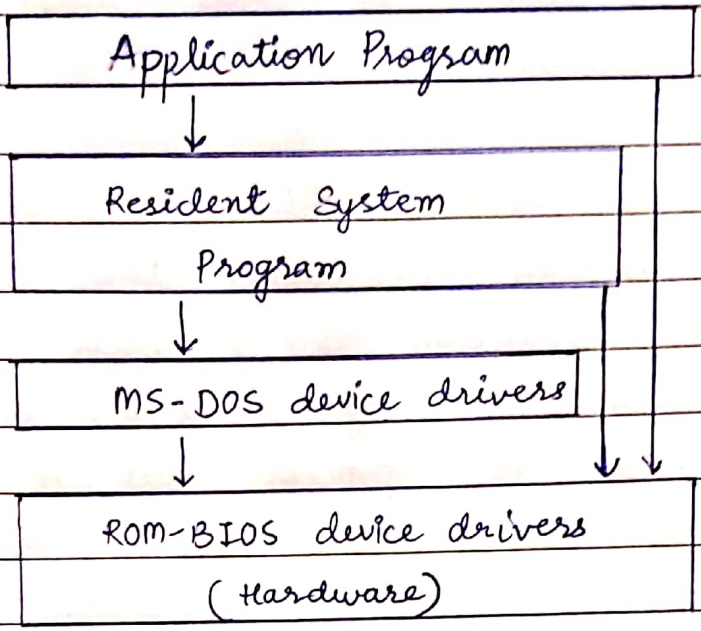
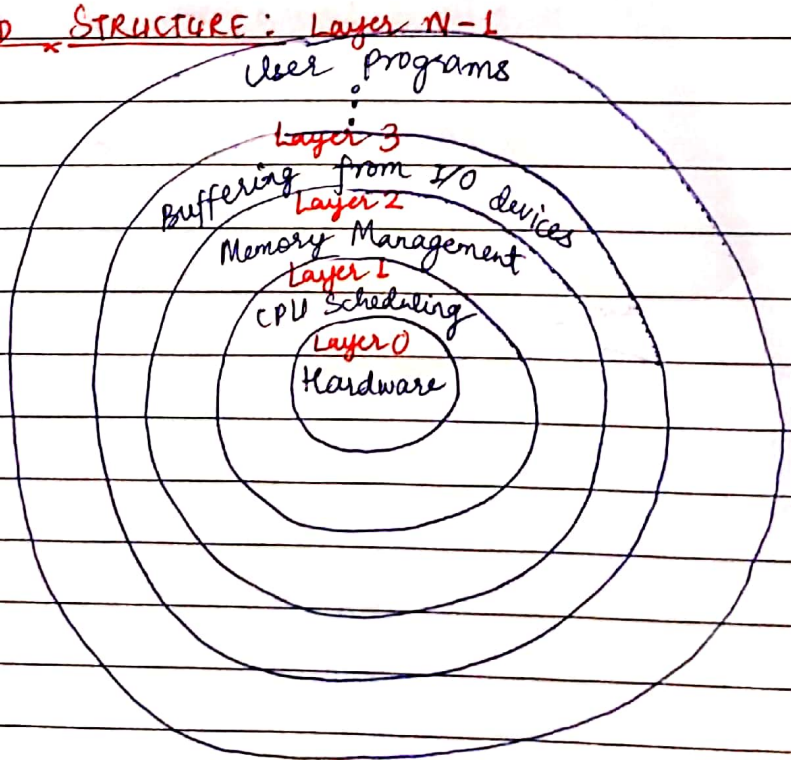
⇒ STRUCTURE OF THE OPERATING SYSTEM

(1.) SIMPLE STRUCTURE :



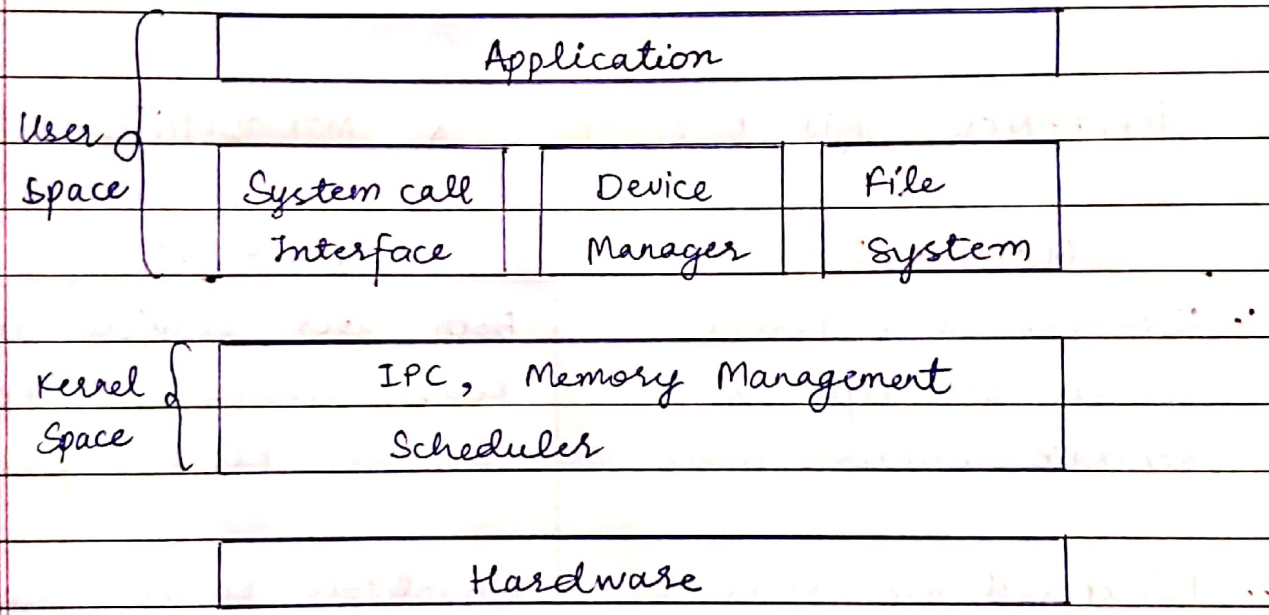
Operating system such as MS-DOS and the original UNIX didn't have well-defined structures. For efficient performance & implementation, an operating system should be partitioned into separate subsystems.

(2.) LAYERED STRUCTURE : Layer N-1



This approach breaks up the OS into different layers. The OS having layer 0 to layer N-1, the bottom layer, Layer 0, is the hardware layer and the highest layer, Layer N-1, is the user interface layer as shown in the figure.
The original UNIX OS used a simple-layered approach.

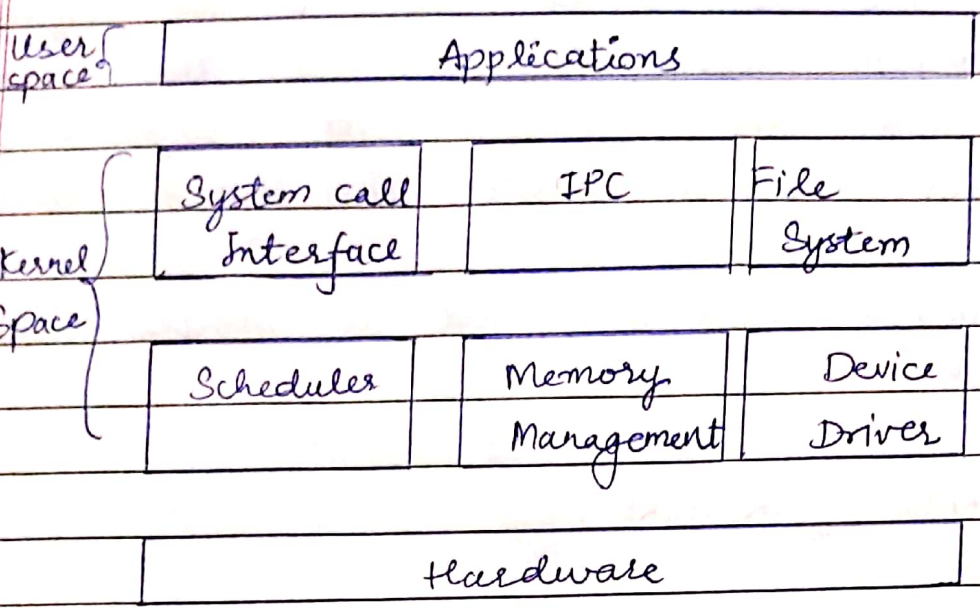
(3.) MICROKERNEL STRUCTURE :



Kernel: It is the core part of the OS, it manages the system resources. It is like a bridge between application & hardware of the computer.

Microkernel is the one in which user services and kernel services are kept in separate address space.

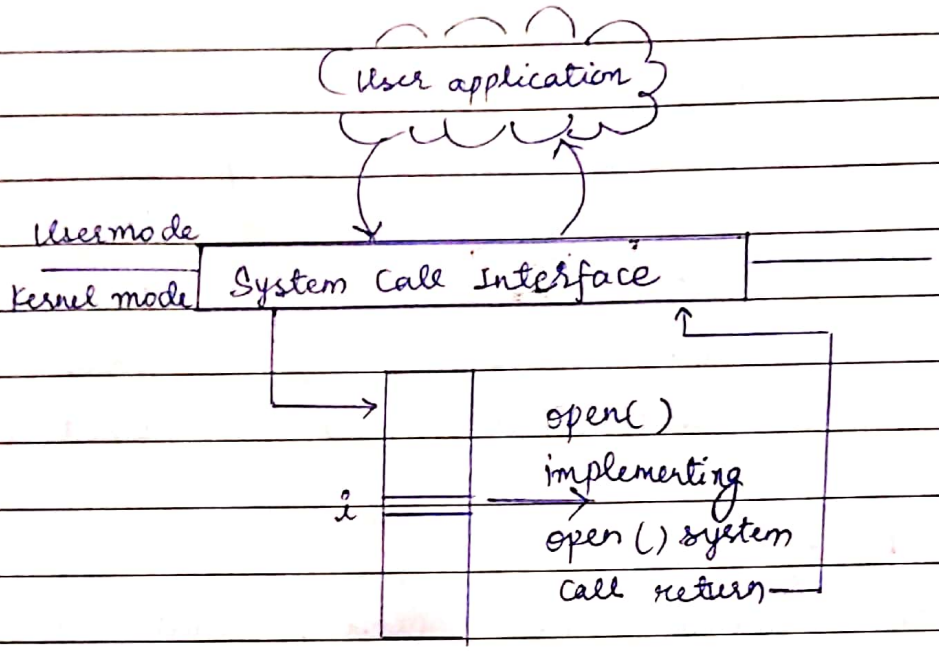
(4.) MONOLITHIC STRUCTURE (KERNEL):



⇒ DIFFERENCE B/W MICROKERNEL & MONOLITHIC

MicroKernel	Monolithic
1. User services & kernel services are kept in separate address space.	Both user services and kernel services are kept in same kernel space.
2. Microkernel are smaller in size.	Monolithic kernels are larges relatively.
3. Slower execution (It has all components in different address space, they have to communicate by IPC).	Faster execution relatively. (It has all components is same address space so system calls are faster).
4. It is easily extendable.	It is hard to extend.
5. If a service crashes, it doesn't affect on working of microkernel.	If a service crashes, the whole system will get crashed.
e.g: Linux, Windows 10, MacOS, iOS, QNX, L4 etc.	eg: Window 1993 or earlier.

⇒ SYSTEM CALLS



System call is a programatic way in which a computer programme requests a service from the kernel of the OS.

A system call is a method for programmes to interact with the OS. A computer program makes a system call when it makes a request to the OS's kernel.

→ SERVICES PROVIDED BY THE SYSTEM CALL :

1. Process Creation & Management
2. Main Memory Management
3. File Access, Directory Access & File System Management
4. Device Handling
5. Protection or Security
6. Networking

→ TYPES OF SYSTEM CALL :

1. PROCESS Control:

Create a process, execute a process, wait event, signal. e.g:

Windows	Linux
CreateProcess()	fork()
ExitProcess()	exit
WaitForSingleObject()	wait()

2. File Management:

Create, delete, open, close, read, write a file, etc.

eg:

Windows	Linux
Createfile()	open()
Readfile()	read()
Writefile()	write()

3. Device Management :

Request device, release device, read device, write device, logically attach or detach device.

4. Information Maintenance :

Get time or date, set time or date, get system data, set system data.

5. Communication :

create or delete communication connection, send or receive messages, transfer status information etc.

(Refer Galvin)

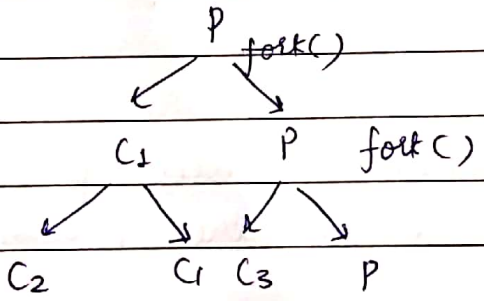
⇒ fork()

fork system call is used for creating a new process which is called 'child process' which runs concurrently with process (which is parent process). A child process uses the same program counter, same CPU registers, same open file which is used in the parent process.

It takes no parameter & returns an integer value.

- If value = -ve, creation of child process was unsuccessful.
- If value = 0, returns to the newly created child process.
- If value = 1, returns to parent process.

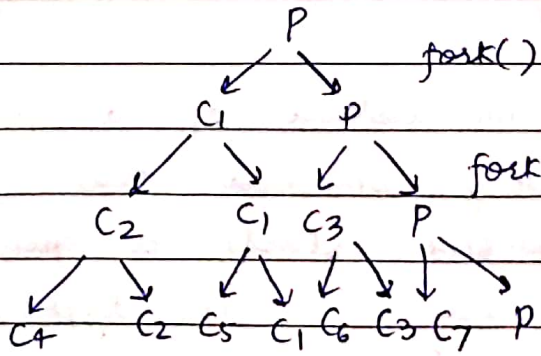
eg:
```
main()
{
fork();
fork();
printf("hello");
}
```



Total no. of child processes = 3
Total no. of processes = 4

Total no. of processes are $2^{\text{no. of times fork appears}}$
Total no. of child processes = $2^{\text{no. of times fork appears}} - 1$

eg:
```
main()
{
fork();
fork();
fork();
printf("Hi");
}
```



Total no. of processes = $2^n = 2^3 = 8$
Total no. of child proc. = $2^n - 1 = 7$

⇒ wait()

It suspends execution of the current process until a child has exited, or, until a signal has delivered whose action is to terminate the current process.

→ waitpid(): It suspends execution of current process until a child as specified by pid arguments has exited or until a signal is delivered.

⇒ exec()

It is a system call which is used to replace current process image with new process image.

⇒ exit()

This function is used for normal process termination. The status of the process is captured for future reference.

⇒ vfork() (Virtual fork)

This system call creates a child process & blocks parent process.

⇒ PROCESS RELATED TERMINOLOGY

1) Process: An instance of a program is called a process. Any command that you give to your Linux machine starts a new process.

Processes are of two types:

- Foreground Process: They run on the screen and need input from the user. e.g: MS Office

- Background Process: They run in the background & usually do not need any user input.
eg: Anti-virus

When a program is loaded into the memory & it becomes a process, it can be divided into four sections:

upgrade & downgrade space	Stack	→ Storage
	↑ ↓	dynamic
	Heap	
	Data	SV & GV (Static & Global variables)
	Text	a.out (execution)

- Stack: The process stack contains the temporary data such as method, function, parameters, return address & local variables.

- Heap: This is dynamically allocated memory to a process during its run time / execution time.

- Data: This section contains global & static variables.

- Text: This includes the current activity represented by the value of program counter & the contents of processor register.

2.) PROCESS CONTROL BLOCK: PCB is a data structure maintained by the OS for every process. The PCB is identified by an Integer process id (pid). A PCB keeps all the information needed to keep

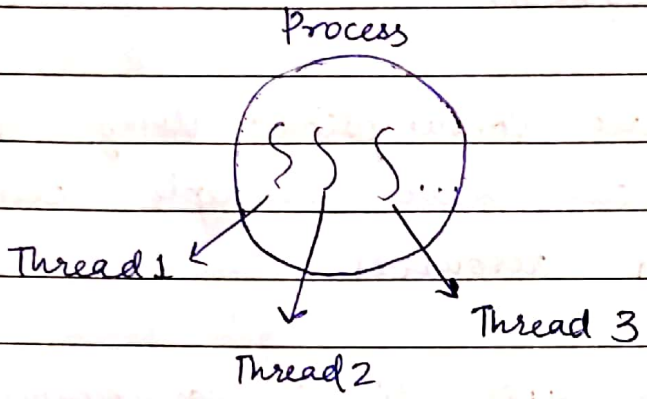
track of the process.

Process ID
State
Pointer
Priority
Program Counter
CPU Registers
I/O Information
Accounting Info

- Process ID : Unique identification for each of the process in the OS.
- Process State : The current state of the process, i.e. whether it is ready, running, waiting or whatever.
- Pointer : A pointer to parent process.
- Program Counter : PC is a pointer to the address of the next instruction to be executed for this process.
- CPU Registers : Various CPU registers where process need to be stored for execution for running state.
- I/O status info : This includes a list of I/O devices allocated to the process.
- Accounting info : This includes amount of CPU used for

process execution, time limit, execution ID etc.

3) THREAD : A thread of execution is the smallest sequence of program instructions that can be managed independently by a scheduler, which is the part of the OS.



⇒ THREAD VS PROCESS

1. Processes are independent while threads exist as subset of a process.
2. Processes have more state information while multiple threads within a process share process state as well as memory.
3. Processes have separate address space, whereas, threads share their address space.
4. Context switching b/w processes is slower as compared to threads.

⇒ MULTITHREADING : It allows multiple threads to exist within the context of 1 process. It can also be applied to one process to enable parallel execution on a multiprocessing system.

⇒ ADVANTAGES

1.) Responsiveness : It allows an application to remain responsive to input.

2.) Faster Execution : It allows to operate fast on computer system.

3.) Lower Resource Consumption : Using threads, an application can serve multiple clients concurrently using fewer resources.

4.) Better System Utilization : For example, a file system using multiple threads can achieve higher throughput.

5.) Simplified Sharing & Communication : Unlike processes, which require a message passing or shared memory mechanism to perform interprocess communication, threads can communicate through data, code & files.

6.) Parallelization : Using multithreading where hundreds of threads run in parallel across large number of CPUs.

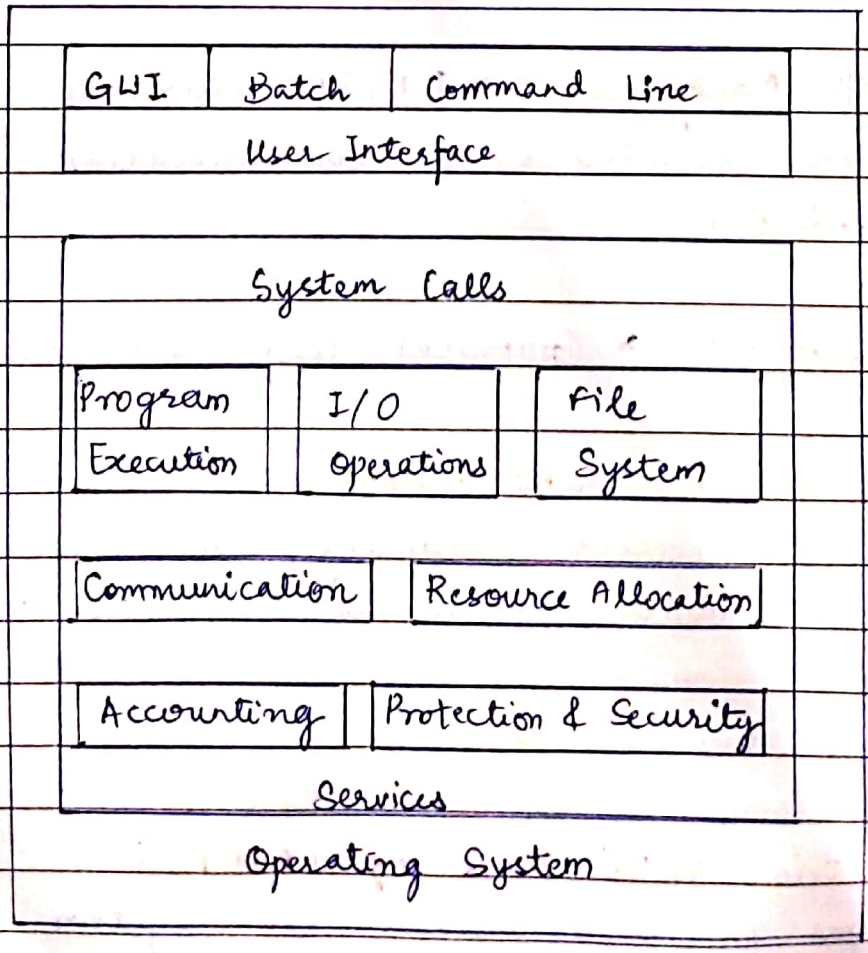
⇒ DISADVANTAGES

1.) Synchronization
2.) Threads can crash a process.

⇒ TYPES OF THREAD

User level Threads	Kernel Level Threads
1.) User threads are implemented by the user.	Kernel threads are implemented by the OS.
2.) OS doesn't recognize user level threads.	Kernel threads are recognized by OS.
3.) Implementation is easier.	Implementation is difficult relatively.
4.) Context switching time is very less.	Context switching time is more.
5.) If one user level thread performs blocking operation then entire process will be blocked.	If one kernel level thread performs blocking operation then another thread can continue execution.

⇒ OPERATING SYSTEM SERVICES AND COMPONENTS



Date / /
Page No.
Shivalal

- **Process Management :** > Process creation, deletion
  > Process Suspension, Resumption.
  > Process Synchronization
  > Process Communication

- **Memory Management :** > Maintain book keeping information
  > Mapping of process to memory locations
  > Allocate & deallocate memory space as requested.
  > I/O device management

- **I/O Device Management :** Disc management functions, storage allocation, fragmentation

- **File System :>** File creation, deletion, updation
  > Mapping of files to secondary storage

- **Protection :** > User Authentication

- **Network Management :** > TCP / IP protocols
  > Connection strategies & communication mechanism
  > User Interface

- **User Interface :** > Graphical User Interface

⇉ MULTIPROCESSOR SCHEDULING

On a multiprocessor scheduling, the scheduler has to decide which process to run and which CPU to run it on.
It is of two dimensions.
Multiprocessing is of two types:

1. **Assymetric Multi Processing (ASMP):**
   Only one processor accesses the system data structures,

reduces the need of data sharing.

2. **Symmetric Multi Processing (SMP):**
Each processor has self scheduling, all processes in common ready queue or each has its own private queue of ready processes.

→ **PROCESSOR AFFINITY :** It enables the binding & unbinding of a processor to a CPU or range of CPUs so processor will execute only on the designated CPU, rather than any other CPU.

- **Hard Affinity** enables processes to run only on a fixed set of one or more processors.

- **Soft Affinity :** The OS will attempt to keep a process on a single processor but it is possible for a process to migrate b/w processors.

Load Balancing
Push & Pull migration
Multithreading ⇒ 1 to 1, 1 to many, many to 1, many to many