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...