UNIT-1

Operating system →

→ Interface b/w user and Hardware.

→ Resource manager and allocator.

→ Resources are :

1. memory (RAM)
2. CPU
3. I/O devices like HDD, keyboard, mouse etc.

Goals → [performance (Effeciency)] vs [convenience]

performance (Effeciency) → developers looks for

convenience → normal users looks for

Types of operating system →

(1) Batch OS →

→ non-interactive

→ Tasks are called as JOBS

→ JOBS are submitted to system and no interaction thereafter.

→ starvation may sufferad by system (effeciency may be low).

→ no-preemption is there i.e. once a JOB is executed it can't be suspended.

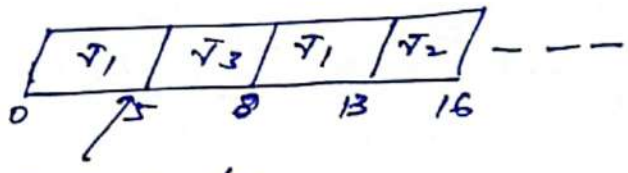
(2) Multi-programming OS →

→ Extension / modification of Batch OS.
→ Interactive but again pre-emption is not there.
→ Jobs / Tasks can be provided as well as i/p can be provided in-between the execution.

(3) Time-sharing / multi-tasking OS →

→ pre-emptive version of multi-programming OS.

T_1 — 10 ms
T_2 — 8 ms
T_3 — 7 ms



T_1 needs I/O
so switched to T_3

→ more complex than multi-programming.

(4) Multi-processing OS →

→ multiple processors are there.
→ Jobs are executed parallely on multiple processors
→ Effeciency is increased.
→ throughput is improved
↳ no. of processes executed per unit time

(5) Real-Time OS → strict time limits and Jobs must be finished within the defined time limits.
→ can be Hard real-time or soft real-time.

Types of Processes →

(1) CPU-Bound process → which requires more CPU time and less I/O time.

(2) I/O-Bound process → which requires more I/O time and less CPU time.

Process management

Program —

→ static entity.

→ piece of code that we write.

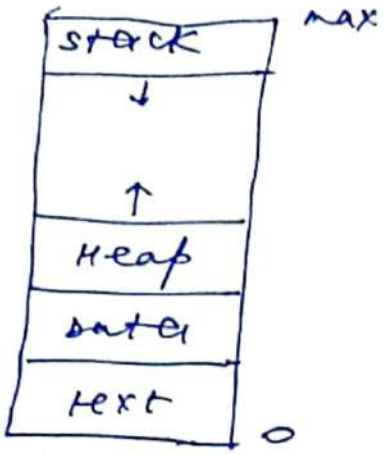
Process —

→ A program in execution

→ Process is created as soon as program is loaded into memory by OS.

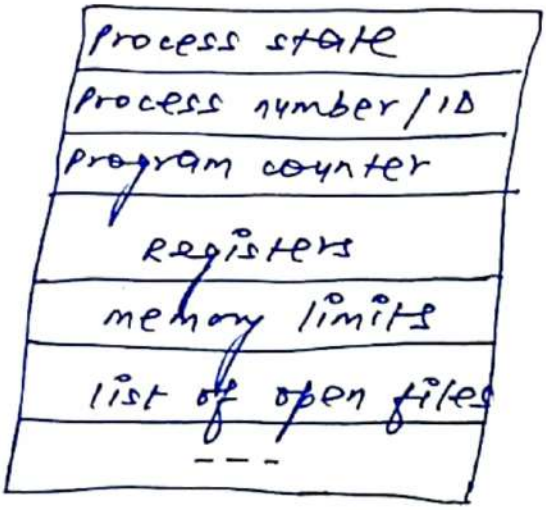
→ Every process has a PCB (Process control Block) or context.

→ Process includes code (text section), current activity represented by program counter, contents of processor registers, stack memory (fn. parameters, return addresses and local variables), data section, heap memory (for Dynamic allocated memory).



→ when a process breaches the memory limits, it is called as segmentation fault.

Process control BLOCK (PCB) →



→ other information in PCB is —

(1) CPU-scheduling information — process priority, pointers to scheduling queues and other scheduling parameters.

(2) memory-management information — values of base and limit registers, paging/segmentation tables etc.

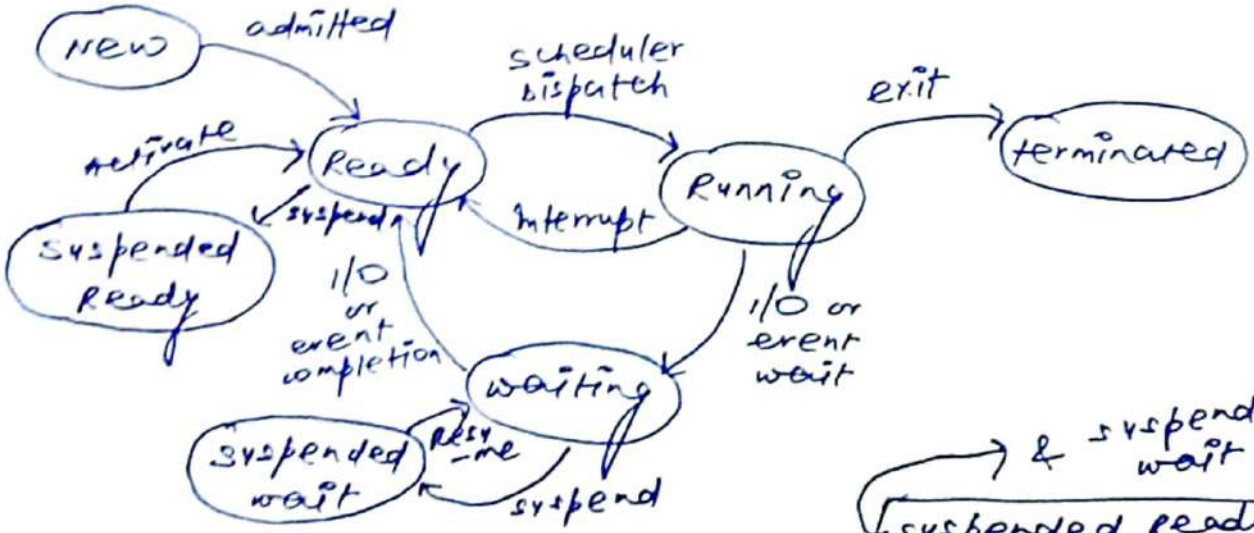
(3) Accounting information — information related to amount of resources used and times for they

have been used.

④ I/O status information → list of I/O devices allocated to process, list of open files etc.

→ Process ID – Allocated as soon as process is created. It is unique.

Process states → state of a process is defined by the current activity and it changes during the execution.



new – Process is being created.

Running – Instructions are executed by CPU (execution Queue)

Ready – Process is waiting to be assigned to processor. (Ready Queue is maintained)

waiting – waiting for some event to be completed. (waiting Queue is maintained)

Terminated – Process has finished execution.

→ minimum no. of states required by any process for

completion are 4 (new → ready → execution → terminated). this is true only for those processes which do not need I/O.

CPU schedulers →

(1) Long-term schedulers → Also called JOB scheduler.

→ long term scheduler ~~takes~~ selects the processes from a pool (maintained in HDD) and loads them into the ready queue.

→ often used in system where more processes are submitted than kept in ready queue.

(2) Short-term scheduler → Also called CPU scheduler.

→ selects processes from ready queue and allocates the CPU for execution.

(3) medium-term scheduler → used in some OS like time-sharing to reduce the degree of multiprogramming.

→ It takes out the processes from the ready queue so that the processes do not remain in contention for CPU.

→ this activity of taking process out of ready queue and later putting back is called as swapping.

OS services →

(1) User interface → Provides user interface in form of GUI or CLI.

(2) Program execution → It helps in the execution of programs by loading them into memory and provides other resources.

(3) I/O operations

(4) File system management → Helps in organizing files in the system.

(5) IPC (Inter-process communication) → Helps in facilitating comm^n b/w processes in the system.

(6) Error detection → Helps in detecting any error during operation related to CPU, memory, error in user program

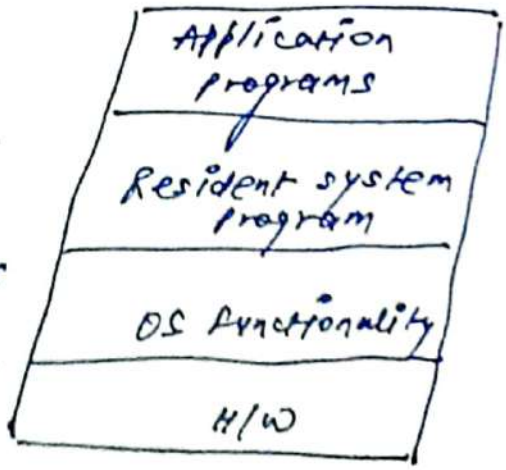
(7) Resource Allocation

(8) Accounting.

OS structure →

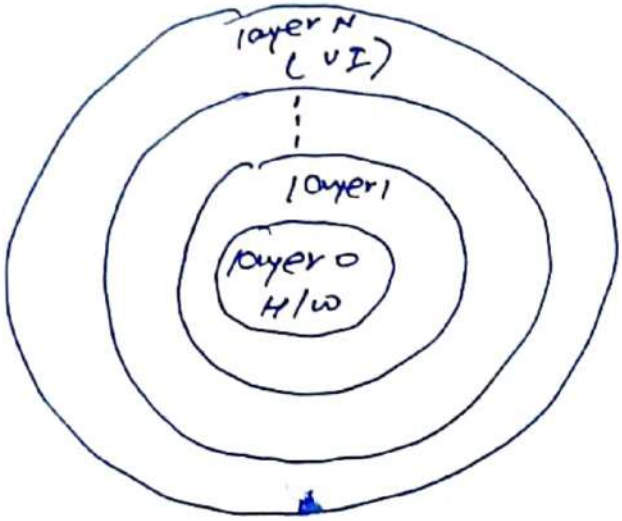
(1) Simple structure →

→ used in small and simple systems.
→ not divided into layers.
→ Examples are MS-DOS & original UNIX OS.
→ only one layer is responsible for all the functionalities.



(2) layered Approach →

→ OS broken into smaller pieces [OS components]
→ easy to debug and manage.
→ Bottom layer is H/w & highest layer is user interface
→ Difficulty is only how to define the layers.



(3) Micro-kernel Approach →

→ Difficult to manage the kernel when. becomes too large.
→ Also a centralized structure impacts the overall performance of kernel.
→ first system was mach developed around 1985 based on micro-kernel approach.
→ It removes all the non-essential components from the kernel and implements them as system & user-level programs.
→ only the main functionality is there with kernel along with communication with the help of msg passing.

UNIX commands →

(1) Internal commands → built-in into the shell. [works faster than External]

e.g:- cd, fg etc.

(2) External command → not built into the shell. when an external command is to be executed, shell looks for its path given in PATH variable.

e.g:- ls, cat etc.

$ help — list all the internal commands.

$ type command_name — to find whether a command is internal or external.

Important commands in LINUX / UNIX →

(1) $ ls — list of files
(2) $ mkdir — creates a file
(3) $ cd — change directory
(4) $ pwd — present working directory
(5) $ who — gives the current users
(6) $ date — current date and time
(7) $ gedit filename — open geditor (GNOME text editor)
$ emacs " — " emacs editor
(8) $ rm — removes a file

(9) $ ls filename — finds file in entire system

(10) $ rmdir — removes a directory

(11) $ touch — creates a file

(12) $ cp source destination — copies a file from source to destination

(13) $ mv source destination — moves a file from source to destination

(14) $ file — displays the type of file / directory (Linux treats everything as a file)

(15) $ man command_name — displays manual for the given command.

System calls →

→ Provides an interface b/w application programs and OS to request for the services provided by OS.

→ Generally available as routines written in C/C++ or assembly languages.

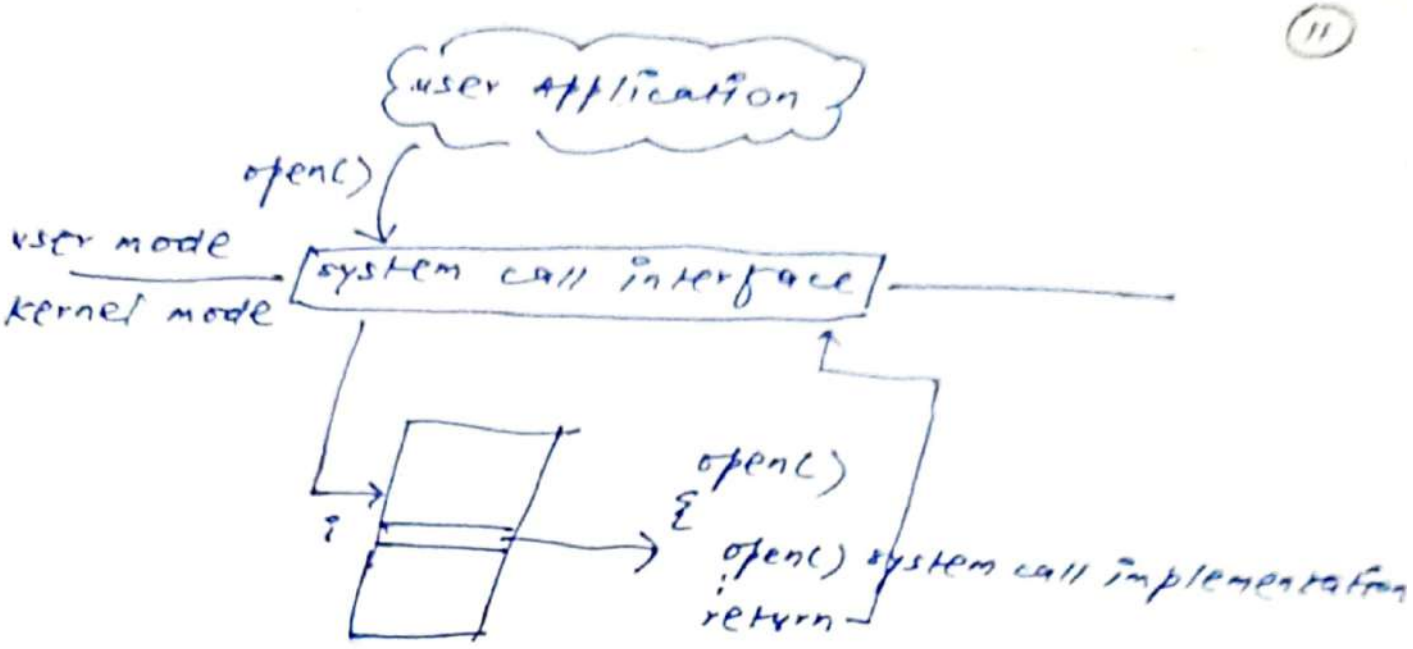
→ To copy the data from one file to other.

(1) sequence of system calls to first have the name of i/p & o/p file and display of prompt msg.

(2) system calls to read the content of file.

(3) To handle the error if there is no i/p file.

(4) and so on.



Types of system calls →

① Process control → load, execute, abort, fork, wait exit, etc.

② File related → open, read, write, close etc.

③ Information → get PID, attributes, get system time and date etc.

④ Communication → Pipe, create/delete connection, send msg, receive msg etc.

* There are typically around 125 system calls in LINUX.
* " " around 700 system calls in windows.

⑤ Device management → request device, release device, read, write, reposition, get/set device attributes etc.

⑥ Protection → chmod(), umask(), chown()

umask() → set file mode creation mask (default permission for created files)

① fork() → fork() is used to create a child process by an existing process. [unistd.h]

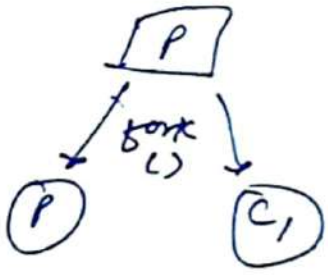
→ fork() → 0 [return to child process]
→ +ve [PID of child to parent process]
→ -ve [child process is not created]

→ if 'n' is the no. of fork(), then no. of child process created are $2^n - 1$ and 1 parent

→ if 2 fork are used then 3 child and 1 parent are there.

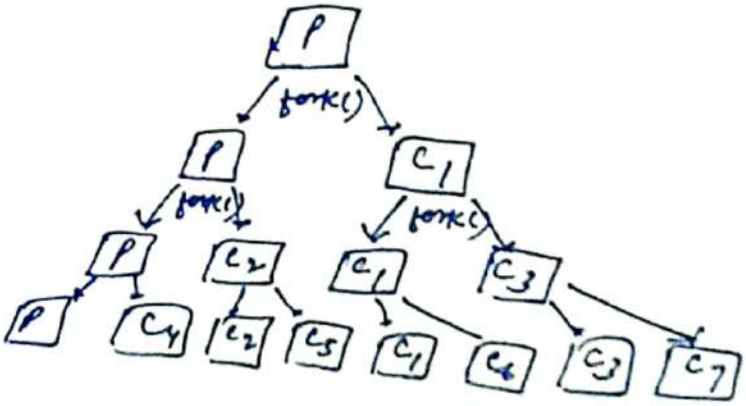
e.g. ⇒

①
```
main()
{
  fork();
  printf("Hello");
  return;
}
```



so, it will print Hello two times.

②
```
main()
{
  fork();
  fork();
  fork();
  printf("Hello");
}
```



* It will print "Hello" 8 times.

(3)
```
main()
{
  if( fork()==0)
    printf(" child process");
  else
    printf(" parent process");
  return 0;
}
```

o/p

parent process
child process

or

child process
parent process

(4)
```
main()
{ int x = 10;
  if( fork()==0)
    printf(" child has x=%d", ++x);
  else
    printf("parent has x=%d", --x);
  return 0;
}
```

o/p

child has x=11
parent has x=9

or

parent has x=9
child has x=11

(3)
```
main()
{
  fork();
  fork() && fork() || fork();
  fork();
  printf(" forked");
  return 0;
}
```

o/p

forked — 20 times

(6)
```
main()
{
  for(i=0; i<4; i++)
    fork();
  printf(" Hello");
  return 0;
}
```

o/p

Hello — 16 times

(6) (7)

```
main()
{
    fork();
    printf("Hi");
    fork();
    printf("Hello");
    return 0;
}
```

o/p

Hi
Hi
Hello
Hello
Hello
Hello

Functions of operating system →

1. CPU / process scheduling
2. Memory management
3. File management
4. Disk scheduling / secondary memory management
5. I/O device management
6. User interface implementation
7. Resource Allocation
8. Accounting
9. Error detection
10. Handling network connections