99;//globally defined var
static int i=999;

4. rodata section (readonly data): it contains constants and string literals.
e.g.

100 – intger constant
100L – long int const
O10 – octal constant
0X15 – hexadecimal constant
'A' - char constant

"sunbeam"
"cdac, pune"

5. code/text section: it contains executable instructions
6. symbol table: it contains info about functions and its vars in a tabular format.

Q. Why an execution of every c program starts from main() function only?
entry point function:

- when we execute a program, loader first verifies file format of an executable file, 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 program into the main memory.

- magic number – it is a constant number (which is a in a hexadecimal format) generated by the compiler which is file format specific.
e.g.
In Linux magic number starts with ELF
In Windows magic number starts with MZ

Q. What is an OS?
- An OS is a **system software (i.e. collection of system programs)** which acts as an interface between user & hardware.
- An OS also acts as an interface between programs (user & application programs) and hardware.
- An OS controls an execution of all running programs, and it also controls hardware devices which are connected to the system, and hence it is also called as a **control program.**
- An OS allocates required resources like main memory, CPU time, IO devices accces to all running programs, it is also called as a **resource allocator.**
- An OS manages limited available resources among all running programs, hence 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 CD/DVD/PD.
1. Kernel: it is a core program/part of an OS which runs continuosly into the main memory and does basic minimal functionalties of it.
e.g. Linux – **vmlinuz**
Windows – ntoskrnl.exe
Kernel is an OS OR OS is a Kernel.
- Kernel is a like heart of an OS.

 # OS DAY-02:
Installation of an OS
- to install an OS onto the machine, means to store OS software (i.e. collection of system programs & application programs which are in a binary format) onto the HDD.
- if an OS want to becomes active, atleast first its core program **i.e. kernel must gets loaded into the main memory**, and to load kernel from HDD into the main memory is done by **bootstrap program**, this process is called as **booting.**

- bootable device: if any storage device (i.e. CD/DVD/PD/HDD) has one special program called as **bootstrap program** in its first sector/boot sector (usually size of sector = 512 bytes), then that device is referred as bootable device.

- If storage device do not contains bootstrap program in its boot sector then it is referred as **non-bootable**.

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

Step-2: the first step of BIOS is **POST(Power On Self Test)**, under POST BIOS checks wheather all peripherals 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** exists in the system and it selects any one bootable device as per the priority decided into the BIOS settings.

Step-4: by default bootstrap loader selects HDD as a bootable device, upon selection of HDD as a bootable device, **bootloader program** which is present inside HDD (in boot sector) gets invokes/executes.

2. System Boot:
Step-5: bootloader program displays list of names of OS's installed onto HDD, from which user need to select any one OS.

Step-6: upon selection of an OS, bootstrap program of that OS which is present inside boot sector gets invokes, which locates the Kernel of that OS and load it into the main memory.

OS:
UNIX
Windows
Android
MAC OSX
Linux
iOS
etc...

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

UNIX – Linux
Linux is UNIX based OS i.e. Linux is a like a UNIX.
- Linux was developed in 1990's, in 1991 it's first kernel version 0.01 released, and was develpeoed by linus torvards as his academic project.

Human Body System:

OS:
- File Subsystem
- Process Control Subsystem: IPC, Scheduling & Memory Management
- System Call Interface
- Hardware Control (HAL: Hardware Abstraction Layer)
- Buffer Cache
- Char Devices & Block Devices
- Device Drivers

- there are 2 major subsystem of UNIX:
1. File Subsystem
2. Process Control Subsystem

- In UNIX file & process these two are very important concepts.
- In UNIX “file has space & process has life”.
- In UNIX whatever that can stored is considered as a file, whereas whatever is active is considered as a process.

Human Body + Soul ==> Living Being => File

Human Body – Soul ==> Dead Body => Proces

- UNIX treats all devices as file

from user point view => KBD – input device
from UNIX system point view => KBD – file (character special device file)

from user point view => Monitor – output device
from UNIX system point view => Monitor – file (character special device file)

from user point view => HDD/PD – storage devices
from UNIX system point view => HD/PD – file (block special device file)

stdin – standard input buffer which is associated with standard input device i.e. kbd.

program -> stdin file
program -> stdout file

stdout

- OS copies data from PD to HDD

HDD/PD – data gets tranfered block by block i.e . sector by sector (512 bytes)

buffer cache: it is a purely software technique, in which **portion of the main memory used by an OS to store most recently accessed disk contents** to get max throughput in min hardware movement.

Kernel: Program
Functions: system calls

slll.c:
main()
create_node( )
add_at_last( )
add_at_first( )

add_at_pos( )
delete_at_first( )
etc...

system calls: are the functions (i.e. functions defined inside kernel program) defined in C, C++ & Assembly language, which provides interface to services made available by the kernel for user.
OR
- if programmer user wants to use services made available by the kernel in his/her program, then it can be used either by giving directly call to system calls or indirectly system calls can be called from inside set of library functions.

- to open a file/to create a new file => OS
fopen( ) lib function C ==> open( ) sys call – function defined inside kernel program i.e. system defined code

- to write data into a file => OS
fwrite( )/fprintf( )/printf( )/fputs( )/fputc( ) ==> write( ) sys call
c++ lib functions ==> write( )

- to read data from file ==> OS
fread( )/fscanf()/scanf()/fgetc()/fgets() ==> read() sys call

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

- to create a new/child process

In UNIX fork( ) sys call
ws – CreateProcess( )
In Linux – fork( ) & clone( )
In Windo
- to terminate proces : **_exit( ) ; exit( ) lib function internally makes call to _exit( ) sys call**
- to suspend a process: wait( ) sys call

- **getpid( )** sys call is used to get pid (process id which is an unique identfier of a process) of calling process.

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

OS DAY-03:

```
//program to add two numbers: user program/user defined code
#include<stdio.h>

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

	//executable statements:
	printf("enter values of n1 & n2: ");//write() sys call – sys defined code
	scanf("%d %d", &n1, &n2);//read() sys call – sys defined code

	res = n1 + n2;
	printf("res = %d\n", res);//write() sys call – sys defined code
	return 0;//successful termination - _exit( ) sys call
}
```

- Whenever sys call gets called the CPU switched from user defined code to system defined code, and hence system calls are also called as **software interrupts/trap.**

Q. What is an interrupt?
- **an interrupt is a signal** recieved by the CPU from any i/o device due to which it stops an execution of one job/process and start executing another job/process.
- Interrupt sent by an io devices called as **hardware interrupt.**

- Throughput an execution of any program the CPU swicthes between user defined code and system defined, and hence we can system runs in 2 modes this mode of operation of the system is called as **dual mode operation.**

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

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

CPU => instructions -> binary format
- the CPU can differentiate between user defined code instructions and system defined code instruction by referrering one bit which onto the CPU maintained by an OS called as **mode bit.**
In user mode => mode bit = 1
In system/kernel mode => mode bit = 0

Process Management:
- When we say an OS does process management, it means
- an OS is responsible for creation of a process, to provide environment for a process to complete execution,
- cpu scheduling, process synchronization, inter process communication etc...
and it also reponsible to exit process

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

System/OS (Linux) point of view:
- Program is a nothing but an executable file/code which has got elf header, bss section, data section, rodata section, code section and symbol table.

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

System point view:
Process is nothing but running program which has got PCB into the main memory (inside kernel space) and has got bss section, code section, data section, rodata section and 2 new sections added by an OS inside user space:
1. stack section
2. heap section

Kernel: it is a core part/program of an OS which runs continuosly into the main memory and does basic minimal functionalities of an OS, and it remains present inside the main memory till we do not shutdown machine, and hence kernel occupies portion of the main memory always
- Main Memory is divided logically into two parts, part of the main memory which is occupied by the kernel is referred as **kernel space**, and whichever part is left other than kernel space will be referred as **user space.**

- when we execute a program (i.e. either we double click on an executable file or ./program.out), loader first verifies file format, if file format maches then only it checks magic number, and if file format as well magic number both matches then only an execution of that program is started OR process gets submitted
- when we say process gets submitted/an execution of a program is started very first one structure gets created into the main memory inside kernel space for that process called as **PCB (Process Control Block),** in which all the information which is required to complete an execution of that process can be kept.

- Upon submission of any process PCB gets created for it into the main memory inside kernel space, and upon exit of any process PCB of that process gets destroyed/removed from the main memory.
- PCB is a structure mainly contains:
- pid: process id – unique identifier
- ppid: pid of parent's process
- PC: Program Counter – it contains an addr of next instruction to be executed
- info about resources allocated for that process
- memory management info
- cpu sched info
- execution context: copy of an execution context on CPU register can also also kept inside PCB of that process.
etc...

if currently the CPU is executing any program, (info about) instructions and data of that program can be kept temporarily into the CPU registers, collectively this info is called as an **execution context.**

etc....

- PCB is also called as PD(Process Descriptor), In Linux PCB is called as TCB(Task Control Block).

- after process submission, process is either active/inactive

- after process submission:
if PCB of a process is 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 a process is into the main memory inside kernel space and program is is not there into the main memory (i.e. it is kept temporarily inside swap area) => **inactive running program.**

if PCB of a program is not there into main memory => program has been exited.

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

- throughout an execution, process goes through diff states, and at a time it may present only in a one state.
- there are total 5 states of process:
1. New State
2. Ready State
3. Running State
4. Waiting State
5. Terminated State

+ Features of an OS:
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 processes that can be submitted into the system at a time.

2. multi-tasking: system in which the CPU can execute multiple processes simulteneously/concurrently (i.e. one after another),
the CPU can execute only one process at a time.
- the CPU excecutes multiple processes concurrently with such a great speed, it seems that/we feels, the CPU executes multiple processes at a time.

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

3. multi-threading: system in which the CPU can execute multiple threads which are of either same process or diff processeses simulteneously / concurrently (i.e. one after another),
the CPU can execute only one thread of any one process at a time.
- the CPU excecutes multiple threads of processes concurrently with such a great speed, it seems that/we feels, the CPU executes multiple threads at a time.
e.g.
youtube: process
thread-1 – downloading
thread-2 – audio streaming
thread-3 – video streaming

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

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

5. multi-user: system in which more than one users/multiple users can loggedin at a time.
e.g. Server OS: Windows Server, Solaris

Ride Bike With Gear:
DAY-01:
step-1: switch on
step-2: to start bike either by click or by kick
step-3: to press cluch fully
step-4: to change gear from neutral to 1
step-5: release cluch slowly & increase acclerator
.
.
.
.

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

"responsiveness to stumuli"

OS DAY-04:
- to keep track on all running programs/processes, an OS maintains few data structures called as **kernel data structures:**
Kernel = Program – functions – system calls, variables constants, data structure
1. job queue: this list contains PCB's of all submitted processes
- upon submission of process PCB gets created into the main memory inside kernel space, and it gets added into the list called as job queue.

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: it contains list of PCB's of processes which are waiting for that particular device.

- an OS maintains dedicated waiting queue for each device
e.g. an OS maintains dedicated waiting queue for hdd, dedicated waiting queue for kbd etc...

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

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

- when the CPU switches from one process to another process is referred as a **context-switch.**
- during context-switch the CPU switches from an execution context of one process into the an execution context of another process.
- whenever an interrupt occurs, first an exceution of current instruction of process which is onto the CPU completed and then its execution context gets saved into its PCB => **state-save**

- priority for a process can be decided by two ways:
1. internally – priority of a processes can be decided by an OS depends on resources required for it.

2. externally – priority of processes can be decided by the user depends on requirement.

- in the following 4 cases CPU scheduler must gets called for effective utilization of the CPU:
case-1 : running state --> terminated state : due to an exit
case-2 : running state --> waiting state : due to an io request
case-3 : running state --> ready state : due to an interrupt
case-4 : waiting state --> ready state : due to an io request completion

- there are 2 types of cpu scheduling:
1. non-preemptive: it is a type cpu scheduling in which, control of the CPU released by the process by its own i.e. voluntarily.
e.g. above case-1 & case-2

2. preemptive: it is a type cpu scheduling in which, control of the CPU taken away forcefully from a process.
e.g. above case-3 & case-4

- there are basic 4 cpu scheduling algo's:
1. fcfs (first come first served) cpu scheduling algorithm
2. sjf (shortest job first) cpu scheduling algorithm:
    sntf (shortest-next-time-first)
    srtf (shortest-remaining-time-first)
3. rr (round robin) cpu scheduling algorithm
4. priority cpu scheduling algorithm

- as there are many/4 cpu scheduling algo's, there is need to decide which algo is efficient and which algo is best suited at specific sitution, and to decide this there are certain criterias referred as cpu scheduling criterias:
- there are total 5 cpu scheduling criterias:
1. cpu utilization (max) : one need to select such an algo in which utilization of the cpu must be as max as possible.

2. throughput(max) : 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 (min): 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(min): 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 (min): 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 time must be as min as possible.

* Execution time/CPU Burst Time: it is the total amount of time spent by the process onto the CPU to complete its execution.
OR CPU Burst Time: total no. of CPU cycles required for the process to complete its execution.

turn-around-time = waiting time + execution time

gant chart – it is bar chart presentation of CPU allocation for processes in terms of CPU cycle numbers.

OS DAY-05:
SNTF
SRTF
- basic logic => process which is having shortest CPU burst time always gets control of the CPU first.

- In multi-programming system => multiple processes can be submitted at a time and during runtime as well processes can keep on submitted.

read queue : 100 processes
there is a 1 process which is having larger cpu burst time

- such process may gets blocked into the ready queue i.e. control of the CPU never gets allocated for that process, this sitution is called as starvation/indifinite blocking.

- to overcome this problem of starvation in SJF ==> Round Robin Algorithm has been designed.

Processes/Jobs	Priority	CPU Burst Time
P1	3	10
P2	1	12
P3	2	5

Process P2 has got highest value -> min priority value

+ IPC(Inter Process Communication)
Q. Why there is a need of an IPC?

2 friends – are PreCAT course from SunBeam, Hinjwadi
PG - Wakad – 10 km away fromHinjwadi
Schedule: 8 TO 1

sharing common bike
independent bikes

whatsapp – common:
students can post doubts
teacher – answer doubts

ipc techniques/models:
1. shared memory model
2. message passing model
further there are 4 message passing ipc techniques:
i. pipe:
- by using pipe mechanism one process can send message to another process, vice-versa is not possible, it is unidirectional communication.
- there are 2 types pipe ipc mechanisms:
1. unnamed pipe: pipe command (|) only related processes can communicates. communicates.

2. named pipe: pipe() sys call – related as well as non-related processes can communicates.

pipe has 2 ends from end -> input can be given and from another end out can be taken

processes which are running in the system also can be catagorised into two catagories:
1. related processes: processes which are of same parent
shell – program => parent process
ls command --> program which displays/lists contents of cur directory. => child process
wc command --> child process
ls & wc are related process

2. non-related proceses: processes which are of different parents

ii. message queue:

iii. Signals – processes which are running in the system also communicates by means sending signals which are having some predefined meaning.
iv. socket

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

1 missed call => predefined meaning
2 missed calls => predefined meaning
3 missed calls => predefined meaning

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

- When we shutdown the machine, an OS sends SIGTERM signal to all processes due to which processes gets terminated normally, but there may exists few process out of them can handle SIGTERM i.e. even after recieving SIGTERM from an OS such processes continues their execution, hence to such processes an OS sends SIGKILL signal due to which processes gets terminated forcefully.

SIGTERM : normal termination
SIGKILL : forcefull termination
SIGSEGV : termination of a process due to segment violation
SIGSTOP : to suspend a process
SIGCONT : to resume suspended process
.
.
etc....

- if any process is trying to access memory which is not allocated for it (e.g. illegal memory access/dangling pointer), an OS sends **SIGSEGV** signal to that process due to which process gets terminated with printing message as **segmentation fault.**

pipe, message queue, signal => by using this ipc mechanisms only processes which are running in the same system can communicates.

- Process which is running on one machine wants to communicates with process which is running on another machine, whereas machines are at remote distance provided connected in a network

+ Process Synchronization/Process Co-ordination:

Q. Why there is a need of Process Synchronization?

Desk: Common NoteBook - 1
A
B
C

- if i ask to all of them to write something in the same notebook on same page and same line at a time => race condition
- to avoid race condition, we can decide their order
Line : PG-DBDA
A
B
C

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

Data Inconsitency:

- An OS does process synchronization/co-ordination by using synchronization tools:
1. semaphore:
- there are 2 types of semaphore:
i. binary semaphore: it can be used when resource can be acquired by only one one process at a time.

ii. classic/counting semaphore: it can be used when resource can be acquired by more than one processes at a time.
It is simply a **counter var**
initial value of counter var "cnt" = max no. of processes that can be acquired resource at time
2 operations can be performed on it
1. increment: this operation can be performed while processes releases the resource.

2. decrement: this operation can be performed while processes acquiring the resource.

2. mutex object: it can be used when resource can be acquired by only one one process at a time.

Critical Section Problem: