Module 1 - Introduction to Operating Systems

Contents

- Introduction to OS
- Operating System Structure,
- Operating System Operations
- Operating System Services,
- User and OS interface,
- System Calls and its types,
- System Programs and its types
- Loaders, Linkers
- Overview of OS design and implementation.

Introduction to Operating System

- An OS is a large & complex system software designed & created piece by piece.
- Each of these pieces are designed & created with carefully defined inputs, outputs & functions
- Generally, now-a-days, all electronic computing systems are having OS in one or another format.
- Personal computers, mobiles, laptops, automated washing machines, ovens, aircraft management systems, etc. doesn't work without OS
- OS is an essential part of any computer system.
- Similarly, a course on OS is an essential part of any CS education

What Operating Systems do ?

- An OS is a collection of programs that manages the computer's hardware.
- It also provides a basis for application programs and acts as an intermediary between the user of a computer and the computer hardware.
- Purpose of OS is to provide an environment in which, users can execute the programs in a convenient & efficient manner.
- SO, any OS has 2 goals
  - Efficient use of computer system
  - User convenience
- But, no OS provides both.
- **Efficient use** is important when a computer system is shared by several users while **user convenience** is important in personal computers

What Operating Systems do?

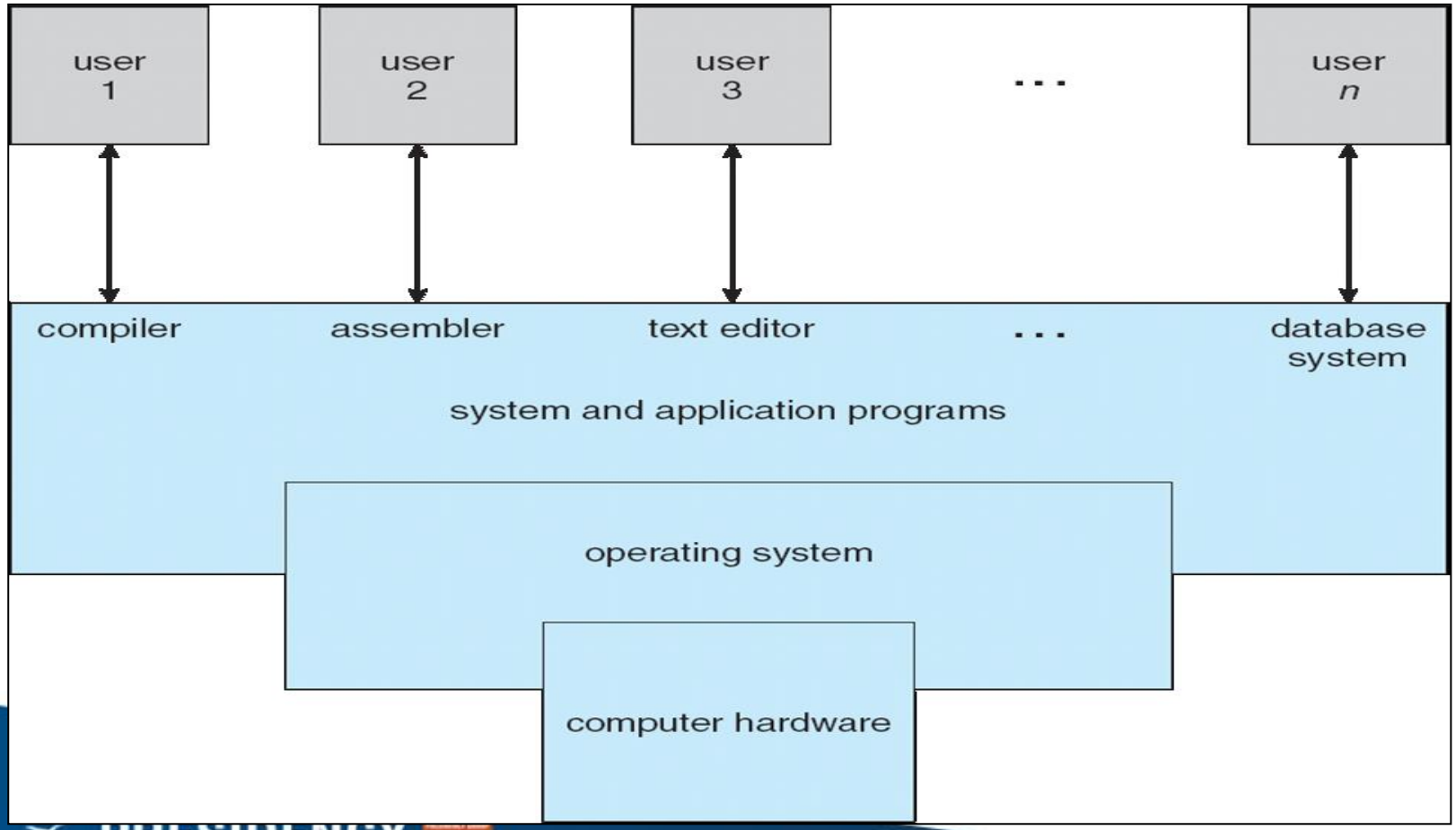
- OS controls the hardware & coordinates its use among the various application programs for the various users
- An OS is similar to government. Like a government, it performs no useful function for itself. It simply provides an environment within which other programs can do useful work.
- An OS is responsible for coordinating all of the computers individual components so that they work together according to a single plan

Computer System Structure

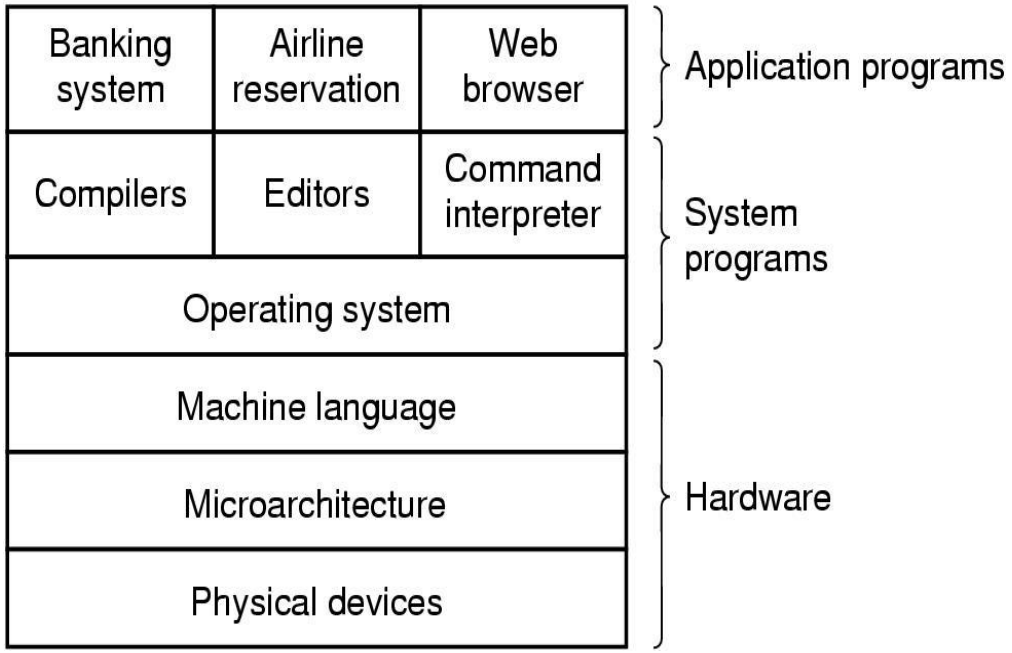
Computer system can be divided into four components:

- **Hardware – provides basic computing resources**
  - **CPU, memory, I/O devices**
- **Operating system**
  - **Controls and coordinates use of hardware among various applications and users**
- **Application programs – define the ways in which the system resources are used to solve the computing problems of the users**
  - **Word processors, compilers, web browsers, database systems, video games**
- **Users**
  - **People, machines, other computers**

Four Components of a Computer System



Introduction



- **A computer system consists of**
  - **hardware**
  - **system programs**
  - **application programs**

Two View points

- An OS can be viewed in 2 ways
  - User view
  - System view
- **User view**
  - Always, user wants their OS in a convenient manner i.e., **ease of use.**
  - So, user view of the computer depends on the interface used i.e., either GUI or Command line.
  - Some users may use PC's. In this type, systems are designed so that only one user can utilize the resources & mostly for ease of use where the attention is mainly on performances & not on the resource utilization.

Users View

1. Stand alone System: Ease of Use & High Performance

2. If the users are at different terminals connected to a mainframe or mini computers : OS is designed to maximize resource utilization.

3. If the users are in workstations, connected to networks and servers : OS is designed for both ease of use and resource availability (files).

Users View

4.Users of hand held systems, expects the OS to be designed for ease of use and performance per amount of battery life.

5. systems like embedded systems used in home devies (like washing m/c) & automobiles do not have any user interaction.

User View

- Some users may use terminals connected to mainframe or minicomputers. Some other users are accessing the same computer through other terminals
  - Here, users at different terminals may exchange information & may share the resources
  - In this case, OS is designed to maximize resource utilization
- In other case, users may sit at workstations connected to networks of other workstations & servers
  - These users have dedicated resources at their end but they also share resources such as networking & servers. Therefore their OS is designed to compromise between individual usability & as well as resource utilization
- But some computers have little user view or no user view at all. For example, embedded computers in home devices & automobiles may have numeric keypads & may turn indicator lights ON or OFF to show some status, but they & their OS are designed primarily to run without user intervention

System View (Computer's point of view)

- From System view**,** OS is the program mainly involved with the hardware. In this context, we can view an OS as a **Resource allocator**.
- Computer resources may be CPU time, memory space, file-storage space, I/O device & so on.
- In order to solve any problem, a computer system has to use one or more Computer resources
  - OS acts as the manager of these resources. OS must decide how to allocate these resources to programs & to the users so that it can operate the computer system efficiently & fairly.
- OS need to **control various I/O devices & user programs** i.e., OS acts as a control program to manage the execution of user program to prevent errors & improper use of the computer

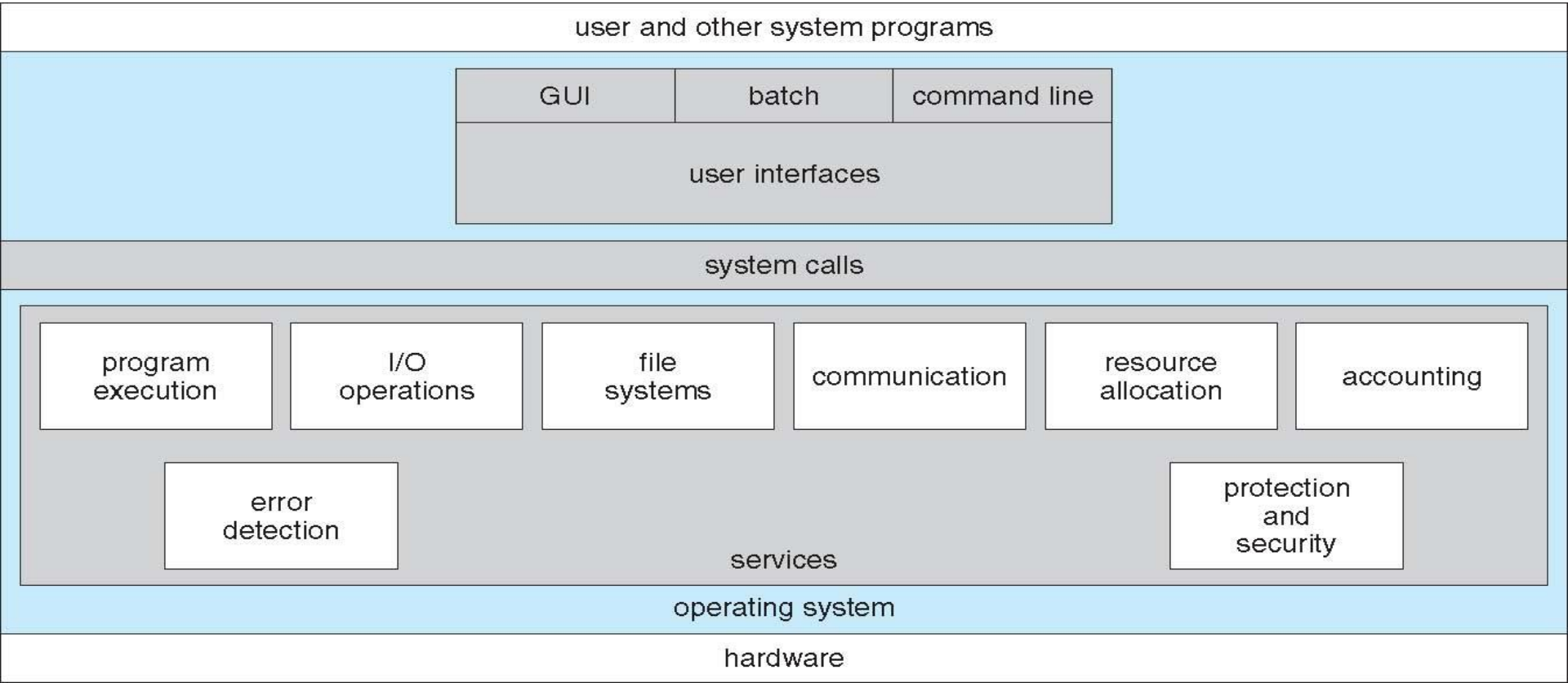
Abstract view of a computer system

1. Hardware – provides basic computing resources (CPU, memory, I/O devices).
2. Operating system – controls and coordinates the use of the hardware among the various application programs for the various users.
3. Applications programs – define the ways in which the system resources are used to solve the computing problems of the users (compilers, database systems, video games, business programs).
4. Users (people, machines, other computers).
5. (General View)

OS provides some services to programs & as well as to users of those programs.

Services provided by an OS differ from one OS to another but there is some common classes

OS provides some services for the convenience of programmer to make programming easy.



Operating System Services

1) User Interface (UI)

- Almost all OS have UI, in order to interact with the computer
- Any OS provides UI in 3 forms

a) Command line interface- uses text commands at the command prompt
b) Batch interface—commands, methods & directives are entered into files & those files are executed
c) GUI—common in almost all the OS

2) Program execution

- OS must be able to load a program from secondary memory into main memory and to run it.
- After execution, OS should end that program either normally (successful case) or abnormally (indicating error)

Operating System Services

3) I/O operations

- Running program may require I/O like file or I/O device.
- For efficiency & protection, users usually cannot control I/O devices directly
- Therefore, OS must provide a means for controlling I/O devices

4) File-system manipulation

- User performs operations on files like, read, write, create, delete ,search by their names.
- Sometimes, programs includes permission management to allow or deny access to files or directories based on file ownership
- For all these, OS should provide some efficient features & facility to perform like system manipulation

Operating System Services

5) Communications

- In some situations, one process needs to exchange information with another process.
- This type of communication may occur between processes that are executing on same computer or between processes running on different computers in the network.
- OS achieves this via shared memory or message passing

6) Error detection

- Errors may occur in
  - CPU
  - Memory hardware(memory full, power failure, etc)
  - I/O devices(parity error on tape, connection failure on network, lack of paper in printer, etc)
  - User program(arithmetic overflow, divide by zero, accessing illegal memory location)

  For each type of errors, OS should take appropriate action to ensure correct & consistent computing

Operating System Services

All the 6 services provided by OS is for user convenience. Some other services which provides efficient operation for the system convenience are

7) Resource allocation

- When multiple users log onto the system or when multiple jobs are running , resources must be allocated to each of them.
- So, it is responsibility of OS to manage the available resources

8) Accounting

- OS should keep track of which users use how many & what kind of resources.
- This record keeping may be used for accounting(statistics or billing)
- It can also be used to improve the system efficiency

9) Protection & Security

- In multi-process environment, it is possible that, one process may interface with the other or with OS itself.
- Some users, stores their information in multiuser computer system or networked computer system, such data should be protected
- External I/O devices must also be protected from invalid access

OS Operations

- Modern OS's are **interrupt driven**.
- Always, events are happened/occurred in the form of interrupt or trap
- A **trap(exception)** is a software generated interrupt which occurs either due to
  - error (division by zero, invalid memory access)
  - Specific request from a user program
- WKT, both hardware & software resources of the computer system are shared by OS & as well as user programs.
- So care should be taken when any error occurs due to user program.

Interrupt

An interrupt temporarily stops or terminates a service or a current process.

An interrupt is **a signal emitted by a device attached to a computer or from a program within the computer**.

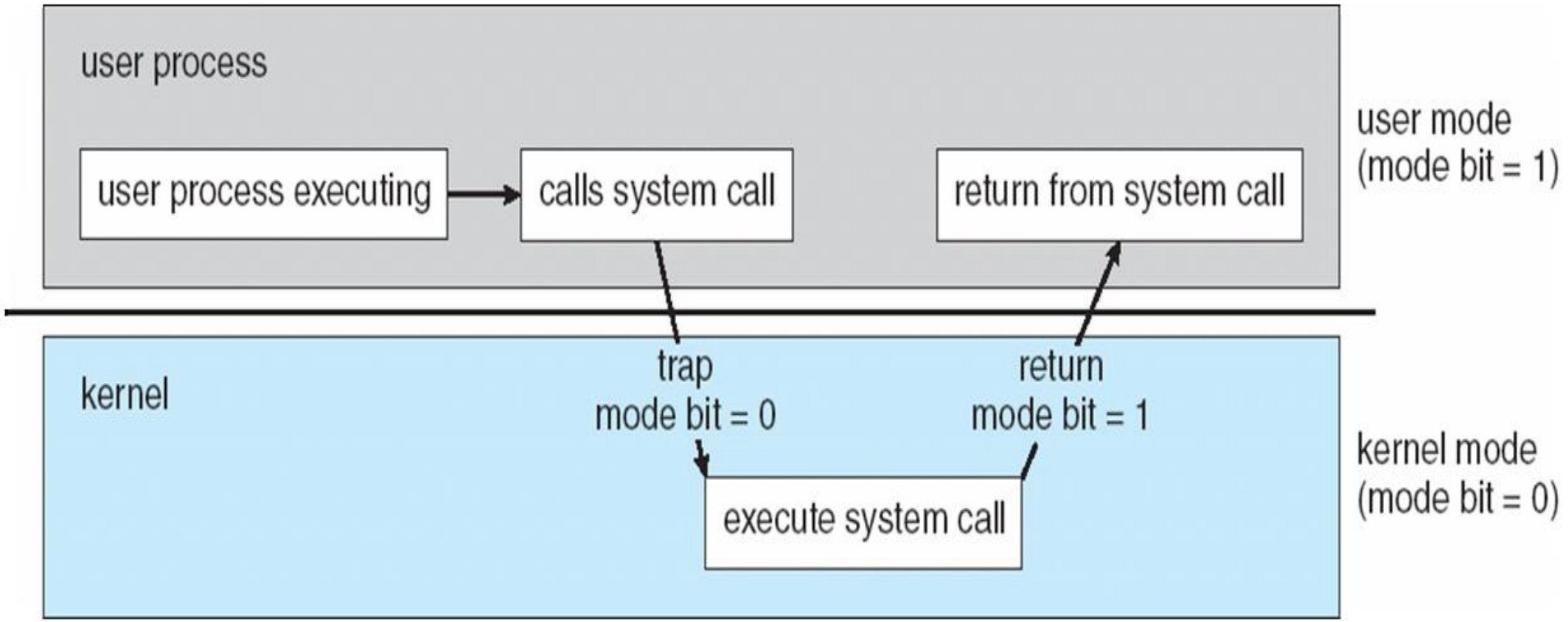
It requires the operating system (OS) to stop and figure out what to do next.

Dual mode operation

In order to ensure proper execution of OS, we must be able to distinguish between execution of OS code & user-defined code

There are 2 modes of operation

1) user mode
2) kernel mode



Dual mode operation

- Single bit called **mode bit** is used to identify mode in which OS is running
- If **mode bit =0** ----- OS is running in kernel mode
- If **mode bit =1** ----- OS is running in user mode
- After booting OS, normally user performs some action i.e., some user process is executing.

whenever user process needs some service, it requests the OS through some system call.

Then, OS switches from user mode to kernel mode by making **mode bit =0**

Then, it executes corresponding system call code

Dual mode operation

- When system starts booting, hardware starts in kernel mode. Then, OS is loaded & starts user applications in user mode.
- Whenever a trap or interrupt occurs, hardware switches from user mode to kernel mode by changing the status of **mode bit =0**
- Thus, whenever OS gains control of the computer, it is in kernel mode.
- System always switches to user mode before passing control to user program.
- Out of all instructions, in the instruction set, some of them are designated as **privileged instructions.** These are executed only in the kernel mode

Privileged instructions

Some common privileged instructions are –

- Input-Output Management
- Switching modes between *user mode and kernel mode*.
- Interrupt management
- **clear memory**
- Set the value of the timer
- modify entries in device status table

Why should "accessing I/O device" be a privileged instruction?

If I/O instructions are entrusted to users, they may misuse them(e.g.overwriting FAT entries, or destroying important disk data).

Now I/O instructions are privileged **so that, OS could check whether you are authorized to do that I/O operation or not, before performing I/O**.

Timer

- When a user executes some program in user mode, it should not stuck in an infinite loop.
- Similarly, process should not stuck in kernel mode for long time
- At regular intervals, it should switch the mode. For this purpose, **timer** is used
- A timer can be set to interrupt the computer after a specified period.
- This period may be fixed (1/60 second) or variable(from 1m sec to 1 sec)
- OS sets the counter. Every time, clock ticks, counter is decremented. When the counter reaches 0, interrupt occurs. Thus, we can use timer to prevent user program from running too long.

User & Operating System interface

- Main functionality of an OS is to provide an interface for the user to interact with the computer.
- 2 commonly used approaches are
  - Command line interface (command interpreter) (Commands)
  - Graphical user interface (GUI) (Graphical Elements)
- Command Line Interface
  - Some OS include this command interpreter in the kernel
  - Others such as windows XP & unix, treat this command interpreter as a special program running when some job is initiated or when user logs in to the system first time.

Command Line Interface contd…..

- On some systems, there are multiple command interpreters, user has to chose one among them.
- In such cases, those interpreters are called as shells
- In CLI, user has to enter the commands at the command prompt.
- For every task, there will be some commands & user has to remember all those commands & their formats. So it is not so user friendly
- ls, cd, cp, mv, rm, mkdir, rmdir, etc

Graphical user interface (GUI)

- GUI is more user friendly than CLI
- Here, user just moves the mouse to position its pointer on images or icons on the screen that represents programs, files, directories & system functions
- Here, everything is represented in graphical form
- For ex, application programs, commands, disk drives, files, etc are presented in the form of icons
- Usually, command is given to the computer by clicking with mouse on the icon
- GUI also provides menus, buttons & other graphical objects to the user to perform different tasks.

Difference between CLI and GUI

Let us talk about the differences between CLI and GUI. To make this topic more understandable and clear, we are comparing both of the terms based on their individual characteristics in a table.

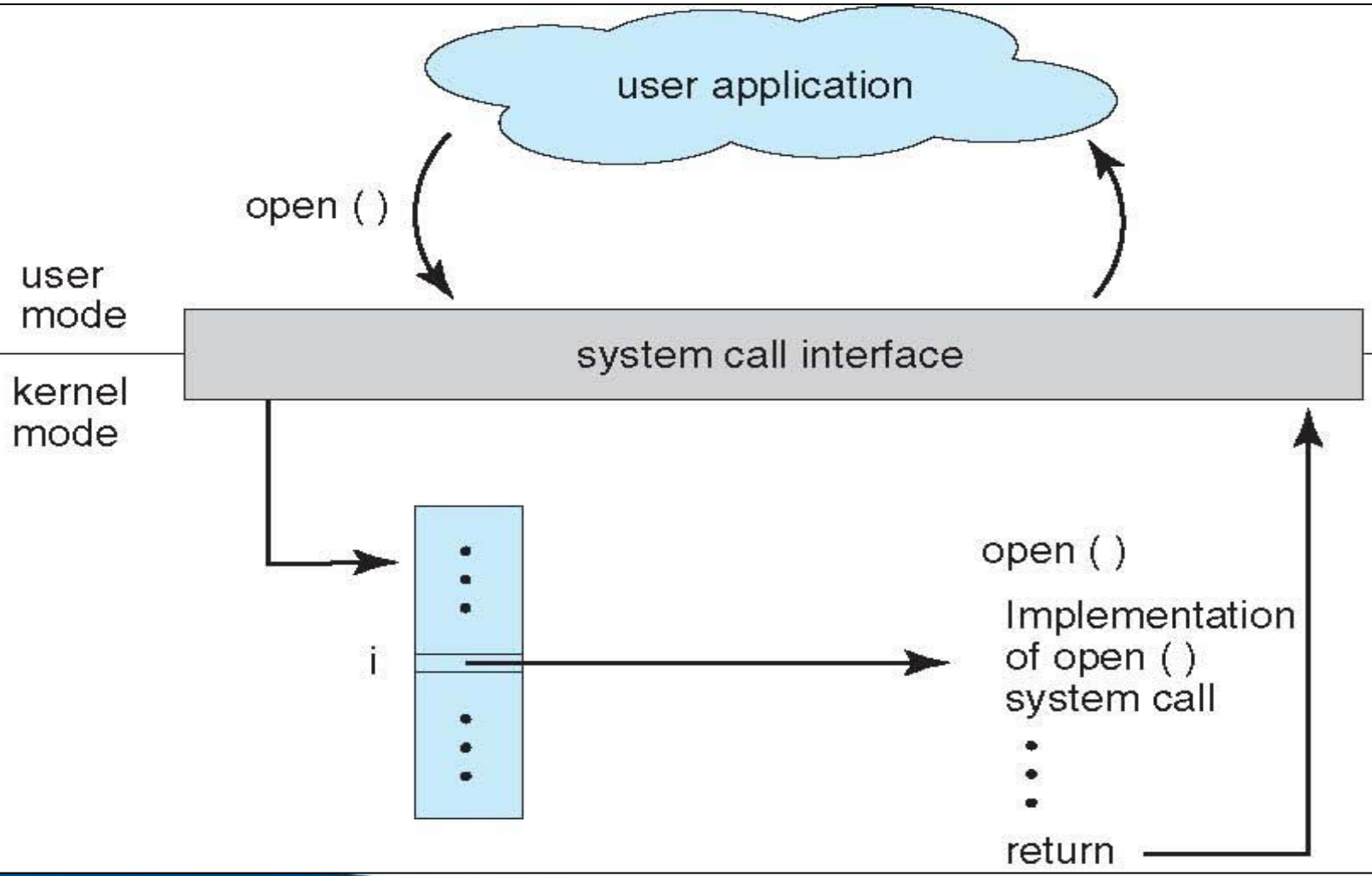
Parameters	CLI	GUI
Ease of Use	It is comparatively more difficult to understand and use.	It is comparatively easier to understand and use.
Memory Consumption	CLI consumes comparatively less memory.	The GUI consumes comparatively more memory.
Level of Precision	Higher precision of work can be obtained using CLI.	GUI offers a lower level of precision.
Speed	It works at a higher speed as compared to the GUI.	It works at a much slower speed as compared to the CLI.
Keyboard and Mouse	Its OS only requires a user's keyboard.	Its OS requires both keyboard and mouse to work.
Modification of Appearance	We cannot change or modify the CLI operating system's appearance.	We can change or modify the GUI operating system's appearance.
Graphics	No graphs are included in CLI.	Graphics are always used in the GUI.
Menus	No menus are provided in CLI.	The GUI OS comes with menus.
Display of Information	The information that the user wants to view is displayed in files and plain text.	The information that the user wants to access is presented in various forms, like plain text, images, videos, gifs, videographs, etc.
Input of Information	The input is usually entered at the command prompt in CLI.	We can input the data anywhere on the computer screen in the case of GUI.
Pointing devices	Pointing devices are not used at all in CLI.	We use pointing devices in the GUI for choosing/selecting the items we want to.
Avoiding Errors	No typing errors or spelling mistakes can be avoided by CLI.	The typing errors or spelling mistakes cannot be avoided by the GUI.

System Calls

- User communicate with the computer through OS. Then, OS has to communicate with the computer resources. This is done through system calls.
- system calls built on top of OS.
- system calls interact with kernel area of OS
- system calls can also interact with hardware part of system like keyboard, mouse, printer, CPU, etc.
- system calls are generally available as routines written in C, C++ & most of the times in assembly language instructions
- "system calls is defined as an interface between OS & a process that allows a process to invoke OS functions"
- 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) /Portable OS interface)

Note that the system-call names used throughout this ppt are generic

Handling user application invoking open() system call



Types of System Calls

- Process control
  - CreateProcess(), fork()----to create the process
  - ExitProcess(), exit()---------to terminate the process
  - end, abort--------to halt the process normally & abnormally
  - 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
- 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, file, or device attributes
- 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

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

System Programs

- 2 types----- application software & system software
- Application software
  - It is a computer software designed to help the user to perform specific tasks
  - Runs on the top of system software
  - It interacts with system software which in turn interacts & makes physical hardware functional
- System software
  - Designed to operate the computer hardware & to provide a platform for running application software
  - It creates his own environment to run itself & run other application
  - Keep running all the times in the computer
  - These are also called as **system utilities** provides a convenient environment for program development & execution

System Programs & its types

1) File Management
   - these programs create, delete, copy, rename, print, dump, list & generally manipulate files & directories
2) Status information
   - Some programs simply ask system for date, time, amount of disk space, number of users or similar status information
   - Others are more complex, providing detailed performance, logging & debugging information
   - Typically, these programs format & print the output to the terminal or other output devices or files or display it in a window of the GUI

System Programs & its types

3) File Modification

- several text editors may be available to create & modify the content of files stored on disk or other storage devices.
- There may also be special commands to search contents of files or perform transformations of the text

4) Programming-language support

- some of the essential system software like compilers, assemblers ,debuggers & interpreters for programming languages are often provided to the user along with the OS

System Programs & its types

5) Program loading & execution

- once a program is assembled or compiled, it must be loaded into memory to be executed.
- System may provide absolute loaders, re-locatable loaders, linkage editors & overlay loaders

6) Communications

- these system programs provide mechanism for creating virtual connections among processes, users & computer systems.
- They allow users to send messages to one another's screens, to browse web pages, to send e-mail messages, to log-in remotely or to transfer files from one machine to another

Loaders

- The Source Program written in assembly language or high level language will be converted to object program, which is in the machine language form for execution.
- This conversion either from assembler or from compiler, contains translated instructions and data values from the source program, or specifies addresses in primary memory where these items are to be loaded for execution.

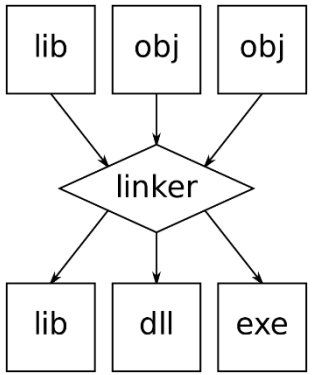
Loaders

- This contains following 3 processes, and they are,

1) **Loading** - which allocates memory location and brings the object program into memory for execution - (Loader)

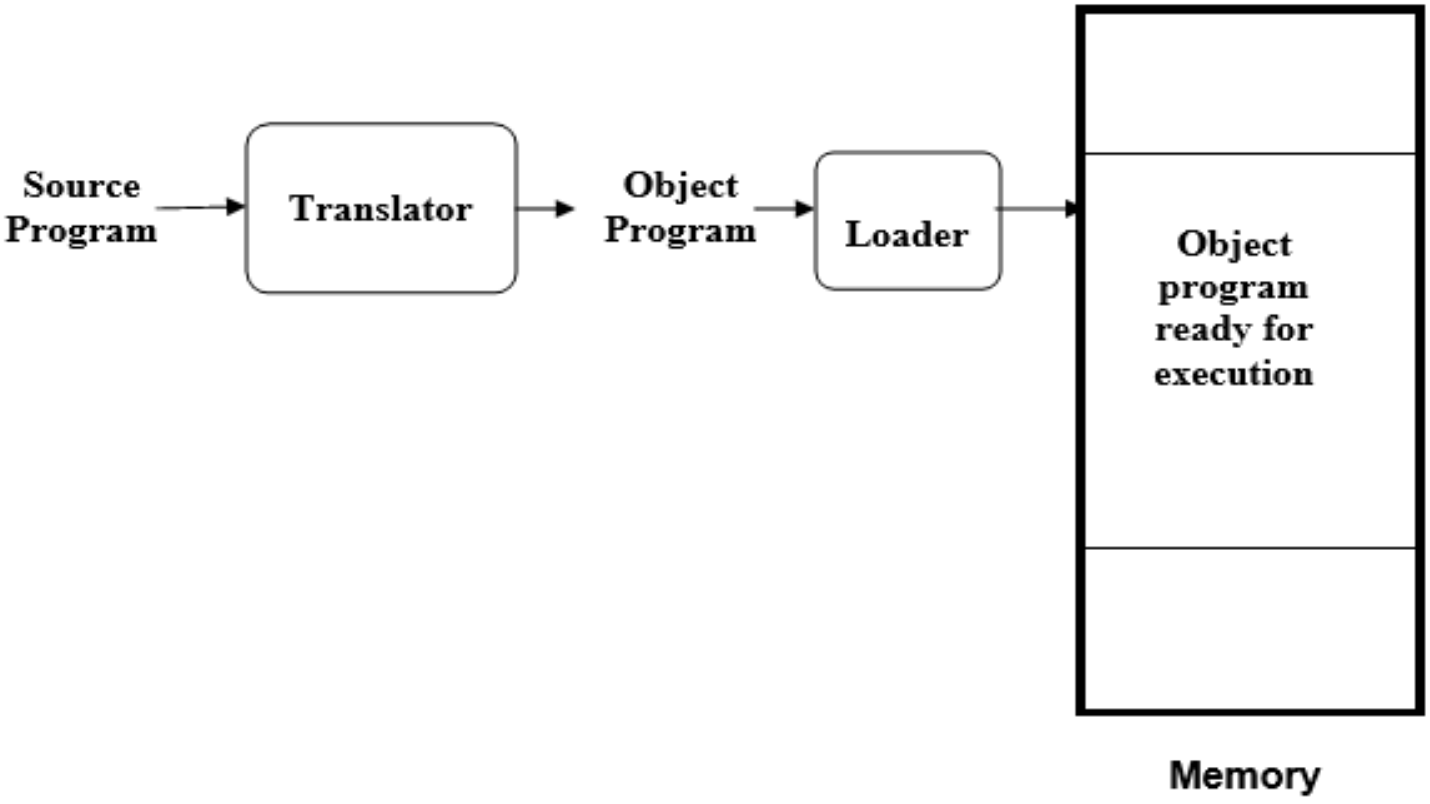
2)Linking- which combines two or more separate object programs and supplies the information needed to allow references between them - (Linker)

3)Relocation - which modifies the object program so that it can be loaded at an address different from the location originally specified - (Linking Loader)



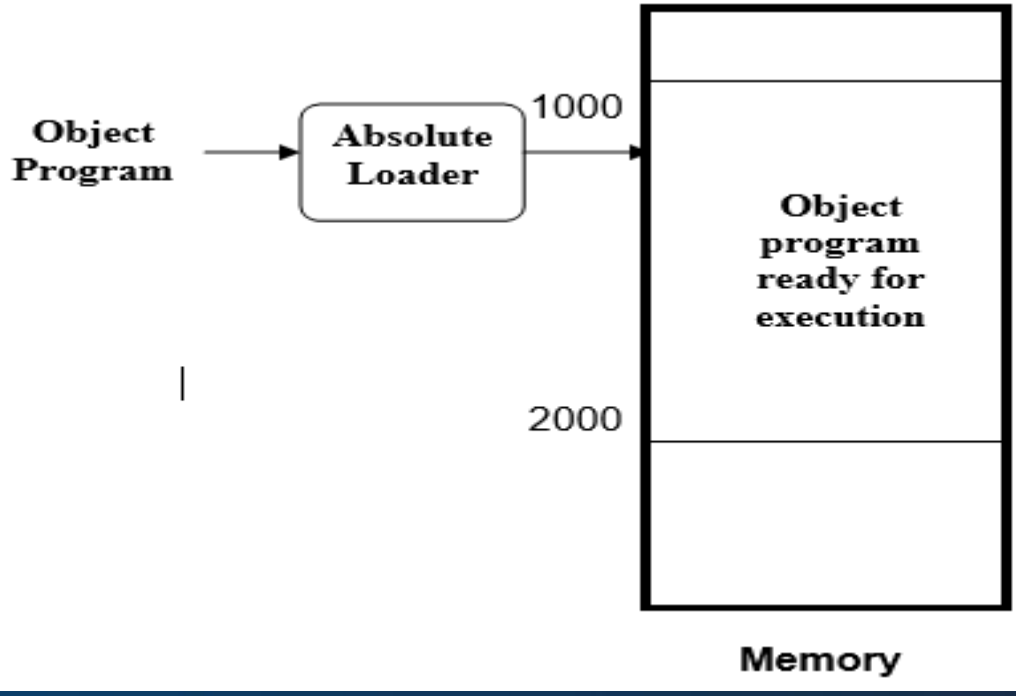
Basic Loader function

A loader is a system program that performs the loading function. It brings object program into memory and starts its execution.



Types of Loaders

- **Absolute Loader**
  - The object code is loaded to specified locations in the memory.
  - At the end the loader jumps to the specified address to begin execution of the loaded program.
  - The advantage of absolute loader is simple and efficient.
  - But the disadvantages are, the need for programmer to specify the actual address, and, difficult to use subroutine libraries.



Types of Loaders

A Simple Bootstrap loader

- When a computer is first turned on or restarted, a special type of absolute loader, called bootstrap loader is executed.
- This bootstrap loads the first program to be run by the computer -- usually an operating system.
- The bootstrap itself begins at address 0. It loads the OS starting address 0x80.
- No header record or control information, the object code is consecutive bytes of memory.

Types of Loaders

Relocatable loaders

- The concept of program relocation is, the execution of the object program using any part of the available and sufficient memory.
- The object program is loaded into memory wherever there is room for it.
- The actual starting address of the object program is not known until load time.
- Relocation provides the efficient sharing of the machine with larger memory and when several independent programs are to be run together.
- It also supports the use of subroutine libraries efficiently. Loaders that allow for program relocation are called relocating loaders or relative loaders.

Types of Loaders

Direct Linking Loader

- Direct linking loader is a general Relocatable loader and perhaps the most popular loading scan presently used.
- It has the advantage of allowing the programmer multiple procedure segments and multiple data segments.
- It has also an advantage that has given to the programmer complete freedom in referencing data or instructions content in other segments.
- This provides flexibility of inter segment referencing and accessing while at the same time allowing independent translation of programs.

Types of loaders

Dynamic Loader

- loader that actually intersect the “calls” and loads the necessary procedure is called over lay super visor or simply flipper. This over all scheme is called Dynamic loading or Load on Call.
- Advantage--no over head is in corrected unless the procedure to be called or referenced is actually used. Also the system can be dynamically re – configured.
- The major draw back is occurred due to the fact that we here postponed most of the binding process until execution time.

Types of loaders

Relocation Loader

- Another function commonly performed by a loader is that of program re – location.
- Relocation is simply moving a program from one area to another in the storage.
- It referred to adjustment of address field and not to movement of a program.
- The task of relocation is to add some constant value to each relative address in the segment the part of a loader which performed relocation is called re – location loader.

Linkers

- A linker is a program in a system, also known as a link editor and binder, which combines object modules into a single object file.
- Generally, it is a program that performs the process of linking; it takes one or multiple object files, which are generated by compiler. And, then combines these files into an executable files.
- Linking is a process that helps to gather and maintain a different piece of code into an executable file or single file.
- With the help of a linker, a specific module is also linked into the system library.

Linkers

- The primary function of the linker is to take objects from the assembler as input and create an executable file as output for the loader, as it helps to break down a large problem into a small module that simplifies the programming task.
- Usually, computer programs are made up of various modules in which all being a compiled computer programs and span separate object files.
- The whole program refers to these different compiled modules with the help of using symbols.
- These separate files are combined by linker into a single executable file. The source code is converted into machine code, and the linking is performed at the last step while compiling the program.

Linkers

- Source code -> compiler -> Assembler -> Object code -> Linker -> Exécutable file -> Loader
- The objects can be collected by linker from a library or runtime library.
- Most of the linker only consists of files in the output that are referenced by other libraries or object files, and they do not include the whole library.
- The process of library linking requires additional modules to be linked with some referenced modules; thus, it may be an iterative process.
- Generally, one or more than one system libraries are linked by default, and libraries are available for different purposes.

Types of linkers

- Static linking is a kind of linking that is performed during the compilation of a source program in which linking is performed before the execution of the file or object.
- The linker produces a result at the time of copying all library routines into the executable image, which is known as static linking.
- It may need more memory storage and disk space. However, when it runs on the system, it does not need the presence of the library that makes it more portable.
- It generates a fully linked object file that would be able to load and run and takes a collection of the Relocatable object file and command-line argument.

Types of linkers

Two major tasks are performed by the static linker,

1)**Symbol resolution:** In this, each symbol has a predefined task, and it associates each symbol exactly with one symbol definition from which they belong to.

2) **Relocation**: Its function is to modify symbol references to the relocated memory location and relocate the code and data section.

Types of linkers

- Dynamic linking: is performed at the run time, in which multiple programs can share a single copy of the library.
- It means, each module having the same object can share information of an object with other modules rather than linking the same object repeatedly into the library.
- These dynamic link libraries are loaded at the time a program is executed; then, it performs a final linking.

Types of linkers

- Although it needs less memory space, there are more chances of error and failure chances.
- In the linking, the needed shared library is held in virtual memory that helps to save random access memory.
- This linking fix the address at run time; also, it allows the users to reposition the code in order to smooth running of code.
- However, it cannot be relocatable all the code.

Types of linkers

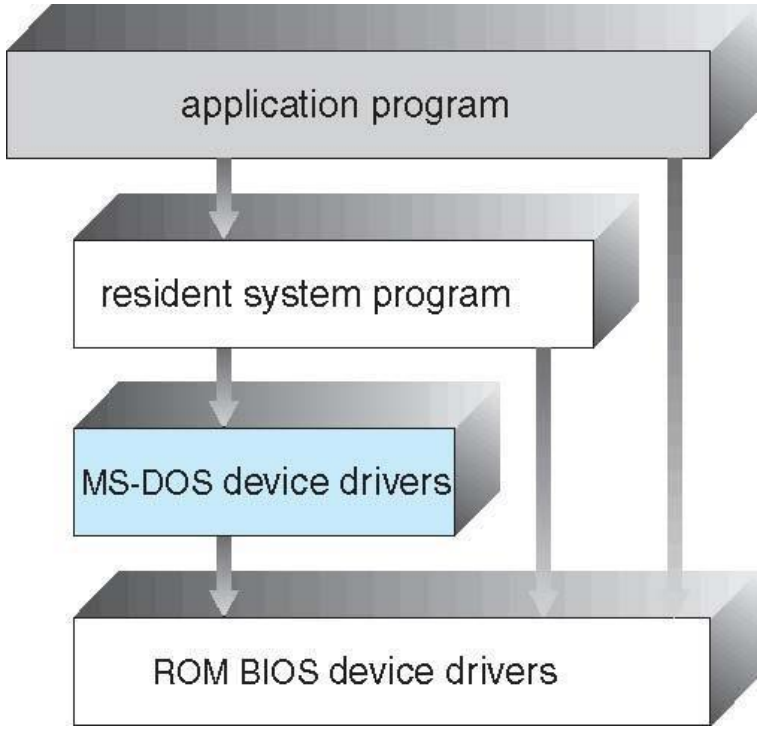
There are two benefits of using the dynamic linking
1) the often-used libraries do not need to store in every single executable file; they only need to store in only one location that helps to save memory and disk space.
2) In the library function, if a bug is corrected with the help of replacing the library, all problems using it dynamically, after restarting them will get benefit from the correction. Otherwise, programs would have to be re-linked first if they include this function by static linking.

Operating System Structure

- OS is an huge collection of programs
- So it is very complex to design, to make functioning, to organize, to manage such a large set of programs
- So, it is better to partition the task into small components rather than having one monolithic system
- Each of these modules should be well-defined portion of the system with carefully defined inputs, outputs & functions
- Various ways to structure ones
  - Simple structure – MS-DOS
  - More complex -- UNIX
  - Layered – an abstrcation
  - Microkernel -Mach

Simple Structure -- MS-DOS

- MS-DOS – written to provide the most functionality in the least space
  - Not divided into modules
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated

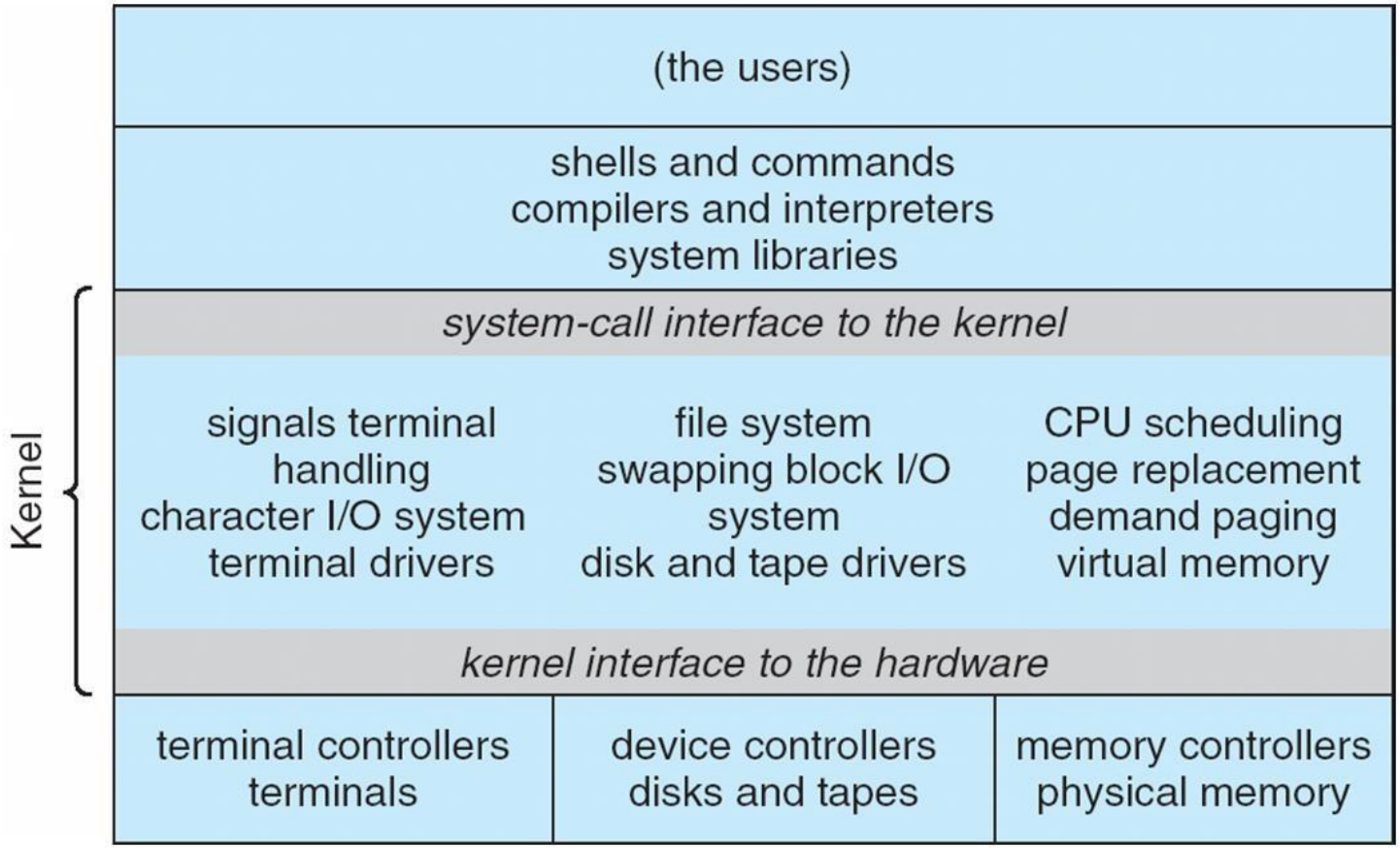


Non Simple Structure -- UNIX

- 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
    - Consists of everything below the system-call interface and above the physical hardware
    - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level

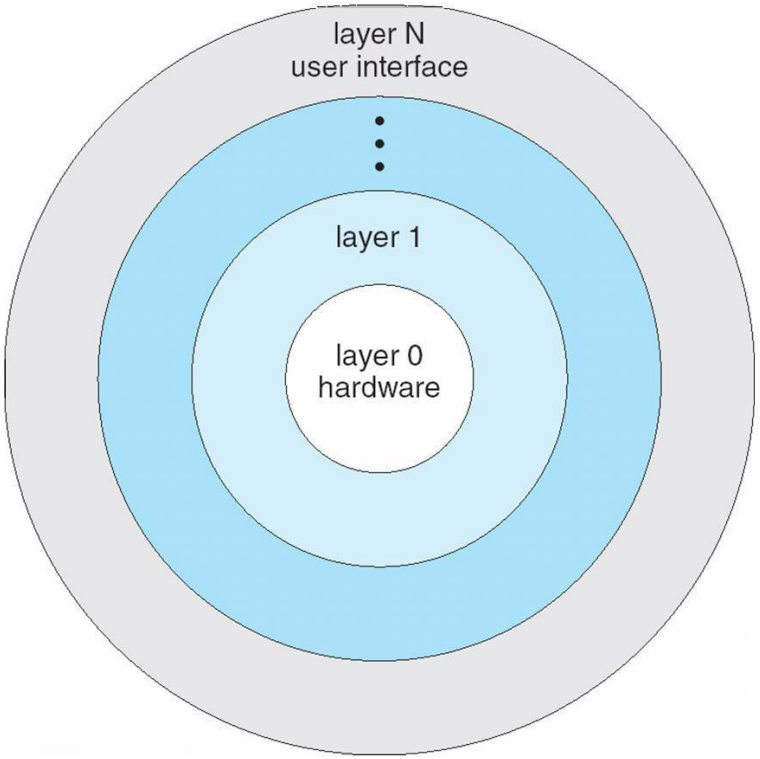
Traditional UNIX System Structure

Beyond simple but not fully layered



Layered Approach

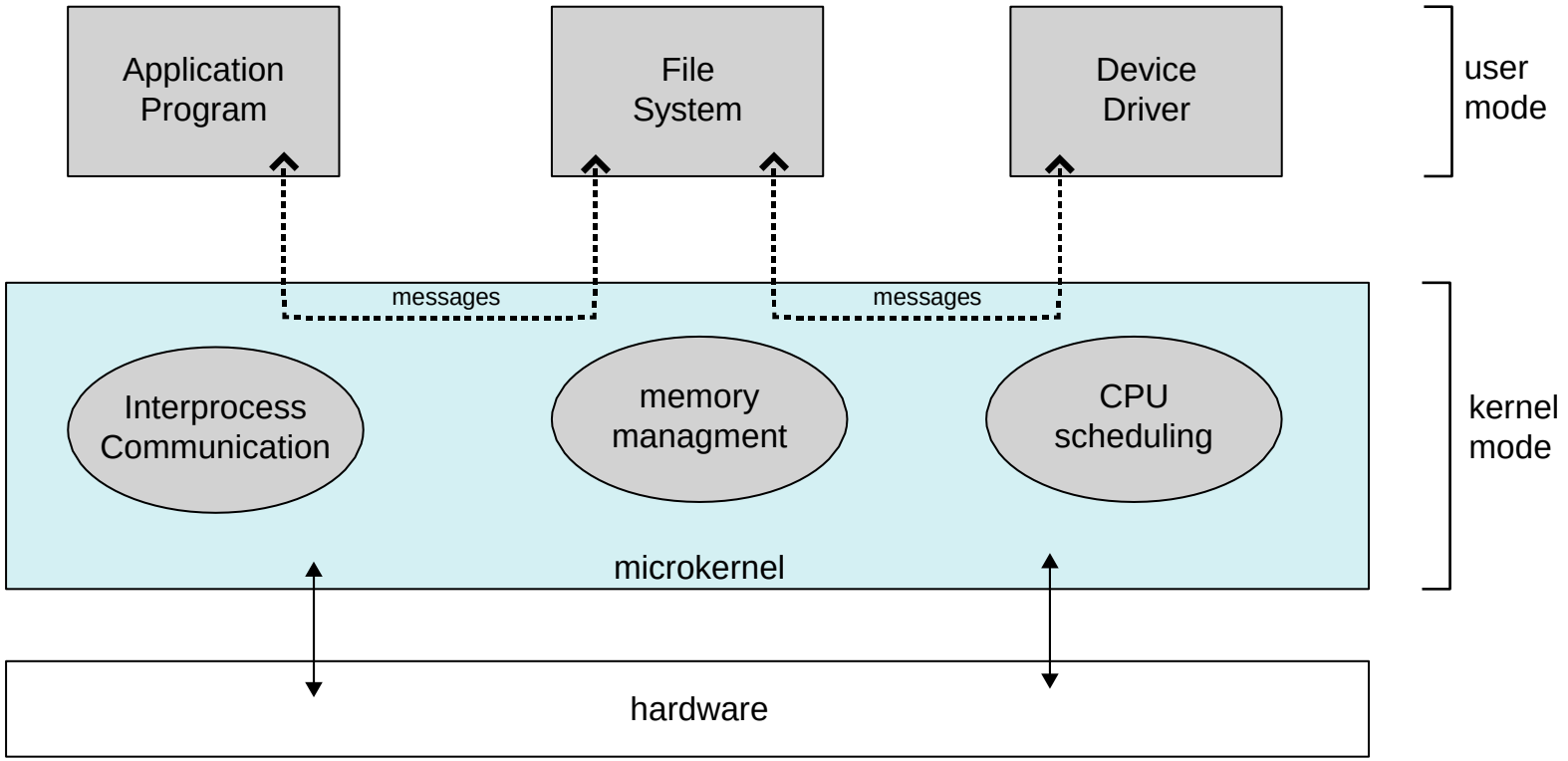
- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers



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**
- Benefits:
  - Easier to extend a microkernel
  - Easier to port the operating system to new architectures
  - More reliable (less code is running in kernel mode)
  - More secure
- Detriments:
  - Performance overhead of user space to kernel space communication

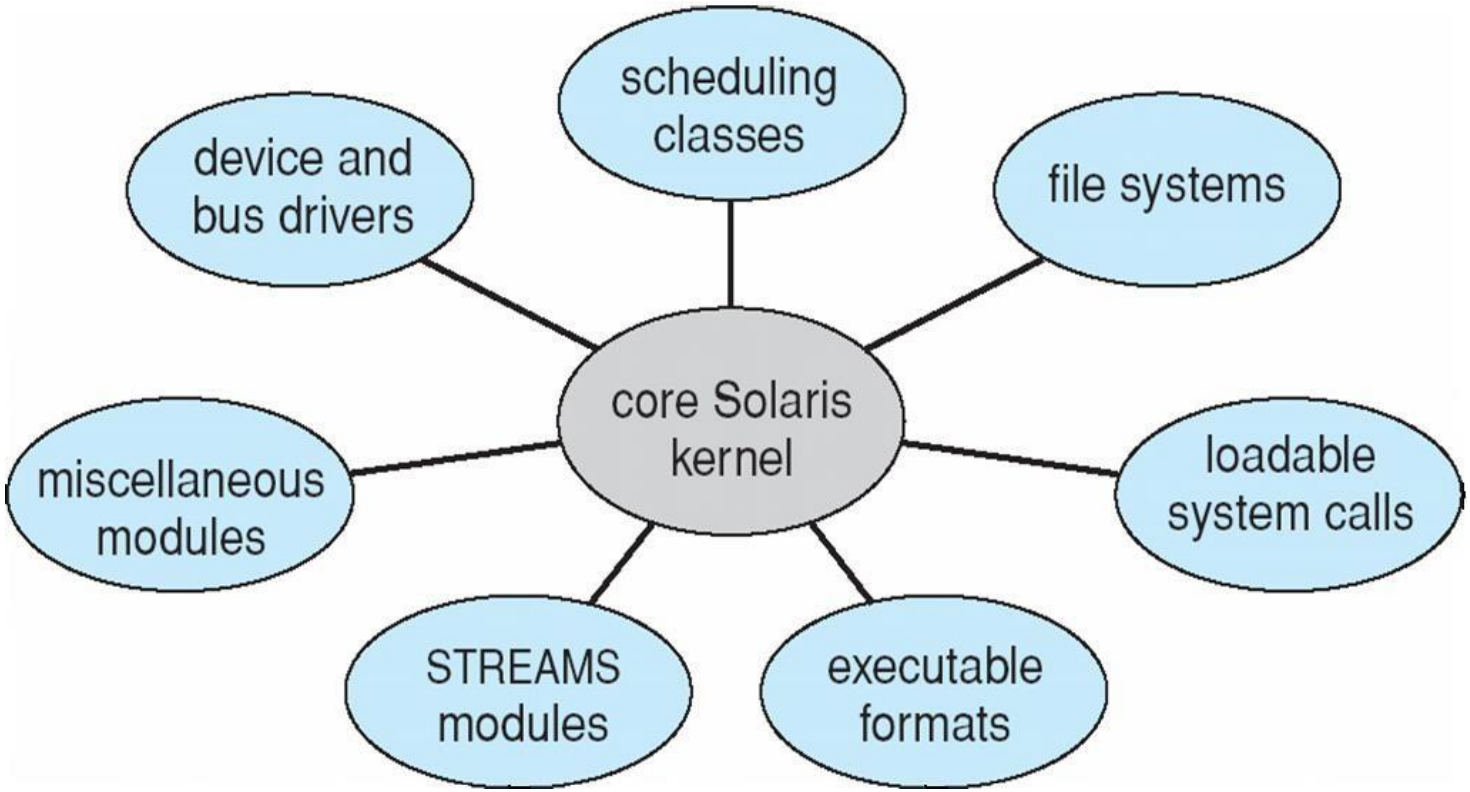
Microkernel System Structure



Modules

- Many modern operating systems implement **loadable kernel modules**
  - Uses object-oriented approach
  - Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel
- Overall, similar to layers but with more flexible
  - Linux, Solaris, etc

Solaris Modular Approach



OS Design & Implementation

- Some of the problems/issues/challenges faced by the designers of OS are
  - Setting the goals
  - Generality
  - Portable
  - Backward Compatibility
- **Design Goals**
  - Very first problem in designing an OS is to define goals & its specifications
  - Due to the modern technology, there is a rapid growth or revolutionary in both hardware & software.
  - So, what we are using today that will be outdated in the next decade. So our OS also should change/update

Design Goals contd…..

- Design of OS will be affected by the choice of hardware & the type of system (batch, time-shared, single user, multi-user, distributed, real-time or general purpose)
- Requirements of an OS can be divided into 2 basic groups 1) **user goals** and 2) **system goals**
- Always user expects, convenient use of the system, easy to learn, reliability, safety & fastness
- In the same way, people who are designing, creating, maintaining & operating the system also expects some of the things from the OS such as easy to design, easy to implement, easy to maintain, flexible, reliable, error free and efficiency

Other goals

- OS designers really do not have a good idea of how their systems will be used, so designers should consider generality
- Modern OS's are generally designed to be portable, i.e., they have to run on multiple hardware platforms. This is another major goal while designing an OS
- Final one is the frequent need to be backward compatibility with some previous OS

Mechanisms & policies

- Mechanisms determines how to do something
- policies determines what will be done
- While designing & implementing an OS, one important principle is the separation of policy from mechanism
- separation of policy from mechanism is important for flexibility
- Policies are likely to change across place or over time

Mechanisms & policies

- Allowing programs to be loaded into the kernel. The mechanism concerns how they are inserted, how they are linked, what system calls they can make, what system calls can be made on them
- The policy is determining who is allowed to load a programs into the kernel & which programs
- As mechanisms are changing, policies can also change.
- But, when policies are changing no need to change mechanisms
- But, change in the policies should not affect the mechanism

Implementation

- Once an OS is designed, it must be implemented.
- Traditionally, OS's have been written in assembly language. Now, they are most commonly written in C or C++
- Advantages of using higher level language are
  - Application programs are also written in high level languages, so OS can be written in more compact
  - Easy to debug
  - Easy to understand
  - Portability
  - Improvements in compiler technology will improve the generated code for the entire OS by simple recompilation
- Only disadvantage of implementing OS in higher level language are reduced speed & increased storage requirements.