Batch Name : PreCAT OM22 FAST TRACK BATCH
Module Name : Operating System Concepts

OS DAY-01:

Section-B – 9 Questions are reserved
All questions are mostly concepts based/theory – no practical/no numericals.
+ GK of OS (Google).

Q. Why there is a need of an OS?
As any user cannot directly interacts with any computer hardware system, and hence there is a need of some interface between user & hardware, so an OS provides this interface.

Q. What is an OS?

- Software: collection of programs
- what is Program?
- there are 3 types of programs:
1. system programs: programs which are part of an OS i.e. inbuilt programs of an OS.
e.g. kernel, cpu scheduler, memory manager, laoder etc....

2. application programs:
e.g. compiler, google chrome, games, notepad, ms office, ide etc....

3. user programs: programs defined by programmer user
e.g. addition.c, hello.cpp, linkedlist.java etc.....

- as any user cannot directly interacts with any OS, an OS provides 2 types of interafaces for user in the form of programs:

1. CUI/CLI (Command User Interface)/Command Line Interface:
- by using this type of interface user can interacts with an OS by means of entering commands in a text format through command line.
e.g.
to compile a program => $gcc program.c
to execute a program => $./program.out OR $./a.out OR $.\a.exe
to copy one file => $cp

In Windows => command prompt => name of the program in Windows that provides CUI => cmd.exe, powershell
In Linux => terminal => name of the program in Linux that provides CUI => shell
In MSDOS => command.com

2. GUI (Graphical User Interface):
- by using this type of interface user can interacts with an OS by means of making an events like right click, left click, double click, click on buttons, exit, menu bar etc .....
double click on icon of an executable file
In Windows – name of the program that provides GUI => explorer.exe
In Linux - name of the program that provides GUI => GNOME/KDE

#include<stdio.h>
declarations of standard i/o functions like printf(), scanf() etc...

#inlcude is a file inclusion preprocessor directive, due to this preprocessor included contents of header file into the source file.

- Header files contains only declarations of lib functions.
- definitions of all lib functions are present inside lib folder in a precompiled object module format.

Compiler Construction
By Aho Ulman

C Programming Language + DS + OS + Hardware Knowledge

Programs – passive entity
What is a Process?
- running instance of a program is called as a process
- when a program gets loaded into the main memory it becomes a process
- active/running program is called as a process
- process is an active entity
- program in execution is called a process

- for an execution of any program it must be loaded into the main memory

- loader: it is a system program (i.e. inbuilt program of an OS), which loads an executable file/code from HDD into the Main Memory.
- to starts an execution of any program => loader (OS)

- dispatcher: it is a system program (i.e. inbuilt program of an OS), which loads an data & instructions of a program from main memory onto the CPU (on CPU registers).

MSOffice/MSCIT

- CPU registers : inbuilt memory of the CPU in which currently executing data & instructions can be kept temporarily.
Why RAM is also called as Main Memory:
For an execution of any program RAM memory is must, and hence it is also called as 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 & hardware.

- An OS also acts an interface between programs (i.e. application programs & user programs) and computer hardware.

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

- An OS manages limited available resources among all running programs, it is also called as a **resource manager.**

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

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

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

Scenario-2:
Machine-1: Linux : program.c --> compile ==> program executable file/code

Machine-2: Windows : program executable file/code => execute ???

Why ???? - file format

- 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).**

What is a file format?
It is a specific way to store/keep data (i.e. data & instructions of a program) into a file.
Way to keep/store data into an executable file is vary from OS to OS.

- When we try to 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 program/executable code from HDD into the main memory.

Linux -> loader ==> ELF
Windows -> loader ==> PE

What is a magic number?
- It is a constant number generated by the compiler which is file format specific i.e. magic number of an executable file in Linux starts with ELF in its eq hexadecimal format.

- magic number of an executable file in Windiows starts with MZ in its eq hexadecimal format.

Marks Zebeski – Windows OS Architect @Microsoft.

magic number is file format specific => file format is OS specific
magic number ==> OS specific

Structure of an ELF file format:
- ELF file format divides an executable file logically into sections, and in each section specific data can be kept.
1. elf header/exe header/primary header:
- main() is also called as entry point function, as an execution of every c program starts from main( ) function.

Q. Why an execution of every C program starts from main( ) only ????
- bydefault **compiler** writes an addr of main( ) function inside exe header as an entry point function.

2. bss section
3. data section
4. rodata section

5. code/text section
6. symbol table

2. bss section: it contains uninitialized global and static vars
int g_num;//global var
int static num;

3. data section: it contains initialized global and static vars
int g_num=99;//global var
int static num=1000;

4. rodata section (read only data section): it contains constants & string literals
e.g.

1000 --> int constant
100L -> long int const
012 -> octal constant
0x10 -> hex consant
'A' -> char constant
1.2f -> float constant
3.5 -> double constant

char str[32] = "sunbeam"; - str is a string variable whose value if "sunbeam" which can be modified later.

char *cptr = "sunbeam"; - sunbeam is string literal which cannot be modified

lavalue required error

"SunBeam"

5. code/text section : it contains executable instructions

6. symbol table: it contains info about functions and its vars in a tabular format.

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

- Kernel is heart of an OS
- Kernel is OS OR OS is Kernel

must functionalities

e.g.
speaking – basic minimal functionality

teaching – extra utility functionality
to give speech - extra utility functionality
to sing - extra utility functionality

breathing -
teaching -

OS DAY-02:
Installation of an OS ==> to install an OS onto the machine is nothing to store OS softwate (i.e. collection of thousands system programs & application programs which are in a binary format) onto the HDD.

- Linux Kernel source code is freely available on internet => Open Source OS

Linux – CD =>

=> If any OS want to becomes active, atleast its core program i.e. kernel must be loaded first into the main memory

- to load kernel from HDD into the main memory is done by bootstrap program is this is called as a booting.

- bootloader program

bootable device: if any storage device contains one special program called as **bootstrap program** in its first sector i.e. in a boot sector (usually size of sector = 512 bytes), then it is referred as bootable device.
and if boot sector of storage device empty => non-bootable.
If in a PD bootstrap program is there in boot sector i.e. first 512 bytes => bootable PD.
If in a PD boot sector i.e. first 512 bytes is empty => non-bootable PD.

There are 2 steps of booting:
1. Machine boot (hardware boots):
step-1: when we switched on the power supply current gets passed to **motherboard**, on which one **ROM memory** is there, which contains one micro-program called as **BIOS (Basic Input Output System)** gets executes first.

step-2: first step of BIOS is **POST(Power On Self Test)**, under POST it check wheather all peripherals are connected properly or not and their working status.

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

2. System Boot:
step-4: upon selection of bootable device as HDD, **bootloader program** which is present in a boot sector of HDD gets invokes, which displays list of names of OS's installed on it, from which user has to select any one OS at a time.

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

iPhone – iOS
MAC Machine – MAC OS X
Desktop OS
different types of OS as per user requirement.

Windows
Linux
UNIX
MACOSX
iOS
Android
Kali Linux
Solaris
etc.....

there are 1000’s of OS available in a market

UNIX:
Q. Why UNIX?

Ken Thompson : B.E. Electricals from UCB
Denies Ritchie : M.Sc. Physics & Maths
AT&T Bell : Transistor, UNIX

- **System arch design** of UNIX is followed in all modern OS (Linux, Windows, MACOSX, android, iOS etc....), and hence UNIX is also referred as mother of all modern OS.

- Mainframe Server Machine ==>

C – B + BCPL

- UNIX can run from nailtop to Super Computer => It was rewritten in C in 1973 => **10,000 = 9000 C + 1000 Assembly.**

- Linux is UNIX like OS, which was invented by linus torvards in 1990’s as his academic project by getting inspired from MINIX OS (UNIX).

- Windows OS designed basically/specially for end users.

UNIX: System Arch Design Of UNIX:

Human Body System:
- Nervous System
- Reproduction System
- Digestive System
.
.
.

OS:
- File Subsystem
- Process Control subsystem
- System call interface
- Hardware Control
- Devices

- there are two major subsystems in any OS:
1. file subsystem
2. process control subsystem

- for any OS – file & process are very imp concepts

In UNIX – file has space & 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.

KBD => input device/hardware
HDD => memory device
Monitor => output device

From UNIX system point of view => KBD is file
From UNIX system point of view => HDD is file
From UNIX system point of view => Monitor is file
From UNIX system point of view => PD is file

UNIX catagorised devices into

- Copy data from HDD to PD => UNIX OS treats this as copy of data from one file to another file.

Human Body – Soul => Dead Body => file

Human Body + Soul => Linving Being – active - process

Kernel Functionalities/Services: Process Management + File & Storage Management +
IO Devices Management + IPC + Scheduling + Memory Management

programmer user => to use kernel services

kernel provides interface in the form of system calls:

Kernel => Program
functions: system calls

calculator: program
main( ) => client function

functionalities: services

addition( )
substraction( )
division( )
multiplication( )

Q. What are system calls?
System calls are the functions defined in C, C++ & assembly, wihch provides interaface of the services made available by the kernel for user.

- if any programmer user wants to use kernel services in his/her program then it can be called indirectly gets called from inside library functions, or directly it can be called by giving call to system calls (system programming).

System calls – system defined code

file handling program in C:
fopen( ) C lib function => to open a file/to create a new file
fopen( ) => open( ) sys call which actually opens a file/creates a new file

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

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

mkdir – command -> to create a new dir/s

system call to create a new process/child process
UNIX => fork( )
Linux => fork( ), clone( )
Windows => CreateProcess( )

- exit( ) C lib function which internally makes call to _exit( ) sys call due to which process gets exited/terminated.

- irrespective of any OS, there are 6 catagories of system calls:
1. file operations system calls: e.g. open( ), close( ), write( ), read( ), lseek( ) etc....

2. device control system calls: e.g. open( ), close( ), write( ), read( ), lseek( ) 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( ), stat( ) etc...
stat( ) sys call is used to get info about the file
getpid( ) sys call gives process id of calling process/program
getppid( ) sys call gives process id parent of calling process/program

6. protection & security system calls: e.g. chmod( ), chown( ) etc...
files & filesystem
chmod( ) - to assign/change access perms by means of changing mode bits
chown ( ) - to change owner of file
etc...

addition of two numbers: user defined code

```
#include<stdio.h>

int main( void )
{
	int n1, n2, res;

	printf( “enter 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;//successful termination
}
```

- whenever system call gets called the CPU stops an execution of user defined code and it starts executing system defined code, and hence system calls are also called as **software interrupts/trap.**

What is an hardware interrupt?
- an interrupt is a signal which recieved by the CPU from any hardware device due to which it stops an execution of one job/process and starts executing another job/proces.

- throughout an execution of any program the CPU swicthes between user defined code and system defined code, and hence we can say system runs in 2 modes:
1. user mode: when the CPU executes user defined code instructions

2. kernel mode/system: when the CPU executes system defined code instructions
dual mode operation of an OS.

OS DAY-03:
- Booting of system
- UNIX history/Linux
- UNIX System Arch Design : system calls, dual mode operation: user mode & kernel mode

- there are 2 major subsystems in any OS:
1. file subsystem
2. process control subsystem

+ Process control subsystem: scheduling, memory management, inter process communication => Process Management

From System Point of view:
- Process is nothing but program which is in the main memory has got one structure gets created by an OS 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 process: stack section & heap section.

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

- kernel gets loaded into the main memory while booting, and it remains present inside the main memory until shutdown, and hence kernel occupied portion of main memory always.

- Portion of the main memory which is occupied by the kernel is referred as a kernel space and whichever part is left other than kernel space is referred as a user space.

Main memory is divided logically into two parts:
1. kernel space
2. user space: user programs can be loaded only inside user space

- When we execute any program, laoder first verifies file format, if file format matches then only it checks magic number and if file format as well as magic no. of both matches then only it starts an execution of that program/process gets submitted.
- when an execution of any program is started/upon process submission, very first one structure gets created into the main memory inside kernel space for that process, this structure is called as PCB (Process Control Block) into which all the info which is required to complete an execution of that program can be kept.

- after complete an execution of a process, PCB gets destoyed/removed from the main memory
no. of PCB's inside the kernel space = no. of processes

PCB => structure: members

An OS creates one PCB per process, PCB mainly contains:
- pid (process id – unique identifier of a process)
- ppid: parent's process id
- pc: program counter – an addr of next instruction to be executed
- memory management info
- cpu scheduling info
- info about resources allocated for that process
- execution context
etc.....

- size of PCB is in MB's

execution context: if the CPU is currently executing any process, then info about that process (i.e. data & instructions of that process) can be kept temporarily into the CPU registers, and collectively this info is called as an execution context, and copy of this is also can be kept inside PCB of that process.

Upon process submission:

PCB gets created for that process into the main memory inside kernel space => running

running program may active or it may be inactive

after process submission, if PCB of that program is into the main memory inside kernel space and program is also there into main memory inside user space ==> active running program.

after process submission, if PCB of that program is into the main memory inside kernel space and program is not there into main memory inside user space (i.e. it can be kept temp into the swap area) ==> inactive running program.

If PCB of a process gets destroyed from main memory => not running

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

- throughout execution, program/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 an execution of more than one programs can be started at a time OR system in which multiple processes can be submitted 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 concurrently/simultaneosly (i.e. one after another).
i.e. the CPU can execute only one process at a time.

- the CPU executes multiple processes with a such a great speed simultanesoly/concurrently, it seems that/we feels that, the CPU executes multiple processes at once => multi-tasking

multi-tasking is also called as a time-sharing => system in which CPU time gets shared among all running programs.

Process – 40 MB

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

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

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

- the CPU executes multiple threads of processes with a such a great speed simultanesoly/concurrently, it seems that/we feels that, the CPU executes multiple threads at once => multi-threading.

Processor => CPU
Uni- processor: 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 than one processor's/CPU's are connected in a closed circuit.
e.g. Linux, Windows etc...

5. multi-user: system in which more than one users can loggedin at a time
e.g. server system

to ride a bike with a gear:
day-01:
step-1: we have to switch-on
step-2: start bike either by kick or by click
step-3: need to press cluch fully
step-4: change gear from neutral to 1st
step-5: slowly need to release cluch and increase an accelerator
.
.
.

day-20:
step-1: we have to switch-on
step-2: start bike either by kick or by click
step-3: need to press cluch fully
step-4: change gear from neutral to 1st
step-5: slowly need to release cluch and increase an accelerator
.
.
.

- responsiveness to stumili ==> reaction time given by the brain to all actions is so quick =>

- to keep track on all running programs, an OS maintains few data structures called as kernel data structures:
- there are 3 kernel data structures:
1. job queue: it contains list of PCB's of all submitted processes.

2. ready queue: it contains list of PCB's of processes which are in the main memory and waiting for the CPU.

- an OS maintains waiting queue per device.
3. waiting queue: it contains list of PCB's of processes which are waiting for that particular device.

- upon proces submission PCB gets created for that process into the main memory inside kernel space and gets added into the job queue.

- job scheduler: it is system program which schedules jobs/processes from job queue to load them onto the ready queue.
- it is also called as long term scheduler

- cpu scheduler: it is system program which schedules job/process from ready queue to load it onto CPU.
- it is also called as short term scheduler – for max utilization of the CPU cpu scheduler program must gets called frequently.

OS DAY-04:
- what is a process & what is PCB and its contents
- process state diagram: new, ready, running, waiting & terminated
- features of an OS: multi-programming, multi-tasking, multi-threading, multi-processor & multi-user
- kernel data structures: job queue, ready queue & waiting queue
- job scheduler & cpu scheduler
- context swicth

- and for max CPU utilization cpu scheduler must be called frequently
- in following 4 cases cpu scheduler must be called:

- there are 2 types of cpu scheduling
1. non-preemptive
2. preemptive

8 AM

Student-A : 7:30 AM.

Went outside

Student-B : Same Seat => forcefully

- 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
3. round robin scheduling algorithm
4. priority cpu scheduling algorithm

- there is a need to decide which algo is an efficient one and which algo is best suited at specific situation, and to decide this there are certain criterias referred as cpu scheduling criterias:
- there are 5 cpu scheduling criterias:
1. cpu utilization (max): one need to select such an algo in which

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

If i asked for 2 students to meet me at 8:00 AM

A => 7:30 AM
B => 8:00 AM

Staff => 8:15 AM

sjf: process which is having min cpu burst time gets control of the CPU first.
shorter processes => highest priority
larger processes => low priority

multi-programming : multiple processes can be submitted into the system at a time and during runtime as well processes gets keeps on submitted.

ready queue = 100 processses, and during runtime as well processes are getting submitted into the ready queue, and these processes are also shorter than process which is already there in a ready queue

if there is exists such a process which is having max cpu burst time

4 algo's : each algo has its own advantages & disadvantages

cpu scheduler program is implemeneted with combined logic of all 4 algo's.

Inter Process Communication:

Why there is a need of an IPC?

What is an IPC?

2 friends => are doging PreCAT from SunBeam, Hinjwadi
leaving in Wakad at different locations, which is 10 Km away from SunBeam

if they are sharing common bike =>

if theye are having own independent bikes

A => SunBeam = 3rd floor

B => SunBeam = ground floor

whatsapp group: doubt – students can post/write

answer/write of that queustion

one process cannot directly sends message to another process, some medium is required for this:
- there are further 4 types in message passing ipc mechanism:
1. pipe:
- pipe has 2 ends, 1 end is for writing and another end for reading

it is an uni-directional communication

- there are 2 tyeps of pipe:
1. unnamed pipe: in this mechanism by using pipe command (|) only related processes can communicates.

2. named pipe: in this mechanism by using pipe( ) sys call related as well as non-related processes can communicates.

- processes can be catagorised into 2 catagories:
1. related processes: processes which are of same parent

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

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

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

SIGTERM – normal termination
SIGKILL – forcefull termination
SIGSEGV – termination of a process due segment violation

pipe, message queue & signal