

Introduction to System software

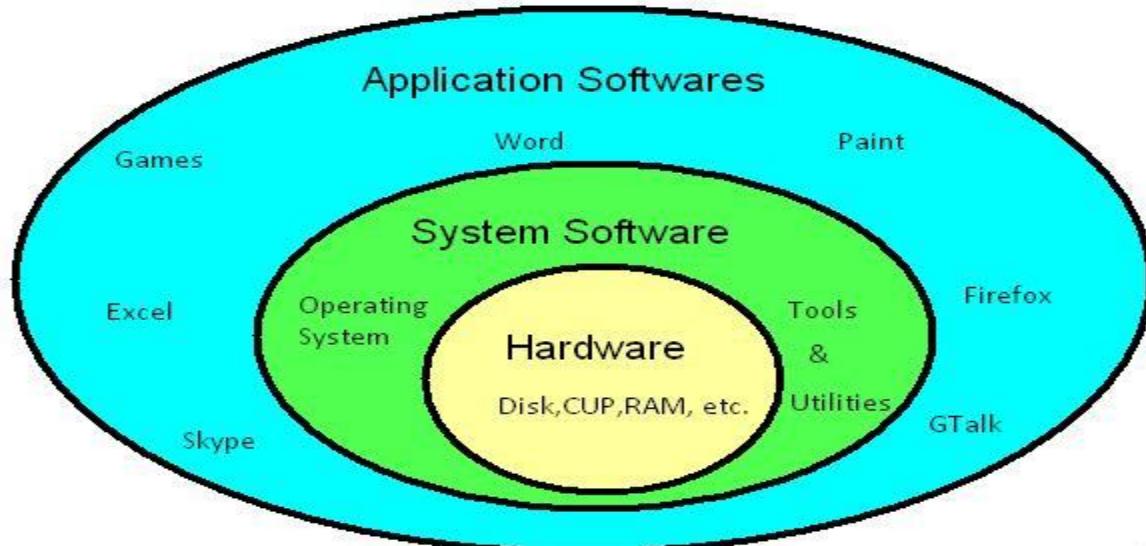
Module 1

System programming

- **System Programming** can be defined as act of building Systems Software using System Programming Languages.

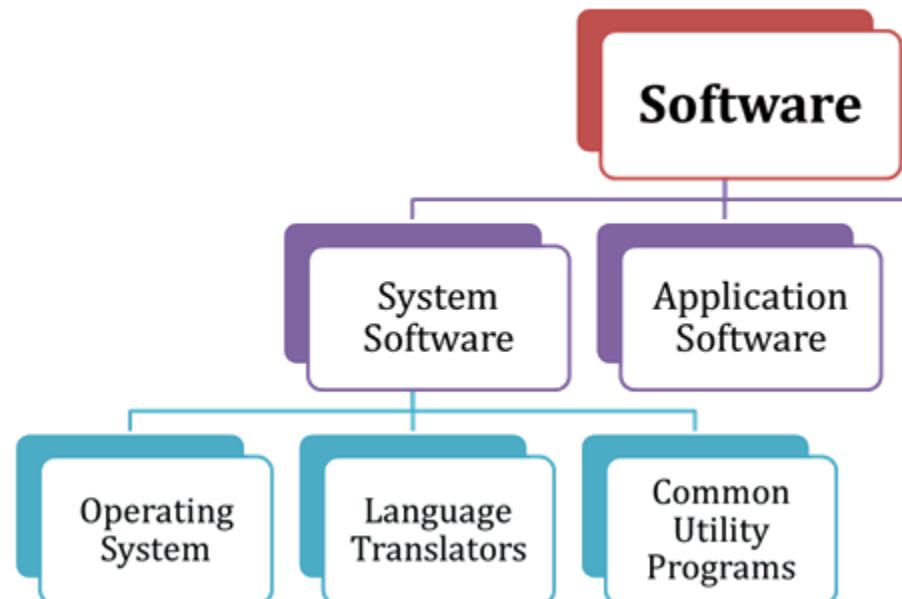
System programming

- According to Computer Hierarchy, one which comes at last is Hardware. Then it is System Programs, and finally Application Programs.
- System software serves as the **interface between the hardware and the end users.**



System programming

- There are two main types of software: systems software and application software.
 - Systems software includes the programs that are dedicated to managing the computer itself.
 - System Software is a set of programs that control and manage the operations of computer hardware. It also helps application programs to execute correctly.



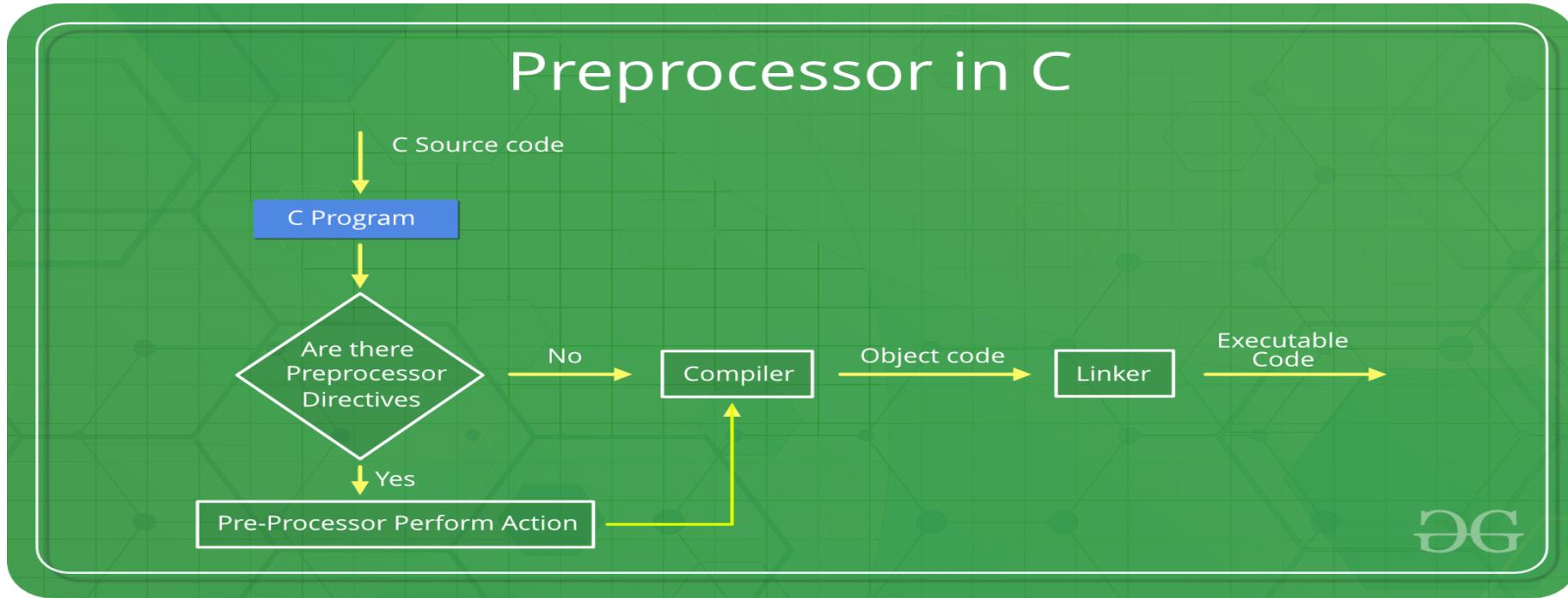
System Programs

Various System Programs are :

- Assemblers
- Loaders
- Linkers
- Macro processor
- Compiler
- Interpreter
- Operating System
- Device drivers

Macro Processors or Pre Processors

- As the name suggests Preprocessors are programs that process our source code before compilation.



Macro Processors or Pre Processors

There are 4 main types of pre-processor directives:

- Macros
- File Inclusion
- Conditional Compilation
- Other directives

Macros

- Macros are a piece of code in a program which is given some name.
 - Whenever this name is encountered by the compiler
 - The compiler replaces the name with the actual piece of code.
 - I.E. It replaces the name with the definition of the macro
- The '#define' directive is used to define a macro.
- Macro definitions need not be terminated by semi-colon(;).

Macros

C program to illustrate macros

```
#include <stdio.h>

// Macro definition
#define LIMIT 5
int main()
{
    // Print the value of macro defined
    printf("The value of LIMIT is %d", LIMIT);
    return 0;
}
```

Output-

The value of LIMIT is 5

File Inclusion

- This type of preprocessor directive tells the compiler to include a file in the source code program.
- There are two types of files which can be included by the user in the program:
 - Header File or Standard files
 - User defined files

File Inclusion

Header File or Standard files:

- These files contains definition of pre-defined functions like `printf()`, `scanf()` etc. These files must be included for working with these functions.
- Different function are declared in different header files.
 - For example C++ standard I/O functions are in ‘`iostream`’ header file whereas
 - Functions which perform string operations are in ‘`string`’ header file.

File Inclusion

Header File or Standard files:

- **Syntax: #include<*file_name*>**
 - where *file_name* is the name of file to be included.
 - The ‘<’ and ‘>’ brackets tells the compiler to look for the file in standard directory.

File Inclusion

User defined files:

- When a program becomes very large, it is good practice to divide it into smaller files and include it whenever needed. These types of files are user defined files.
- These files can be included as: `#include "filename"`

File Inclusion

#include<filename.h>

- By using this syntax, when we are including header file then it will be loaded from default directory i.e. C:\TC\INCLUDE.
- Generally, by using this syntax we are including pre-defined header files.

#include “filename.h”

- Generally, by using this syntax we are including user-defined header files.
- It is loaded from the current working directory.

Conditional Compilation Directives

- Are type of directives which helps
 - To compile a specific portion of the program or
 - To skip compilation of some specific part of the program
 - Based on some conditions.
- This can be done with the help of two preprocessing commands '**ifdef**' and '**endif**'.

Conditional Compilation Directives

Syntax:

```
#ifdef macro_name
```

```
statement1;
```

```
statement2;
```

```
statement3;
```

```
.
```

```
.
```

```
.
```

```
statementN;
```

```
#endif
```

- If the macro with name as '*macroname*' is defined
 - Then the block of statements will execute normally
 - Else the compiler will simply skip this block of statements.

Conditional Compilation Directives

```
#include <stdio.h>

#define YEARS_OLD 10

int main()
{
#define YEARS_OLD
    printf("TechOnTheNet is over %d years old.\n", YEARS_OLD);
#undef YEARS_OLD
    printf("TechOnTheNet is a great resource.\n");
    return 0;
}
```

Output

```
/tmp/PG7rwWF95Ud.o
TechOnTheNet is over 10 years old.
TechOnTheNet is a great resource.
```

- In the C Programming Language, the #ifdef directive allows for conditional compilation.
- The preprocessor determines if the provided macro exists before including the subsequent code in the compilation process.

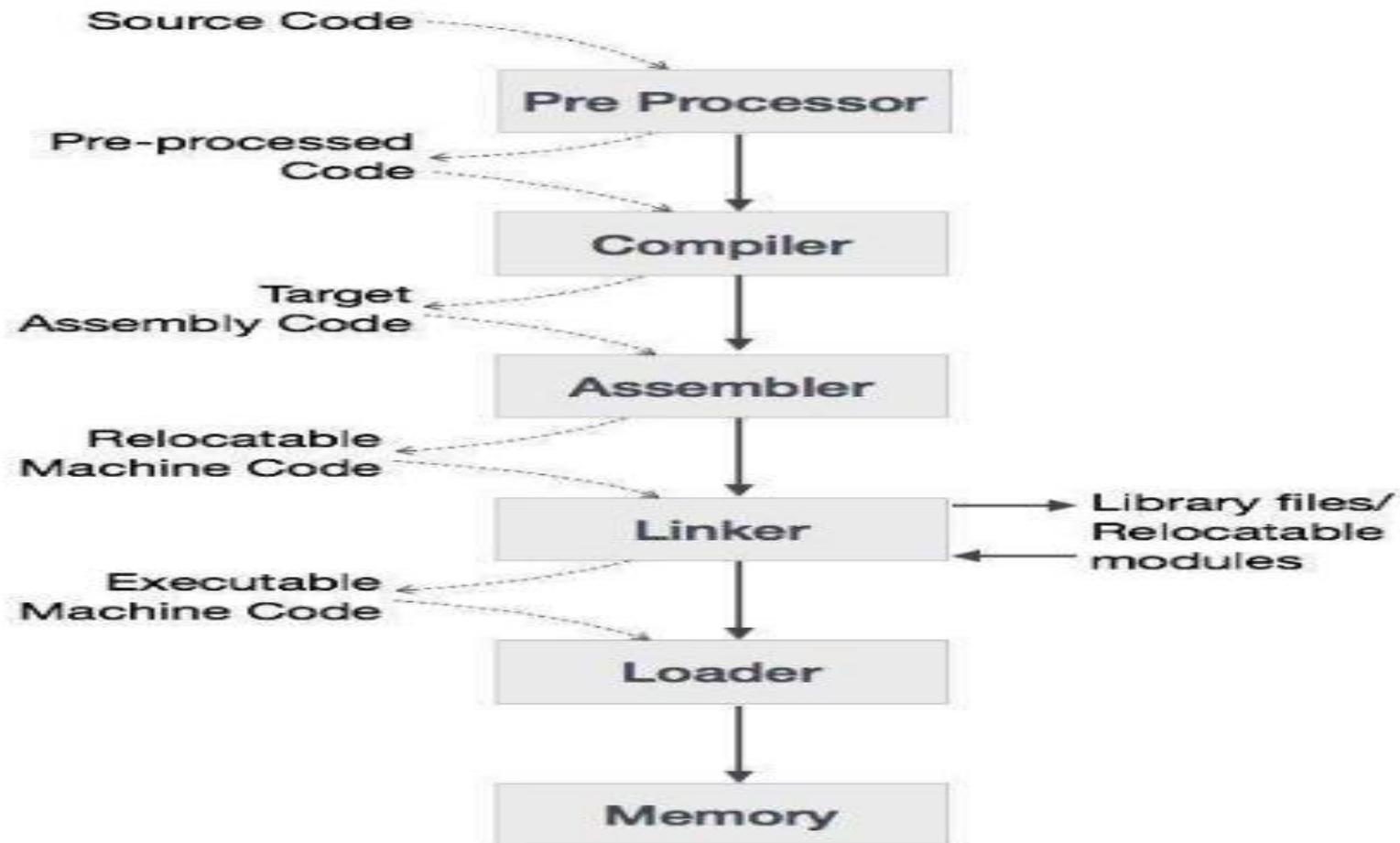
Other directives

- Apart from the above directives there are two more directives which are not commonly used.
- These are:
 - **#*undef* Directive**
 - **#*pragma* Directive**

Other directives

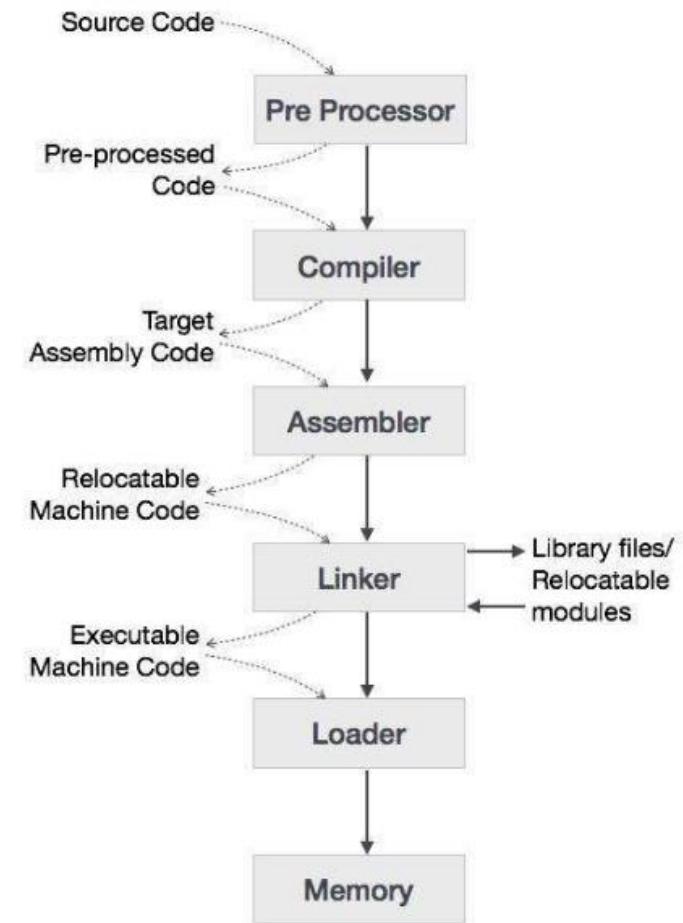
- **#undef Directive:**
 - The #undef directive is used to **undefine an existing macro**. This directive works as:
 - #undef LIMIT
 - Using this statement will **undefine** the existing macro LIMIT.
 - After this statement every “#ifdef LIMIT” statement will evaluate to **false**.
- **#pragma Directive:**
 - This directive is a special purpose directive and is used to **turn on or off some features**.
 - This type of directives are compiler-specific, i.e., they vary from compiler to compiler.

How a program, using C compiler, is executed on a host machine ?

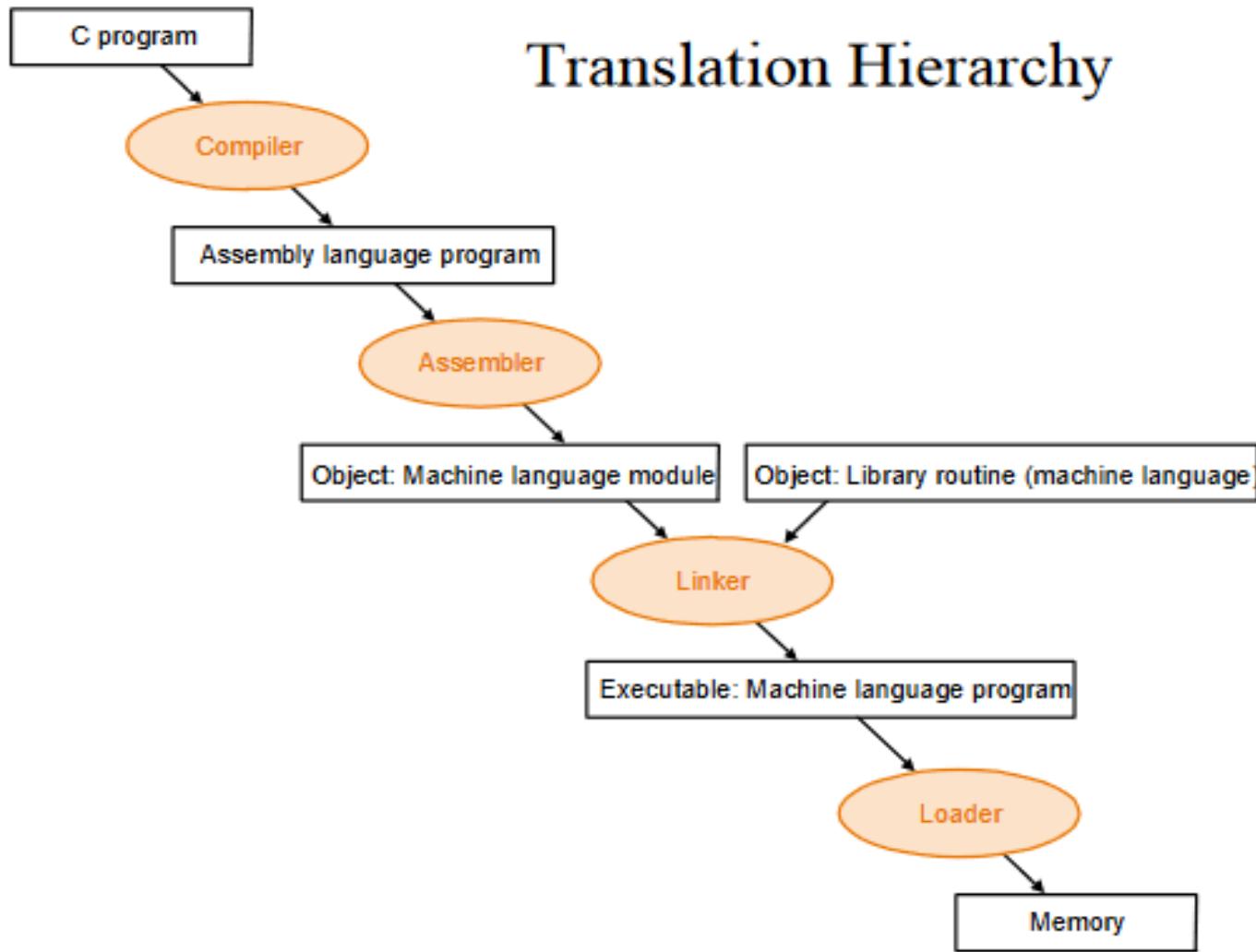


How a program, using C compiler, is executed on a host machine ?

- User writes a program in C language (high-level language).
- The C compiler, compiles the program and translates it to assembly program.
- An assembler then translates the assembly program into machine code (object file).
- A linker tool is used to link all the parts of the program together for execution (executable machine code).
- A loader loads all of them into memory and then the program is executed.

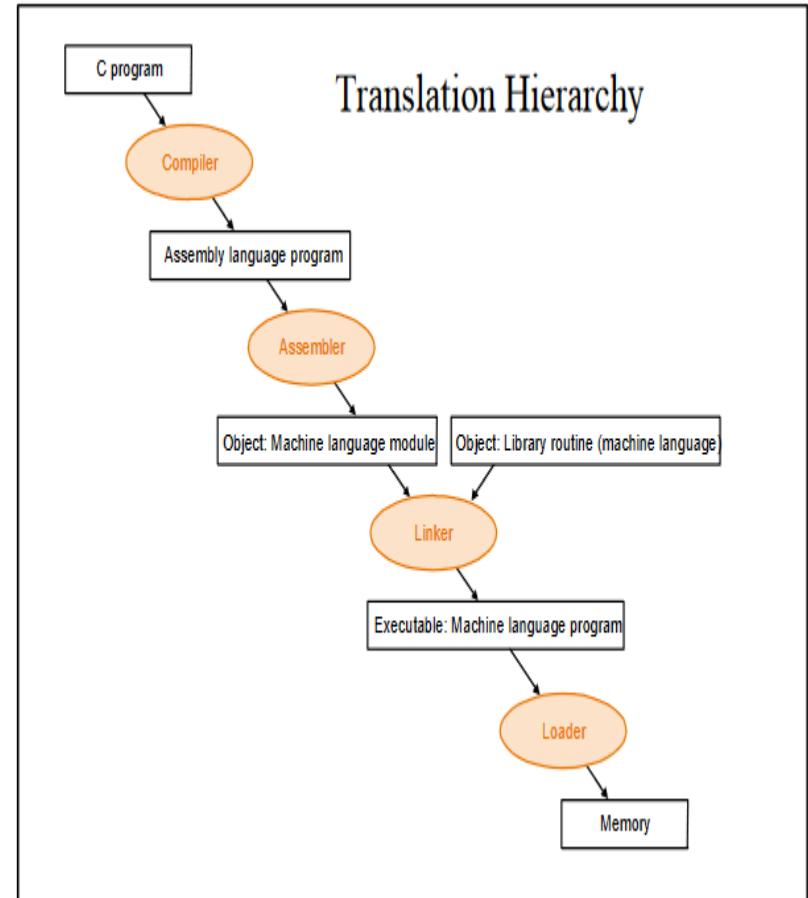


Translation Hierarchy



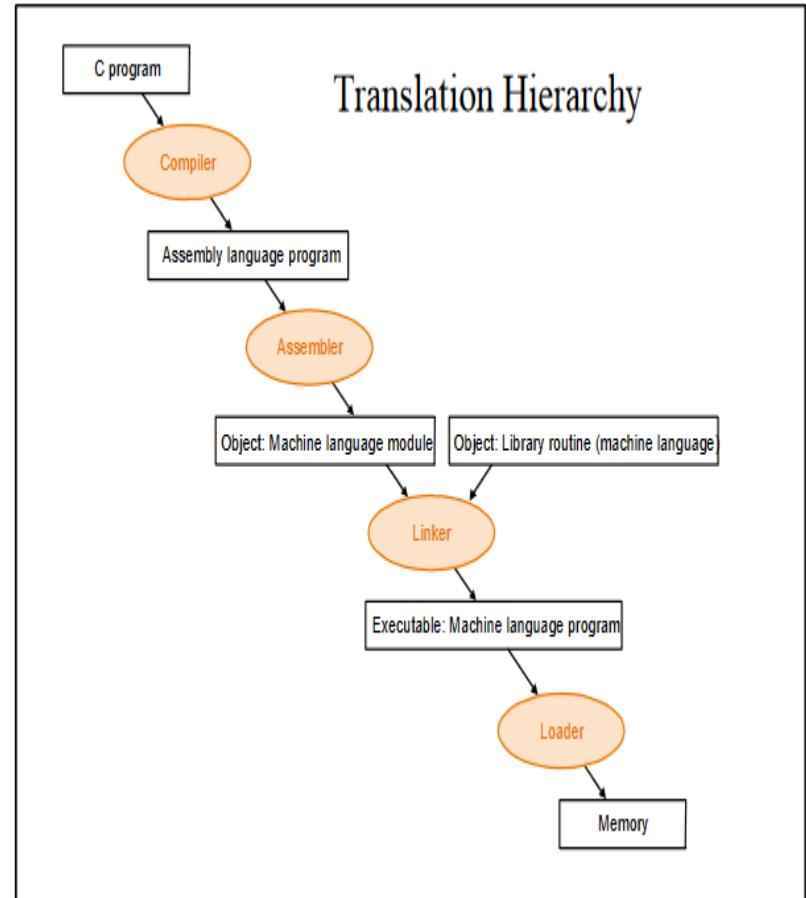
Compiler

- A compiler is a software that
 - translates the code written in one language to some other language
 - without changing the meaning of the program.



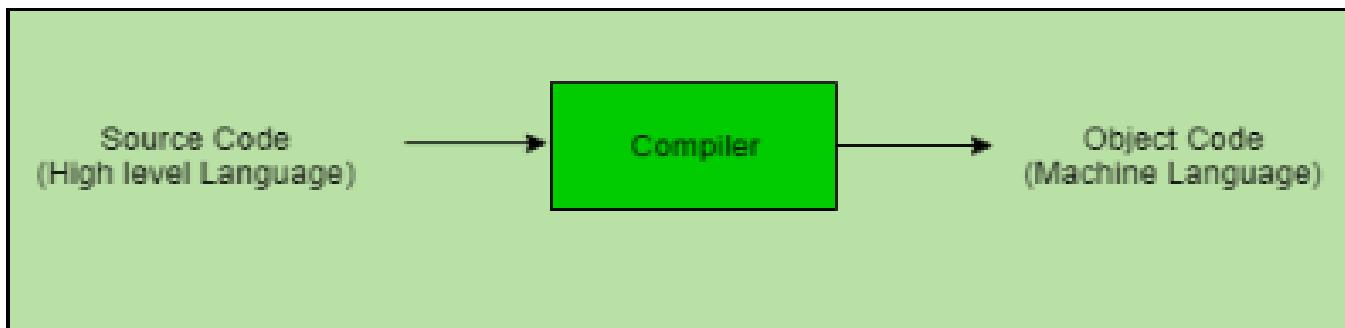
Compiler

- The compiler is also said to
 - make the target code efficient and
 - optimized in terms of time and space.
- In this example , “C” compiler converts high-level language program into assembly language



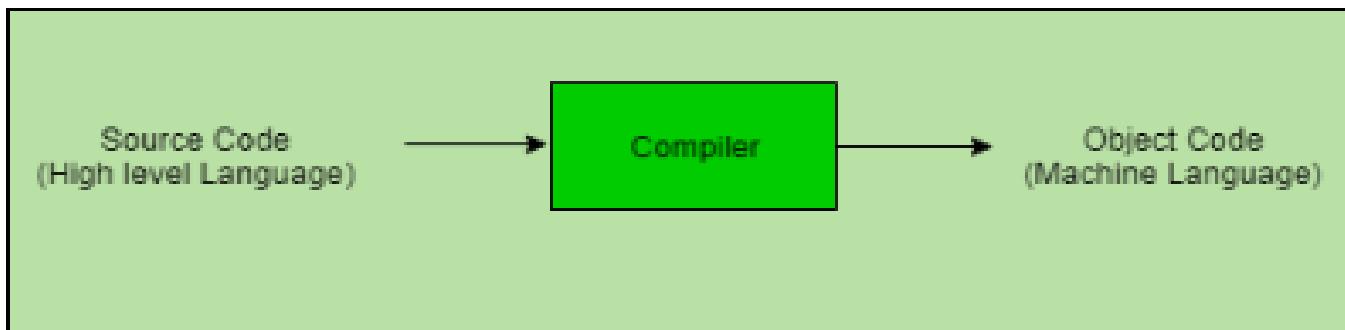
Compiler

- The language processor that reads the complete source program written in high level language as a whole in one go and translates it into an equivalent program in machine language is called as a Compiler.



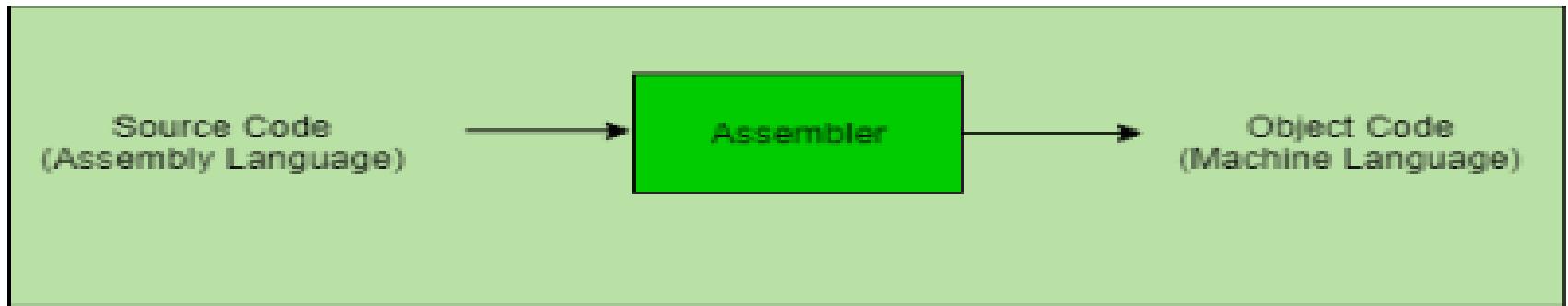
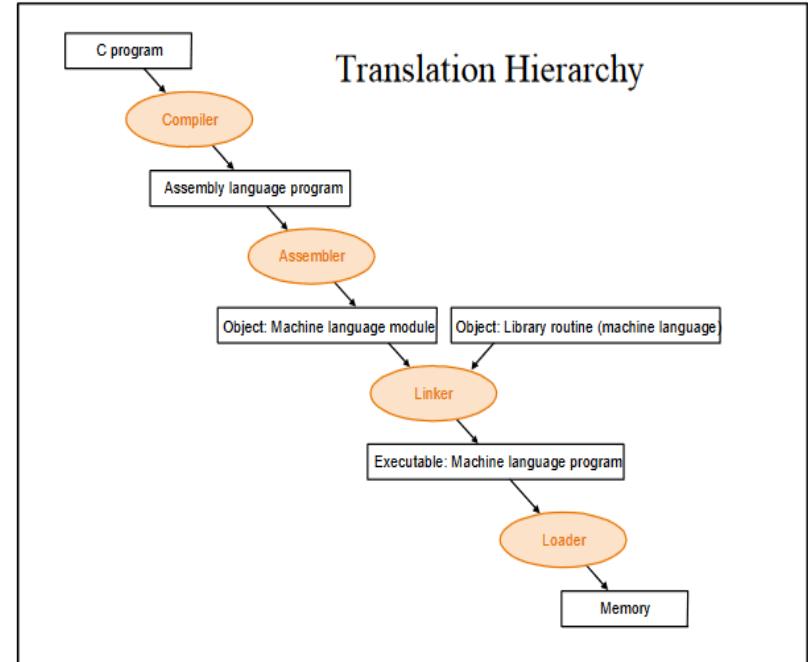
Compiler

- In a compiler, the source code is translated to object code successfully if it is free of errors.
- The compiler specifies
 - the errors at the end of compilation with line numbers when there are any errors in the source code.
 - The errors must be removed before the compiler can successfully recompile the source code again.



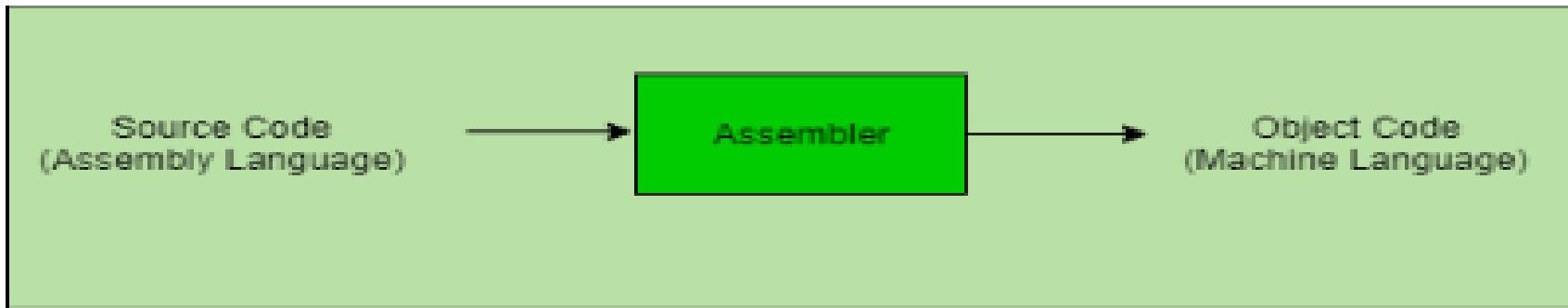
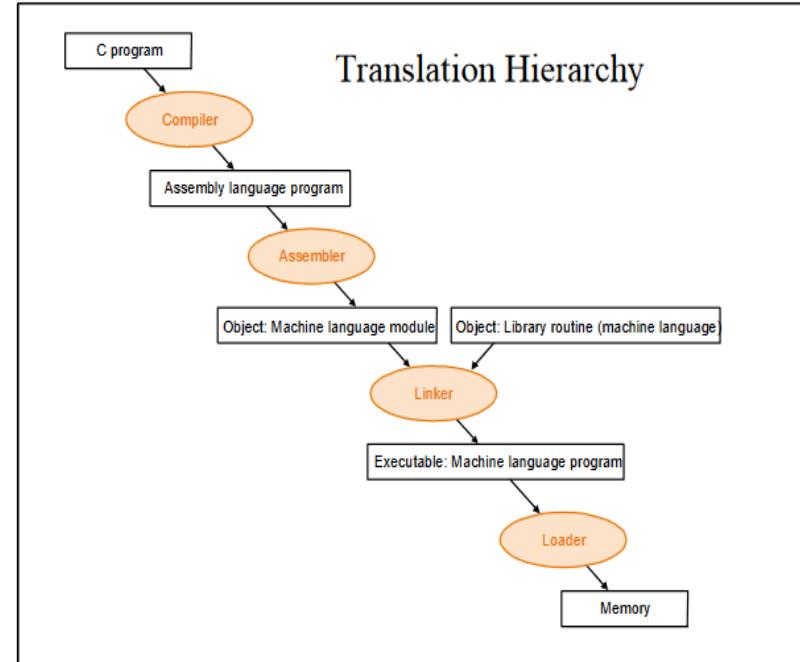
Assembler

- A program that converts assembly language programs into object files

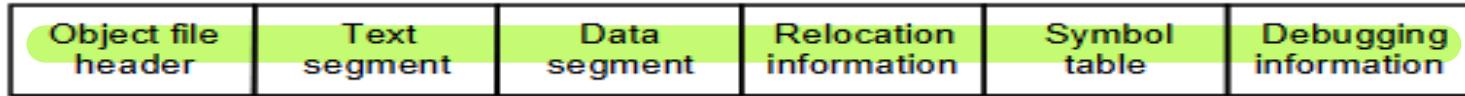


Assembler

- The output of an assembler is called an object file
- Object files contain a combination of
 - machine instructions,
 - data, and
 - information needed to place instructions properly in memory.



Object file format



- Object file header describes the size and position of the other pieces of the file
- Text segment contains the machine instructions
- Data segment contains binary representation of data in assembly file
- Relocation info identifies instructions and data that depend on absolute addresses
- Symbol table associates addresses with external labels and lists unresolved references
- Debugging info

Interpreter

- An interpreter, like a compiler, translates high-level language into low-level machine language.

Interpreter

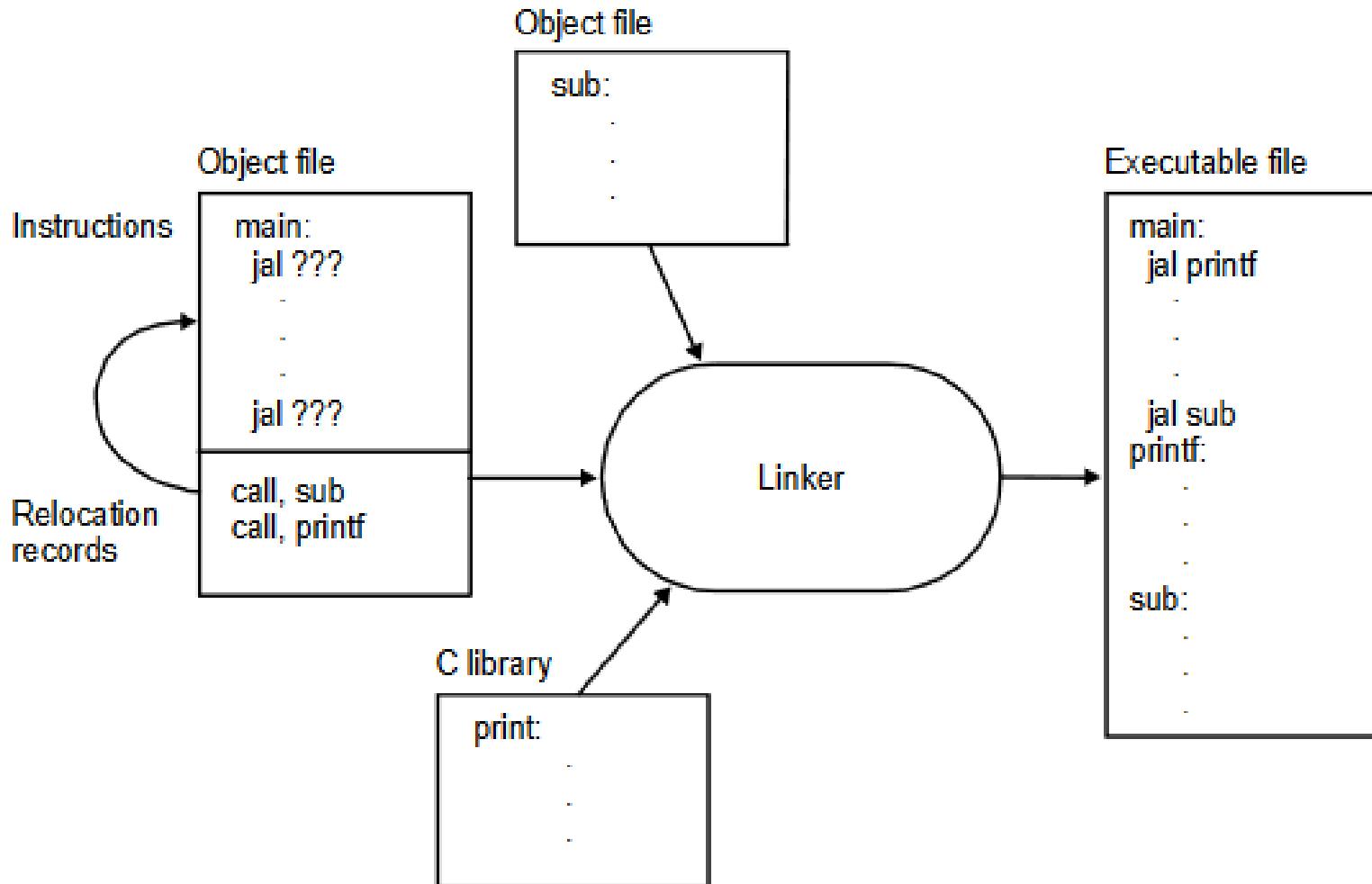
- The difference lies in the way they read the source code or input.
 - A compiler reads the whole source code at once, creates tokens, checks semantics, generates intermediate code, executes the whole program and may involve many passes.
 - An interpreter reads a statement from the input, executes it, then takes the next statement in sequence.

Interpreter

- If an error occurs,
 - an interpreter stops execution and reports it. The interpreter moves on to the next line for execution only after removal of the error.
 - whereas a compiler reads the whole program even if it encounters several errors.
- **Example:** Perl, Python and Matlab.

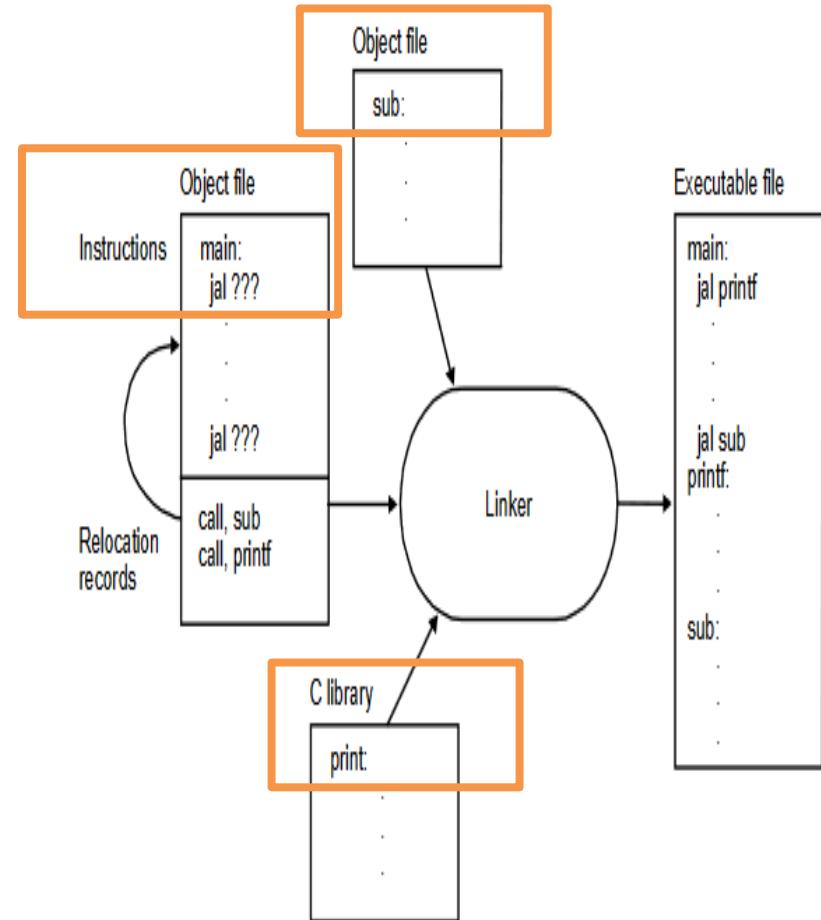
COMPILER	INTERPRETER
A compiler is a program which converts the entire source code of a programming language into executable machine code for a CPU.	interpreter takes a source program and runs it line by line, translating each line as it comes to it.
Compiler takes large amount of time to analyze the entire source code but the overall execution time of the program is comparatively faster.	Interpreter takes less amount of time to analyze the source code but the overall execution time of the program is slower.
Compiler generates the error message only after scanning the whole program, so debugging is comparatively hard as the error can be present anywhere in the program.	Its Debugging is easier as it continues translating the program until the error is met
Generates intermediate object code.	No intermediate object code is generated.
Examples: C, C++, Java	Examples: Python, Perl

Linker



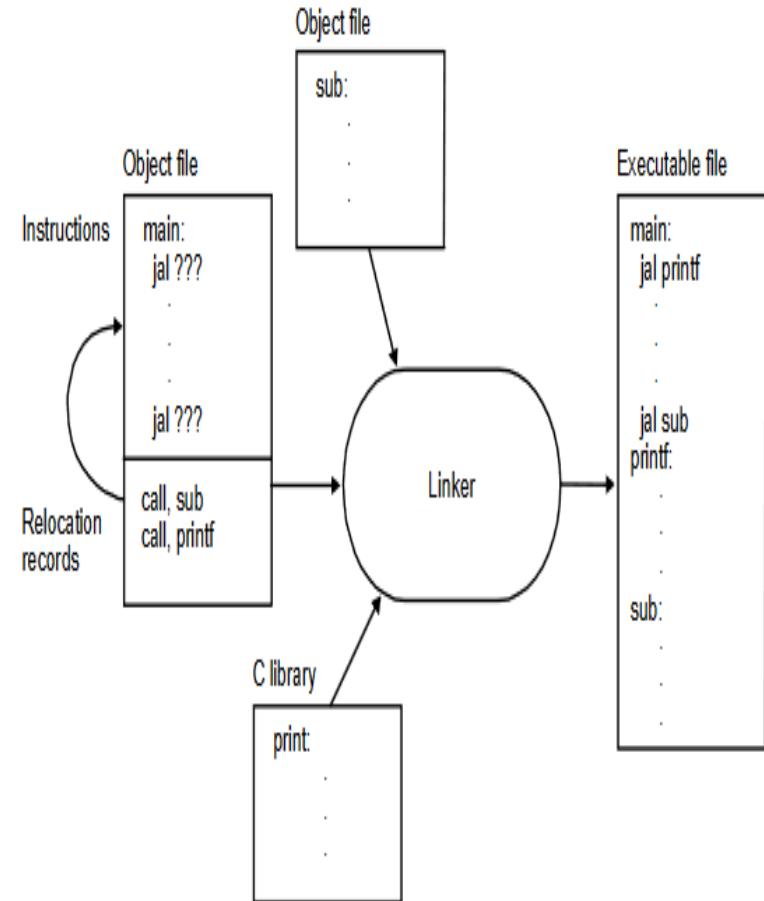
Linker

- Tool that merges the object files produced by separate compilation or assembly and creates a single executable file
- Linker are also called link editors.



Linker

- A linker tool is used to link all the parts of the program together for execution (executable machine code).



EXTERN STORAGE CLASS

PGM1.CPP

```
#include<stdio.h>
#include "pgm2.cpp"
int i;
void show(void);
int main()
{
    i=10;
    show();
    printf("\n Value of i in
pgm1.cpp=%d",i);
    return 0;
}
```

PGM2.CPP

```
extern int i;
void show()
{
    printf("\n value of i in
pgm2.cpp=%d",i);
}
```

OUTPUT??

EXTERN STORAGE CLASS

PGM1.CPP

```
#include<stdio.h>
#include "pgm2.cpp"
int i;
void show(void);
int main()
{
    i=10;
    show();
    printf("\n Value of i in
pgm1.cpp=%d",i);
    return 0;
}
```

PGM2.CPP

```
extern int i;
void show()
{
    printf("\n value of i in
pgm2.cpp=%d",i);
}
```

OUTPUT-

```
C:\TC\BIN>tc.exe
```

```
value of i in pgm2.cpp=10
Value of i in pgm1.cpp=10
```

Linker

- The major tasks–
 - To search and locate referenced routines in a program
 - Searches the program to find library routines used by program, e.g. printf(), math routines,...

Loader in C/C++

- Loader is the program of the operating system
 - which loads the executable from the disk into the primary memory(RAM) for execution.
- It allocates the memory space to the executable module in main memory and then transfers control to the beginning instruction of the program .

Loader in C/C++

- It places programs into memory and prepares them for execution.
- Loading a program involves reading the contents of the executable file containing the program instructions into memory,
- Then carrying out other required preparatory tasks to prepare the executable for running.
- Once loading is complete, the operating system starts the program by passing control to the loaded program code.

[https://en.wikipedia.org/wiki/Loader_\(computing\)](https://en.wikipedia.org/wiki/Loader_(computing))

Relocation

- The available memory is generally shared among a number of processes in a multiprogramming system,
- So it is not possible to know in advance which other programs will be resident in main memory at the time of execution of this program.
- Swapping the active processes in and out of the main memory enables the operating system to have a larger pool of ready-to-execute process.

-GeeksforGeeks

Relocation

- When a program gets swapped out to a disk memory, then it is not always possible that when it is swapped back into main memory then it occupies the previous memory location,
- Since the location may still be occupied by another process.
- We may need to relocate the process to a different area of memory. Thus there is a possibility that program may be moved in main memory due to swapping.

-GeeksforGeeks

<https://www.geeksforgeeks.org/requirements-of-memory-management-system/>

Relocation

- **Relocation** is the process of assigning load addresses for position-dependent code and data of a program and adjusting the code and data to reflect the assigned addresses.
- Relocation is typically done by the linker at link time, but it can also be done at load time by a relocating loader.

-Wikipedia

GATE CS 1998 | Question 25

In a resident- OS computer, which of the following system software must reside in the main memory under all situations?

- (A) Assembler**
- (B) Linker**
- (C) Loader**
- (D) Compiler**

GATE CS 1998 | Question 25

In a resident- OS computer, which of the following system software must reside in the main memory under all situations?

- (A) Assembler
- (B) Linker
- (C) Loader
- (D) Compiler

Answer: (C)

Explanation: Loader is the part of an operating system that is responsible for loading programs and libraries. it places the programs into memory and also prepares them for execution.

Loading a program involves tasks such as reading the contents of the executable file containing the program instructions into memory, and then carrying out other required preparatory tasks to prepare the executable for running so any software must reside in the main memory under all situations.

The operating system starts the program by passing control to the loaded program code once the loading process is completed.

Option (C) is correct.

Q) A linker program

- a. places the program in the memory for the purpose of execution.
- b. relocates the program to execute from the specific memory area allocated to it.
- c. links the program with other programs needed for its execution.
- d. interfaces the program with the entities generating its input data.

Q) A linker program

- a. places the program in the memory for the purpose of execution.
- b. relocates the program to execute from the specific memory area allocated to it.
- c. links the program with other programs needed for its execution.
- d. interfaces the program with the entities generating its input data.

Ans) c)

What is an Operating System?

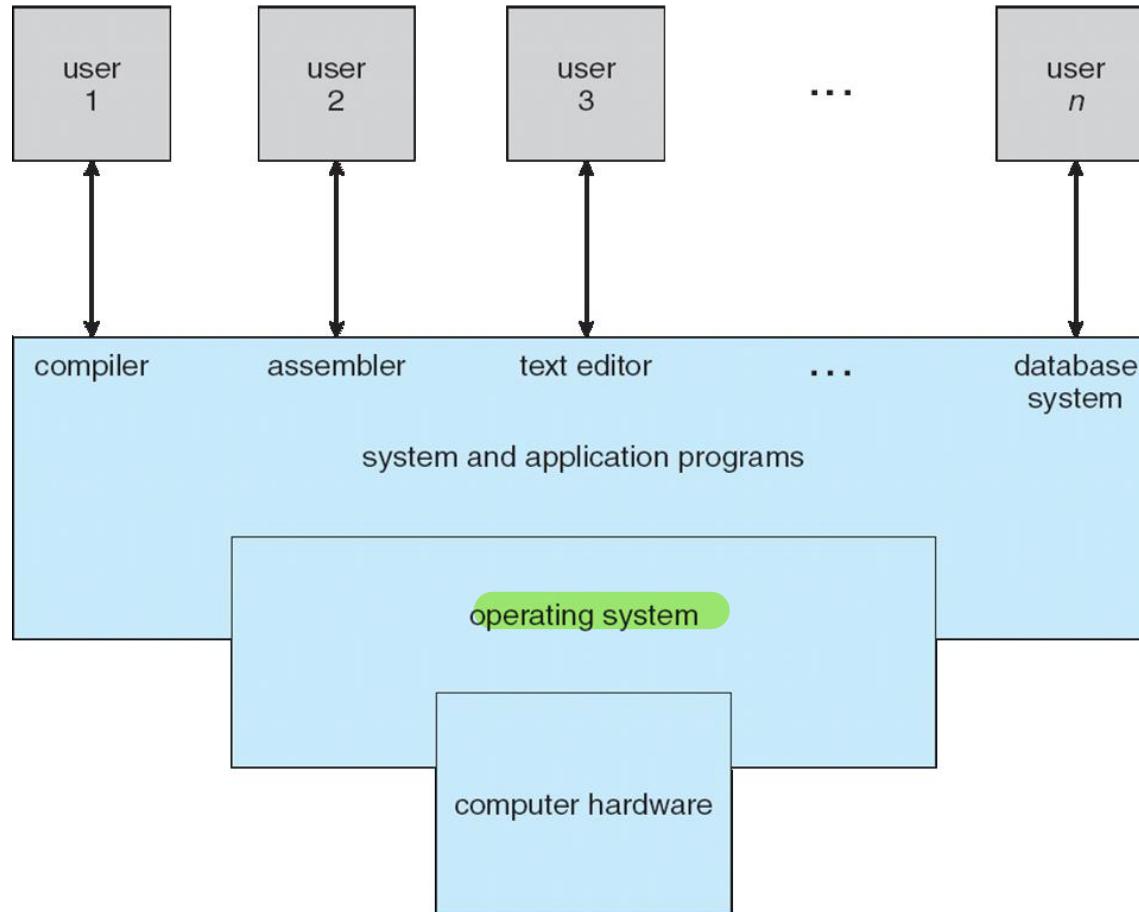
What is an Operating System?

- A program that acts as an intermediary between a user of a computer and the computer hardware.
- Provides an environment in which a user can execute programs in a convenient and efficient manner.

What is an Operating System?

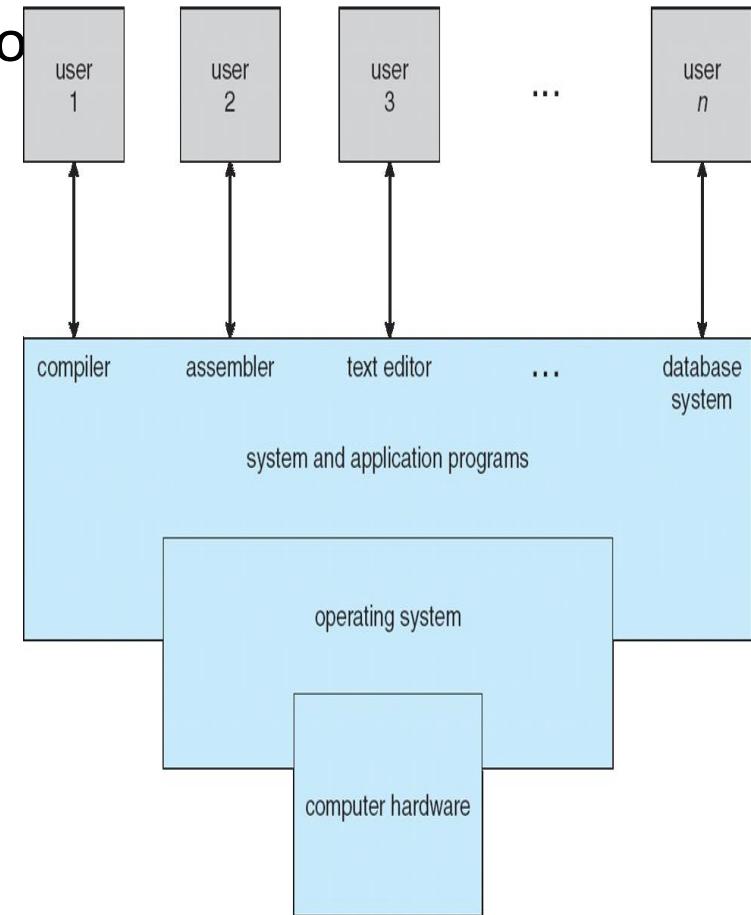
- Operating system goals:
 - Execute user programs and make solving user problems easier
 - Make the computer system convenient to use
 - Use the computer hardware in an efficient manner

Four Components of a Computer System



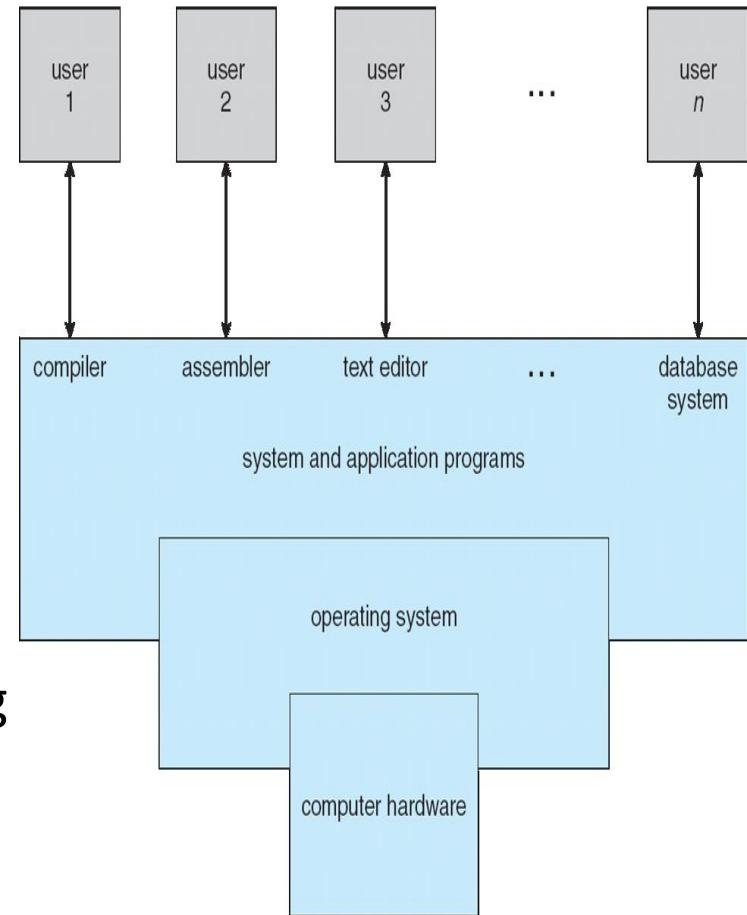
Computer System Structure

- Computer system can be divided into four components:
 - Hardware
 - Operating system
 - System & Application programs
 - Users



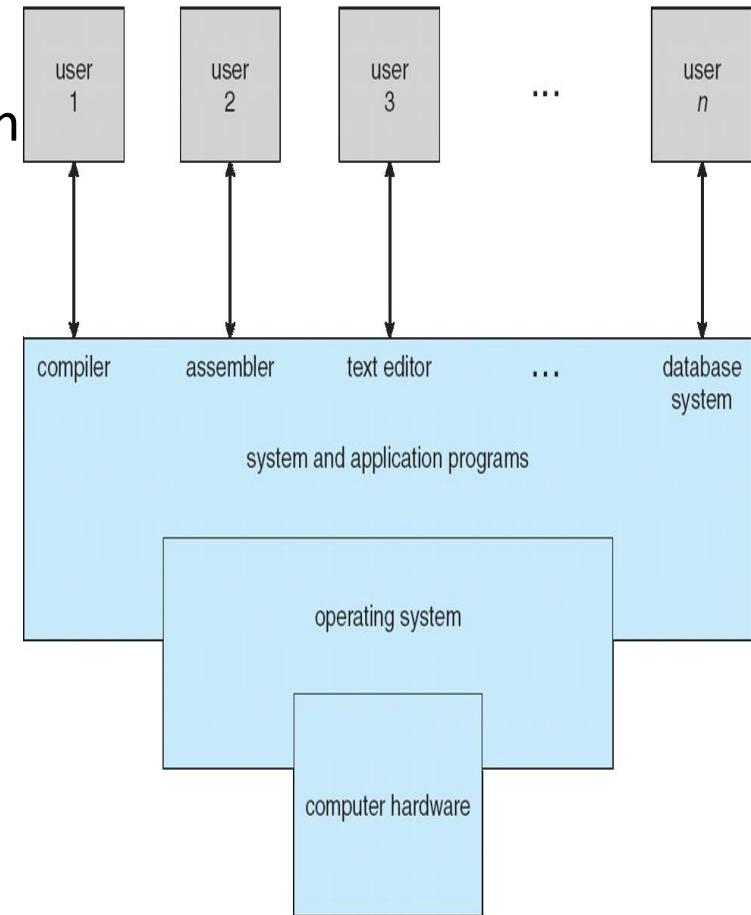
Computer System Structure

- Hardware – provides basic computing resources
 - CPU,
 - memory,
 - I/O devices
- Operating system
 - Controls and
 - Coordinates use of hardware among various applications and users

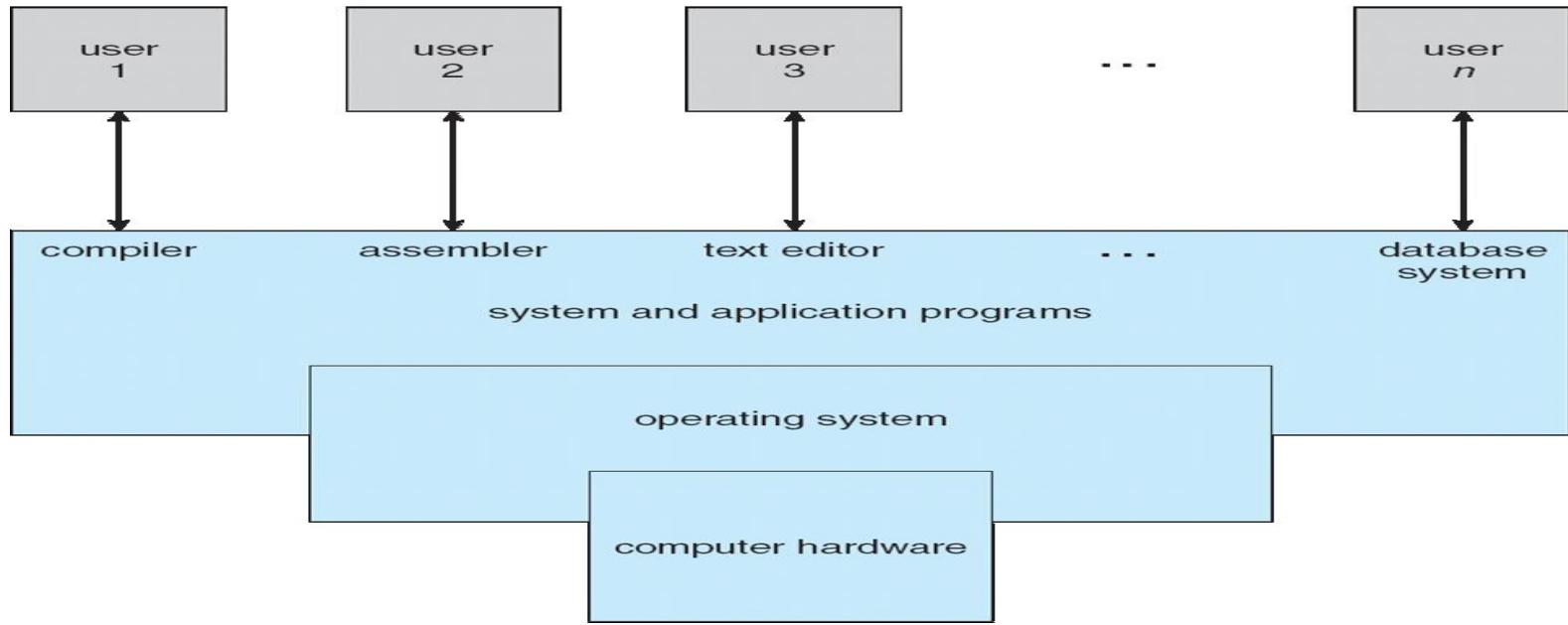


Computer System Structure

- Application programs
 - define the ways in which the system resources are used
 - to solve the computing problems of the users
 - Word processors,
 - web browsers,
 - database systems,
 - video games
- Users
 - People, machines, other computers



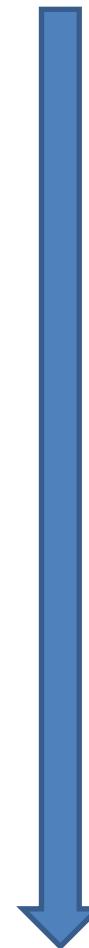
Four Components of a Computer System



OS controls and coordinates the use of hardware among the various application programs for the various users.

Evolution of Operating System

- **Mainframe System**
 - Batch Systems
 - Multiprogrammed Systems
 - Time Sharing Systems
- **Desktop Systems**
 - Multiprocessor Systems
 - Symmetric Multiprocessor
 - Asymmetric Multiprocessor
 - Distributed Systems
 - Client Server Systems
 - Peer to Peer Systems
 - Clustered Systems
 - Asymmetric clustering
 - Symmetric clustering
 - Real Systems
 - Hard Real Time System
 - Soft Real Time System
- **HandHeld Systems**



Mainframe System

- First computers to tackle commercial and scientific applications.
- Growth from Batch systems to Multi-programmed to Timesharing(Multitasking system)

??



??



Batch system

- Early computers were physically enormous machines run from a console.
 - Input devices –
 - Card readers and
 - Tape drivers.
 - O/P devices-
 - Line printers,
 - Tape drivers,
 - Card Punches.

Tape drives

- A tape drive is a data storage device that reads and writes data on a magnetic tape.
 - Magnetic tape data storage is typically used **for offline, archival data storage.**
 - Tape media generally has a **favorable unit cost and a long archival stability.**
- **Magnetic tapes are typically placed in plastic covers referred to as cassettes.**



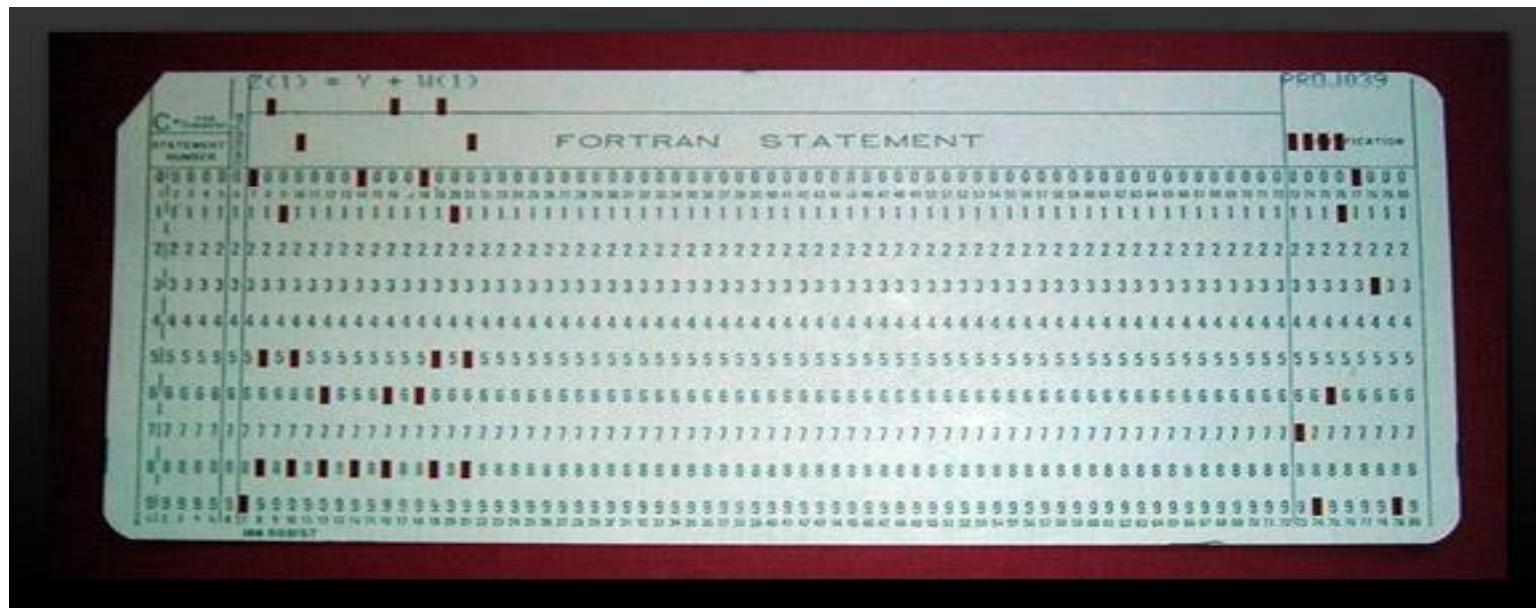
Tape drives

- A tape drive provides sequential access storage, unlike a hard disk drive, which provides direct access storage.
 - A disk drive can move to any position on the disk in a few milliseconds,
 - but a tape drive must physically wind tape between reels to read any one particular piece of data.
 - As a result, tape drives have very large average access times.
 - **However, tape drives can stream data very quickly off a tape when the required position has been reached.**

Punch card/A punched card

- Is a piece of stiff paper that can be used to contain digital data
 - represented by the presence or absence of holes in predefined positions.
 - Punched cards were widely used through much of the 20th century for
 - data input, output, and storage.

Card from a Fortran program

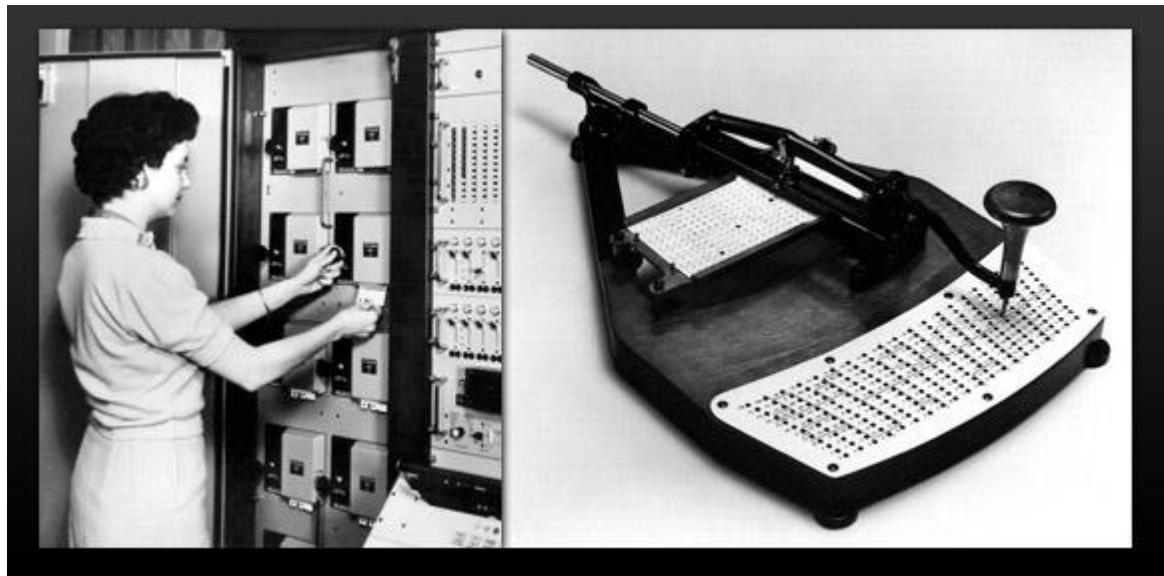


Punch card/A punched card

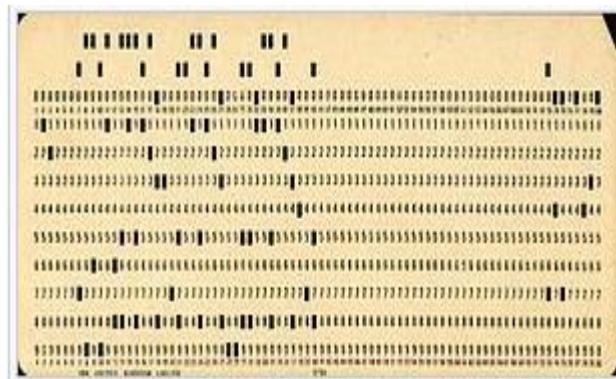
- Many early digital computers used punched cards for input of both
 - computer programs and
 - data.
- While punched cards are now obsolete as a storage medium, as of 2012, some voting machines still use punched cards to record votes.

Above left: Punch card reader.

Above right: Punch card writer.



Punch card Reader, A punched card



A 12-row/80-column IBM punched card from the mid-twentieth century



A deck of punched cards comprising a computer program



Card reader/punch

Courtesy :<https://www.ibiblio.org/comphist/node/57>,https://en.wikipedia.org/wiki/Punched_card

Batch system

- User did not interact with the computer systems.
 - User prepared a job which consisted of
 - the programs,
 - data and
 - control information ,
 - submitted it to the computer operator.
 - The job was usually in the form of punch cards.
- At some later time, the output appeared.
 - The o/p consisted of
 - result of the program, as well as
 - a dump of the final memory and
 - register contents for debugging.

Batch system

- To speed up processing,
 - operators batched together jobs with similar needs and
 - ran them throughout the computer as a group.

Batch system

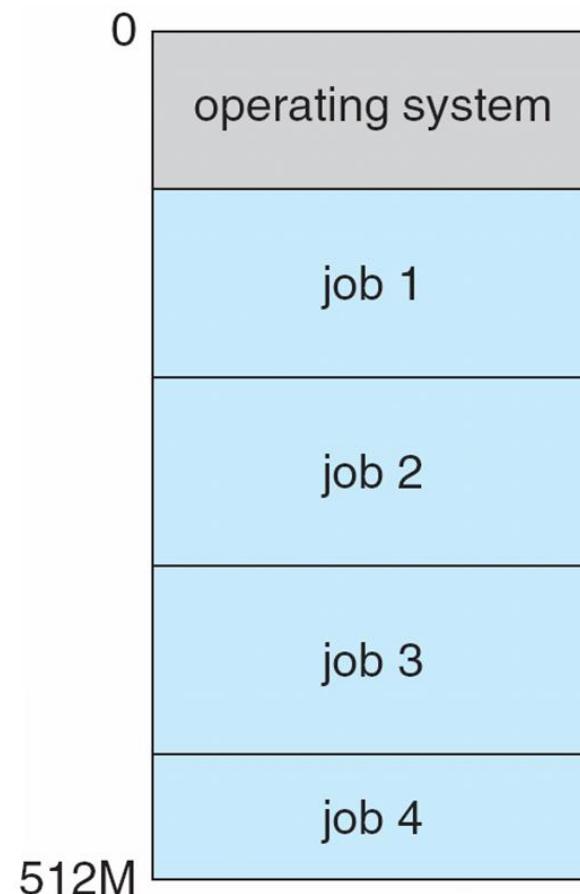
- 1) The programmers would leave their programs with the operator.
- 2) The operator would
 - 1) sort programs into batches with similar requirements and
 - 2) would run each batch as the computer became available.
- 3) The O/P would be sent back to the appropriate programmer

Memory Layout for Multiprogrammed System

- Bring multiple programs in RAM

Now,

- How to execute these programs?



Multiprogramming

Multi programmed –Non-Preemptive?

Multitasking-Preemptive?

Multiprogramming

Class-Student Example?

S1

S2

S3

S4

S5

S6

S7

S8

S9

S10

Multitasking

Class-Student Example?

S1

S2

S3

S4

S5

S6

S7

S8

S9

S10

Non-multi programmed system

Non-multi programmed system's working –

- As soon as one job leaves the CPU and goes for I/O task the CPU becomes idle.
- **The CPU keeps waiting and waiting until this job (which was executing earlier) comes back and resumes its execution with the CPU.** So CPU remains free for all this while.
- Now it has a **drawback that the CPU remains idle for a very long period of time.** Also, other jobs which are waiting to be executed might not get a chance to execute because the CPU is still allocated to the earlier job.

Multiprogramming

Multi programmed system's working –

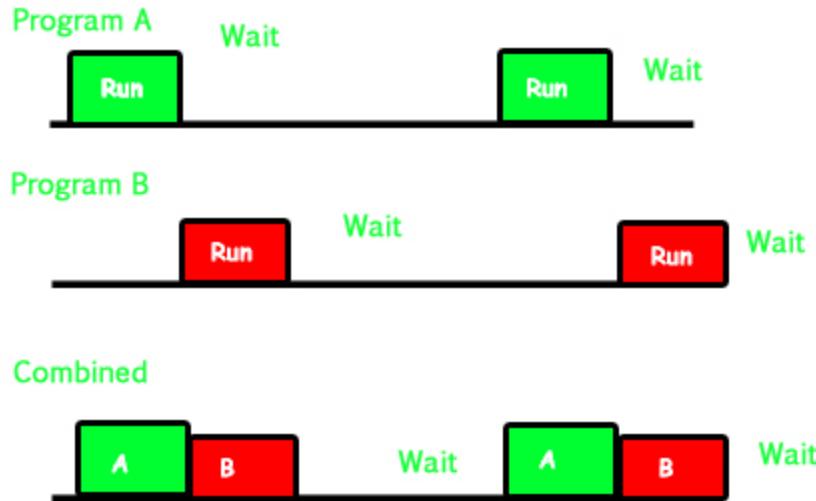
- In a multi-programmed system, as soon as one job goes for an I/O task, the Operating System interrupts that job, chooses another job from the job pool, gives CPU to this new job and starts its execution.
- The previous job keeps doing its I/O operation while this new job does CPU bound tasks.
- Now say the second job also goes for an I/O task, the CPU chooses a third job and starts executing it.

Multiprogramming

- **Multiprogramming** needed for efficiency
 - Single Program cannot keep CPU and I/O devices busy at all times
 - Multiprogramming increases CPU utilization by organizing jobs (code and data) so that CPU always has one to execute.
 - A subset of total jobs in system is kept in memory
 - One job selected and run via **job scheduling**
 - When it has to wait for some task, such as I/O, OS switches to another job
 - In non- multiprogrammed system, The CPU would sit idle.

Multiprogramming

- Program A runs for some time and then goes to waiting state. In the mean time program B begins its execution.



- So the CPU does not waste its resources and gives program B an opportunity to run.

Drawbacks

- Multiprogrammed ,batched system provided an environment
 - where the various system resources were utilized effectively,
 - **but it did not provide for user interaction with the computer system.**

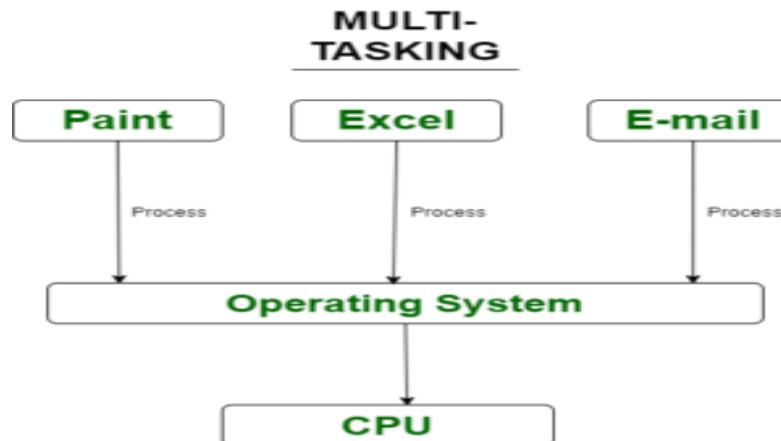
Timesharing (multitasking)

- Timesharing (multitasking) is logical extension of multiprogramming
- CPU switches jobs so frequently that users can interact with each job while it is running, creating interactive computing.

-Galvin

- CPU executes multiple jobs by switching among them typically using a small time quantum
- The switches occur so frequently that the users can interact with each program while it is running.

- GeeksforGeeks



Timesharing (multitasking)

- An interactive computer system provides direct communication between the user and the system.
- **Response time** should be short , < 1 second
- Each user has at least one program executing in memory
⇒ **process**
- If several jobs ready to run at the same time ⇒ **CPU scheduling**
- If processes don't fit in memory, **swapping** moves them in and out to run
- **Virtual memory** allows execution of processes not completely in memory

Timesharing (multitasking)

- More complex than Multi programmed OS

Timesharing (multitasking)

- To obtain a reasonable response time,
 - jobs may have to be swapped in and out of main memory to the disk.
 - Virtual memory is used
 - which allows execution of a job that may not be completely in memory.
 - Programs can be longer than physical memory.

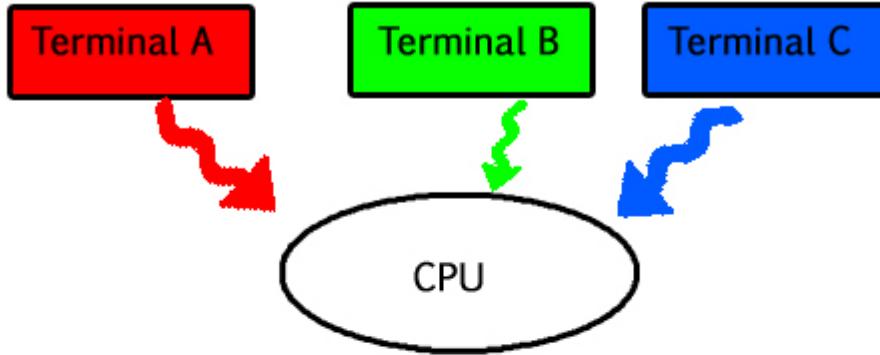
Timesharing (multitasking)

- In a time sharing system, each process is assigned some specific quantum of time for which a process is meant to execute.
- Say there are 4 processes P1, P2, P3, P4 ready to execute.
- Say time quantum of 5 nanoseconds (5 ns).
- As one process begins execution (say P2), it executes for that quantum of time (5 ns).
- After 5 ns the CPU starts the execution of the other process (say P3) for the specified quantum of time.

Timesharing (multitasking)

- Thus the CPU makes the processes to share time slices between them and execute accordingly.
- As soon as time quantum of one process expires, another process begins its execution.
- **The context switch is so fast that the user is able to interact with each program separately while it is running.**
- This way, the user is given the illusion that multiple processes/ tasks are executing simultaneously.

Timesharing (multitasking)



- So for multitasking to take place, **firstly there should be multiprogramming i.e. presence of multiple programs ready for execution.**
- And **secondly the concept of time sharing.**
- **Responsiveness Increases**

Multiprogramming Vs Timesharing (multitasking)

Sr.no	Multiprogramming	Multi-tasking
1.	Both of these concepts are for single CPU.	Both of these concepts are for single CPU.
2.	Concept of Context Switching is used.	Concept of Context Switching and Time Sharing is used.
3.	In multiprogrammed system, the operating system simply switches to, and executes, another job when current job needs to wait.	The processor is typically used in time sharing mode. Switching happens when either allowed time expires or where there other reason for current process needs to wait (example process needs to do I/O).
4.	Multi-programming increases CPU utilization by organising jobs .	In multi-tasking also increases CPU utilization, it also increases responsiveness.
5.	The idea is to reduce the CPU idle time for as long as possible.	The idea is to further extend the CPU Utilization concept by increasing responsiveness Time Sharing.

ISRO | ISRO CS 2007 | Question 25

What is the name of the technique in which the operating system of a computer executes several programs concurrently by switching back and forth between them?

- (A) Partitioning
- (B) Multi-tasking
- (C) Windowing
- (D) Paging

ISRO | ISRO CS 2007 | Question 25

What is the name of the technique in which the operating system of a computer executes several programs concurrently by switching back and forth between them?

- (A) Partitioning
- (B) Multi-tasking
- (C) Windowing
- (D) Paging

Answer: (B)

Explanation: In a multitasking system, a computer executes several programs simultaneously by switching them back and forth to increase the user interactivity. Processes share the CPU and execute in an interleaving manner. This allows the user to run more than one program at a time.

GATE | GATE-CS-2002 | Question 46

Which combination of the following features will suffice to characterize an OS as a multi-programmed OS?

- (a) More than one program may be loaded into main memory at the same time for execution.
- (b) If a program waits for certain events such as I/O, another program is immediately scheduled for execution.
- (c) If the execution of program terminates, another program is immediately scheduled for execution.

- (A) a
- (B) a and b
- (C) a and c
- (D) a, b and c

GATE | GATE-CS-2002 | Question 46

Answer: (B)

Which combination of the following features will suffice to characterize an OS as a multi-programmed OS? **Explanation:**

- (a) More than one program may be loaded into main memory at the same time for execution.
- (b) If a program waits for certain events such as I/O, another program is immediately scheduled for execution.
- (c) If the execution of program terminates, another program is immediately scheduled for execution.

- (A) a
- (B) a and b
- (C) a and c
- (D) a, b and c

(a) More than one program may be loaded into main memory at the same time for execution.

True: Only done in a multiprogrammed OS, not in single programmed OS

(b) If a program waits for certain events such as I/O, another program is immediately scheduled for execution.

True: Only done in a multiprogrammed OS, not in single programmed OS

(c) If the execution of program terminates, another program is immediately scheduled for execution.

False: Done in both Multiprogrammed and single programmed OSs

Multiprocessor Systems

- Also called Parallel System or Tightly coupled System grew in importance.
- More than one processor in close communication, sharing
 - the computer bus,
 - the clock and
 - sometimes memory and peripheral devices.

Multiprocessor Systems

Advantages:-

- **Increased Throughput**
- **Economy of scale**
- **Increased Reliability**

Multiprocessor Systems

Advantages:-

- **Increased Throughput**
 - **To Get more work done.**
 - The speedup ratio with N processors is not N, it is less than N.
 - Overhead is incurred in keeping all parts working correctly.
 - Plus Contention for shared resources lowers the expected gains.

Multiprocessor Systems

Advantages:-

- **Economy of scale**
 - **Save more money** as they share
 - peripherals,
 - mass storage,
 - power supplies.

Multiprocessor Systems

Advantages:-

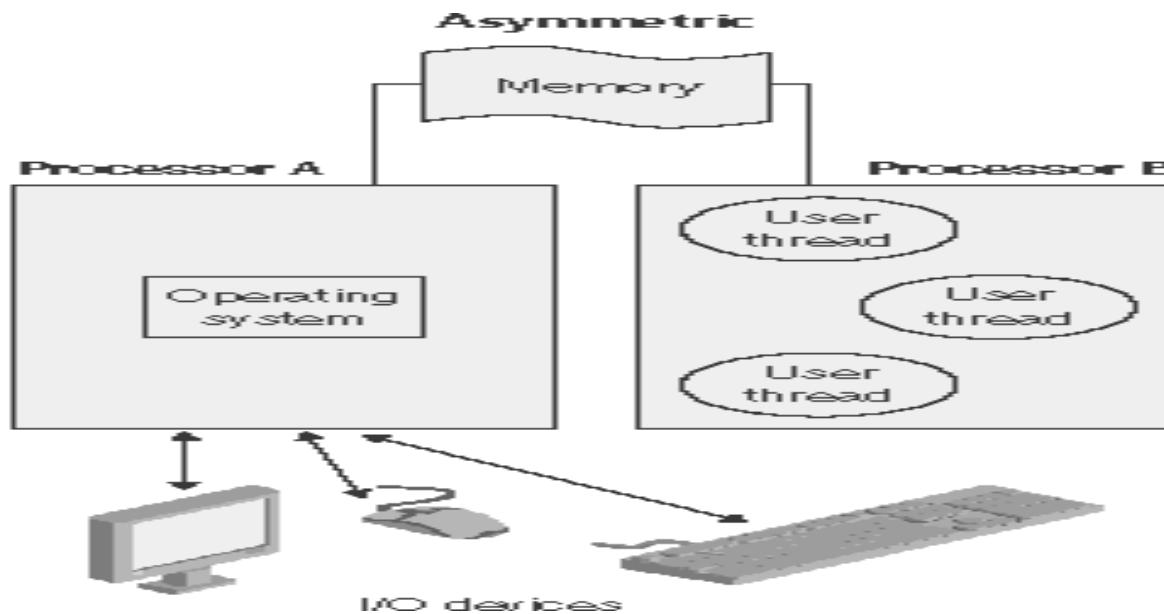
- **Increased Reliability**
 - If functions can be distributed properly among several processors, then
 - **the failure of one processor will not halt the system,**
 - only slow it down.

Multiprocessor Systems

- Symmetric Multiprocessor
 - Each processor runs an identical copy of the OS and
 - These copies communicate with one another as needed.
- Asymmetric Multiprocessor
 - Each processor is assigned a specific task.
 - A master processor controls the system,
 - The other processor either look to the master for instruction or have predefined tasks.
 - The master allocates the work to the slave processor.

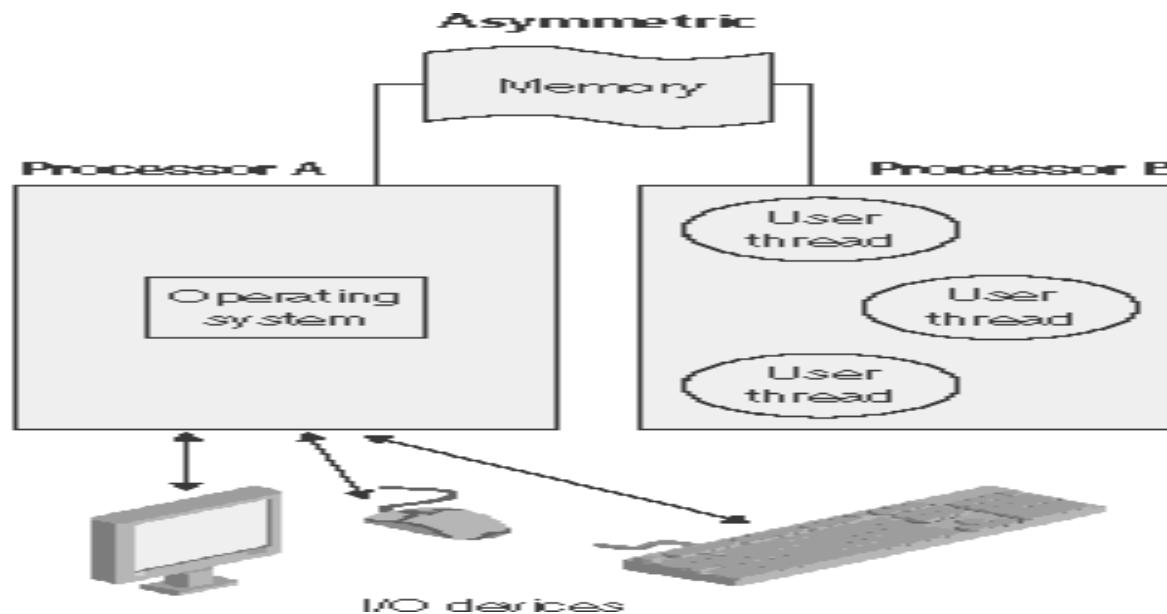
Asymmetric multiprocessing

- ASMP
 - The operating system typically sets aside one or more processors for its exclusive use.
 - The remainder of the processors run user applications.



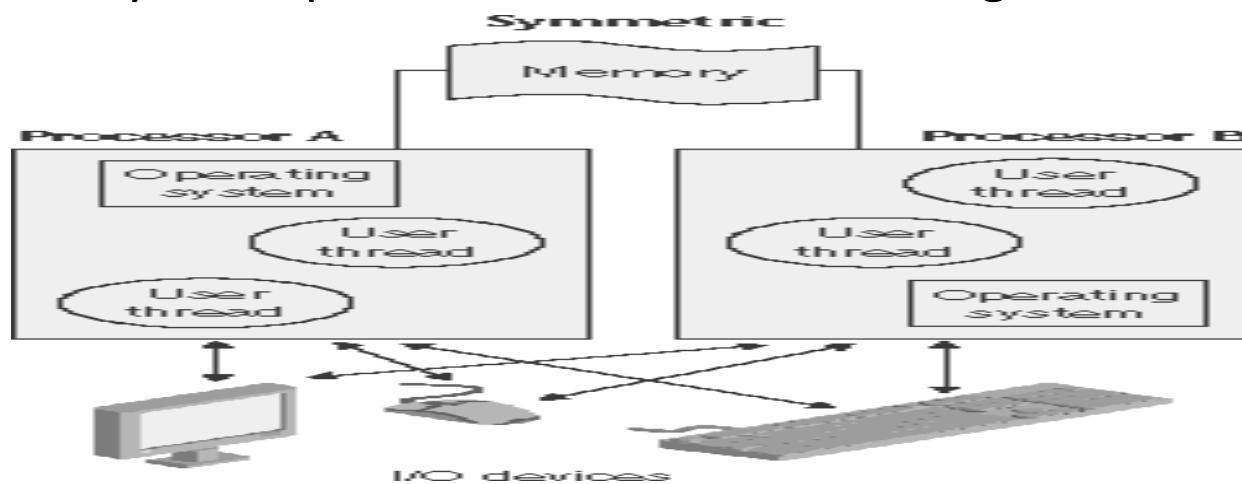
Asymmetric multiprocessing

- In the ASMP model,
 - If the processor that fails is an operating system processor,
 - The whole computer can go down.



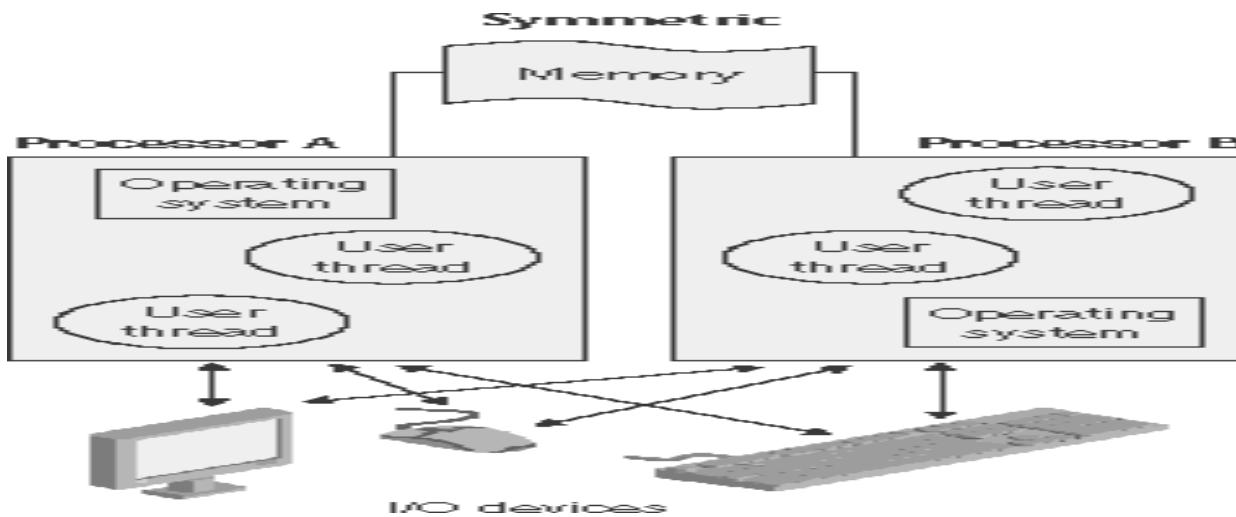
Symmetric Multiprocessing

- Symmetric multiprocessing (SMP) technology is used to get higher levels of performance.
- In symmetric multiprocessing, any processor can run any type of thread. The processors communicate with each other through shared memory.
- SMP systems provide better load-balancing and fault tolerance.



Symmetric Multiprocessing

- Because the operating system threads can run on any processor, the chance of hitting a CPU bottleneck is greatly reduced.
 - All processors are allowed to run a mixture of application and operating system code.
 - A processor failure in the SMP model only reduces the computing capacity of the system.



Difference Between Asymmetric and Symmetric Multiprocessing:

ASYMMETRIC MULTIPROCESSING	SYMMETRIC MULTIPROCESSING
In asymmetric multiprocessing, the processors are not treated equally.	In symmetric multiprocessing, all the processors are treated equally.
Tasks of the operating system are done by master processor.	Tasks of the operating system are done in individual processor
No Communication between Processors as they are controlled by the master processor.	All processors communicate with another processor by a shared memory.
In asymmetric multiprocessing, process are master-slave.	In symmetric multiprocessing, the process is taken from the ready queue.
Asymmetric multiprocessing systems are cheaper.	Symmetric multiprocessing systems are costlier.
Asymmetric multiprocessing systems are easier to design	Symmetric multiprocessing systems are complex to design

Multiprocessor Systems

- **Tightly coupled System**
- More than one processor in close communication
- **Sharing the computer bus, the clock and sometimes memory and peripheral devices.**

Distributed Systems

- **Loosely Coupled Systems**
- Collection of processors
- **Do not share memory or a clock.**
- **Each processor has its own local memory.**
- The processors communicate with one another through various communication lines such as **high speed buses or telephone lines.**

Distributed Systems

- Collection of separate, possibly heterogeneous, systems networked together
 - Network is a communications path,
 - TCP/IP most common network protocol
 - Most OS support TCP/IP including the Windows and UNIX OS.

Distributed Systems

- Networks are typecast based on the distances between their nodes
 - Personal Area Network (**PAN**)
 - Local Area Network (**LAN**)
 - Metropolitan Area Network (**MAN**)
 - Wide Area Network (**WAN**)

Distributed Systems

- Networks are typecast based on the distances between their nodes
 - **Personal Area Network (PAN)**-Short distance of several feet like Bluetooth Devices
 - **Local Area Network (LAN)**- within a room, a floor, or a building
 - **Metropolitan Area Network (MAN)**-link building within a city
 - **Wide Area Network (WAN)**- exists between buildings, cities or countries

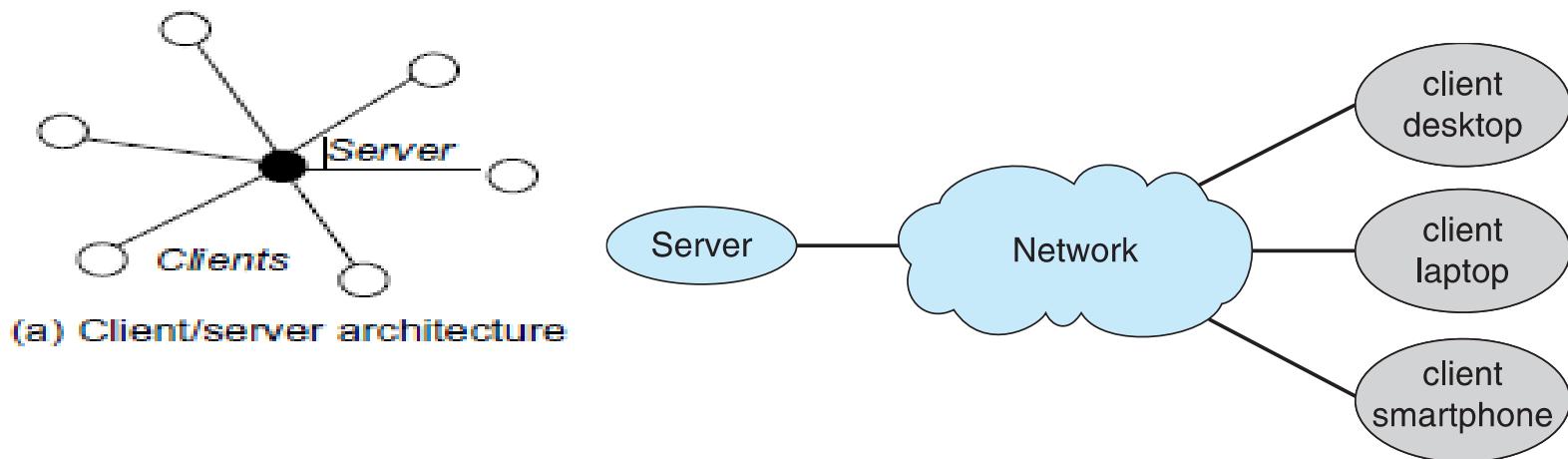
Distributed Systems

- Types
 - Client-Server Systems
 - Peer-to-Peer Systems

Client-Server Systems

□ Client-Server Computing

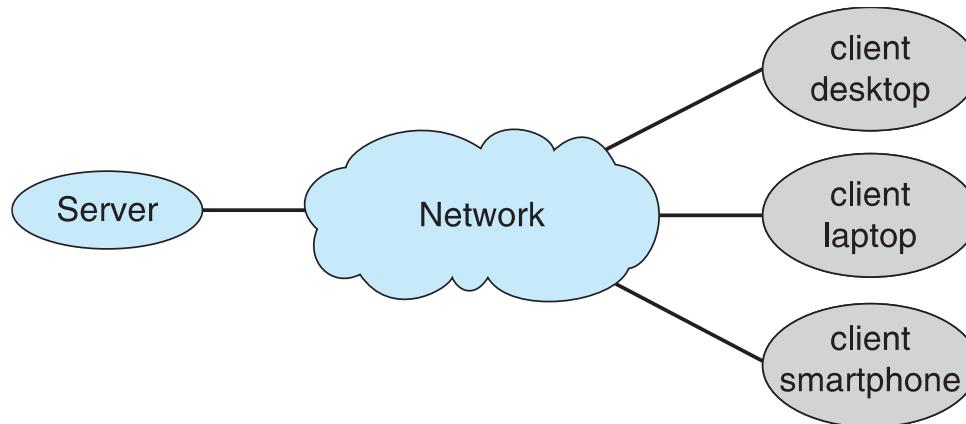
- Centralized system act as server systems to satisfy requests generated by client systems
- User interface handled by smart PCs.



Client-Server Systems

- Server systems can be categorized as :

- ▶ **Compute-server system** provides an interface to client to request services (i.e., database), perform action, send back results to the client
- ▶ **File-server system** provides file system interface for clients to store and retrieve files i.e. create, update, read and delete files

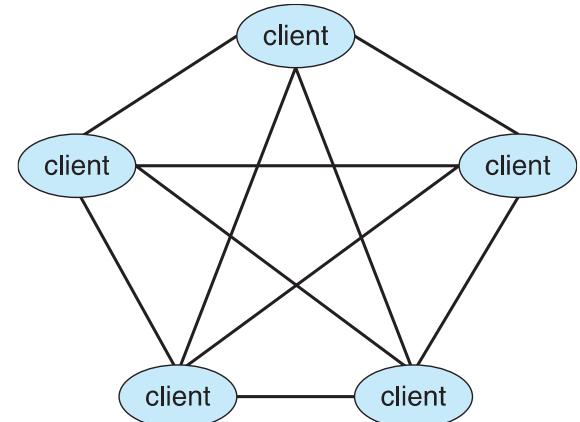


P2P Networks

- An alternative to the traditional client / server architecture where there typically is a single server and many clients.

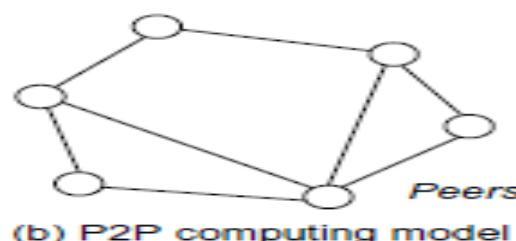
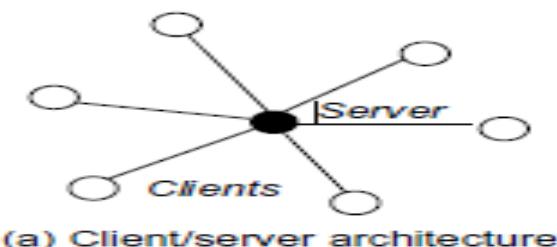
Peer-to-Peer

- Another model of distributed system
- P2P does not distinguish clients and servers
 - **All nodes are considered peers**
 - **Why Peers?**



Peer-to-Peer

- Why Peers?
 - “A communications model in which **each party has the same capabilities** and
 - All play **equal roles**.
 - **Each node may act as client, server or both**
- **No peer has global view** of the entire system.



P2P Networks

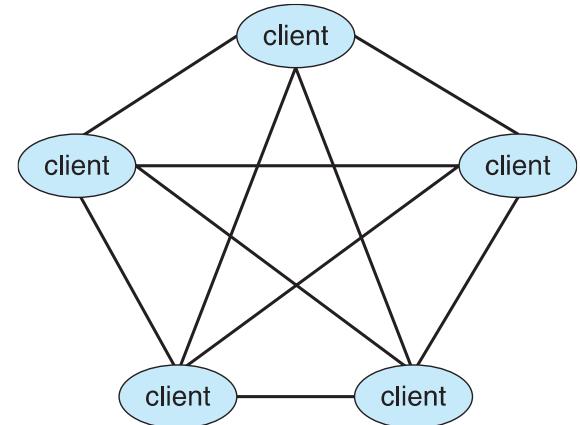
- Peers-
 - Exchange resources with one another directly.
 - Either party can initiate a communication session”

P2P Networks

- P2P Networks are autonomous and self-organised with free participation from peers.
- A P2P computing system is formed with peer hosts that are fully distributed without using a centralized server.

Peer-to-Peer

- Node must join P2P network
 - Registers its service with central lookup service on network, or
 - Broadcast request for service and respond to requests for service via ***discovery protocol***
- Examples such as Skype



P2P Networks

- Used widely in
 - Messaging,
 - Online Chatting,
 - Video streaming and
 - Social networking.

Peer-to-peer networking with BitTorrent

- BitTorrent is by far
 - the most popular peer-to-peer programs ever.
- ??

Peer-to-peer networking with BitTorrent

- BitTorrent is one of the most common protocols
 - for transferring large files,
 - such as digital video files containing TV shows or video clips or digital audio files containing songs.

BitTorrent Clients

- To send or receive files,
 - a person uses a BitTorrent client on their Internet-connected computer.
- A BitTorrent client is a
 - computer program that implements the BitTorrent protocol.
 - Popular clients include
 - µTorrent, Xunlei, Transmission, qBittorrent, Vuze, Deluge, BitComet and Tixati.

BitTorrent Clients

- FrostWire



- µTorrent



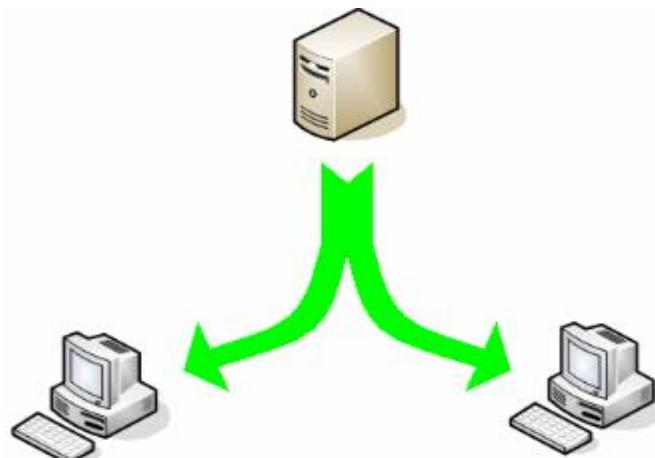
- Tonido



- Deluge



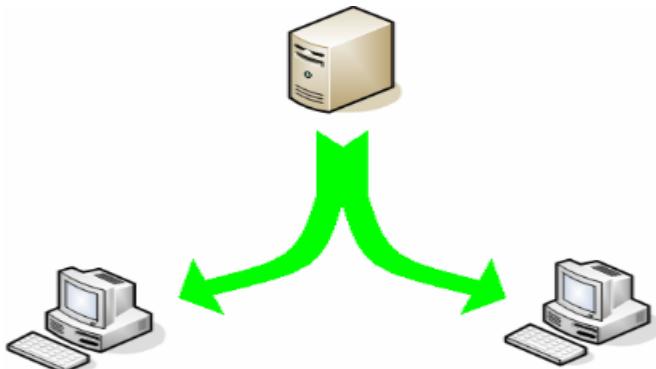
Typical Client Server



Downloading File from the server

P2P??

Typical Client Server



- In the figure, The time for the download to finish will be two times the time if only one peer was downloading from the server.

Basic Idea behind BitTorrent

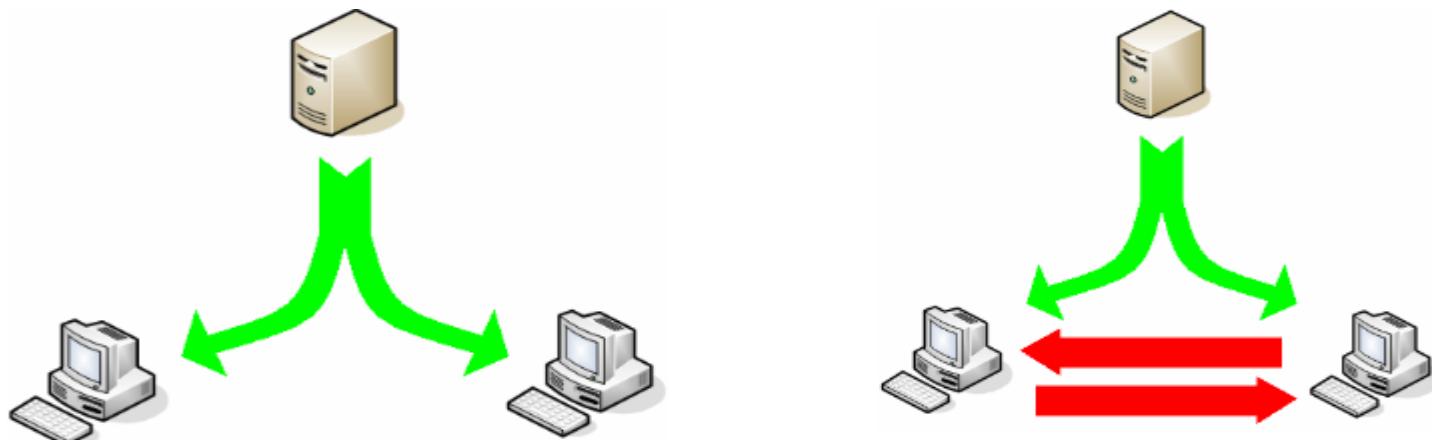


Figure 4 – The basic flow of the BitTorrent protocol.

Student Answersheets ??

Basic Idea behind BitTorrent

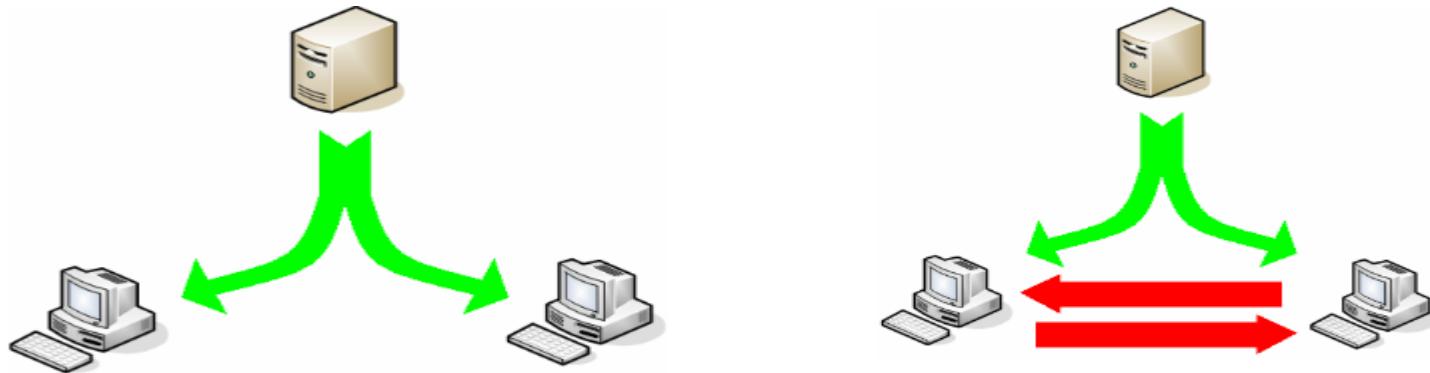
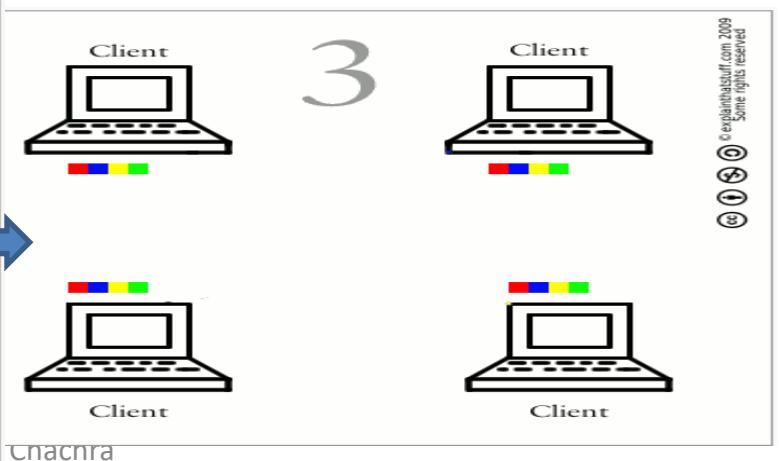
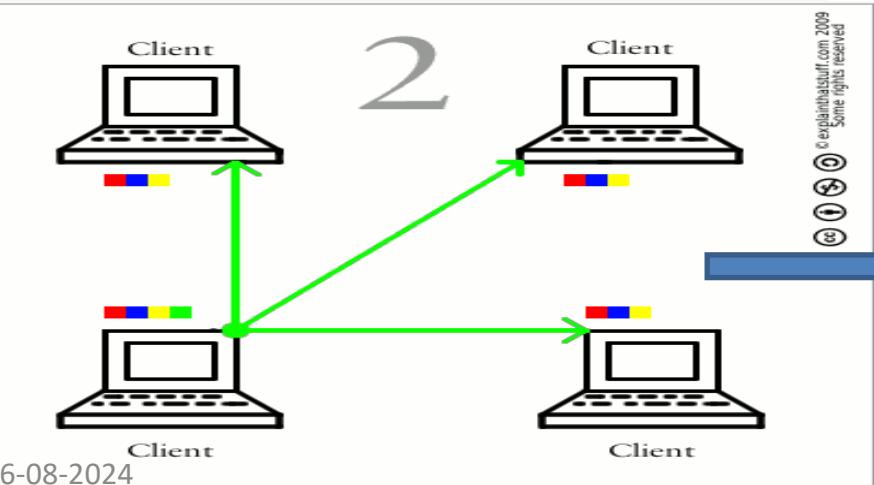
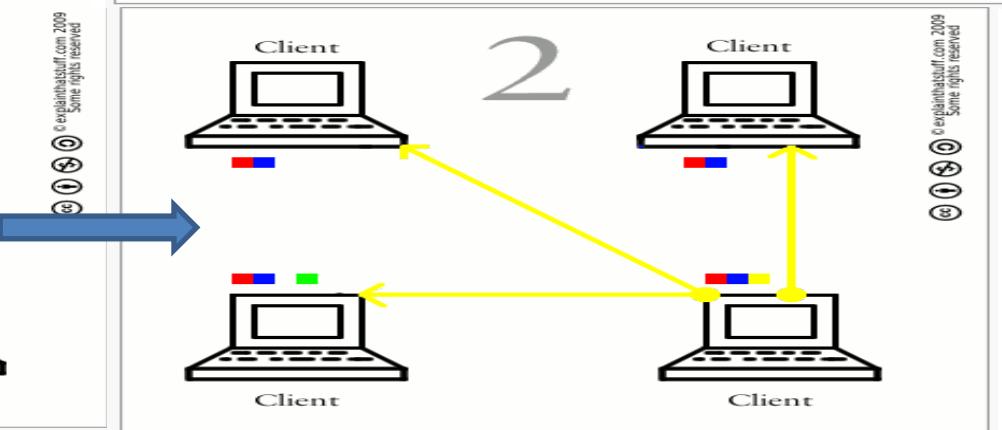
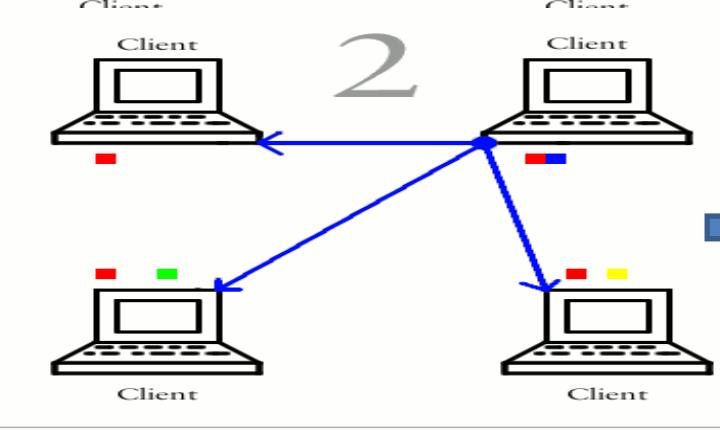
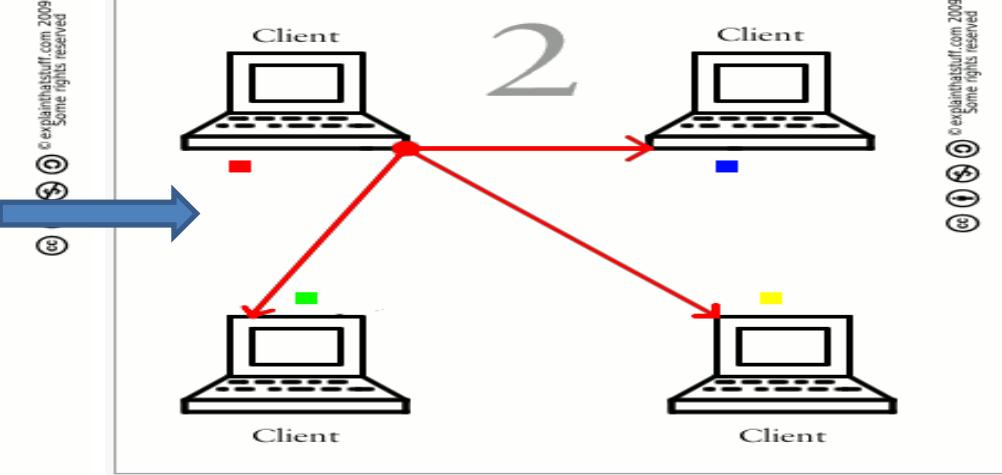
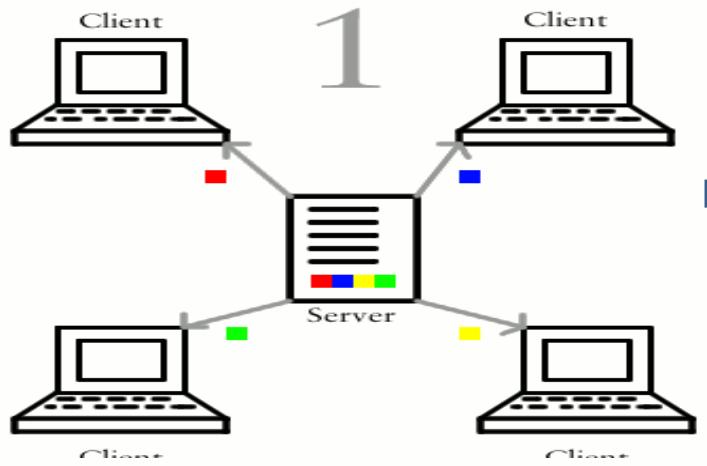


Figure 4 – The basic flow of the BitTorrent protocol.

- The figure on the right shows an approach similar to BitTorrent.
 - By splitting the file
 - Send one part to each peer,
 - Let the peers download the part they are missing from each other,
 - Both download time and load on the server is reduced.
- Of course, the BitTorrent protocol is much more sophisticated than this simple example, but this shows the basic idea.



Real Time Systems

- Special purpose OS
- Used when rigid time requirements have been placed on the processing.
- Often used as control device in Dedicated applications.

Real Time Systems

- Real-time operating systems (RTOS) are used in environments where a large number of events, mostly external to the computer system, must be accepted and processed in a short time or within certain deadlines.

Real Time Systems

Examples of the real-time operating systems:

- Airline traffic control systems,
- Command Control Systems,
- Industrial control,
- Telephone switching equipment,
- Flight control, and
- Real-time simulations.

Real Time Systems

- Real Time Systems functions correctly only if it returns the correct result within its time constraints.
- In Contrast-
 - Time Sharing Systems, where it is desirable but not mandatory to respond quickly.
 - Batch System, which may have no time constraints at all.

Real Time Systems

Two types of Real-Time Operating System which are as follows:

- Hard Real-Time Systems
- Soft Real-Time Systems

Hard Real-Time Systems:

- These OSs are meant for the applications where time **constraints are very strict** **and even the shortest possible delay is not acceptable.**
- These systems are built for saving life like **automatic parachutes or air bags** which are required to be readily available in case of any accident.

Soft Real-Time Systems:

- These OSs are for applications where for **time-constraint is less strict.**

Hard Real-Time Systems:

- This type of system can never miss its deadline.
- Missing the deadline may have disastrous consequences.

Soft Real-Time Systems:

- This type of system can miss its deadline occasionally with some acceptably low probability.
- Missing the deadline have no disastrous consequences.

Hard Real-Time Systems:

- The usefulness of result produced by a hard real time system decreases abruptly if tardiness increases.

Examples:

- **Satellite launch system**
- **Railway signalling system**

Soft Real-Time Systems:

- The usefulness of result produced by a soft real time system decreases gradually with increase in tardiness.

Examples:

- **Telephone switches.**
- **Multimedia Transmission and Reception**
- **DVD Player**
- **Web browsing**
- **Gaming-Computer Games**

Tardiness - How late a real time system completes its task with respect to its deadline.

HRTS- Healthcare and Patient Monitoring

- How quickly data is processed in healthcare can often mean the difference between life and death.
- Real-time systems are key to ensuring data from patient monitoring systems, such as heart rate monitors, is available to clinicians when and where they need it to keep patients safe and healthy.

RTS



HARD REAL TIME SYSTEM

SOFT REAL TIME SYSTEM

In hard real time system, the size of data file is small or medium.

In soft real time system, the size of data file is large.

In this system response time is in millisecond.

In this system response time are higher.

Peak load performance should be predictable.

In soft real time system, peak load can be tolerated.

In this system safety is critical.

In this system safety is not critical.

A hard real time system is very restrictive.

A Soft real time system is less restrictive.

In case of an error in a hard real time system, the computation is rolled back.

In case of an soft real time system, computation is rolled back to previously established a checkpoint.

Satellite launch, Railway signaling system etc.

DVD player, telephone switches, electronic games etc.

Match the following:

List – I	List – II
(a) Spooling	(i) Allows several jobs in memory to improve CPU utilization
(b) Multiprogramming	(ii) Access to shared resources among geographically dispersed computers in a transparent way
(c) Time sharing	(iii) Overlapping I/O and computations
(d) Distributed computing	(iv) Allows many users to share a computer simultaneously by switching processor frequently

codes:

	(a)	(b)	(c)	(d)
(1)	(iii)	(i)	(ii)	(iv)
(2)	(iii)	(i)	(iv)	(ii)
(3)	(iv)	(iii)	(ii)	(i)
(4)	(ii)	(iii)	(iv)	(i)

(A) (1)

(B) (2)

(C) (3)

(D) (4)

Match the following:

List – I	List – II
(a) Spooling	(i) Allows several jobs in memory to improve CPU utilization
(b) Multiprogramming	(ii) Access to shared resources among geographically dispersed computers in a transparent way
(c) Time sharing	(iii) Overlapping I/O and computations
(d) Distributed computing	(iv) Allows many users to share a computer simultaneously by switching processor frequently

codes:

	(a)	(b)	(c)	(d)
(1)	(iii)	(i)	(ii)	(iv)
(2)	(iii)	(i)	(iv)	(ii)
(3)	(iv)	(iii)	(ii)	(i)
(4)	(ii)	(iii)	(iv)	(i)

- (A) (1)
(B) (2)
(C) (3)
(D) (4)

Answer: (B)

Explanation:

- Spooling provides Overlapping I/O and computations.
- Multiprogramming allows several jobs in memory to improve CPU utilization.
- Time sharing allows many users to share a computer simultaneously by switching processor frequently.
- Distributed computing accesses to shared resources among geographically dispersed computers in a transparent way.

So, option (B) is correct.

SPOOL is an acronym for **simultaneous peripheral operations on-line**.

- A large buffer from hard disk
 - Buffer can store the data through I/O
 - Because I/O is slow and CPU is fast.
- In a computer system peripheral equipments, such as printers and punch card readers etc (batch processing), are very slow relative to the performance of the rest of the system.

SPOOL is an acronym for **simultaneous peripheral operations on-line**.

- In computing, spooling is a specialized form of multi-programming for the purpose of copying data between different devices
- It is usually used for mediating between a computer application and a slow peripheral, such as a printer.
- A dedicated program, the **spooler**, maintains an orderly sequence of jobs for the peripheral and feeds it data at its own rate.

ISRO | ISRO CS 2017 – May | Question 71

Which of the following statement is true?

- (A) Hard real time OS has less jitter than soft real time OS
- (B) Hard real time OS has more jitter than soft real time OS
- (C) Hard real time OS has equal jitter as soft real time OS
- (D) None of the above

ISRO | ISRO CS 2017 – May | Question 71

Which of the following statement is true?

- (A)** Hard real time OS has less jitter than soft real time OS
- (B)** Hard real time OS has more jitter than soft real time OS
- (C)** Hard real time OS has equal jitter as soft real time OS
- (D)** None of the above

Answer: (A)

Explanation: Jitter is the variation / displacement between the signals or data being sent.

Hard real operating systems deal with more sensitive systems which require strict time deadlines like engine control systems, satellite launching systems etc while soft real operating systems do not require strict timing constraints and a bit delay is permissible, like mobile phones, online database systems. So hard real operating systems require minimised jitter.

ISRO | ISRO CS 2008 | Question 51

What is the name of the operating system that reads and reacts in terms of actual time.

- A. Batch system
- B. Quick response system
- C. Real time system
- D. Time sharing system

ISRO | ISRO CS 2008 | Question 51

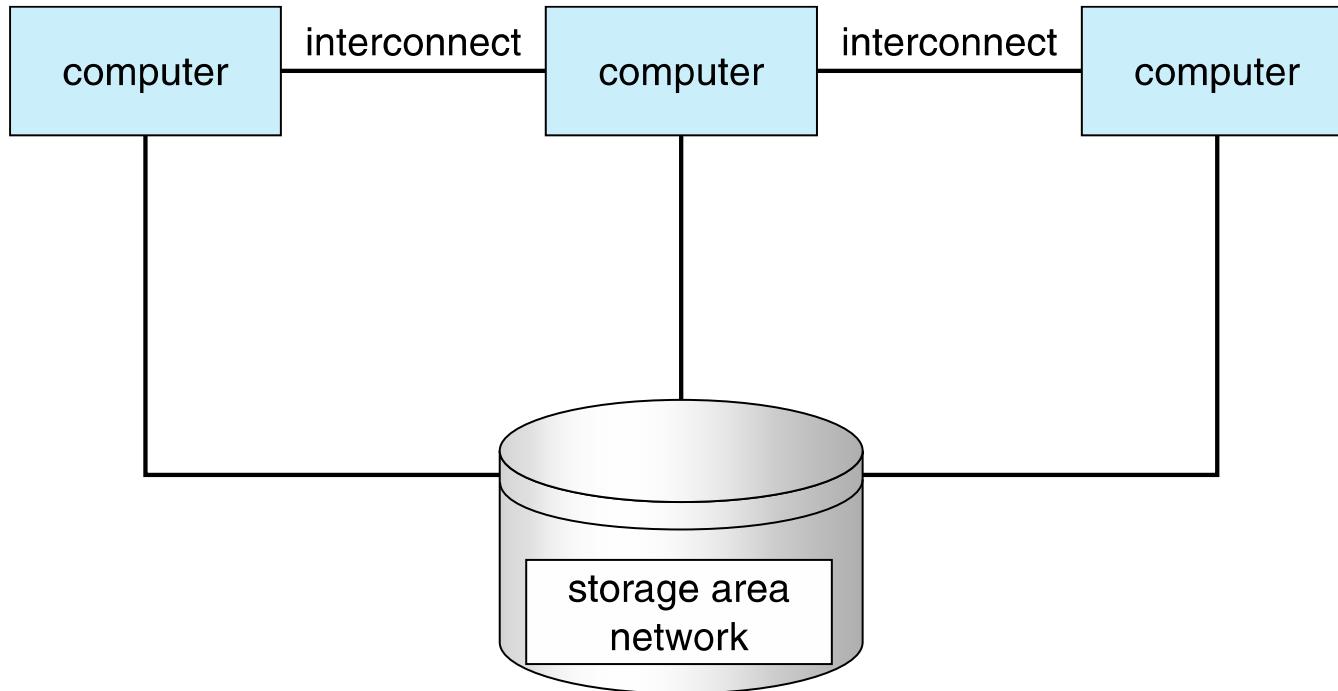
What is the name of the operating system that reads and reacts in terms of actual time.

- A. Batch system
- B. Quick response system
- C. Real time system
- D. Time sharing system

Answer: (C)

Explanation: A real-time operating system is an operating system that guarantees to process events or data by a specific moment in time.
Option (C) is correct.

Clustered Systems



- Like multiprocessor systems, but multiple systems working together Usually sharing storage via a **storage-area network (SAN)**

Clustered Systems

- Provides a **high-availability** service which survives failures
- Clusters are used for **high-performance computing (HPC)**
 - Applications must be written to use **parallelization**
- Some have **distributed lock manager (DLM)** to avoid conflicting operations

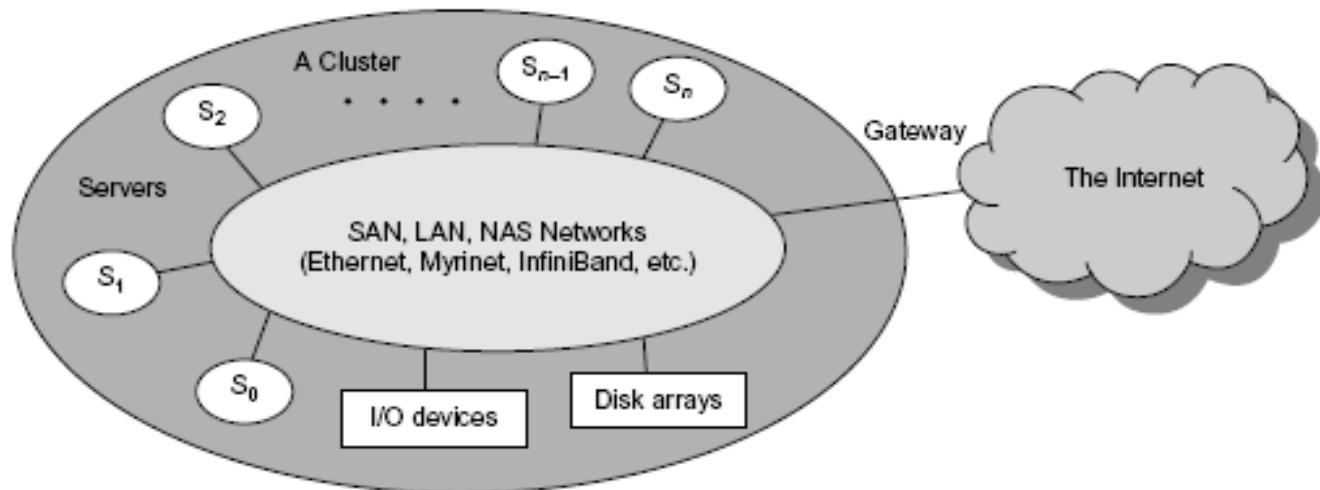
Cluster for Massive Parallelism

- A collection of
 - interconnected stand alone computers
 - which can work together collectively and
 - cooperatively as a single integrated computing resource pool.
- Deployed to
 - improve speed and/or reliability over that provided by a single computer, while
 - being much more cost effective than single computer of comparable speed or reliability

Cluster Architecture

Through hierarchical construction using a SAN, LAN or WAN, one can build scalable clusters with an increasing number of nodes.

The cluster is connected to the internet via a virtual private network(VPN) gateway.



Single System Image

- An ideal cluster should merge multiple system images into a single system image(SSI).
- Cluster designers desire a cluster operating system to support SSI at various levels, including the sharing of CPUs, memory and I/O across all cluster nodes.
- SSI is an illusion created by software or hardware that presents a collection of resources as one integrated, powerful resource.
- SSI makes the cluster appear like a single machine to the user.

SSI

- Suppose a workstation has:
 - A 300 Mflops/second processor
 - 512 MB of memory
 - 4 GB disk
 - Support 50 active users and 1,000 processes.
- By clustering 100 such workstations, we should get a single system that is equivalent to one huge workstation, or a megastation, that has:
 - 30 Gflops/second processor
 - 50 GB of memory
 - 400 GB disk
 - Support 5,000 active users and 100,000 processes
- This is an appealing goal, but it is very difficult to achieve.
- SSI techniques are aimed at achieving this goal.

- **Asymmetric clustering** has one machine in hot-standby mode
- **Symmetric clustering** has multiple nodes running applications, monitoring each other

Clustered Systems

- Asymmetric clustering-
 - One m/c is in hot standby mode while the other (**primary node**) is running applications.
 - The hot standby host does nothing but monitor the active server.
 - If that server fails, the hot standby host becomes the active server.
 - The **standby node** is powered on (hot) and running some monitoring programs to communicate heartbeat signals to check the status of the primary node, but is not actively running other useful workloads.

Clustered Systems

- Symmetric clustering-
 - Two or more hosts are running applications and they are monitoring each other.
 - More efficient, as it uses all of the available hardware.

OS Objectives/Operations

- Process Management Activities
- Memory Management
- Storage Management
- Mass-Storage Management
- Protection System

Process Management Activities

The operating system is responsible for the following activities in connection with process management:

- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication
- Providing mechanisms for deadlock handling

Memory Management

- To execute a program
 - All (or part) of the instructions must be in memory
 - All (or part) of the data that is needed by the program must be in memory.
- Memory management determines what is in memory and when
 - Optimizing CPU utilization and computer response to users

Memory Management

- Memory management activities
 - Keeping track of the parts of memory are currently being used and by whom
 - Deciding which processes (or parts thereof) and data to move into and out of memory
 - Allocating and deallocating memory space as needed

Storage Management

- OS provides uniform, logical view of information storage
 - Abstracts physical properties to logical storage unit - **file**
 - Each medium is controlled by device (i.e., disk drive, tape drive)
 - Varying properties include
 - **access speed,**
 - **capacity,**
 - **data-transfer rate,**
 - **access method (sequential or random)**

File System Storage Management

- File-System management
 - Files usually organized into directories
 - Access control on most systems to determine who can access what
 - OS activities include
 - Creating and deleting files and directories
 - Primitives to manipulate files and directories
 - Mapping files onto secondary storage
 - Backup files onto stable (non-volatile) storage media

Mass-Storage Management

- Usually **disks** used to store
 - data that does not fit in main memory or
 - **data that must be kept for a “long” period of time**
- Entire speed of computer operation hinges on disk subsystem and its algorithms
- OS activities
 - Free-space management
 - Storage allocation
 - **Disk scheduling**

Mass-Storage Management

- Some storage need not be fast
 - Tertiary storage includes optical storage, magnetic tape
 - Still must be managed – by OS or applications
 - Varies between WORM (write-once, read-many-times) and RW (read-write)

Performance of Various Levels of Storage

Level	1	2	3	4	5
Name	registers	cache	main memory	solid state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25 - 0.5	0.5 - 25	80 - 250	25,000 - 50,000	5,000,000
Bandwidth (MB/sec)	20,000 - 100,000	5,000 - 10,000	1,000 - 5,000	500	20 - 150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape

Movement between levels of storage hierarchy can be explicit or implicit

Protection and Security

- **Protection** – any mechanism for controlling access of processes or users to resources defined by the OS
- **Security** – defense of the system against internal and external attacks
 - Huge range, including
 - denial-of-service,
 - worms,
 - viruses,
 - identity theft,
 - theft of service

Protection and Security

- Systems generally first distinguish among users, to determine who can do what
 - User identities (user IDs, security IDs) include name and associated number, one per user
 - User ID then associated with all files, processes of that user to determine access control
 - Group identifier (group ID) allows set of users to be defined and controls managed, then also associated with each process, file
 - Privilege escalation allows user to change to effective ID with more rights

Protection and Security

- A mechanism that ensures that the files, memory segments, CPU and other resources
- can be operated on by only those processes that have gained proper authorization from the OS.

Operating System Functions/Services

- One set of operating-system services **provides functionalities** that are helpful to the user:
 - **User interface** - Almost all operating systems have a user interface (**UI**).
 - Varies between **Command-Line (CLI)**, **Graphics User Interface (GUI)**, **Batch**
 - **Program execution** - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
 - **I/O operations** - A running program may require I/O, which may involve a file or an I/O device

Operating System Functions/Services (Cont.)

- One set of operating-system services provides functions that are helpful to the user (Cont.):

File-system manipulation –

Programs need to

- read and write files and directories,
- create and delete them,
- search them,
- list file Information,
- permission management.

Operating System Functions/Services (Cont.)

- One set of operating-system services provides functions that are helpful to the user (Cont.):

Communications –

- Processes may exchange information,
 - on the same computer or
 - between computers over a network
- Communications may be via
 - shared memory or
 - through message passing (packets moved by the OS)

Operating System Functions/Services (Cont.)

- One set of operating-system services provides functions that are helpful to the user (Cont.):
 - **Error detection** – OS needs to be constantly aware of possible errors
 - May occur in the CPU and memory hardware, in I/O devices, in user program
 - For each type of error, OS should take the appropriate action to ensure correct and consistent computing
 - Debugging facilities

Operating System Functions/Services (Cont.)

- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing
 - Resource allocation
 - Accounting
 - Protection and security

Operating System Functions/Services (Cont.)

Resource allocation –

- When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
 - Many types of resources -
 - **CPU cycles,**
 - **main memory,**
 - **file storage,**
 - **I/O devices.**
 - Some have
 - **special allocation codes**(CPU cycles, main memory, file storage) while
 - others (such as I/O devices) may have much **more general request** and release code.

Operating System Functions/Services (Cont.)

Accounting –

- To keep track of
 - which users
 - use how much and
 - what kinds of computer resources
- This record can be used for billing purposes or
 - for simply accumulating usage statistics.

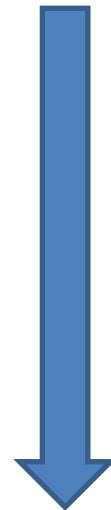
Operating System Functions/Services (Cont.)

Protection and security –

- **Protection** involves ensuring that all access to system resources is controlled
- **Security** of the system from outsiders is also important, requires
 - user authentication,
 - extends to defending external I/O devices from invalid access attempts,
 - recording all such connections for detection of break ins.

Operating System Structure

- General-purpose OS is very large program
- Various ways to structure
 - Simple structure – MS-DOS
 - More complex -- UNIX
 - Layered – an abstraction
 - Microkernel -Mach



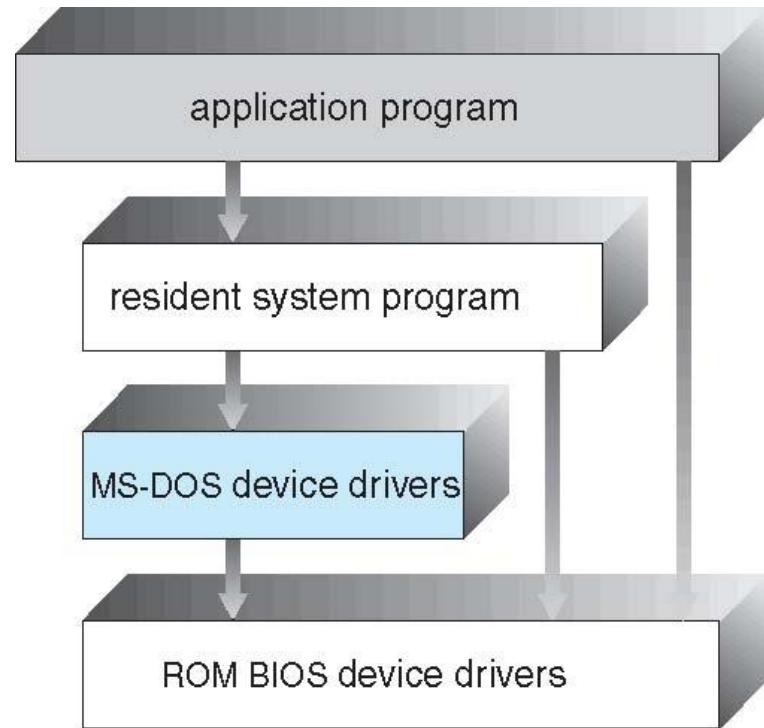
Simple Structure -- MS-DOS

- MS-DOS – written to provide **the most functionality** in the least space
 - Designed and implemented by a few people who had no idea that it would become so popular
 - Written by Tim Paterson



Simple Structure -- MS-DOS

- MS-DOS – written to provide the most functionality in the least space
 - Because of the limited hardware it was build on,
 - it was Not divided into modules carefully
 - Although MS-DOS has some structure,
 - its interfaces and levels of functionality
 - are not well separated



Simple Structure -- MS-DOS

- **Development?**
 - Microsoft Disk Operating System
 - OS for x86-based PCs mostly developed by Microsoft.
 - Collectively, MS-DOS,
 - some operating systems attempting to be compatible with MS-DOS, are sometimes referred to as "DOS" (which is also the generic acronym for disk operating system)

Simple Structure -- MS-DOS

- Competitors?
 - It was gradually superseded by OS offering a graphical user interface (GUI),
 - in various generations of the graphical Microsoft Windows operating system.

Simple Structure -- MS-DOS

- Competitors?
 - Although **MS-DOS and PC DOS** were initially developed in parallel by Microsoft and IBM,
 - the **two products diverged after twelve years, in 1993, with recognizable differences in compatibility, syntax, and capabilities.**

Simple Structure -- MS-DOS

- Closure?
 - MS-DOS went through **eight versions, until development ceased in 2000**
 - Versions
 - MS-DOS 1.x
 - MS-DOS 2.x
 - MS-DOS 3.x
 - MS-DOS 4.0 / MS-DOS 4.x
 - MS-DOS 5.x
 - MS-DOS 6.x
 - MS-DOS 7 (as part of Windows 9x)
 - Localized versions

Non Simple Structure -- UNIX

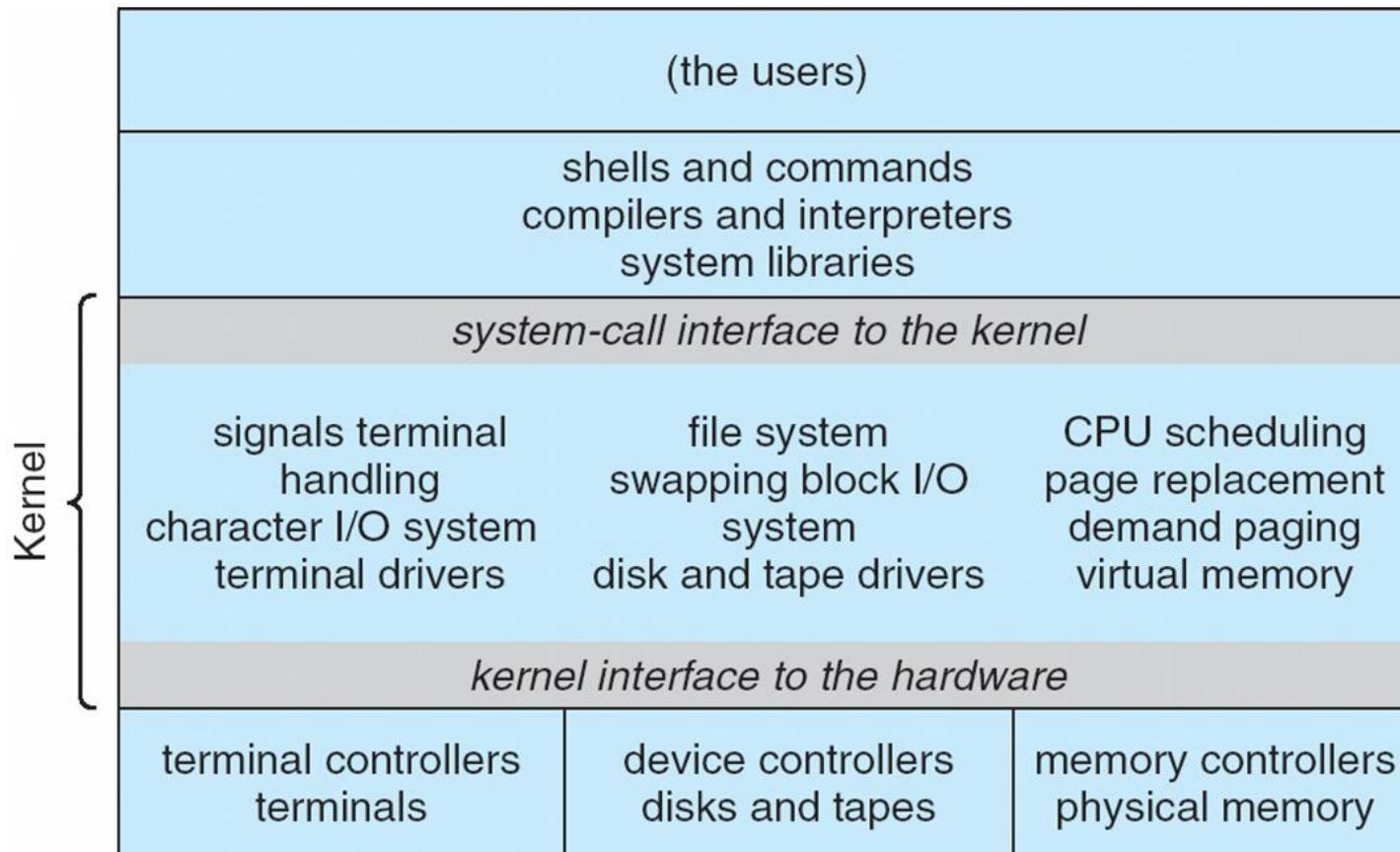
UNIX – **limited by hardware functionality**, the original UNIX operating system had **limited structuring**.

The UNIX OS consists of two separable parts

- **Systems programs**
- **The kernel**

Traditional UNIX System Structure

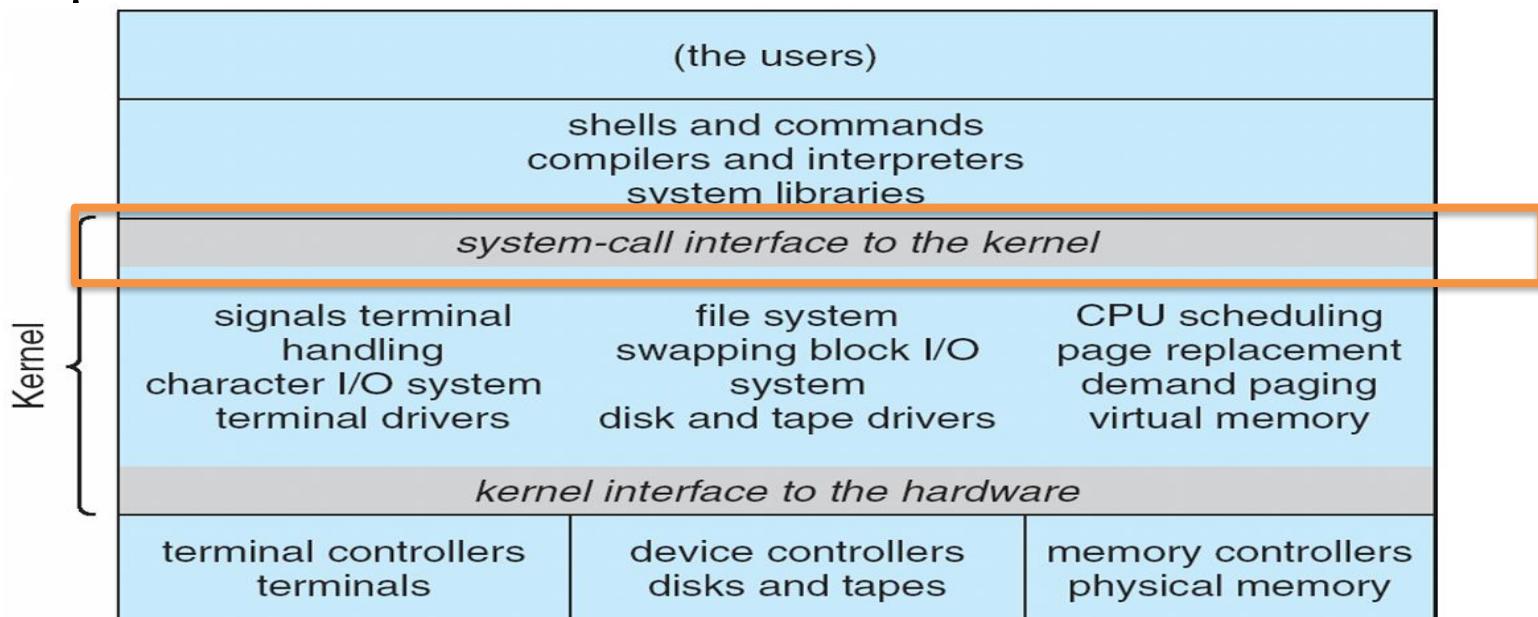
Beyond simple but not fully layered



Non Simple Structure -- UNIX

The kernel

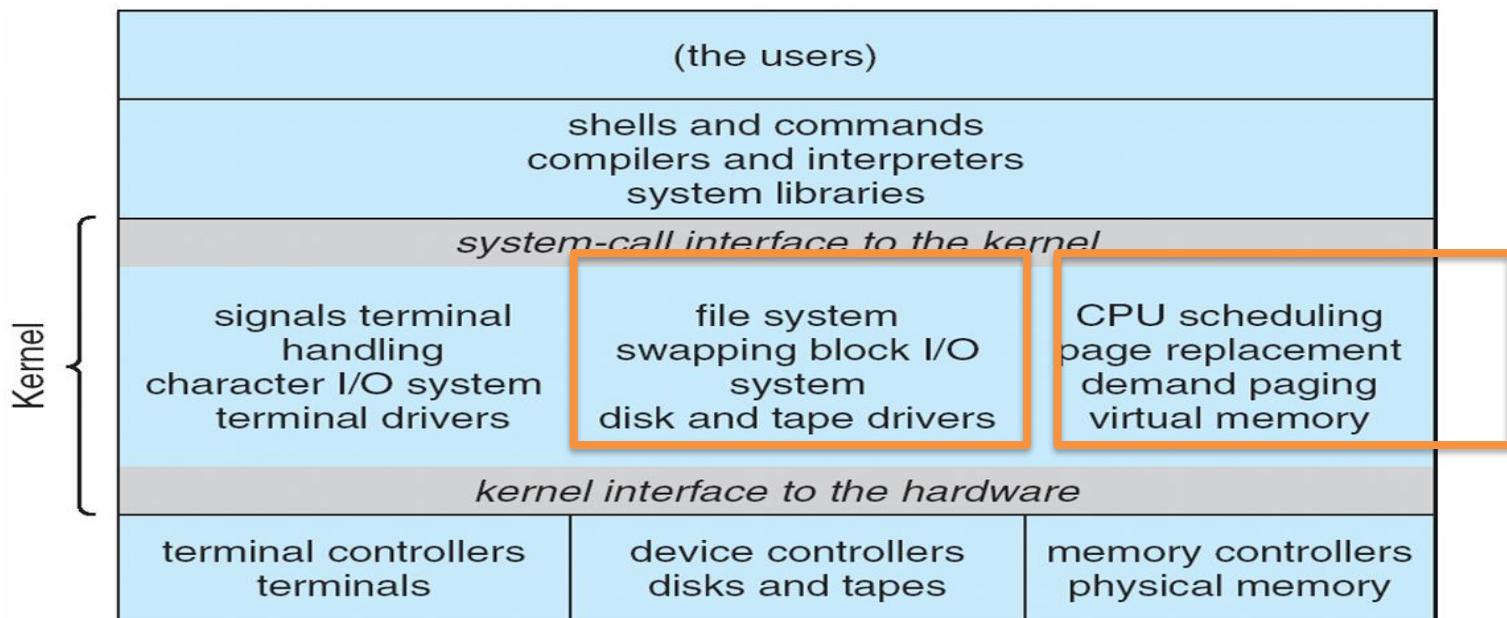
- Consists of everything **below the system-call interface and above the physical hardware**
- Further separated into series of interfaces and device drivers which were added and expanded over the years as UNIX evolved.



Non Simple Structure -- UNIX

The kernel

- Provides the **file system, CPU scheduling, memory management, and other functions**; a large number of functions for one level
- An enormous amount of functionality to be combined into one level,
- This makes **UNIX difficult to enhance**,
 - as **changes in one level could adversely affect other areas**.



UNIX System Structure

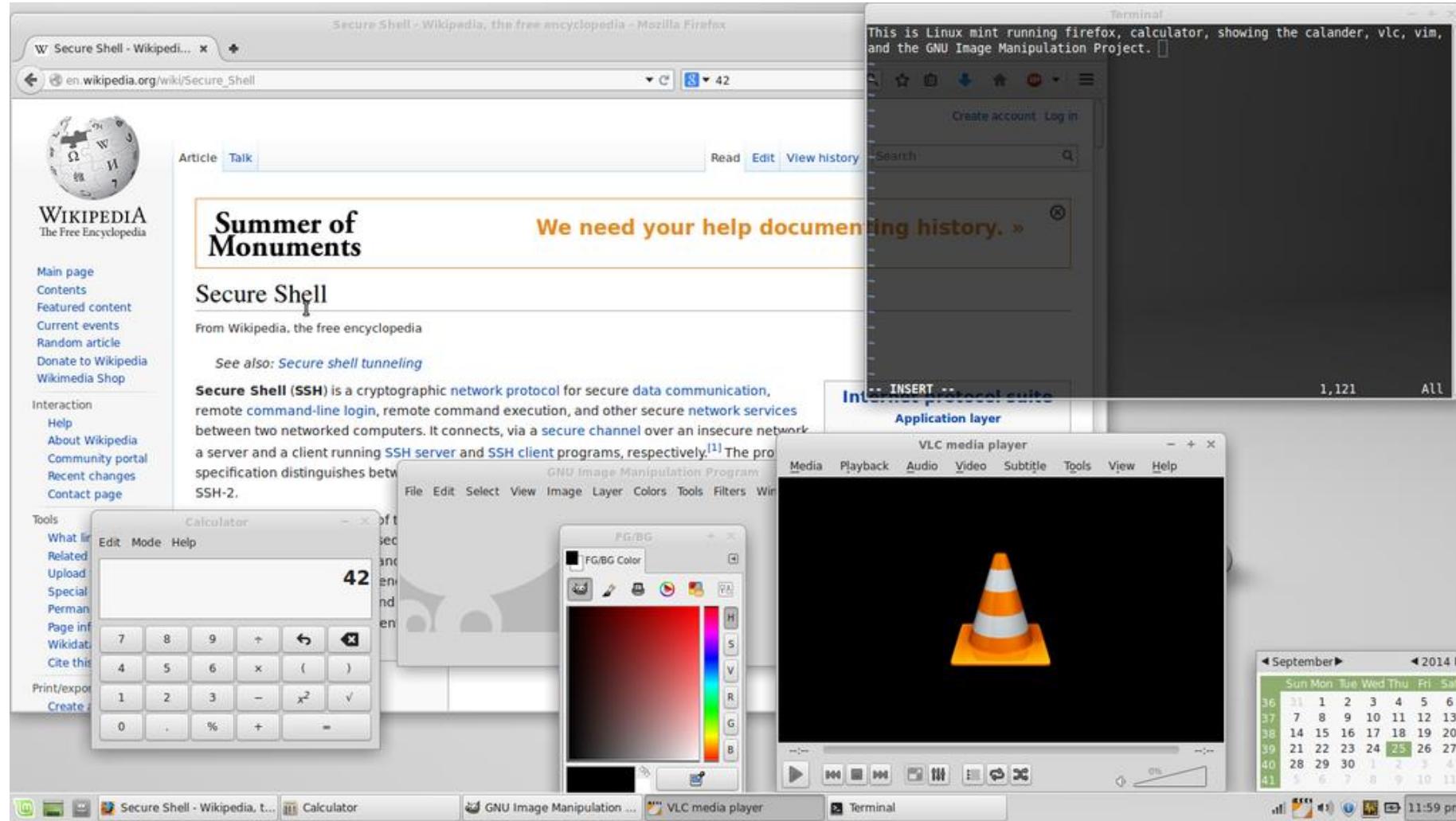
- **New versions of UNIX are designed to use more advanced hardware.**
- Given proper hardware support,
 - OS may be broken into pieces
 - that are smaller and more appropriate than those allowed
 - by the original MS-DOS or UNIX systems.

UNIX

- Unix is a family of multitasking, multiuser computer operating systems that derive from the original AT&T Unix,
- Development starting in the **1970s at the Bell Labs research center by Ken Thompson, Dennis Ritchie, and others**

MULTITASKING

Multitasking is the concurrent execution of multiple tasks (also known as processes) over a certain period of time. New tasks can interrupt already started ones before they finish



MULTIUSER

MULTIUSER

Some multi-user operating systems such as Windows versions from the Windows NT family support simultaneous access by multiple users .

For example, via Remote Desktop Connection as well as the Local user

An example is a Unix or Unix-like system where multiple remote users have access (such as via a serial port or Secure Shell) to the Unix shell prompt at the same time.

UNIX

- There are many different versions of UNIX, although they share common similarities.
- **The most popular varieties of UNIX are Sun Solaris, GNU/Linux, and MacOS X.**

- Some people think Unix and Linux as synonyms, but that's not true.

- **Linux is the clone of Unix. It has several features similar to Unix, still have some key differences.**
- **Linux is a family of open-source Unix-like operating systems based on the Linux kernel**

Difference between Linux and Unix??

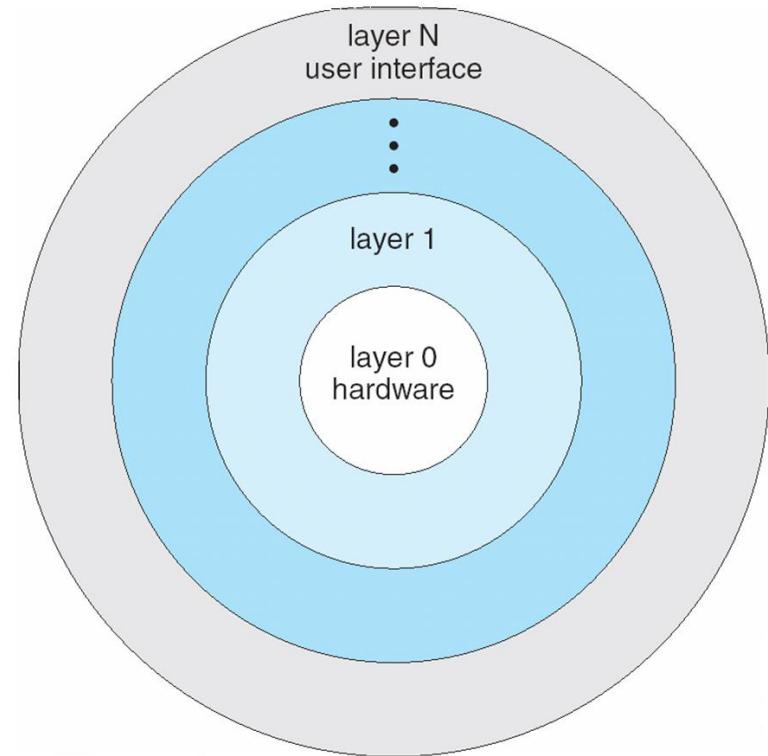
Difference between Linux and Unix

Comparison	Linux	Unix
Definition	It is an open-source operating system which is <i>freely available to everyone</i> .	It is an operating system which <i>can be only used by its copyrighters</i> .
Examples	It has different distros like Ubuntu, Redhat, Fedora, etc	IBM AIX, HP-UX and Sun Solaris.
Users	Nowadays, Linux is in great demand. Anyone can use Linux whether a home user, developer or a student.	It was developed mainly for servers, workstations and mainframes.
Usage	Linux is used everywhere from servers, PC, smartphones, tablets to mainframes and supercomputers.	It is used in servers, workstations and PCs.
Cost	Linux is <i>freely distributed, downloaded, and distributed through magazines also</i> . And priced distros of Linux are also cheaper than Windows.	Unix copyright vendors decide different costs for their respective Unix Operating systems.
Development	As it is open source, it is developed by sharing and <i>collaboration of codes by world-wide developers</i> .	Unix was developed by <i>AT&T Labs, various commercial vendors and non-profit organizations</i> .
Manufacturer	Linux kernel is developed by the <i>community of developers from different parts of the world</i> . Although the father of Linux, Linus Torvalds oversees things.	Unix has three distributions IBM AIX, HP-UX and Sun Solaris. Apple also uses Unix to make OSX operating system.

GUI	Linux is command based but some distros provide GUI based Linux. Gnome and KDE are mostly used GUI.	Initially it was command based OS, but later Common Desktop Environment was created. Most Unix distributions use Gnome.
Interface	The default interface is BASH (Bourne Again SHeLL). But some distros have developed their own interfaces.	It originally used Bourne shell. But is also compatible with other GUIs.
File system support	Linux supports more file system than Unix.	It also supports file system but lesser than Linux.
Coding	Linux is a Unix clone, behaves like Unix but doesn't contain its code.	Unix contain a completely different coding developed by AT&T Labs.
Operating system	Linux is just the kernel.	Unix is a complete package of Operating system.
Security	It provides higher security. Linux has about 60-100 viruses listed till date.	Unix is also highly secured. It has about 85-120 viruses listed till date
Error detection and solution	As Linux is open-source, whenever a user post any kind of threat, developers from all over the world start working on it. And hence, it provides faster solution.	In Unix, users have to wait for some time for the problem to be resolved.

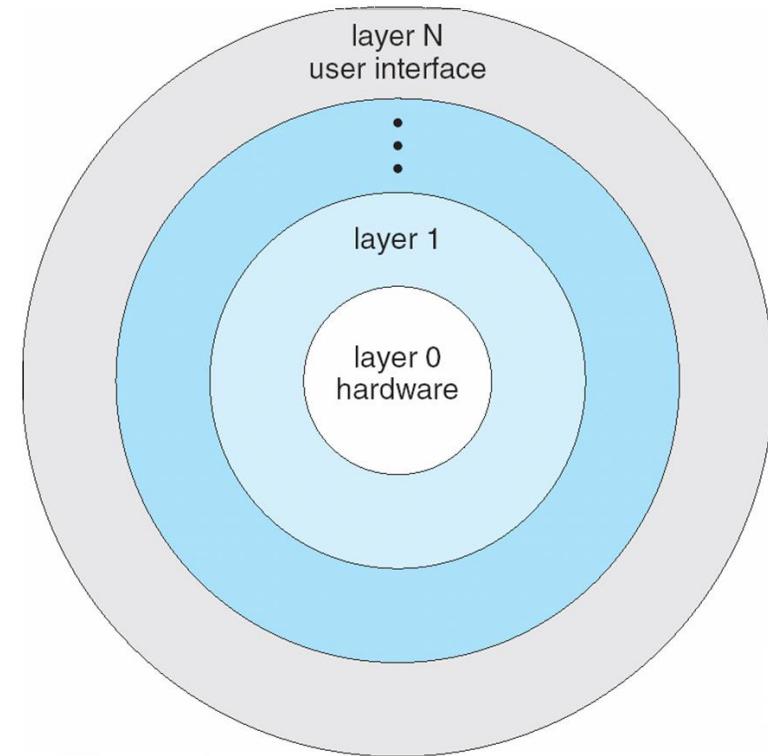
Layered Approach

- The operating system is divided
 - into a number of layers (levels),
 - each built on top of lower layers.
 - **The bottom layer (layer 0), is the hardware;**
 - **The highest (layer N) is the user interface.**



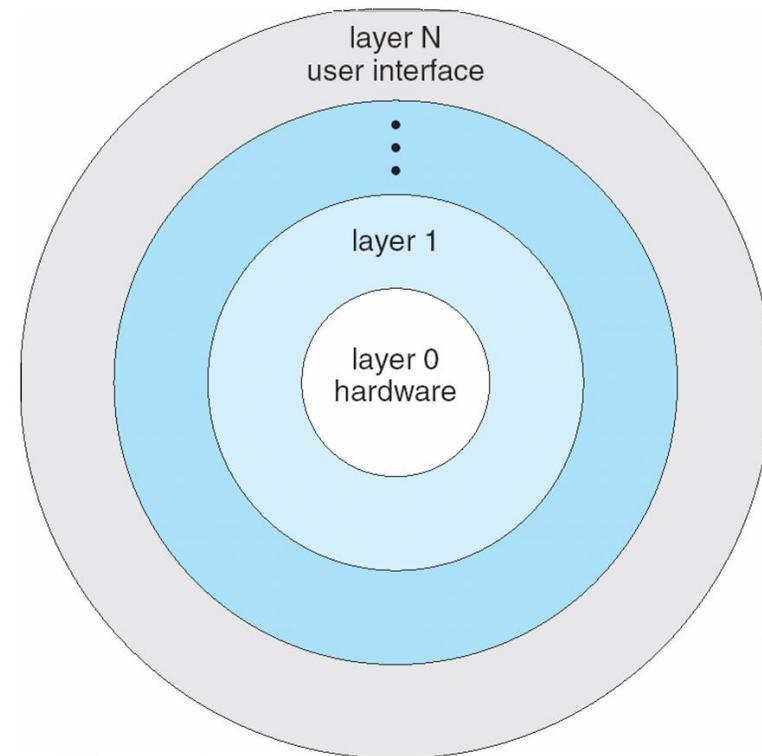
Layered Approach

- An OS layer say M,
 - **consists of data structures and**
 - **a set of routines**
 - **that can be invoked by higher layers.**
 - **Layer M can invoke operations on lower layers.**
- With modularity,
 - layers are selected such that
 - **each uses functions and services of only lower-level layers**



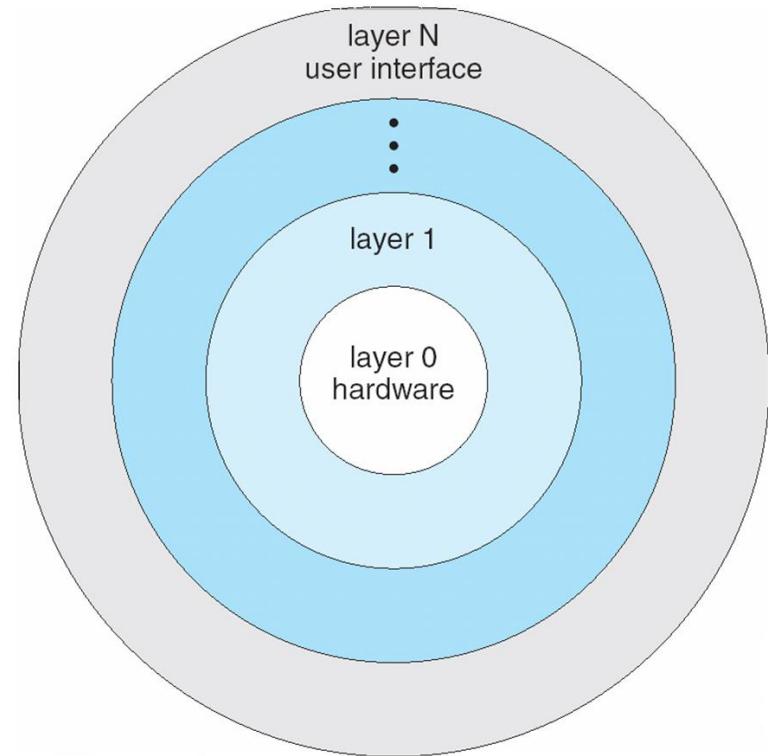
Layered Approach

- It simplifies debugging and system verification.
- **The 1st layer can be debugged**
 - **without any concern for the rest of the system.**
- Once the 1st layer is debugged,
 - its correct functioning can be assumed while
 - the second layer is being debugged and so on.



Layered Approach

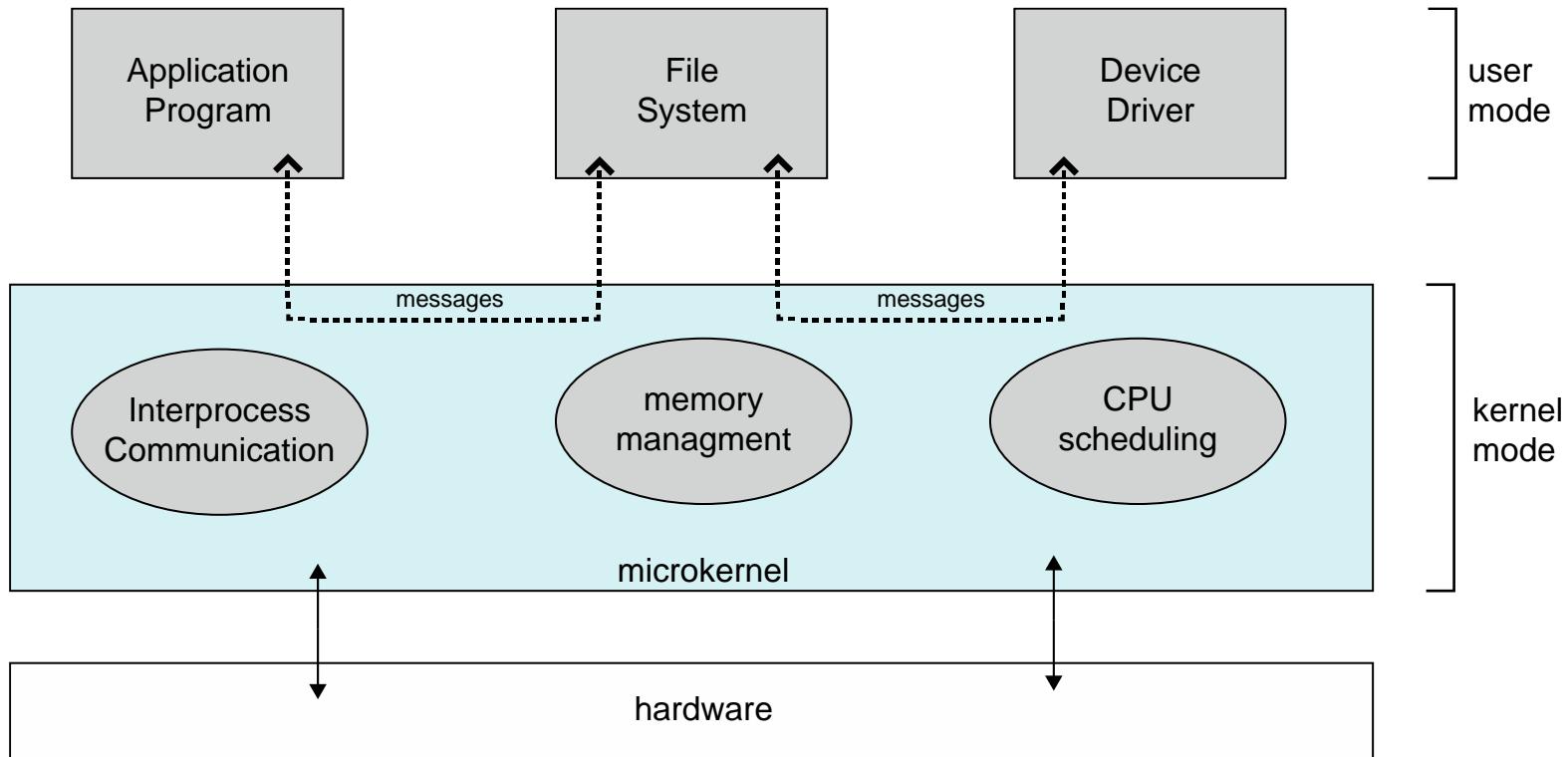
- If error arises during debugging of a particular layer,
 - **the error must be on that layer,**
 - because the **layers below it are already debugged.**



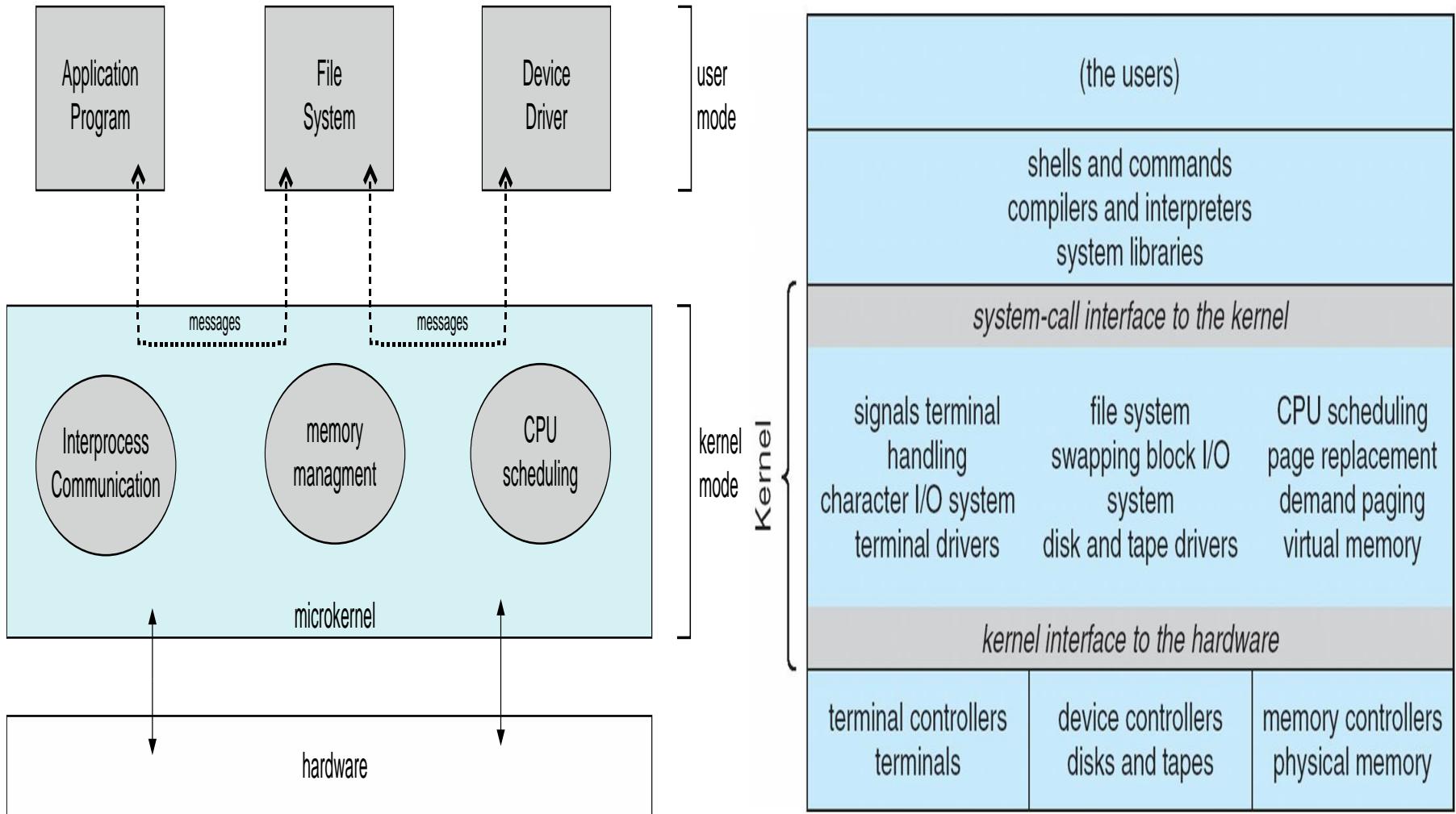
Microkernel System Structure

- Moves as much from the kernel into user space
- **Mach** example of **microkernel**
 - Mac OS X kernel (**Darwin**) partly based on Mach
- Communication takes place between user modules using **message passing**

Microkernel System Structure



Comparison-Microkernel System UNIX Kernel



Microkernel System Structure

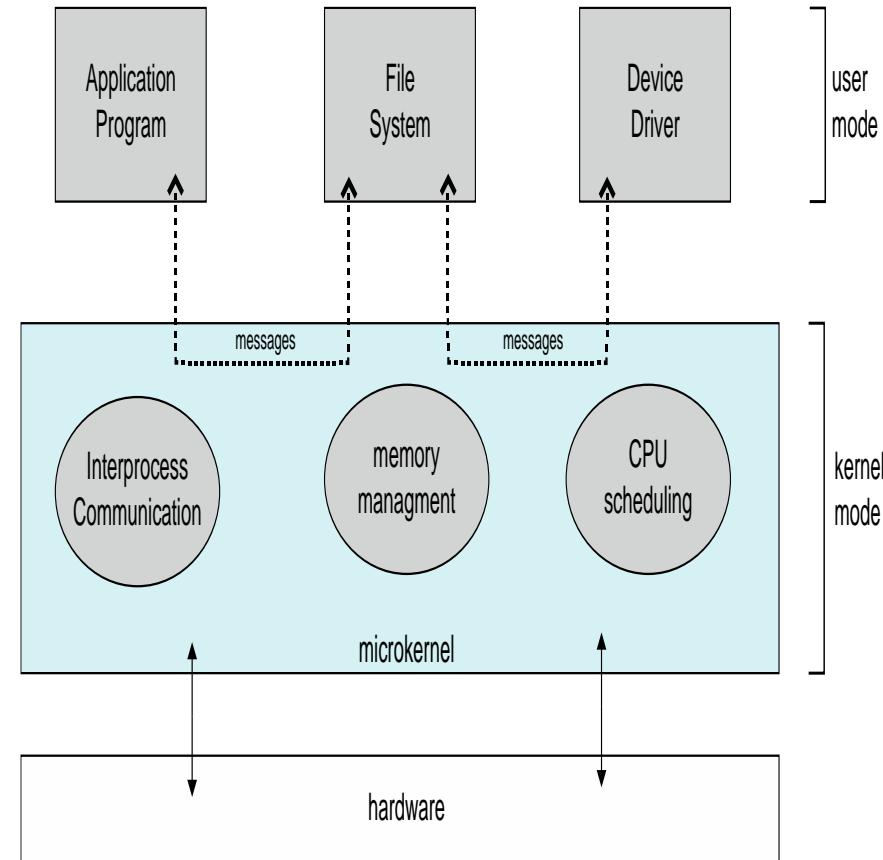
How?

- **Remove all non essential components from the kernel and**
 - implementing them as system and user level programs.
- **Result=Smaller=Micro Kernel**

Microkernel System Structure

How?

- Microkernels provide
 - **minimal process and memory management, communication facility.**
- Main fn
 - **Provide a communication facility**
 - between client program and various services that are also running in user space,
 - **Communication provided by Message passing**



Microkernel System Structure

When ? Where?

- As the UNIX OS expanded, kernel became large and difficult to manage.
- In mid **1980s**, **researchers at Carnegie Mellon University developed an OS called Mach**,
 - that modularizes the kernel using **The Microkernel approach**.

Microkernel System Structure

- Benefits:
 - **Easier to extend** a microkernel
 - **All new services added to user space , no modification of the kernel**
 - **Easier to port** the operating system to new architectures
 - **As no modifications to kernel, changes tend to be fewer, a smaller kernel**

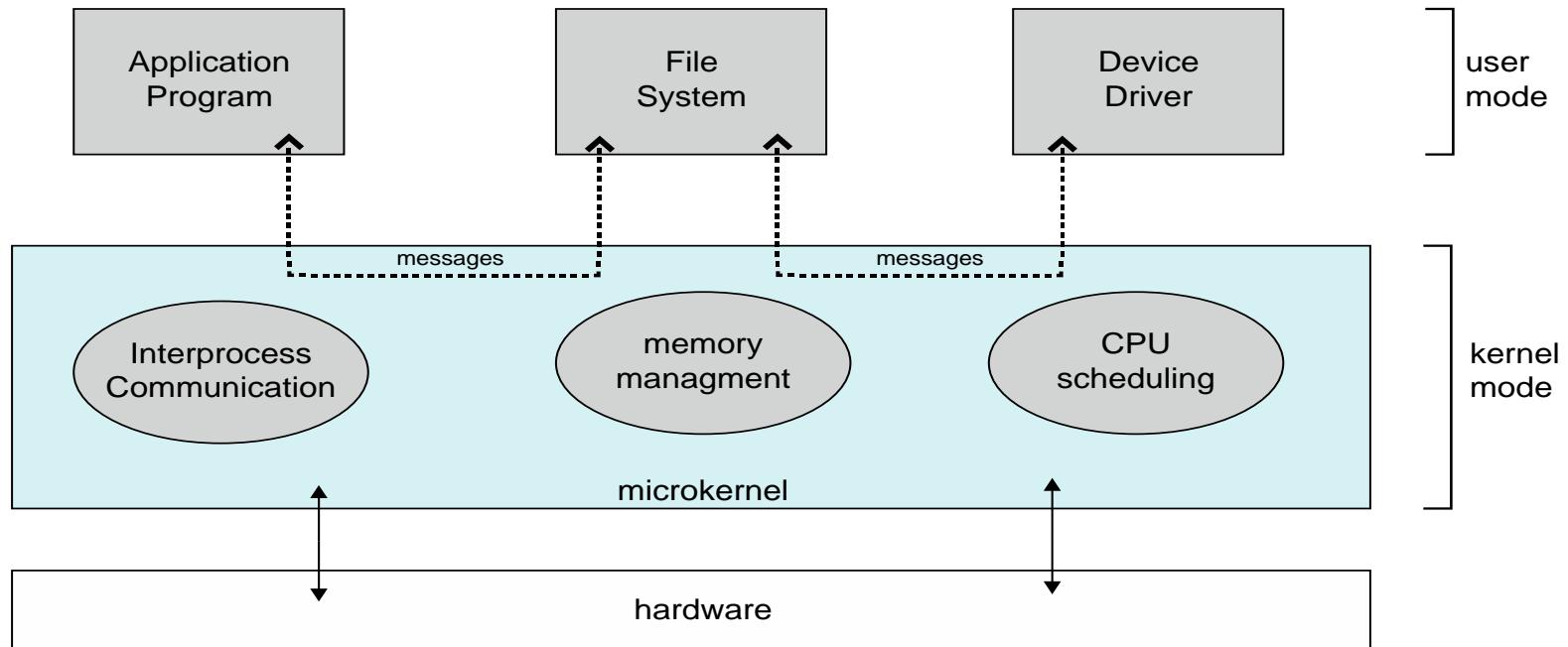
Microkernel System Structure

- Benefits:
 - More reliable (less code is running in kernel mode) and More secure
 - **More services running as user rather than kernel,**

- **If a service fails, the rest of the operating system remains untouched.**

Microkernel System Structure

- Detriments:
 - **Performance overhead** of user space to kernel space communication



Linux Kernel

- Think about it like this.
- The kernel is a **busy personal assistant** for a **powerful executive** (the **hardware**).
- It's the assistant's job :-
- to **relay messages** and requests (**processes**) from employees and the public (**users**) to the **executive**,
- to **remember what** is stored **where** (**memory**), and
- to **determine who** has **access** to the **executive** at any **given time** and for **how long**.

Courtesy: <https://www.redhat.com/en/topics/linux/what-is-the-linux-kernel>

Linux Kernel

- Linux Kernel a **busy personal assistant** that
 - **manages** all system resources and
 - that **interacts** directly with the computer **hardware.**
- The **Linux® kernel** is
 - **the main component of a Linux operating system (OS) and**
 - **is the core interface between a computer's hardware and its processes.**
 - It **communicates** between the **two**, managing resources as efficiently as possible

Linux Kernel

- The **kernel** is so named because—
 - like a **seed inside a hard shell**
 - it **exists** within the **OS** and
 - **controls all the major functions** of the **hardware**



Linux Kernel

- We need much more than this kernel
 - to produce a full operating system.

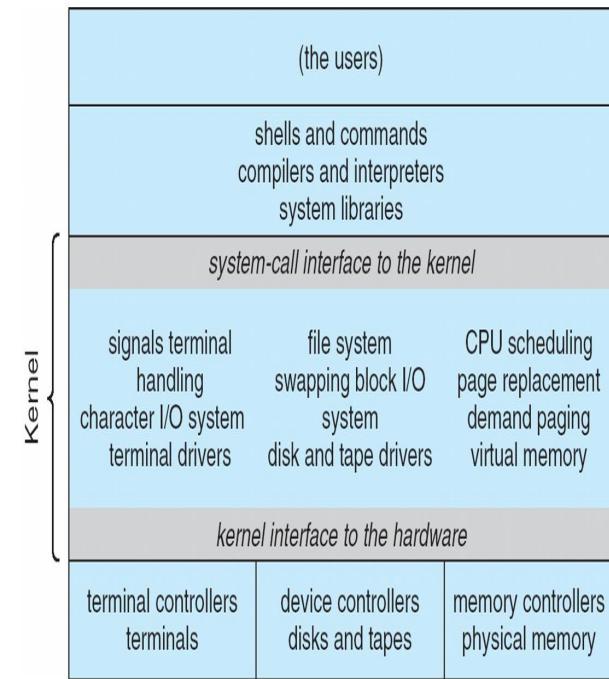
- The Linux kernel
 - **forms the core of the Linux project, but**
 - **other components make up the complete Linux operating system**

Linux Kernel

- **Linux kernel** is composed
 - entirely of code written from scratch specifically for the Linux project,
- Whereas the **Linux system** includes
 - a multitude of components some written from scratch
 - other borrowed from other development projects and
 - others created in collaboration with other teams.

Linux Kernel

- The kernel, if implemented properly,
 - is **invisible to the user**,
 - **working in its own little world known as kernel space**,
 - where it **allocates memory** and
 - **keeps track** of where everything is stored.
- **What the user sees—like web browsers and files—are known as the user space.**
- **These applications interact with the kernel**
 - through a **system call interface (SCI)**.

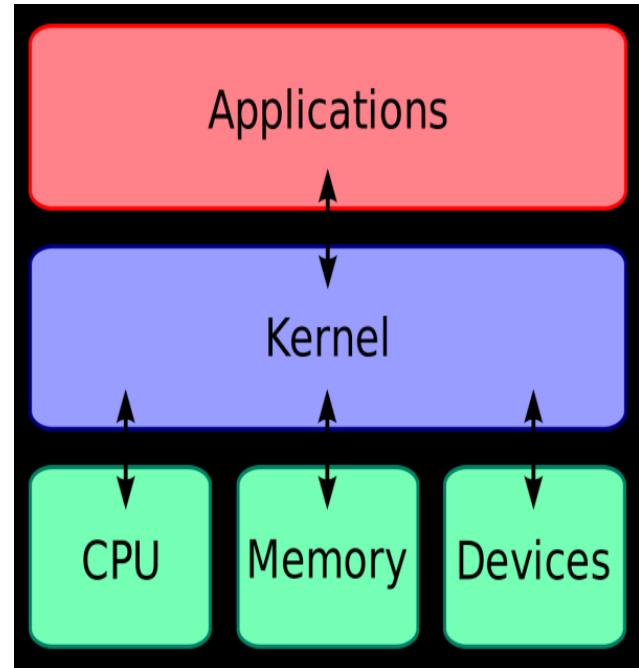


Courtesy: <https://www.redhat.com/en/topics/linux/what-is-the-linux-kernel>

Where the kernel fits within the OS?

To put the kernel in context, you can think of a Linux machine as having 3 layers:

- The hardware
- The Linux kernel: The core of the OS. (See? It's right in the middle.) It's software residing in memory that tells the CPU what to do.
- User processes: These are the running programs that the kernel manages.
 - User processes are what collectively make up user space.
 - The kernel also allows these processes and servers to communicate with each other (known as inter-process communication, or IPC).



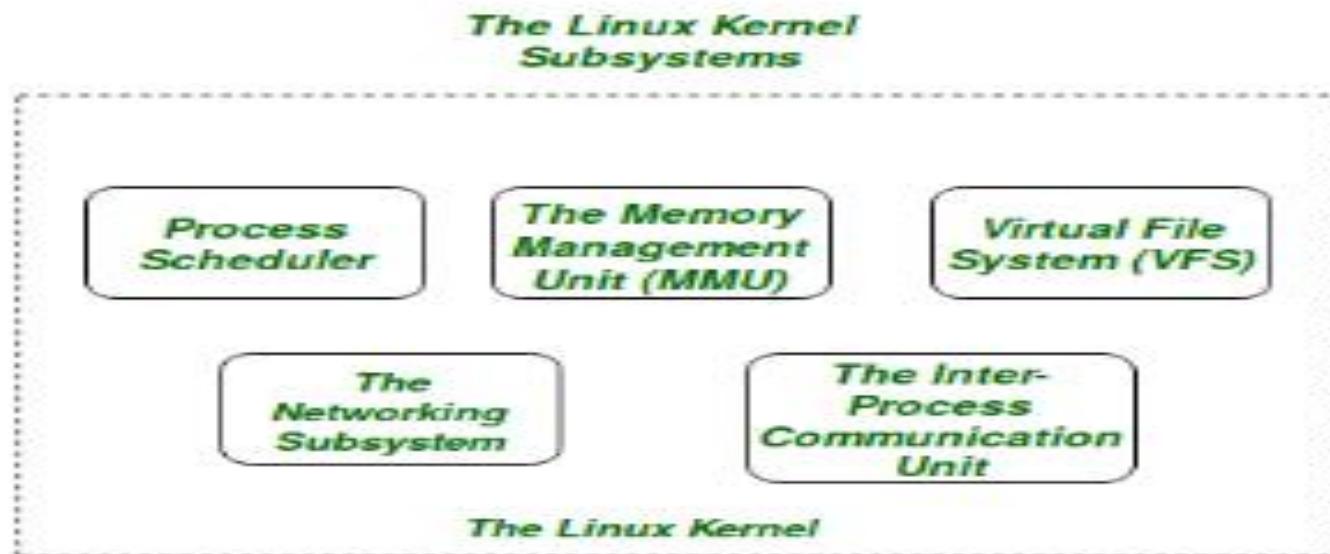
Courtesy: <https://www.redhat.com/en/topics/linux/what-is-the-linux-kernel>

Linux Kernel

The **Core Subsystems** of the **Linux Kernel** are as follows:

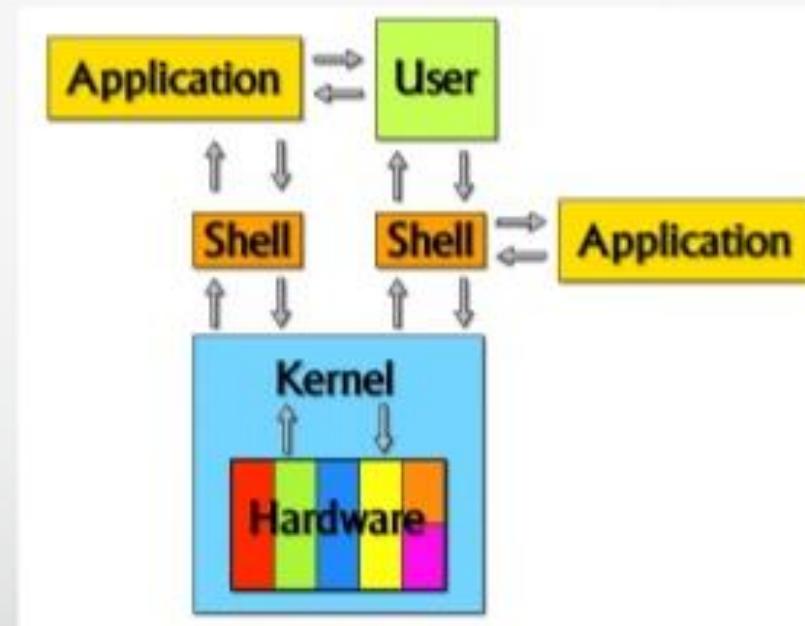
- The Process Scheduler/ Process management
- The Memory Management Unit (MMU)
- The Virtual File System (VFS) /File management
- The Networking Unit
- Inter-Process Communication Unit
- Device management/ I/O management

DIN MVP



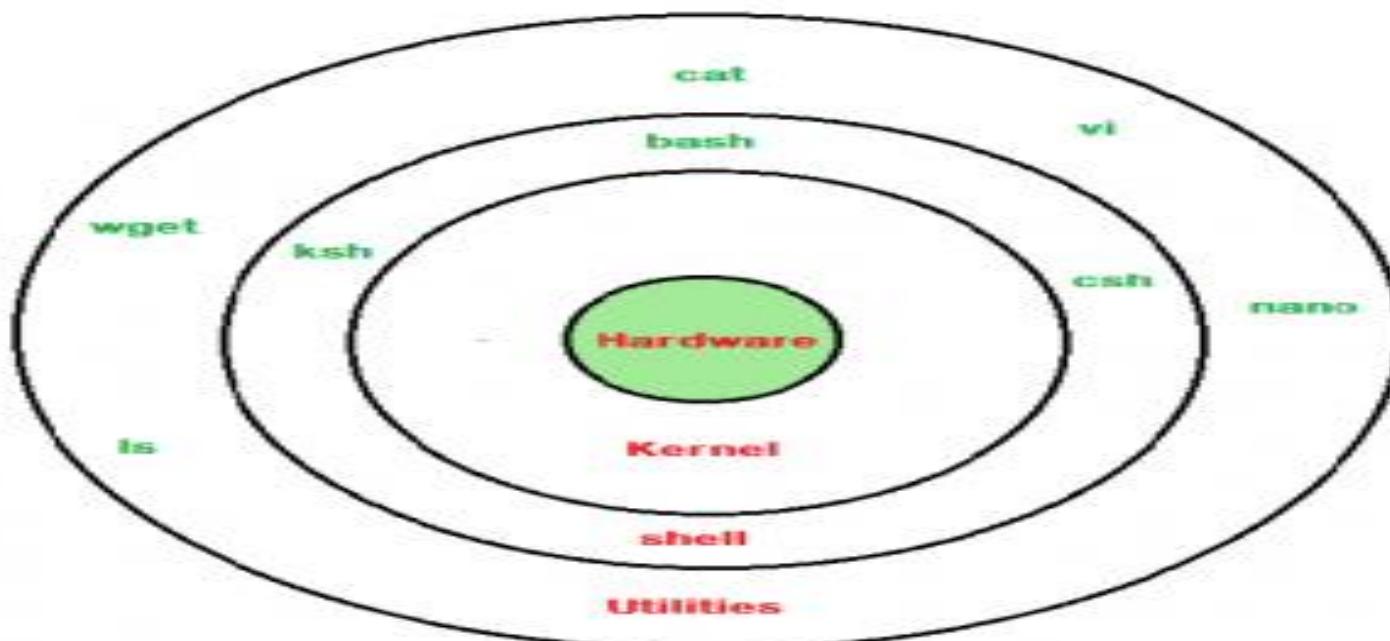
What Is Kernel?

- central module of an operating system (OS)
- the part of the operating system that loads first, and it remains in main memory.
- can be contrasted with a shell, the outermost part of an operating system that interacts with user commands.
- Manage your hardware and system resources



Linux Kernel

- It is often mistaken that **Linus Torvalds has developed Linux OS**,
 - but actually he is only responsible for development of Linux kernel.
 - **Complete Linux system = Kernel + GNU system utilities and libraries + other management scripts + installation scripts.**



What is Shell?

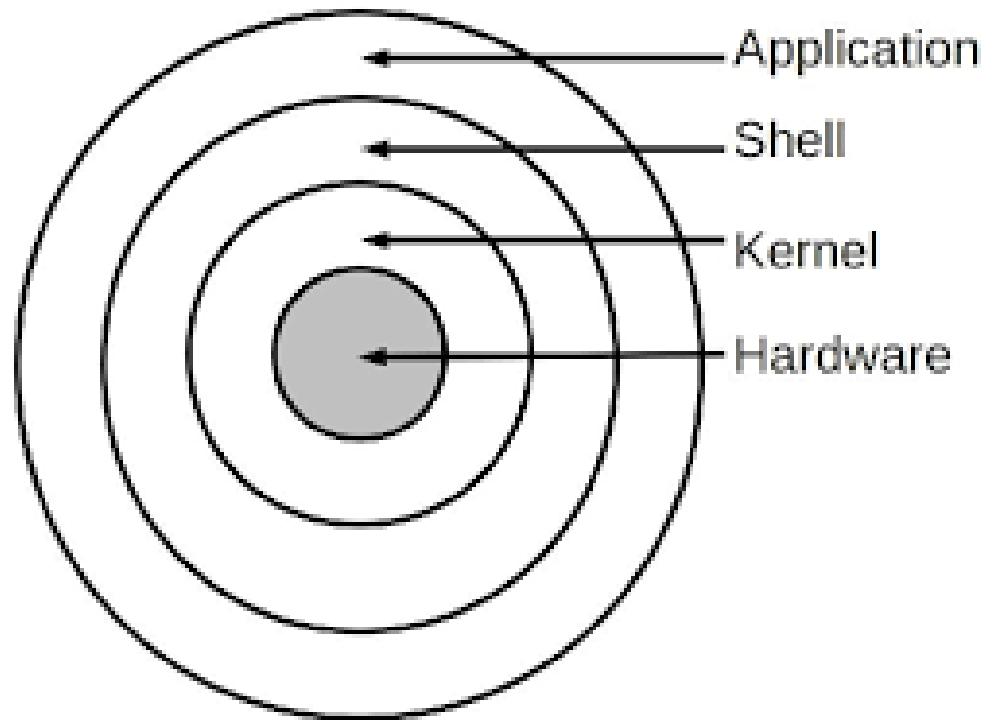
- A shell is **special user program** which provide an **interface** to **user** to **use operating system services**.
 - Shell accept **human readable commands** from user and **convert them into something which kernel can understand.**

What is Shell?

- It is a command language interpreter that execute commands read from input devices such as keyboards or from files.
- The shell gets started when the user logs in or start the terminal.

What is Shell?

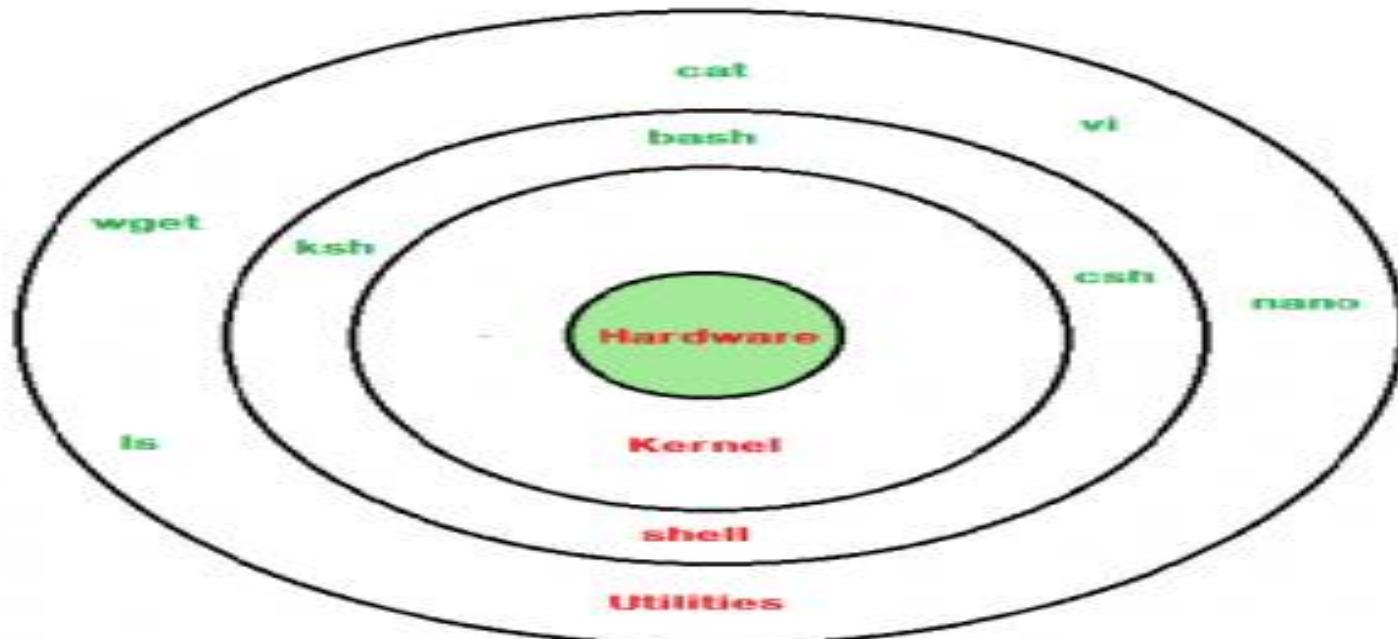
- It is named a shell because it is the outermost layer around the operating system.



What is Shell?

Shell is broadly classified into two categories-

- Command Line Shell
- Graphical shell

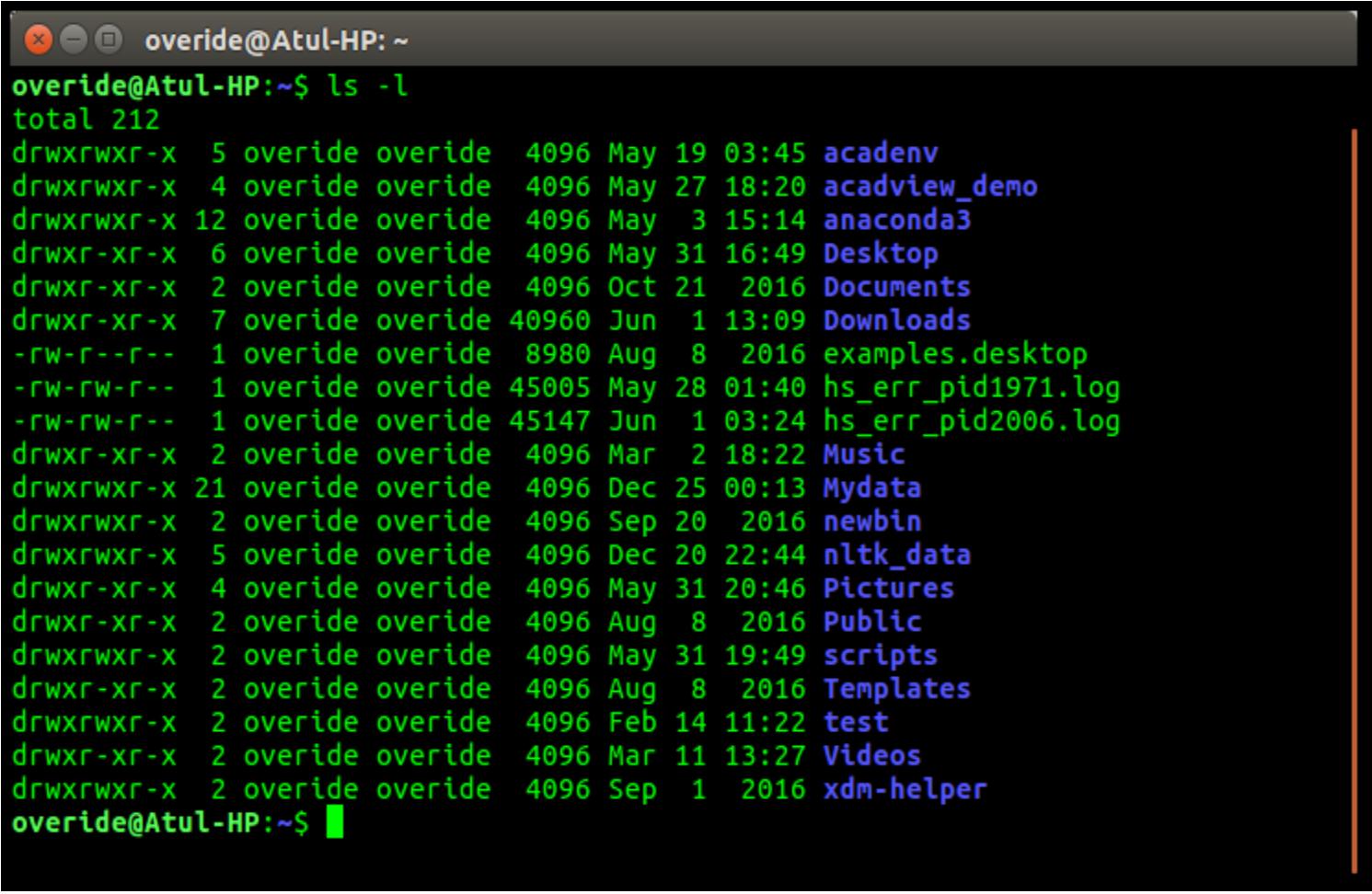


Command Line Shell

- Shell can be accessed by user using a command line interface.
- A special program called
 - Terminal in linux/macOS or
 - Command Prompt in Windows OS
 - is provided to type in the **human readable commands** such as “cat”, “ls” etc. for execution. The result is then displayed on the terminal to the user.

Command Line Shell

- A terminal in Ubuntu 16.4 system looks like this –



```
override@Atul-HP:~$ ls -l
total 212
drwxrwxr-x  5 override override  4096 May 19  03:45 acadenv
drwxrwxr-x  4 override override  4096 May 27 18:20 acadview_demo
drwxrwxr-x 12 override override  4096 May   3 15:14 anaconda3
drwxr-xr-x  6 override override  4096 May 31 16:49 Desktop
drwxr-xr-x  2 override override  4096 Oct 21  2016 Documents
drwxr-xr-x  7 override override 40960 Jun   1 13:09 Downloads
-rw-r--r--  1 override override  8980 Aug   8  2016 examples.desktop
-rw-rw-r--  1 override override 45005 May 28 01:40 hs_err_pid1971.log
-rw-rw-r--  1 override override 45147 Jun   1 03:24 hs_err_pid2006.log
drwxr-xr-x  2 override override  4096 Mar   2 18:22 Music
drwxrwxr-x 21 override override  4096 Dec 25  00:13 Mydata
drwxrwxr-x  2 override override  4096 Sep 20  2016 newbin
drwxrwxr-x  5 override override  4096 Dec 20 22:44 nltk_data
drwxr-xr-x  4 override override  4096 May 31 20:46 Pictures
drwxr-xr-x  2 override override  4096 Aug   8  2016 Public
drwxrwxr-x  2 override override  4096 May 31 19:49 scripts
drwxr-xr-x  2 override override  4096 Aug   8  2016 Templates
drwxrwxr-x  2 override override  4096 Feb 14 11:22 test
drwxr-xr-x  2 override override  4096 Mar 11 13:27 Videos
drwxrwxr-x  2 override override  4096 Sep   1  2016 xdm-helper
override@Atul-HP:~$
```

Command Line Shell

- Working with command line shell is bit difficult for the beginners because it's hard to memorize so many commands.
- **It is very powerful, it allows user to store commands in a file and execute them together. This way any repetitive task can be easily automated.**
- **These files are usually called**
 - **Batch files in Windows and**
 - **Shell Scripts in Linux/macOS systems.**

Graphical Shells

- Provide means for manipulating programs based on graphical user interface (GUI),
 - by allowing for operations such as opening, closing, moving and resizing windows, as well as switching focus between windows.
- **Window OS or Ubuntu OS can be considered as good example which provide GUI to user for interacting with program.**
- User do not need to type in command for every actions.

Graphical Shells

- Command-line shells require the user to be familiar with commands and their calling syntax, and to understand concepts about the shell-specific scripting language
- Graphical shells place a low burden on beginners and are easy to use.
- Since they also come with certain disadvantages, most GUI-enabled operating systems also provide CLI shells.

Shell Prompt

- The prompt, **\$**, which is called the **command prompt**, is issued by the shell.
- While the prompt is displayed, you can type a command.
- Shell reads your input after you press **Enter**.
- It determines the command you want to be executed by looking at the first word of your input. A word is an unbroken set of characters. Spaces and tabs separate words.

FINDING OUT YOUR SHELL

- When you are provided with a UNIX account, the system administrator chooses a shell for you.
 - To find out which shell was chosen for you, look at your prompt.
- 1) If you have a **\$** prompt, you're probably in a **Bash**, **Bourne** or a **Korn** shell.
 - 2) If you have a **%** prompt, you're probably in a **C shell**.

FINDING OUT YOUR SHELL

- Using the Echo command



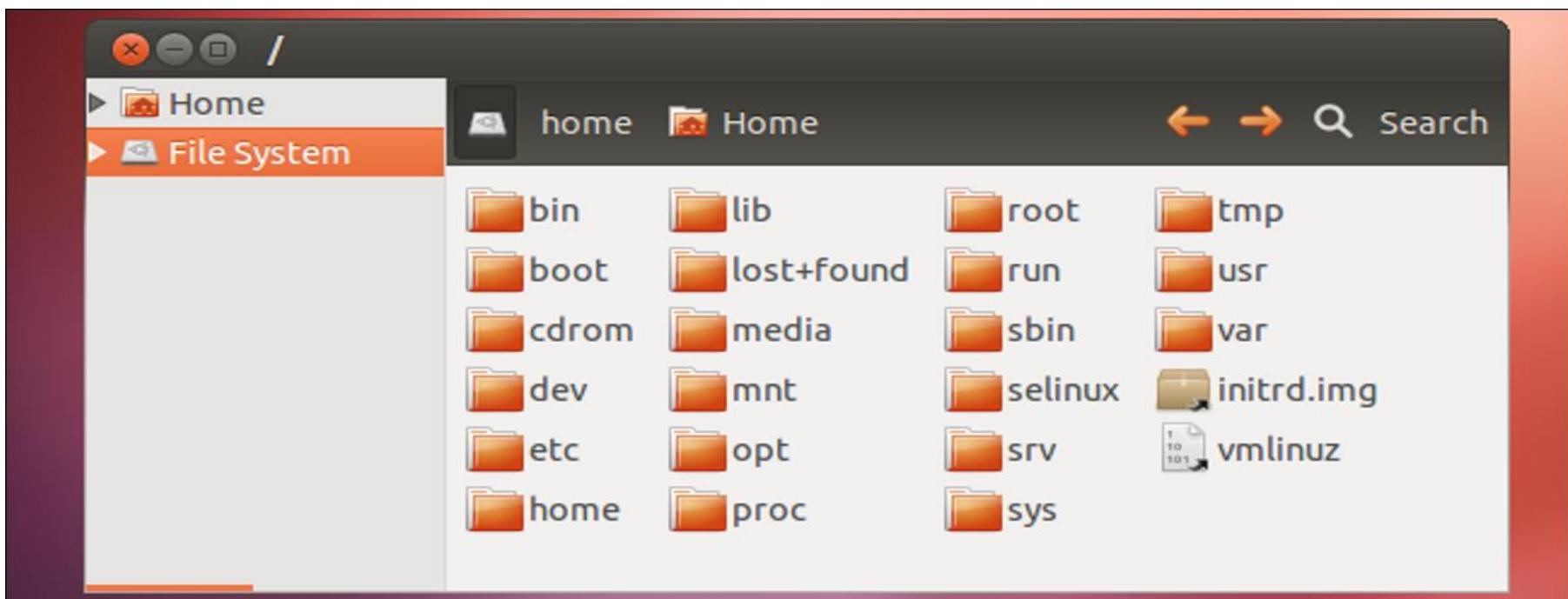
```
/bin/bash  
kjsce_dbs116@kjscedbs116:~$ echo $SHELL  
/bin/bash
```

List all the shells on your PC?

- Using the Cat command

List all the shells on your PC?

- The list of all the shells which are currently installed in our Linux system is stored in the **'shells' file which is present in /etc folder of the system.**
- It has read-only access by default and is modified automatically whenever we install a new shell in our system.



/etc – Configuration Files

- The /etc directory contains configuration files, which can generally be edited by hand in a text editor.
- Note that the /etc/ directory contains system-wide configuration files
- User-specific configuration files are located in each user's home directory.

List all the shells on your PC?

- The cat command displays the various installed shells along with their installation paths.

```
kjsce_dbs116@kjscedbs116:~$ cat /etc/shells
# /etc/shells: valid login shells
/bin/sh
/bin/bash
...
/bin/rbash
/bin/dash
kjsce_dbs116@kjscedbs116:~$ chsh
```

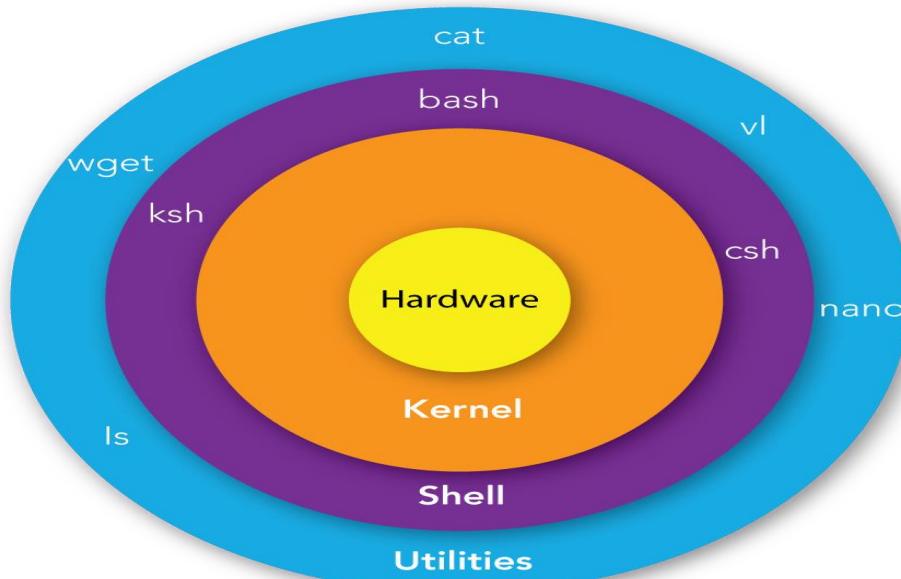
Change the shell

- **Using chsh Utility**
- chsh is the utility to change a user's login shell. chsh provides the -s option to change the user's shell.

```
sh-5.1$ grep nishant /etc/passwd
nishant:x:1000:1000::/home/nishant:/bin/sh
sh-5.1$ chsh -s /bin/bash nishant
Changing shell for nishant.
Password:
Shell changed.
sh-5.1$ grep nishant /etc/passwd
nishant:x:1000:1000::/home/nishant:/bin/bash
sh-5.1$
```

Shells

- There are **several shells** are available for Linux systems.
- Each shell does the **same job** but **understand different commands and provide different built in functions.**



Shell Types

- In UNIX there are two major types of shells:
 - The Bourne shell.
 - **If you are using a Bourne-type shell, the default prompt is the \$ character.**
 - The C shell.
 - **If you are using a C-type shell, the default prompt is the % character.**

<https://www.tutorialspoint.com/unix/unix-what-is-shell.htm>

The Bourne Shell

- The original UNIX shell written in the **mid-1970s by Stephen R. Bourne while he was at AT&T Bell Labs in New Jersey.**
- **Original UNIX shell-The Bourne shell was the first shell to appear on UNIX systems, thus it is referred to as "the shell".**
- Denoted as **sh**
- It is **faster** and more preferred.

<https://www.tutorialspoint.com/unix/unix-what-is-shell.htm>

The Bourne Shell

- It lacks features for interactive use like the ability to recall previous commands.
- It also lacks built-in arithmetic and logical expression handling.
- It is default shell for Solaris OS.

CSH (C SHeLL)

- The C Shell
- Denoted as csh
- Bill Joy created it at the University of California at Berkeley.
- It incorporated features such as aliases and command history.
- It includes helpful programming features like built-in arithmetic
- C-like expression syntax-shell syntax and usage are very similar to the C programming language.

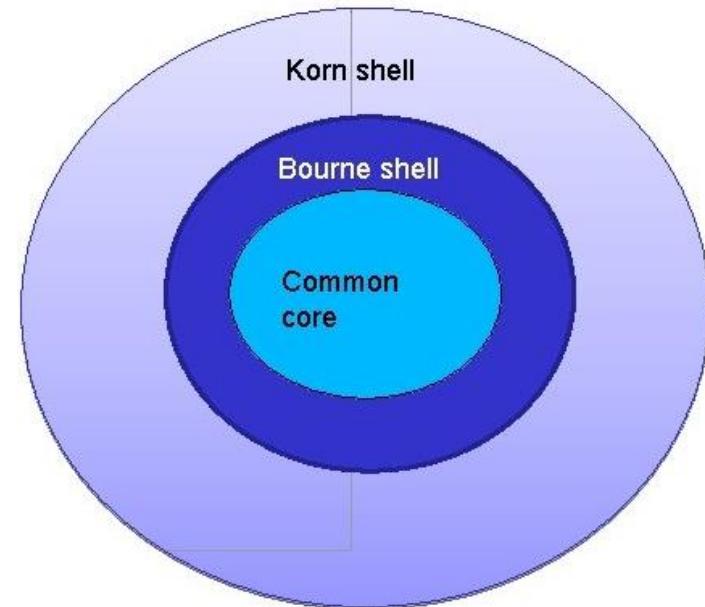
Shell Types

- There are again various subcategories for Bourne Shell which are listed as follows –
 - **Bourne shell (sh)**
 - **Korn shell (ksh)**
 - **Bourne Again shell (bash)**
 - **POSIX shell**
- The different C-type shells follow –
 - **C shell (csh)**
 - **TENEX/TOPS C shell (tcsh)**

<https://www.tutorialspoint.com/unix/unix-what-is-shell.htm>

The Korn Shell

- It is denoted as **ksh**
- It Was written by David Korn at AT&T Bell Labs.
- It is a superset of the Bourne shell. So it supports everything in the Bourne shell.



The Korn Shell

- It has interactive features.
- It includes features like built-in arithmetic and C-like arrays, functions, and string-manipulation facilities.
- It is faster than C shell. It is compatible with script written for C shell.
- The Korn Shell also was the base for the POSIX Shell standard specifications etc.

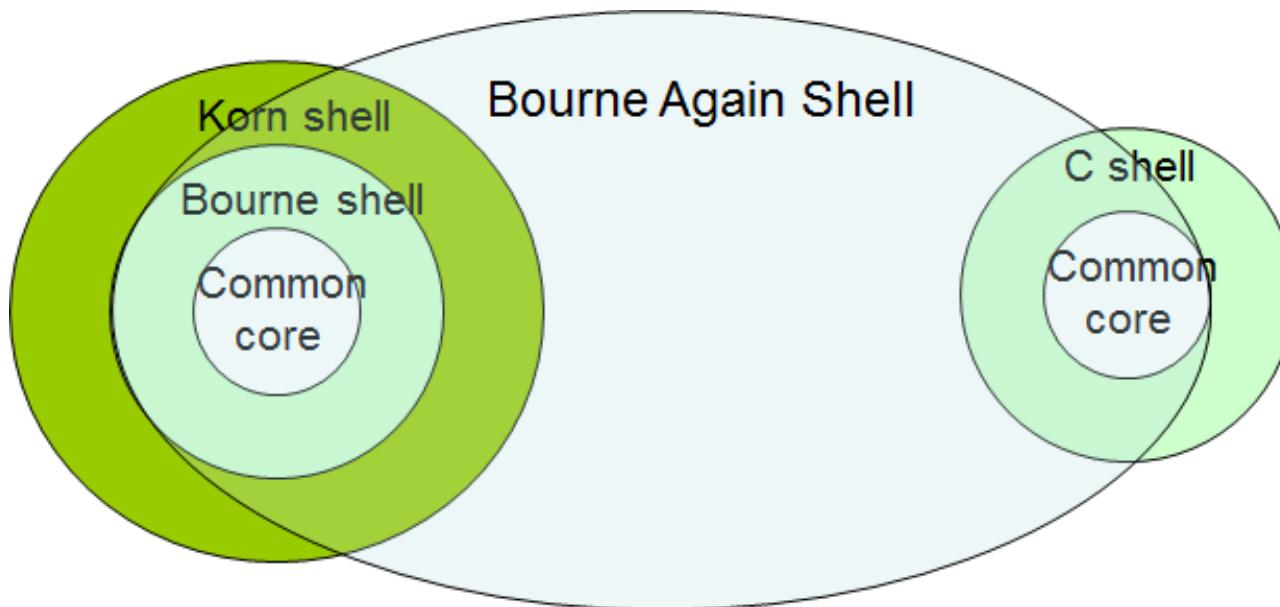
BASH

- BASH= Bourne Again Shell
- The shell's name is an acronym for Bourne Again Shell,
- A pun on the name of the Bourne shell that it replaces and
- The notion of being "born again".

[https://en.wikipedia.org/wiki/Bash_\(Unix_shell\)#:~:text=Bash%20is%20a%20Unix%20shell,ported%20to%20Linux%2C%20alongside%20GCC.](https://en.wikipedia.org/wiki/Bash_(Unix_shell)#:~:text=Bash%20is%20a%20Unix%20shell,ported%20to%20Linux%2C%20alongside%20GCC)

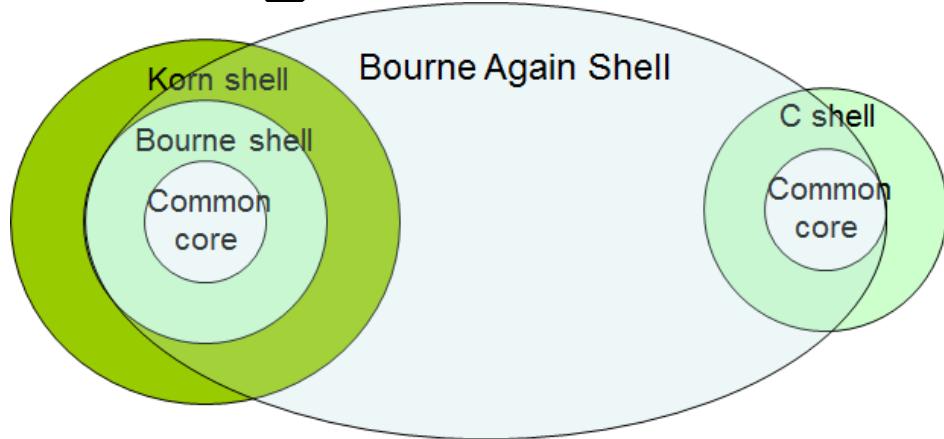
GNU Bourne-Again Shell

- The Bash command syntax is a superset of the Bourne shell command syntax.
- Bash command syntax includes ideas drawn from the KornShell (ksh) and the C shell (csh) such as command line editing, command history (history command), the directory stack, and POSIX command substitution syntax



GNU Bourne-Again Shell

- Denoted as **bash**
- It is compatible to the Bourne shell.
- It includes features from Korn and Bourne shell.
- It is most widely used shell in Linux systems.
- It is used as **default login shell in Linux systems and in macOS**. It can also be installed on Windows OS.



GNU ?

GNU ?

- **GNU is an operating system** and also an extensive collection of utility programmes wholly free software and also the project within which the free software concept originated
- **GNU is a recursive acronym for "GNU's Not Unix!"**
- **Chosen because GNU's design is Unix-like, but differs from Unix by being free software and containing no Unix code.**

Shell Scripting

- Usually shells are interactive that mean,
 - they accept command as input from users and execute them.
 - to execute a bunch of commands routinely,
 - **so we have to type in all commands each time in terminal.**

Shell Scripting

- As shell can also take commands
 - **as input from file we can write these commands in a file**
 - can execute them in shell to avoid this repetitive work.
 - These files are called **Shell Scripts or Shell Programs.**
 - Shell scripts are **similar to the batch file in MS-DOS.**
 - Each shell script is saved with **.sh file extension** eg.
myscript.sh

Shell Scripting

- A shell script have syntax just like any other programming language.
- A shell script comprises following elements –
 - **Shell Keywords** – if, else, break etc.
 - **Shell commands** – cd, ls, echo, pwd, touch etc.
 - **Functions**
 - **Control flow** – if..then..else, case and shell loops etc.

The Shell Definition Line

- The shell definition line tells the system
 - **what shell should be used** to interpret the script's commands, and
 - where the program (shell) is located.
- The shell definition line tells the system to use the **Korn Shell when executing this script.**
 - (#!/bin/ksh)

Shell Script Comments

- Comments, or non-command code,
- In a shell script, **Comments begin with the # (pound) character**

Permissions for script

- Before you can run the script, you need to make the shell script file, an executable by using the UNIX chmod command.

Permissions for script

- The following command will allow only the user (owner) of the script to execute it:
\$ chmod u+x script1
- If you wanted to allow everyone (all) to execute the script, you would use this command:
\$ chmod a+x script1

Executing script

- After the script file has been made an executable with the chmod command, you can run the script in a new shell by giving the path to the script:

```
$ ./script1
```

- This (./) would be the path to **script1 if you are in the same directory as the script.**
- If you were **in a different directory than the script**, you could use one of the following commands to run it:

```
$ /home/student1/script1
```

or

```
$ /bin/ksh /home/student1/script1
```

Executing script

- On some online portals, Execute it as

```
$ sh script1.sh
```

Sample program

- vi hello.sh
- To go into edit mode:
 - **press ESC and type I**
- To save a file
 - **press ESC and type :w fileName**
- To save a file and quit:
 - **press ESC and type :wq**
 - Or
 - **press ESC and type :x**
- Save and close the file.
- You can run the script as follows:
./hello.sh
- Sample outputs:
bash: ./hello.sh: Permission denied
- Type **chmod u+x hello.sh**
- And execute the program again

Chmod command

Chmod command

The '*chmod command*' is used to change the access permissions of a file. The syntax is as follows:

chmod ugo+rwx filename Where,

- u : Users
- g : Groups
- o : Others
- + : Adds the permission
- - : Removes the permission
- = : Overwrites current permissions
- r : Read permission
- w : Write permission
- x : Execute permission

Displaying Output

- Two methods for displaying output to standard output:

echo "Text Line 1"

- The print command is the replacement for the echo command.
 - You can use the print command when you are writing shell scripts because
 - it is more powerful than the echo command, and
 - its syntax has been standardized on multiple operating systems.

Echo command

- **echo** - display a line of text.

Example-2:

To print value of x, where x=10.

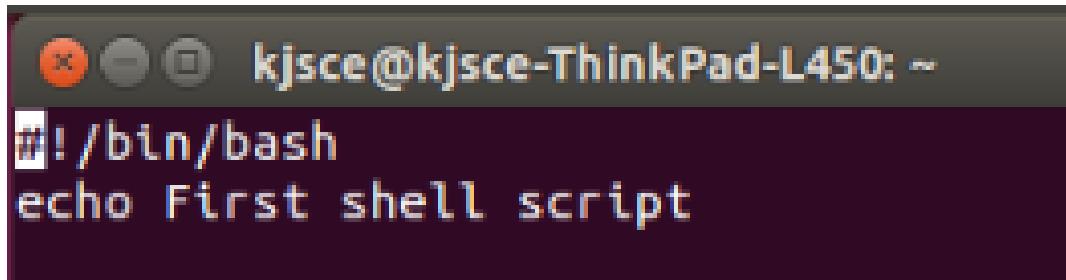
\$ echo \$x

output:

10

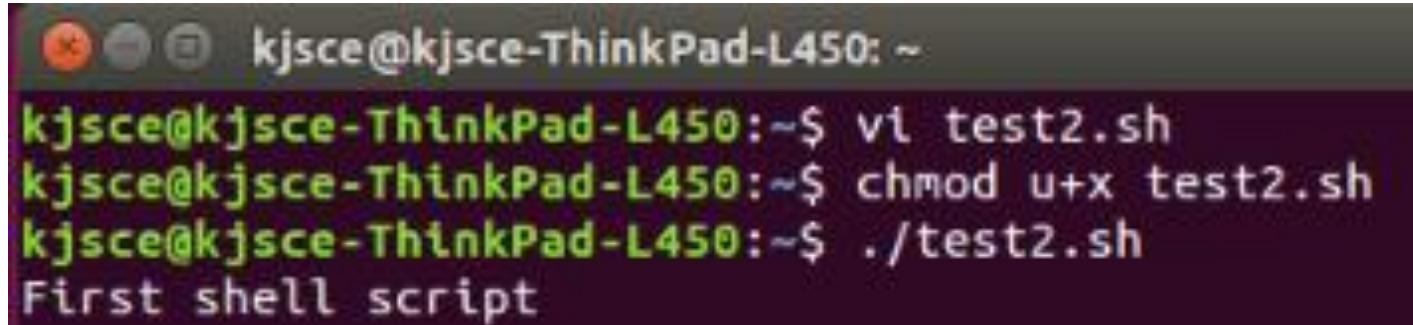
Echo command

- To display a line of text
- test2.sh-



```
#!/bin/bash
echo First shell script
```

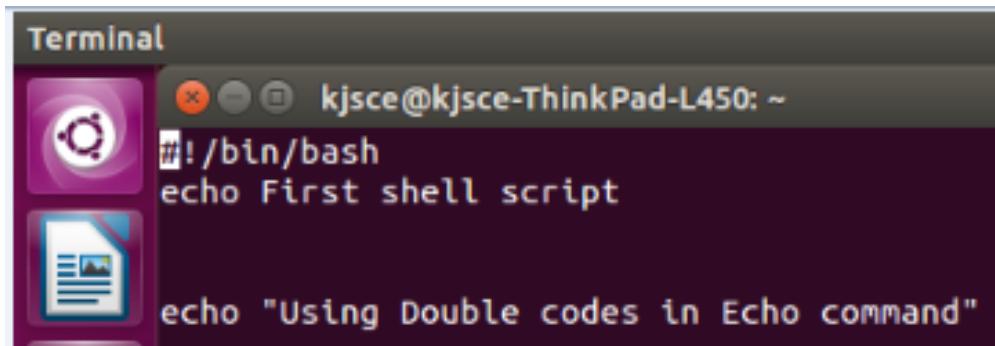
- Output-



```
kjsce@kjsce-ThinkPad-L450:~$ vi test2.sh
kjsce@kjsce-ThinkPad-L450:~$ chmod u+x test2.sh
kjsce@kjsce-ThinkPad-L450:~$ ./test2.sh
First shell script
```

Echo command

- **Using double quotes with echo**
- test2.sh-



```
Terminal
kjsce@kjsce-ThinkPad-L450: ~
#!/bin/bash
echo First shell script

echo "Using Double codes in Echo command"
```

- Output-



```
kjsce@kjsce-ThinkPad-L450:~/> ./test2.sh
First shell script
Using Double codes in Echo command
```

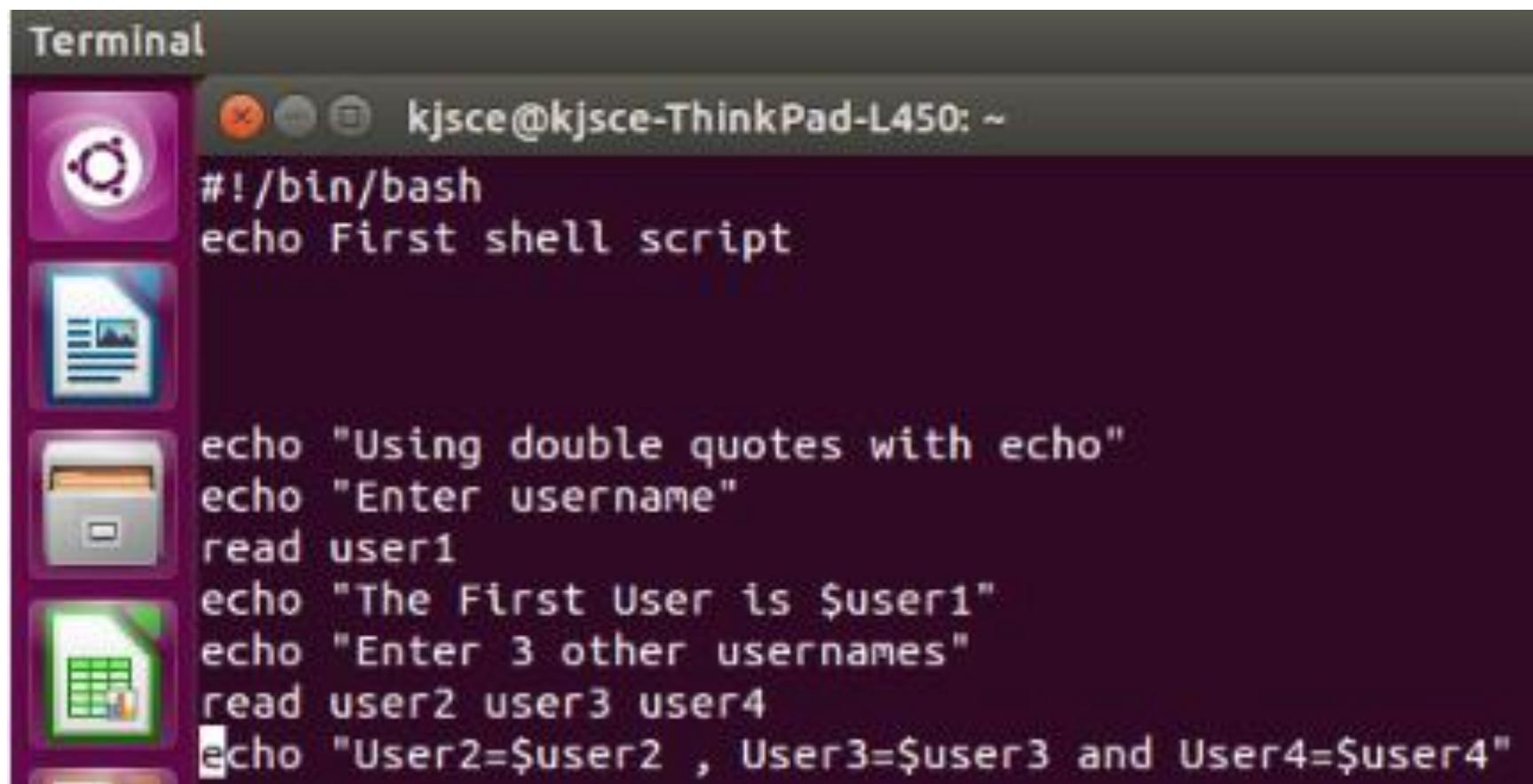
Echo command

- **Using double quotes with echo**
- That the use of double quotes ("") characters affect how spaces and TAB characters are treated, for example:

```
$ echo Hello      World  
Hello World  
  
$ echo "Hello      World"  
Hello      World
```

Read command

- To read input from the user
- test2.sh-



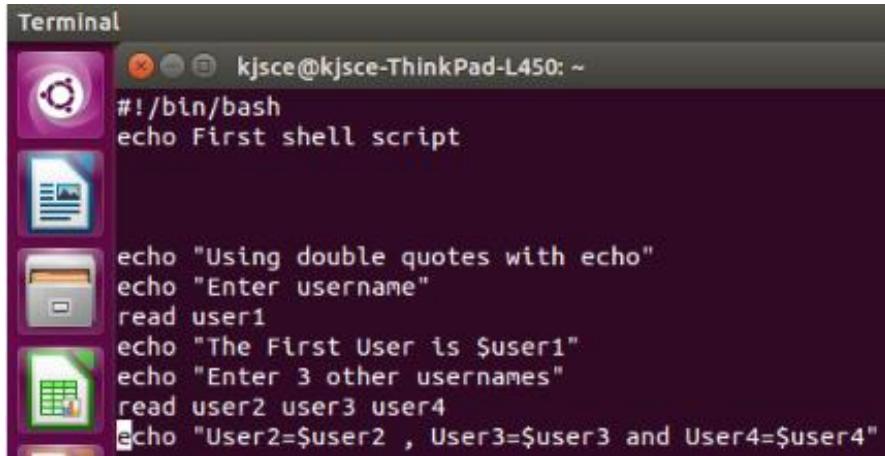
The image shows a terminal window on an Ubuntu desktop environment. The title bar says "Terminal". The window contains a shell script named "test2.sh" with the following content:

```
#!/bin/bash
echo First shell script

echo "Using double quotes with echo"
echo "Enter username"
read user1
echo "The First User is $user1"
echo "Enter 3 other usernames"
read user2 user3 user4
echo "User2=$user2 , User3=$user3 and User4=$user4"
```

Read command

- To read input from the user
- test2.sh-



```
Terminal kjsce@kjsce-ThinkPad-L450: ~
#!/bin/bash
echo First shell script

echo "Using double quotes with echo"
echo "Enter username"
read user1
echo "The First User is $user1"
echo "Enter 3 other usernames"
read user2 user3 user4
echo "User2=$user2 , User3=$user3 and User4=$user4"
```

- Output-
- ```
kjsce@kjsce-ThinkPad-L450:~$./test2.sh
First shell script
Using double quotes with echo
Enter username
SDC
The First User is SDC
Enter 3 other usernames
AAG JMR RAN
User2=AAG , User3=JMR and User4=RAN
```

```
kjsce@kjsce-ThinkPad-L450:~$ █
```

# How to define variables

- We can define a variable by using the syntax `variable_name=value`.
- To get the value of the variable, add \$ before the variable.

# Arithmetic expression

- Arithmetic expansion allows the evaluation of an arithmetic expression and the substitution of the result.
- The format for arithmetic expansion is:

**$\$(\text{expression})$**

- Reading man bash says that the old format:

**$\$[\text{expression}]$**

- is deprecated and will be removed.
- Otherwise they should be equivalent.

# Shell Script to Add Two Integers

```
#!/bin/bash
Calculate the sum of two integers with pre initialize values
in a shell script
a=10
b=20
sum=$(($a + $b))
echo $sum
```

# Arithmetic Operators

| Operator           | Description                                                           | Example                                |
|--------------------|-----------------------------------------------------------------------|----------------------------------------|
| + (Addition)       | Adds values on either side of the operator                            | `expr \$a + \$b` will give 30          |
| - (Subtraction)    | Subtracts right hand operand from left hand operand                   | `expr \$a - \$b` will give -10         |
| * (Multiplication) | Multiplies values on either side of the operator                      | `expr \$a \* \$b` will give 200        |
| / (Division)       | Divides left hand operand by right hand operand                       | `expr \$b / \$a` will give 2           |
| % (Modulus)        | Divides left hand operand by right hand operand and returns remainder | `expr \$b % \$a` will give 0           |
| = (Assignment)     | Assigns right operand in left operand                                 | a = \$b would assign value of b into a |
| == (Equality)      | Compares two numbers, if both are same then returns true.             | [ \$a == \$b ] would return false.     |
| != (Not Equality)  | Compares two numbers, if both are different then returns true.        | [ \$a != \$b ] would return true.      |

# Relational Operators

| Operator   | Description                                                                                                                          | Example                      |
|------------|--------------------------------------------------------------------------------------------------------------------------------------|------------------------------|
| <b>-eq</b> | Checks if the value of two operands are equal or not; if yes, then the condition becomes true.                                       | [ \$a -eq \$b ] is not true. |
| <b>-ne</b> | Checks if the value of two operands are equal or not; if values are not equal, then the condition becomes true.                      | [ \$a -ne \$b ] is true.     |
| <b>-gt</b> | Checks if the value of left operand is greater than the value of right operand; if yes, then the condition becomes true.             | [ \$a -gt \$b ] is not true. |
| <b>-lt</b> | Checks if the value of left operand is less than the value of right operand; if yes, then the condition becomes true.                | [ \$a -lt \$b ] is true.     |
| <b>-ge</b> | Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then the condition becomes true. | [ \$a -ge \$b ] is not true. |
| <b>-le</b> | Checks if the value of left operand is less than or equal to the value of right operand; if yes, then the condition becomes true.    | [ \$a -le \$b ] is true.     |

# Shell Script –Using Decision constructs

## Syntax

if [ expression ]

then

    Statement(s) to be executed if expression is true

else

    Statement(s) to be executed if expression is not true

fi

# Conditional Expressions

- It is very important to understand that all the conditional expressions should be inside square braces with spaces around them,
- For example
- [ \$a == \$b ] is correct whereas,
- [\$a==\$b] is incorrect.

# Conditional Expressions

- It is very important to understand that all the conditional expressions should be placed inside square braces with spaces around them.
- For example,
- [ \$a <= \$b ] is correct whereas,
- [\$a <= \$b] is incorrect.

# Conditional Expressions

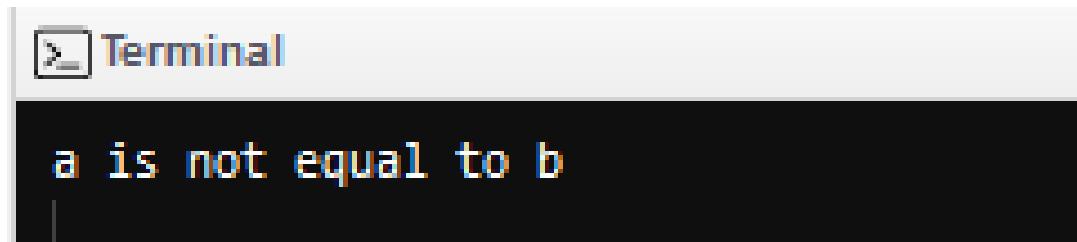
- -lt is Less than which is used for condition checking
- < is used for Reading input from the files.
- The statement [ \$num < 10 ] is just syntactic sugar for the command test \$num < 10, which means:
  - Run the command test with one parameter (the content of the shell variable num), and
  - associate the standard input of this invocation with a file named 10

# Shell Script –Using Decision constructs

Example-

```
1 #!/bin/sh
2
3 a=10
4 b=20
5
6 if [$a == $b]
7 then
8 echo "a is equal to b"
9 else
10 echo "a is not equal to b"
11 fi
```

Output-

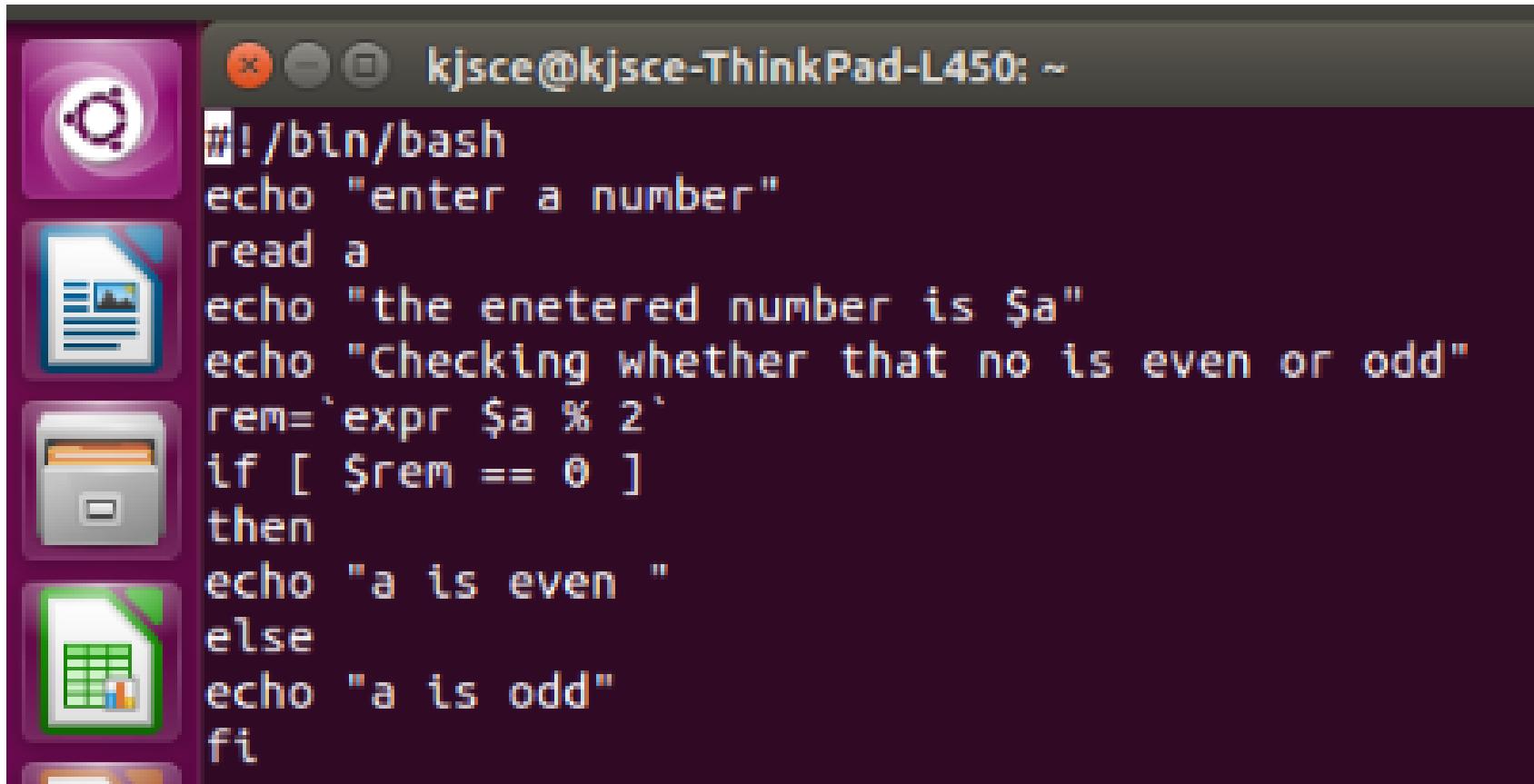


A screenshot of a terminal window titled "Terminal". The window displays the output of a shell script. The script compares two variables, \$a and \$b, which are both set to 10. The "else" block is executed, resulting in the output "a is not equal to b".

```
a is not equal to b
```

## Shell Script to check if a number is even or odd

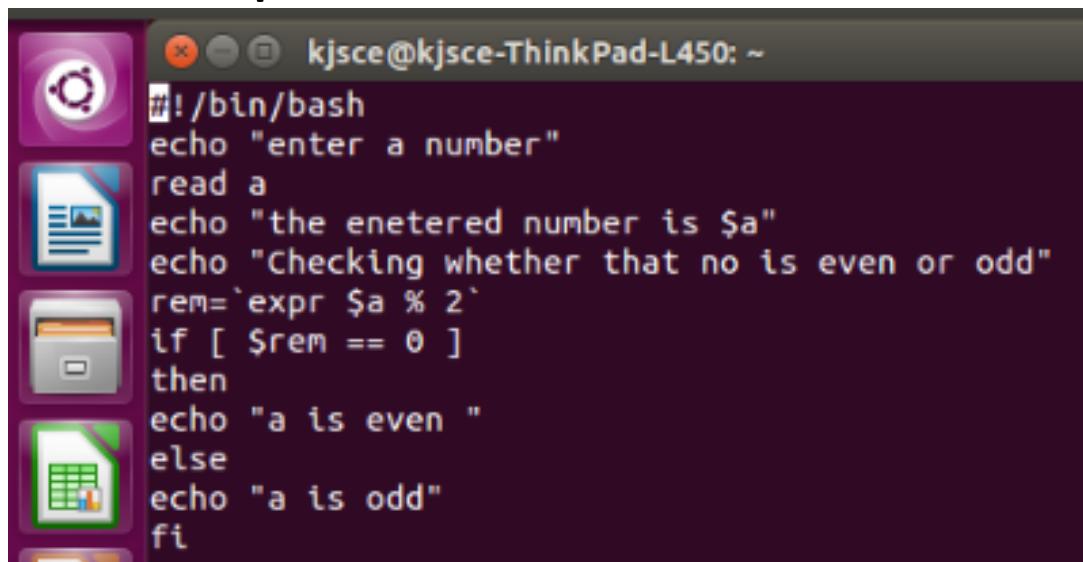
- *expr* is an external program used by Bourne shell.
- It uses *expr* external program with the help of backtick.
- The *backtick(`)* is actually called command substitution.



The image shows a screenshot of a Linux desktop environment, specifically Ubuntu, with a terminal window open. The terminal window has a dark background and contains the following shell script:

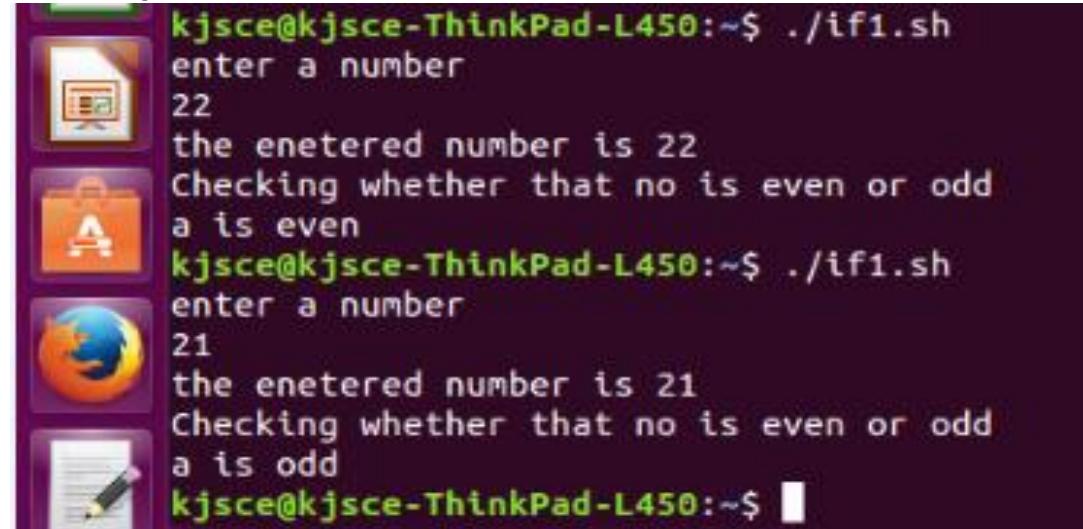
```
#!/bin/bash
echo "enter a number"
read a
echo "the enetered number is $a"
echo "Checking whether that no is even or odd"
rem=`expr $a % 2`
if [$rem == 0]
then
echo "a is even "
else
echo "a is odd"
fi
```

- **Shell Script to check if a number is even or odd**



```
#!/bin/bash
echo "enter a number"
read a
echo "the enetered number is $a"
echo "Checking whether that no is even or odd"
rem=`expr $a % 2`
if [$rem == 0]
then
echo "a is even "
else
echo "a is odd"
fi
```

- **Output-**



```
kjsce@kjsce-ThinkPad-L450:~$./if1.sh
enter a number
22
the enetered number is 22
Checking whether that no is even or odd
a is even
kjsce@kjsce-ThinkPad-L450:~$./if1.sh
enter a number
21
the enetered number is 21
Checking whether that no is even or odd
a is odd
kjsce@kjsce-ThinkPad-L450:~$
```

- *expr* is an external program used by Bourne shell.
- It uses *expr* external program with the help of backtick.
- The *backtick(`)* is actually called command substitution.

- Shell Script to check if a number is even or odd



```
#!/bin/bash
echo "enter a number"
read a
echo "the enetered number is $a"
echo "Checking whether that no is even or odd"
rem=`expr $a % 2`
if [$rem == 0]
then
echo "a is even "
else
echo "a is odd"
fi

1 clear
2 echo "-----EVEN OR ODD IN SHELL SCRIPT-----"
3 echo -n "Enter a number: "
4 read t
5 echo -n "RESULT: "
6 if [`expr $t % 2` == 0]
7 then
8 echo "$t is even."
9 else
10 echo "$t is odd."
11 fi
```

- **Shell Script to calculate hra and ta**

- Shell Script to calculate hra and ta

Desktop

```
ksce@kjsce-ThinkPad-L450: ~
#!/bin/sh
echo "enter basic salary"
read basic
echo "if basic is less than 40,000 then hra=10% and ta=25% else hra=15% and ta=35%"
if [$basic -lt 40000]
then hra=`expr 10 * $basic / 100`
ta=`expr 25 * $basic / 100`
echo "hra is $hra"
echo "ta is $ta"
else
hra=`expr 15 * $basic / 100`
ta=`expr 35 * $basic / 100`
echo "hra is $hra"
echo "ta is $ta"
fi
```

- *expr* is an external program used by Bourne shell.
- It uses *expr* external program with the help of backtick.
- The *backtick(`)* is actually called command substitution.

- Shell Script to calculate hra and ta

```
Desktop
x kjsce@kjsce-ThinkPad-L450: ~
#!/bin/sh
echo "enter basic salary"
read basic
echo "if basic is less than 40,000 then hra=10% and ta=25% else hra=15% and ta= 35%"
if [$basic -lt 40000]
then hra=`expr 10 * $basic / 100`
ta=`expr 25 * $basic / 100`
echo "hra is $hra"
echo "ta is $ta"
else
hra=`expr 15 * $basic / 100`
ta=`expr 35 * $basic / 100`
echo "hra is $hra"
echo "ta is $ta"
fi
```

- Output-

```
kjsce@kjsce-ThinkPad-L450:~$./ifelse.sh
enter basic salary
50000
if basic is less than 40,000 then hra=10% and ta=25% else hra=15% and ta= 35%
hra is 7500
ta is 17500
kjsce@kjsce-ThinkPad-L450:~$./ifelse.sh
enter basic salary
35000
if basic is less than 40,000 then hra=10% and ta=25% else hra=15% and ta= 35%
hra is 3500
ta is 8750
```

- The backslash (\) character is used to mark these special characters so that they are not interpreted by the shell, but passed on to the command being run (for example, echo).

```
$ echo "A quote is \", backslash is \\, backtick is `."
A quote is ", backslash is \, backtick is `.
```

# For Loop Syntax

```
sssit@JavaTpoint: ~
#!/bin/bash

for ((cond1; cond2; cond3))
do
 echo "statement"
done
```

condition1 indicates **initialization**,  
cond2 indicates **condition** and cond3 indicates **updation**.

# For Loop Eg

```
sssit@JavaTpoint: ~
```

```
#!/bin/bash
```

```
for ((i=10; i >= 1; i--))
do
 echo "$i"
done
```

O/P-

```
sssit@JavaTpoint: ~
```

```
sssit@JavaTpoint:~$./random.sh
```

```
10
9
8
7
6
5
4
3
2
1
```

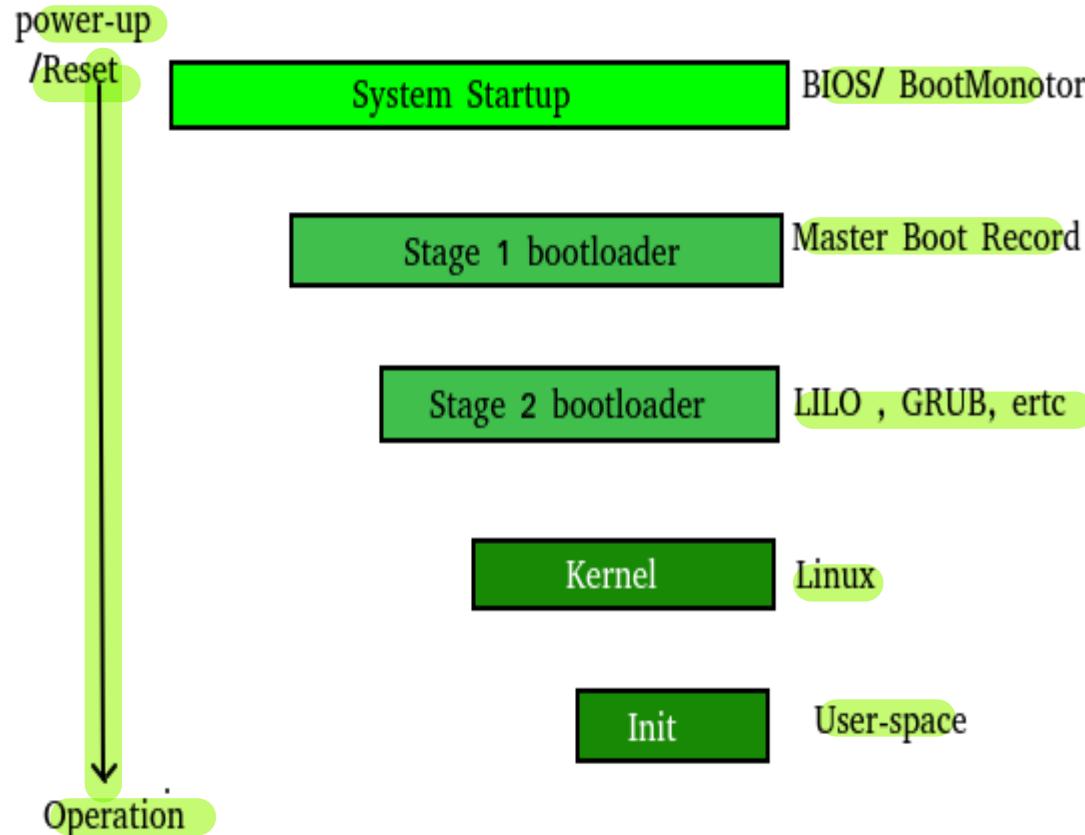
```
sssit@JavaTpoint:~$
```

# System Boot

# System Boot

- The boot process is something that
  - happens every time
  - you turn your computer on.
  - You don't really see it,
  - because it happens so fast.
- You press the power button,
  - come back a few minutes later and
  - Windows XP, or Windows 10, or whatever Operating System you use
  - is all loaded.

# System Boot



# System Boot- The working

- The CPU initializes itself
  - after the power in the computer is first turned on.
  - This is done by triggering a series of clock ticks
  - that are generated by the system clock.

# System Boot- The working

- After this, the CPU looks for the system's ROM BIOS
  - This first instruction is stored in the ROM BIOS and
  - it instructs the system to
    - run POST (Power On Self Test)
    - in a memory address that is predetermined.
- Since ROM is read only,
  - it cannot be infected by a computer virus.

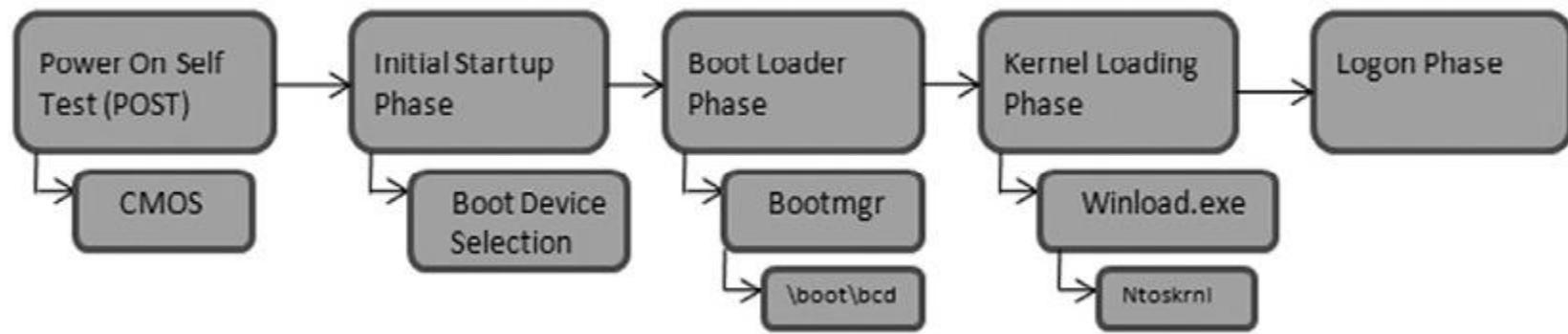
# System Boot- The working

- POST first checks
  - the BIOS chip and
  - then the CMOS RAM.
  - If there is no battery failure is detected by POST,
  - then it continues to initialize the CPU.
- POST also checks
  - the hardware devices,
  - secondary storage devices such as hard drives, ports etc.
  - And other hardware devices such as the mouse and keyboard.
  - This is done to make sure they are working properly.

After POST makes sure that all the components are working properly, then the BIOS starts Boot Strap program.

# Post method

- To perform this check, the POST
  - Sends out a standard command that says to all the devices, "Check yourselves out!"
  - All the standard devices in the computer then run their own internal diagnostic
  - The POST doesn't specify what they must check.
  - The quality of the diagnostic is up to the people who made that particular device.



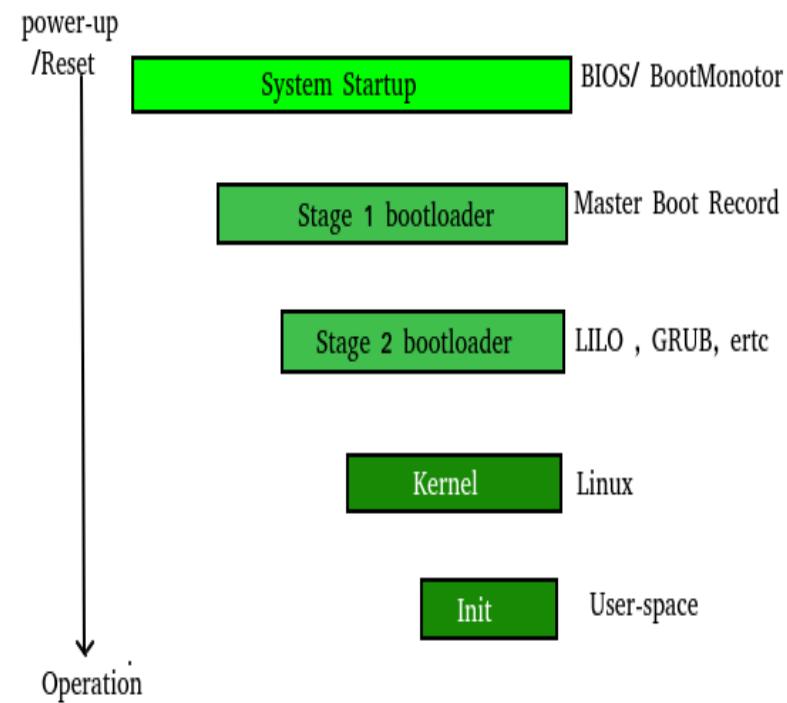
Courtesy : <http://www.c-jump.com/CIS24/Slides/Booting/Booting.html>

# System Boot- The working

- For most computers, Boot Strap is stored in BIOS ROM.
  - Changing the Bootstrap code requires changing the ROM hardware chips.

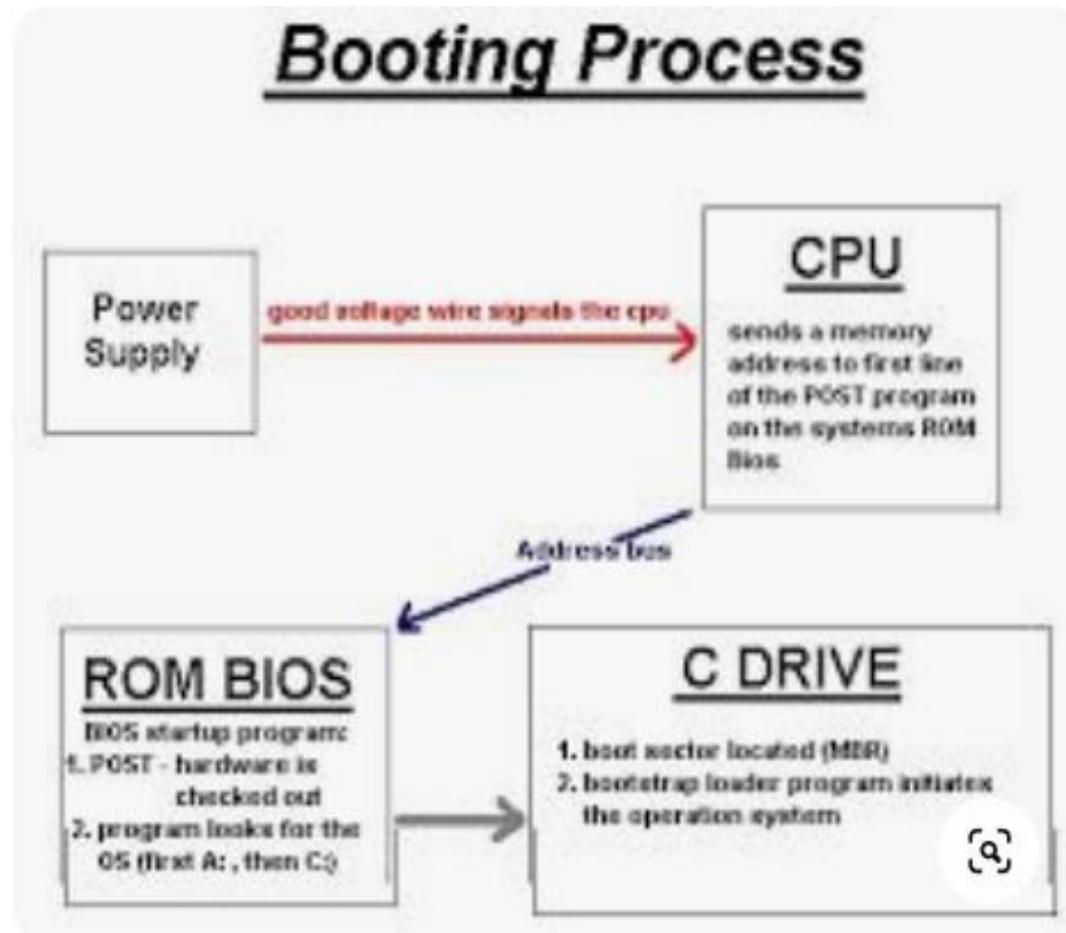
Alternative:

- A tiny Bootstrap loader program is stored in the ROM,
  - whose only job is to bring in a full bootstrap program from the disk.
  - The full bootstrap pgm can be changed easily.



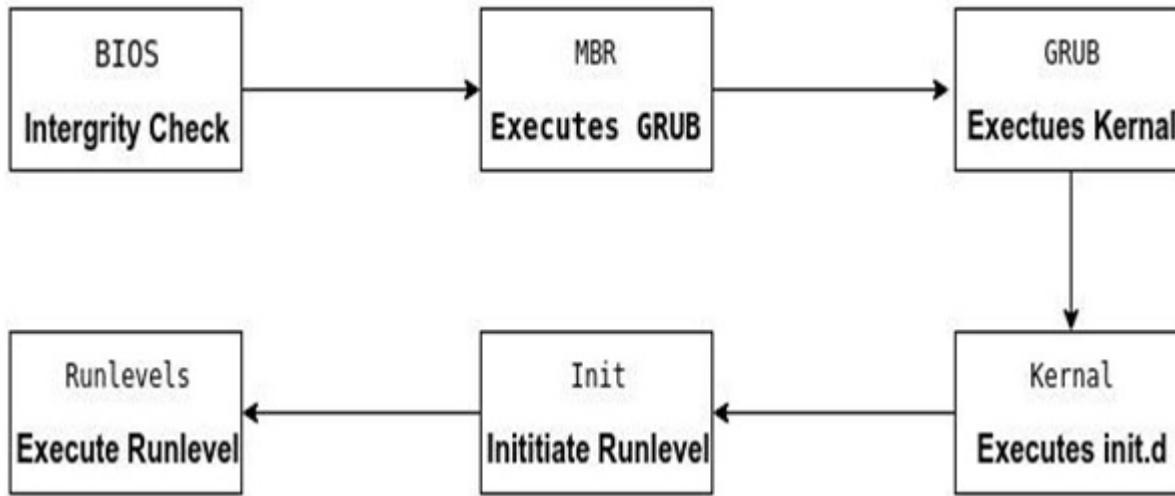
# System Boot- The working

- The Bootstrap pgm
  - is stored in a partition called the boot blocks ,
  - at a fixed location on the disk.
- A disk that has a boot partition
  - is called a Boot Disk or System Disk.



# System Boot- The working

- The Bootstrap program is loaded into memory and
  - starts its execution.
- The bootstrap program
  - finds the OS kernel on disk,
  - loads that Kernel into memory and
  - jumps to an initial address
  - to begin the OS execution.



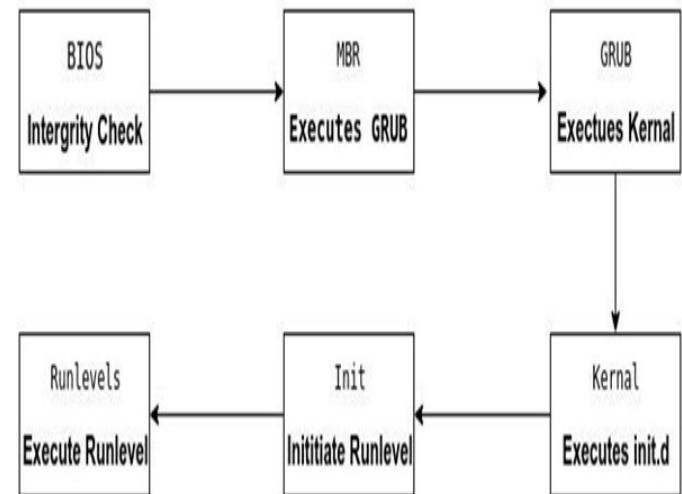
## Master Boot Record

The Master Boot Record (MBR) is the information which is in the first sector of any hard disk.

It holds information about GRUB (or LILO in old systems).

## init

- This is the last step of the **booting process**.
- It decides the run level by looking at the / etc / inittab file.
- The initial state of the operating system is decided by the run level.



# init

Following are the run levels of Operating System:

Level

0 - System Halt

1 - Single user mode

2 - Multiuser, without NFS(Network File System)

3 - full multiuser mode

4 - unused

5 - Full multiuser mode with network and X display manager(X11)

6 - Reboot

We would set the default run level to either 3 or 5.

- The step after it is to
  - 1) Start up various daemons that support networking and other services.
  - 2) X server daemon manages display, keyboard, and mouse.
  - 3) You can see a Graphical Interface and
  - 4) A login screen is displayed during X server daemon is started.

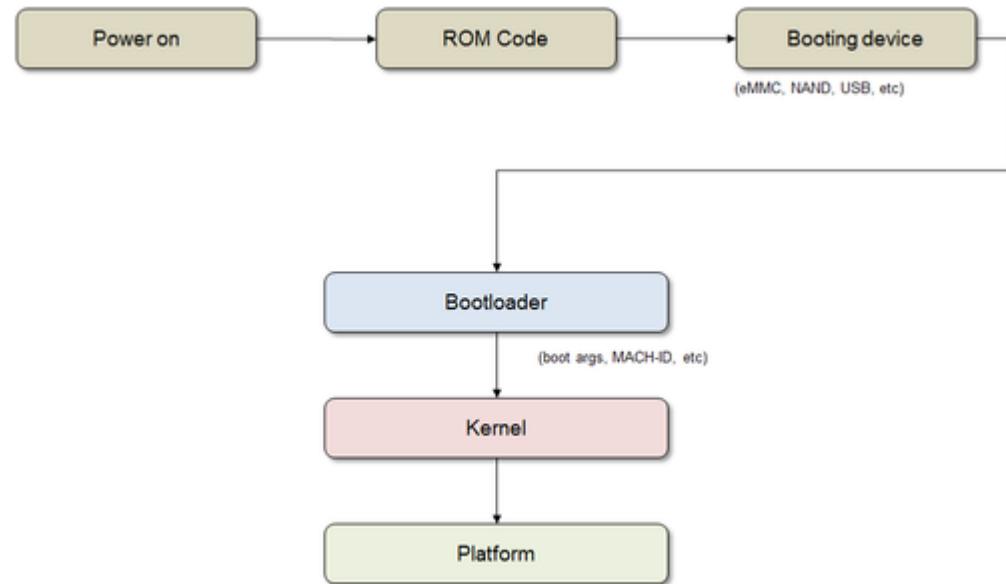
- Daemons are **processes that run unattended**.
- They are constantly in the background and are available at all times.
- Daemons are usually started when the system starts, and they run until the system stops.

# System Boot

- When power is initialized on system,
  - execution starts at a fixed memory location,
  - **Firmware ROM used to hold initial boot code**
- Operating system must be
  - made available to hardware
  - so hardware can start it

# System Boot

- Sometimes two-step process where
  - **Boot block** at fixed location loaded by ROM code,
  - which loads bootstrap program from disk
  - Then loader continues with its job



# System Boot

- **Bootstrap loader**
  - Small piece of code,
  - stored in **ROM** or **EEPROM** (electrically erasable programmable read-only memory)
  - locates the kernel,
  - loads it into memory, and
  - starts it

# System Boot

- Common bootstrap loader,
  - **GRUB**, (the GRand Unified Bootloader)
  - allows selection of kernel from
    - multiple disks,
    - versions,
    - kernel options
- Kernel loads and
  - system is then **running**

# Failure during boot

- If the computer cannot boot, we will get a boot failure error.
  - This error indicates that the computer is not passing POST or
  - **a device in the computer, such as the hard drive or memory, has failed.**
  - **You may also hear a beep code to identify which hardware is failing during the POST.**
  - An error message or blue screen may show on the screen as operating system files cannot be loaded, due to not being found or being corrupt.

What is the name given to the process of initializing a microcomputer with its operating system?

- (a) Cold booting
- (b) Booting
- (c) Warm booting
- (d) Boot recording
- (e) None of the above

What is the name given to the process of initializing a microcomputer with its operating system?

- (a) Cold booting
- (b) Booting **(Ans)**
- (c) Warm booting
- (d) Boot recording
- (e) None of the above

- Cold Boot-
  - To perform a cold boot (also called a "hard boot") means to start up a computer that is turned off.
- Warm Boot-
  - It is often used in contrast to a cold boot, which refers to restarting a computer once it has been turned on.
- While a warm boot and cold boot are similar,
  - a cold boot performs a more complete reset of the system than a warm boot.

- Warm Boot-
  - If the computer hangs because of some reason while working, and demands to be restarted to make it working.
  - The process of reset/restart of the computer system is called as warm booting.
  - It is done with the help of reset button or keys (Ctrl+Alt+Del).

## Difference between Cold Booting and Warm Booting:

| S.NO. | COMPARISON                | COLD BOOTING                                  | WARM BOOTING                                             |
|-------|---------------------------|-----------------------------------------------|----------------------------------------------------------|
| 1.    | Initialized by            | Power button.                                 | Reset button or by pressing Ctrl+Alt+Del simultaneously. |
| 2.    | Performed                 | Frequent basis.                               | Not very common.                                         |
| 3.    | Alternate names           | Hard Booting, Cold start and dead start.      | Soft Booting.                                            |
| 4.    | POST (Power On Self Test) | Included.                                     | Not included.                                            |
| 5.    | Basic                     | Turning a computer ON from a powerless state. | Resetting a computer from already in running state.      |
| 6.    | Consequence               | Does not affect the data or other hardware.   | Can severely affect the system causing the data loss.    |

Consider the following statements :

### **UGC-NET | UGC-NET CS 2017 Nov – III | Question 73**

(a) UNIX provides three types of permissions

- \* Read
- \* Write
- \* Execute

(b) UNIX provides three sets of permissions

- \* permission for owner
- \* permission for group
- \* permission for others

Which of the above statement/s is/are true ?

- (A)** only (a)
- (B)** only (b)
- (C)** Both (a) and (b)
- (D)** Neither (a) nor (b)

Consider the following statements :

### UGC-NET | UGC-NET CS 2017 Nov – III | Question 73

(a) UNIX provides three types of permissions

- \* Read
- \* Write
- \* Execute

(b) UNIX provides three sets of permissions

- \* permission for owner
- \* permission for group
- \* permission for others

Which of the above statement/s is/are true ?

- (A) only (a)
- (B) only (b)
- (C) Both (a) and (b)
- (D) Neither (a) nor (b)

**Answer: (C)**

**Explanation:** UNIX provides Read, Write and Execute permission on files

UNIX provides three sets of permissions

permission for owner

permission for group

permission for others

## Question 56

In Unix, the command to enable execution permission for file “myfile” by all is

- 
- (A) Chmod ugo + X myfile
  - (B) Chmod a + X myfile
  - (C) Chmod + X myfile
  - (D) All of the above

## **UGC-NET | UGC NET CS 2015 Dec – III | Question 56**

In Unix, the command to enable execution permission for file “mylife” by all is \_\_\_\_\_.

- (A) Chmod ugo + X myfile
- (B) Chmod a + X myfile
- (C) Chmod + X myfile
- (D) All of the above

**Answer: (D)**

**Explanation:** In Unix, the command to enable execution permission for file “mylife” by all are:

Chmod + X myfile  
Chmod a + X myfile  
Chmod ugo + X myfile  
So, option (D) is correct.

# System Calls

# System Calls

- Provide the Interface between a process and the OS

# System Calls

## WHEN?

- System calls are usually made when a process in user mode requires access to a resource.
- Then it requests the kernel to provide the resource via a system call.

# System Calls

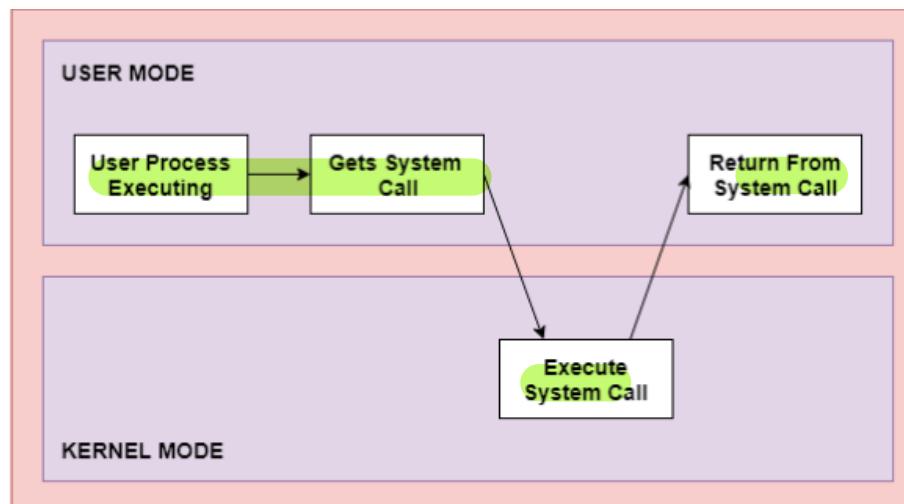
## WHEN?

System calls are required in the following situations –

- 1) If a file system requires the creation or deletion of files.
- 2) Reading and writing from files also require a system call.
- 3) Creation and management of new processes.
- 4) Network connections also require system calls. This includes sending and receiving packets.
- 5) Access to a hardware devices such as a printer, scanner etc. requires a system call.

# System Calls

- 1) The processes execute normally in the user mode until a system call interrupts this.
- 2) Then the system call is executed on a priority basis in the kernel mode.
- 3) After the execution of the system call, the control returns to the user mode
- 4) Execution of user processes can be resumed.



# System Calls

- **Generally available as Assembly language instructions.**
- **Certain systems allow system calls to be made directly from a higher level language program,**
  - In this case, they resemble predefined functions or subroutine calls

# System Calls

- Languages like C,C++, Perl have been defined to replace Assembly language for system programming.
  - They allow system calls to be made directly from programs.
- Unix system call may be invoked directly from C or C++ program.

# System Calls

- Mostly accessed by programs via a high-level Application Programming Interface (API) rather than direct system call use
- Three most common APIs are
  - Win32 API for Windows,
  - POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and
  - Java API for the Java virtual machine (JVM)

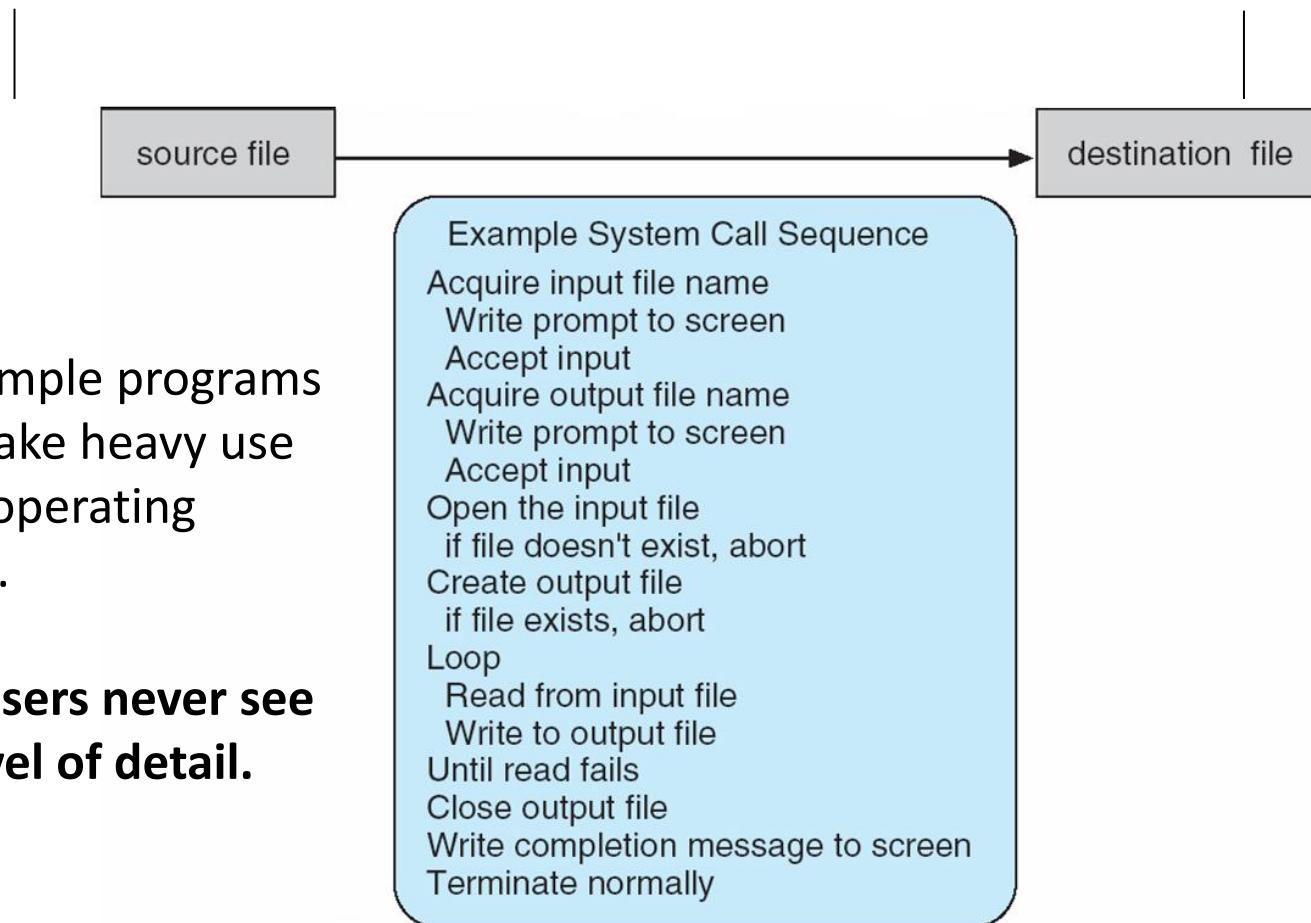
# Example of System Calls

- System call sequence to copy the contents of one file to another file



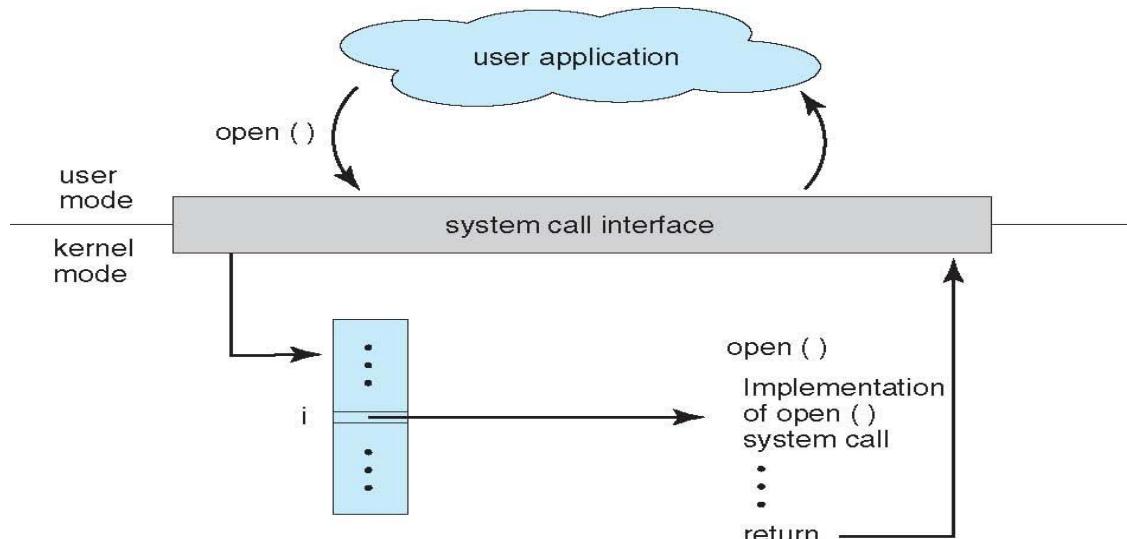
# Example of System Calls

- System call sequence to copy the contents of one file to another file



# System Call Implementation

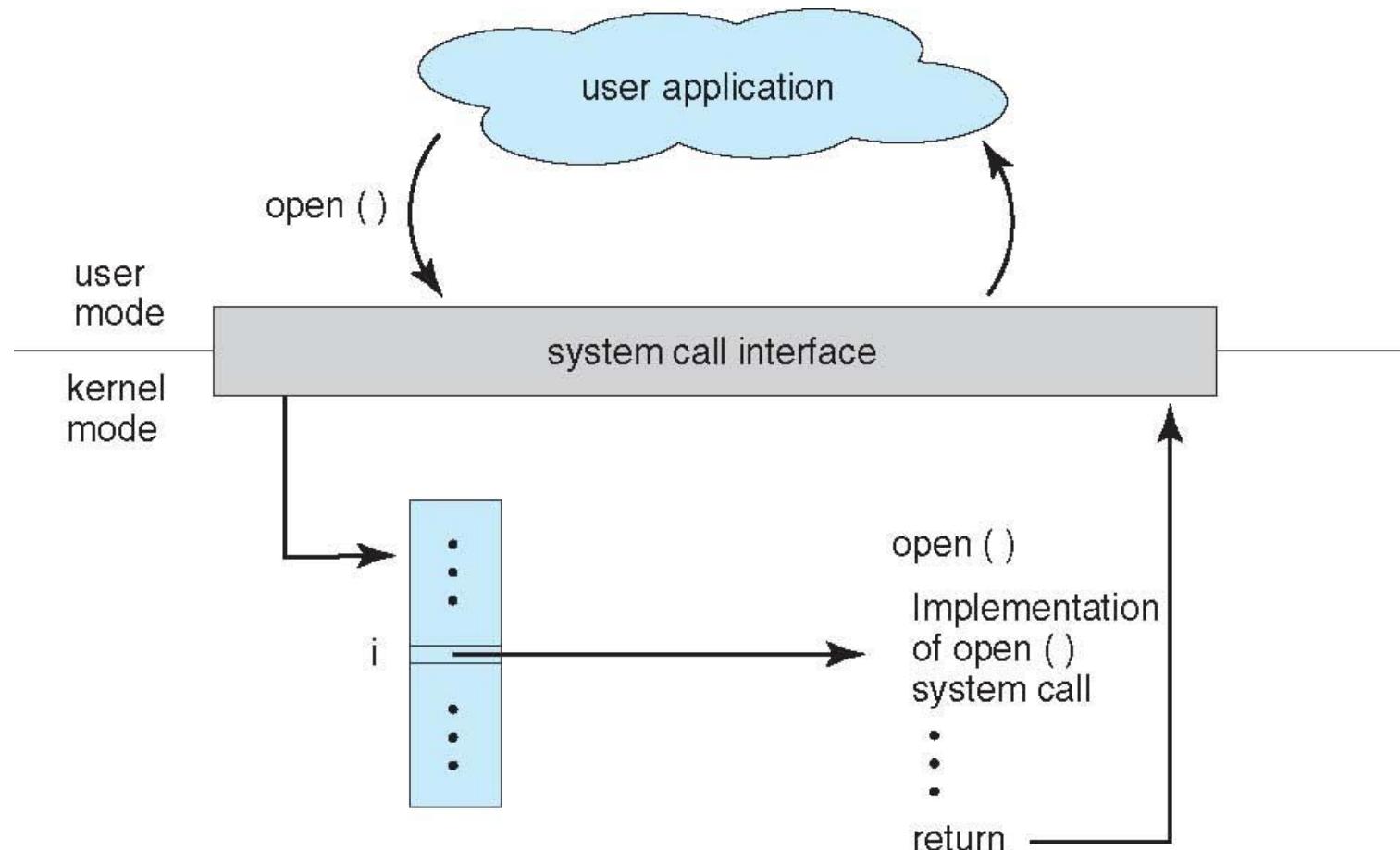
- Typically, a number is associated with each system call
  - System-call interface maintains a table indexed according to these numbers
- The system call interface
  - invokes the intended system call in OS kernel and
  - returns status of the system call and any return values



# System Call Implementation

- **The caller need know nothing about how the system call is implemented**
  - Just needs to obey API and **understand what OS will do as a result of call**
  - **Most details of OS interface hidden from programmer by API**
    - Managed by run-time support library (set of functions built into libraries included with compiler)

# API – System Call – OS Relationship



# System Call Parameter Passing

- System Calls occur in different ways
- Often, more information is required than simply identity of desired system call
  - **Exact type and amount of information vary according to OS and call**
  - Eg- To get input,
    - **we need to specify file or device to use as the source and**
    - the address and **length of the memory buffer** into which the input should be read.

# Example of Standard API

## ***EXAMPLE OF STANDARD API***

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the man page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count)
```

return value      function name      parameters

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

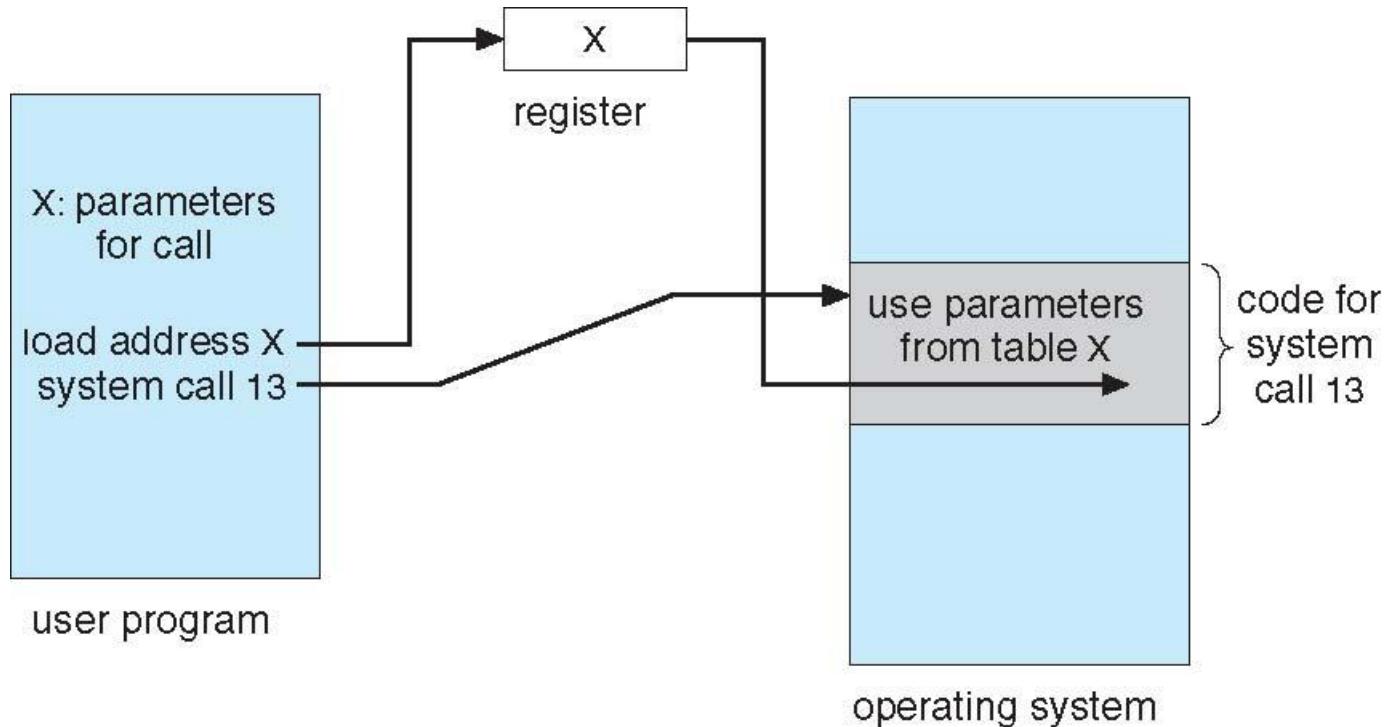
- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns -1.

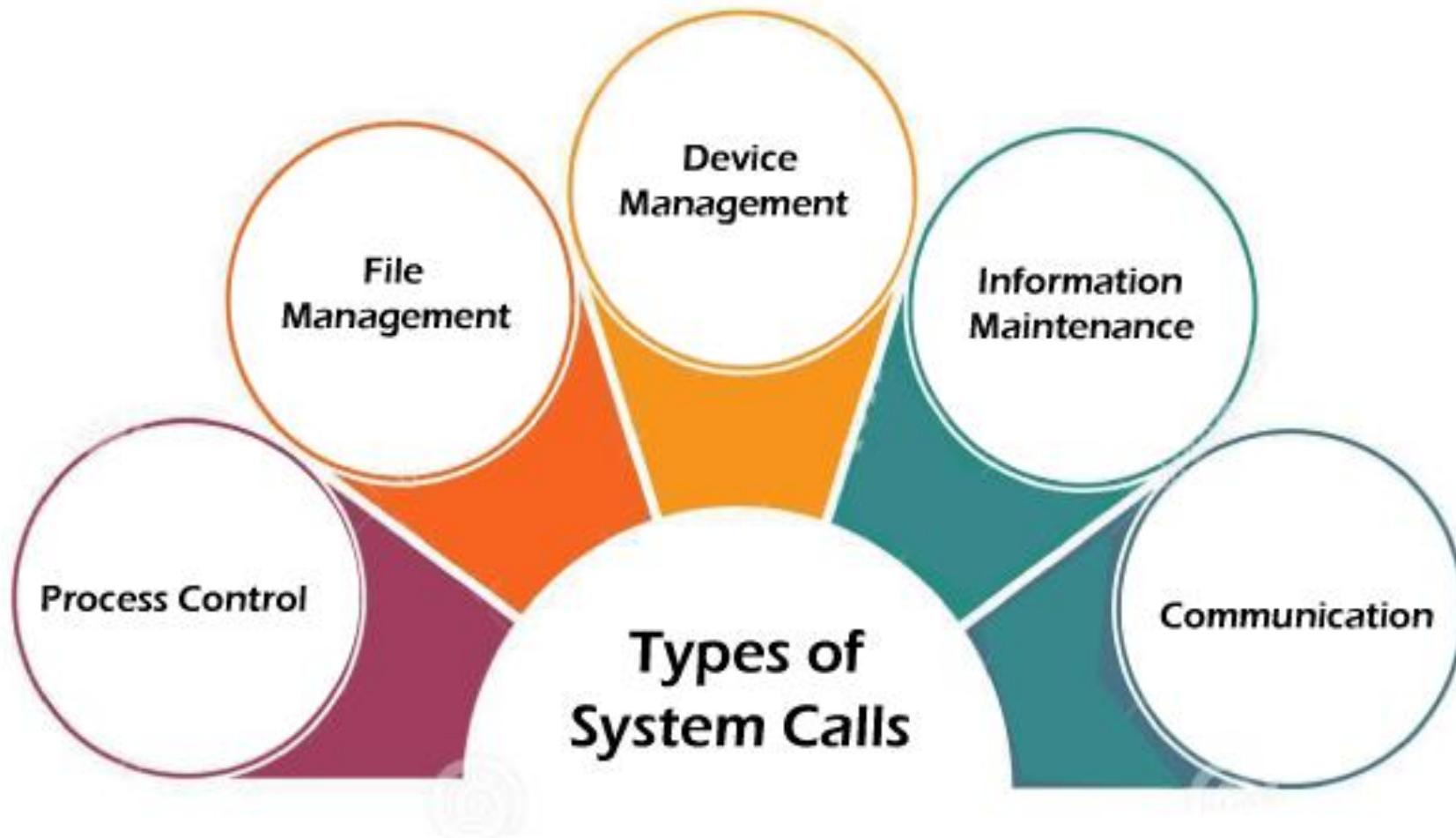
# System Call Parameter Passing

- Three general methods used to pass parameters to the OS
  - Simplest: **pass the parameters in registers**
    - In some cases, may be more parameters than registers
  - **Parameters stored in a block, or table, in memory, and address of block passed as a parameter in a register**
    - This approach taken by Linux and Solaris
  - **Parameters placed, or pushed, onto the stack by the program and popped off the stack by the operating system**
  - Block and stack methods do not limit the number or length of parameters being passed

# Parameter Passing via Table



# Types of System Calls



# Types of System Calls

- Process control
  - **create process, terminate process**
  - **end, abort**
  - load, execute
  - **get process attributes, set process attributes**
  - **wait for time**
  - wait event, signal event
  - allocate and free memory
  - Dump memory if error
  - **Debugger** for determining **bugs, single step** execution
  - **Locks** for managing access to shared data between processes

# Types of System Calls

- **File management**
  - **create file, delete file**
  - **open, close file**
  - **read, write, reposition**
  - **get and set file attributes**

# Types of System Calls

- Device management
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices

# Types of System Calls (Cont.)

- Information maintenance
  - **get time or date, set time or date**
  - get system data, set system data
  - **get and set process (process id), file, or device attributes**

# Types of System Calls (Cont.)

- Communications
  - **create, delete communication connection**
  - **send, receive messages**
  - If **message passing model to host name or process name**
    - From client to server
  - **Shared-memory model create and gain access to memory regions**
  - transfer status information
  - attach and detach remote devices

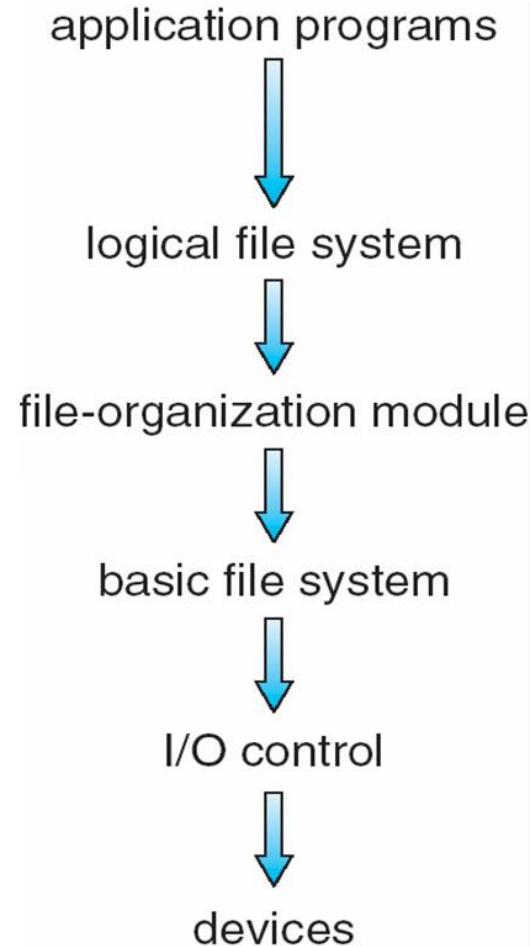
# Types of System Calls (Cont.)

- Protection
  - **Control access to resources**
  - **Get and set permissions**
  - Allow and deny user access

# Examples of Windows and Unix System Calls

|                                | <b>Windows</b>                                                                      | <b>Unix</b>                            |
|--------------------------------|-------------------------------------------------------------------------------------|----------------------------------------|
| <b>Process Control</b>         | CreateProcess()<br>ExitProcess()<br>WaitForSingleObject()                           | fork()<br>exit()<br>wait()             |
| <b>File Manipulation</b>       | CreateFile()<br>ReadFile()<br>WriteFile()<br>CloseHandle()                          | open()<br>read()<br>write()<br>close() |
| <b>Device Manipulation</b>     | SetConsoleMode()<br>ReadConsole()<br>WriteConsole()                                 | ioctl()<br>read()<br>write()           |
| <b>Information Maintenance</b> | GetCurrentProcessID()<br>SetTimer()<br>Sleep()                                      | getpid()<br>alarm()<br>sleep()         |
| <b>Communication</b>           | CreatePipe()<br>CreateFileMapping()<br>MapViewOfFile()                              | pipe()<br>shmget()<br>mmap()           |
| <b>Protection</b>              | SetFileSecurity()<br>InitializeSecurityDescriptor()<br>SetSecurityDescriptorGroup() | chmod()<br>umask()<br>chown()          |

# Layered File System



# Device Drivers

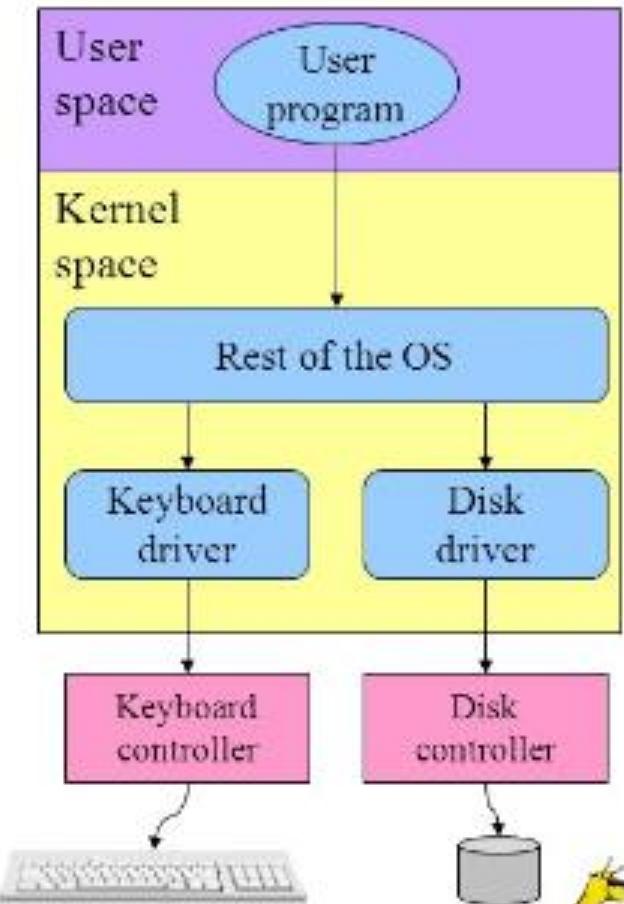
- **Device drivers** manage I/O devices at the I/O control layer
- **Device driver** controls the physical device
- The I/O control consists of
  - device drivers and
  - interrupt handlers
  - to transfer information between the main memory and the disk system.

# Device Drivers

- A device driver
  - can be thought of as a translator.
  - **Its input consists of high level commands such as “retrieve block 123”**
  - **Given commands like “read drive1, cylinder 72, track 2, sector 10, into memory location 1060”**
  - **outputs low-level hardware specific commands to hardware controller**

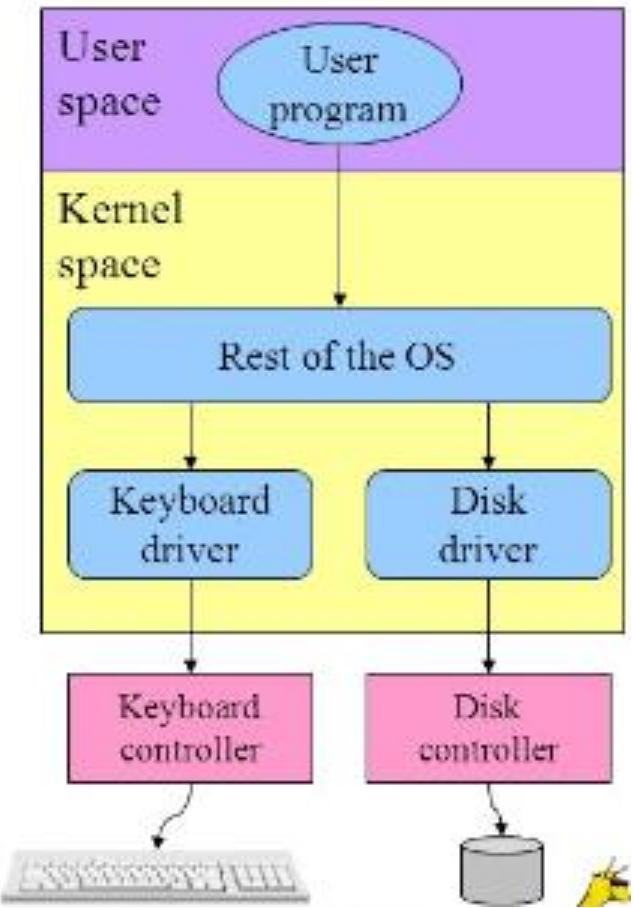
# Device Drivers

- Its output consists of
  - low level hardware specific instructions
  - that are used by the hardware controller
  - which interfaces I/O device to the rest of the system.
- The device driver
  - writes specific bit patterns
  - to special locations in the I/o controller's memory
  - to tell the controller on which device location to act and
  - what actions to take.



# Device Drivers

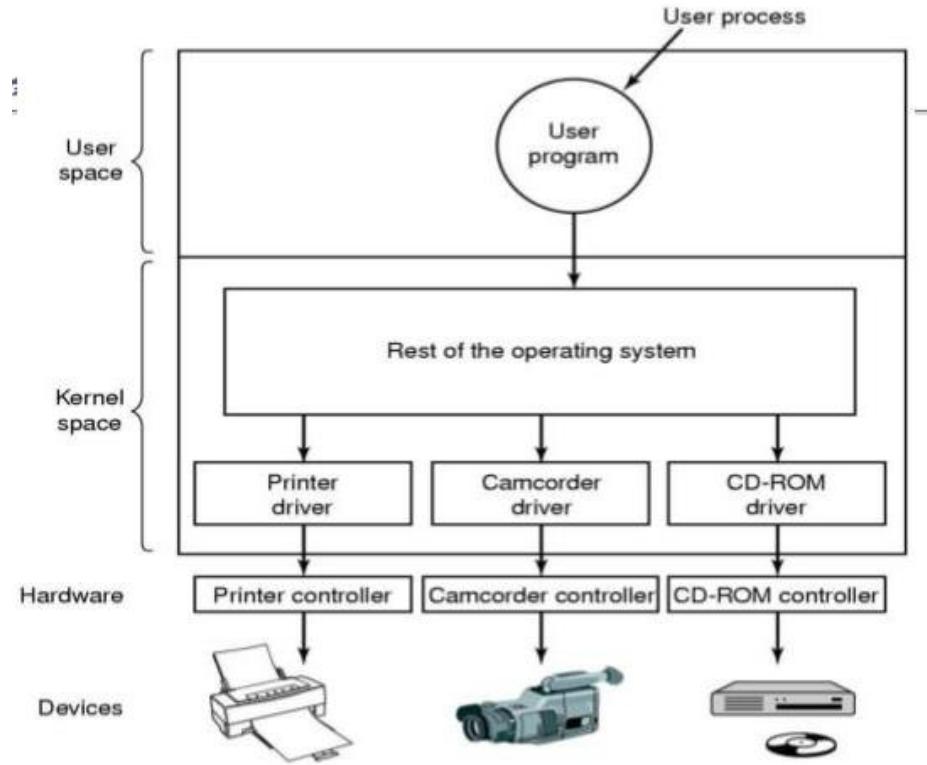
- Device drivers are
  - **software modules that can be plugged into an OS to handle a particular device.**
- Operating System takes help from
  - device drivers to handle all I/O devices.



# Device Controllers

- The Device Controller works like
  - an interface between a device and a device driver.
- There is always a device controller and a device driver
  - for each device to communicate with the Operating Systems.

## Device Drivers



Courtesy:[https://www.tutorialspoint.com/operating\\_system/os\\_io\\_hardware.htm](https://www.tutorialspoint.com/operating_system/os_io_hardware.htm)

# Device Controllers

- I/O units (Keyboard, mouse, printer, etc.) typically consist of
  - **a mechanical component and**
  - **an electronic component**
  - **where electronic component is called the device controller.**

Courtesy:[https://www.tutorialspoint.com/operating\\_system/os\\_io\\_hardware.htm](https://www.tutorialspoint.com/operating_system/os_io_hardware.htm)

# Device Controllers

- **A Electrical component/device controller controls the device**
  - May be able to handle multiple devices.
  - **May be more than one controller per mechanical component. (Eg-Hard drive)**
- Controller's Tasks-
  - Convert serial bit stream to block of bytes,
  - **Perform error correction as necessary**
  - **Make device available to main memory**

Courtesy:[https://www.tutorialspoint.com/operating\\_system/os\\_io\\_hardware.htm](https://www.tutorialspoint.com/operating_system/os_io_hardware.htm)

# Operating System Design and Implementation

- Design and Implementation of OS not “solvable”, but some approaches have proven successful
- **Internal structure of different Operating Systems can vary widely**
- Start the design **by defining goals and specifications**
- Affected by **choice of hardware, type of system**

# Operating System Design and Implementation

- **User** goals and **System** goals
  - User goals – operating system should be
    - convenient to use,
    - easy to learn,
    - reliable,
    - safe, and
    - fast

# Operating System Design and Implementation

- System goals – operating system should be
  - easy to design, implement, and maintain,
  - as well as flexible,
  - reliable,
  - error-free, and
  - efficient

# Operating System Design and Implementation (Cont.)

- Important principle to separate  
**Policy:** *What* will be done?  
**Mechanism:** *How* to do it?
- Policies decide what will be done, Mechanisms determine how to do something
- **The separation of policy from mechanism is a very important principle,**
  - it allows maximum **flexibility if policy decisions are to be changed later (example – timer)**
- Specifying and designing an OS is highly creative task of **software engineering**

# Implementation

- Much variation
  - Early OSes in assembly language
  - Then system programming languages like Algol, PL/1
  - Now C, C++
- Actually usually a mix of languages
  - Lowest levels in assembly
  - Main body in C
  - Systems programs in C, C++, scripting languages like PERL, Python, shell scripts
- More high-level language easier to **port** to other hardware
  - But slower

# OS Design Considerations for Multiprocessor and Multicore architectures

# Symmetric Multiprocessor

- In an SMP system,
  - the **kernel can execute on any processor**, and
    - typically each processor does self-scheduling from the pool of available processes or threads.
  - The kernel can be constructed
    - **as multiple processes or**
    - **as multiple threads,**
    - allowing **portions of the kernel to execute in parallel.**

# Symmetric Multiprocessor OS

- In an SMP system,
  - The SMP approach complicates the OS.
  - The OS designer must deal with the complexity due to
    - sharing resources (like data structures) and
    - coordinating actions (like accessing devices)
    - from multiple parts of the OS executing at the same time.

# **OS Design Considerations for Symmetric Multiprocessor OS:**

- 1) Simultaneous concurrent processes or threads**
- 2) Scheduling**
- 3) Synchronization**
- 4) Memory management**
- 5) Reliability and fault tolerance**

# OS Design Considerations for Symmetric Multiprocessor OS:

## Simultaneous concurrent processes or threads:

- Kernel routines need to be reentrant
  - to allow several processors
  - to execute the same kernel code simultaneously.
- With multiple processors executing the same or different parts of the kernel,
  - kernel tables and management structures must be managed properly
  - to avoid data corruption or invalid operations.

# OS Design Considerations for Symmetric Multiprocessor OS:

## Scheduling:

- Any processor may perform scheduling,
  - which complicates the task of enforcing a scheduling policy and
  - **assuring that corruption of the scheduler data structures is avoided.**
- If kernel-level multithreading is used, then
  - The opportunity exists to schedule **multiple threads from the same process simultaneously on multiple processors.**

# OS Design Considerations for Symmetric Multiprocessor OS:

## Synchronization:

- With multiple active processes having potential access
  - to shared address spaces or shared I/O resources,
  - care must be taken to provide effective synchronization.
- **Synchronization is a facility that enforces**
  - mutual exclusion and
  - event ordering.
- **A common synchronization mechanism**
  - used in multiprocessor operating systems is locks.

# OS Design Considerations for Symmetric Multiprocessor OS:

## Memory management:

- Memory management on a multiprocessor must deal with
  - **all of the issues found on uniprocessor computers**
- In addition, the OS needs to **exploit**
  - the available **hardware parallelism** to achieve the best performance.

# OS Design Considerations for Symmetric Multiprocessor OS:

## Memory management:

- The paging mechanisms on different processors must be coordinated to enforce consistency
  - when several processors share a page or segment and
  - to decide on page replacement.
  - The reuse of physical pages is the biggest problem of concern;
    - **that is, it must be guaranteed**
    - **that a physical page can no longer be accessed with its old contents**
    - **before the page is put to a new use.**

# OS Design Considerations for Symmetric Multiprocessor OS:

## Reliability and fault tolerance:

- The OS should provide **graceful degradation**
  - in the face of processor failure.
- The scheduler and other portions of the OS
  - **must recognize the loss of a processor and**
  - **restructure management tables accordingly**

# OS Design Considerations for Symmetric Multiprocessor OS:

## Reliability and fault tolerance:

- Because multiprocessor OS design issues generally involve extensions to
  - solutions to multiprogramming uniprocessor design problems,
  - we do not treat multiprocessor operating systems separately.

# OS Design Considerations for Multicore architectures

- The considerations for multicore systems include
  - all the design issues discussed so far in this section for SMP systems.
- But additional concerns arise.
  - The issue is one of the scale of the potential parallelism.

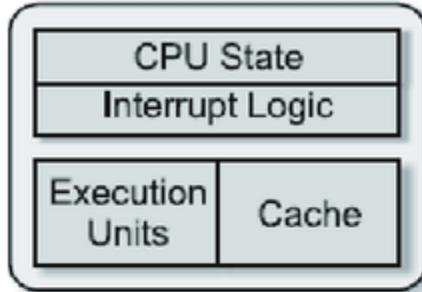
# Multicore architectures

- Multicore vendors offer systems with up to eight cores on a single chip.
- With each succeeding processor technology generation,
  - the number of cores and
  - the amount of shared and dedicated cache memory increases,
  - so that we are now entering the era of “many-core” systems.

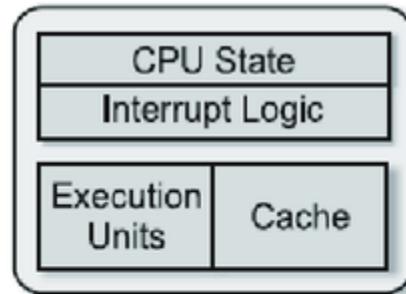
# Unicore vs Multicore

- A processor with a single core is called a Unicore processor.
- But a processor with two or more cores is called a Multicore processor.

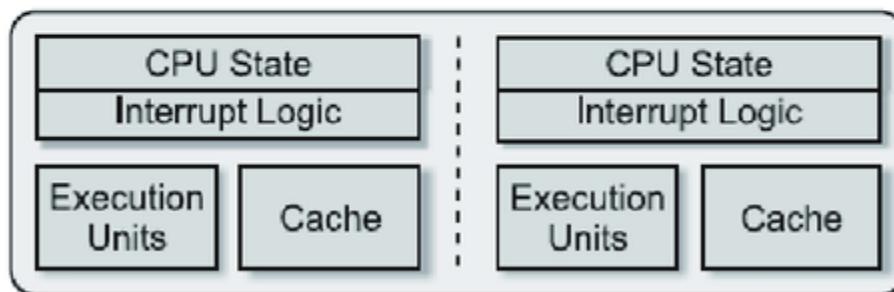
# Multicore



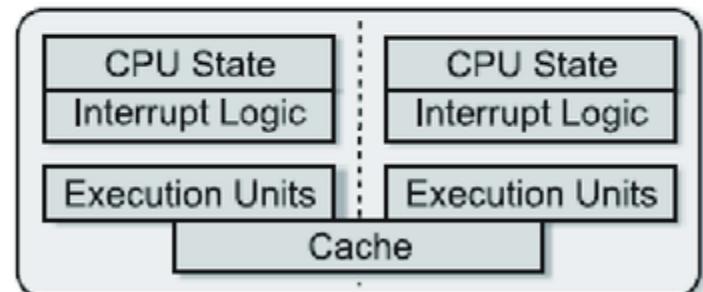
(a) Single core



(b) Multiprocessor



(c) Multi-core

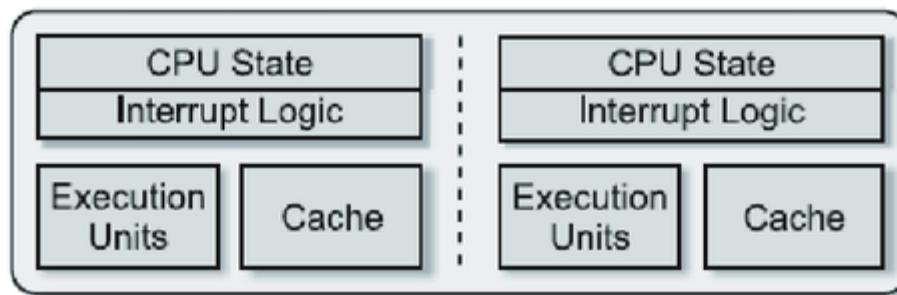


(d) Multi-core with shared cache

[https://www.researchgate.net/figure/Simple-comparison-of-single-core-multi-processor-and-multi-core-architectures-after\\_fig3\\_266489075](https://www.researchgate.net/figure/Simple-comparison-of-single-core-multi-processor-and-multi-core-architectures-after_fig3_266489075)

# Multicore

- The cores of a multicore processor can individually read and execute program instructions at the same time.
- This increases the speed of execution of the programs and supports parallel computing.
- Eg: Quadcore processors, Octacore processors, etc.



(c) Multi-core

# OS Design Considerations for Multicore architectures

## **PARALLELISM WITHIN APPLICATIONS**

- Most applications can be subdivided into multiple tasks that can execute in parallel,
  - With these tasks subdivided into multiple processes,
  - Perhaps each with multiple threads.
- The difficulty is that the developer must decide
  - how to **split up the application work into independently executable tasks.**
  - That is, the developer must decide **what pieces can or should be executed asynchronously or in parallel.**

# OS Design Considerations for Multicore architectures

## ***PARALLELISM WITHIN APPLICATIONS***

- It is primarily the compiler and the programming language features
  - that support the parallel programming design process.
- But, the OS can support this design process, at minimum,
  - by efficiently allocating resources among parallel tasks as defined by the developer.

# OS Design Considerations for Multicore architectures

## ***VIRTUAL MACHINE APPROACH***

- With the **ever-increasing number of cores on a chip**,
  - the attempt to multiprogram individual cores to support multiple applications may be a misplaced use of resources.

# OS Design Considerations for Multicore architectures

## ***VIRTUAL MACHINE APPROACH***

- If instead, **we allow one or more cores to be dedicated to a particular process** and
  - then **leave the processor alone to devote its efforts to that process,**
  - we avoid much of the **overhead of task switching and scheduling decisions.**

# OS Design Considerations for Multicore architectures

## ***VIRTUAL MACHINE APPROACH***

- The multicore OS could then act as a hypervisor
  - that makes a high-level decision **to allocate cores to applications**
  - but does little in the way of resource allocation beyond that.