OPERATING SYSTEM
Chapter-2

CHAPTER:2

OPERATING SYSTEM STRUCTURES

CONTENTS:

- 2.1 OPERATING SYSTEM SERVICES
- 2.3 SYSTEM CALLS
- 2.4 TYPES OF SYSTEM CALLS
- 2.5 SYSTEM PROGRAMS
- 2.7 OPERATING SYSTEM STRUCTURE

OBJECTIVES:

- To describe the services an operating system provides to users, processes, and other systems
- To discuss the various ways of structuring an operating system
- To explain how operating systems are installed and customized and how they boot

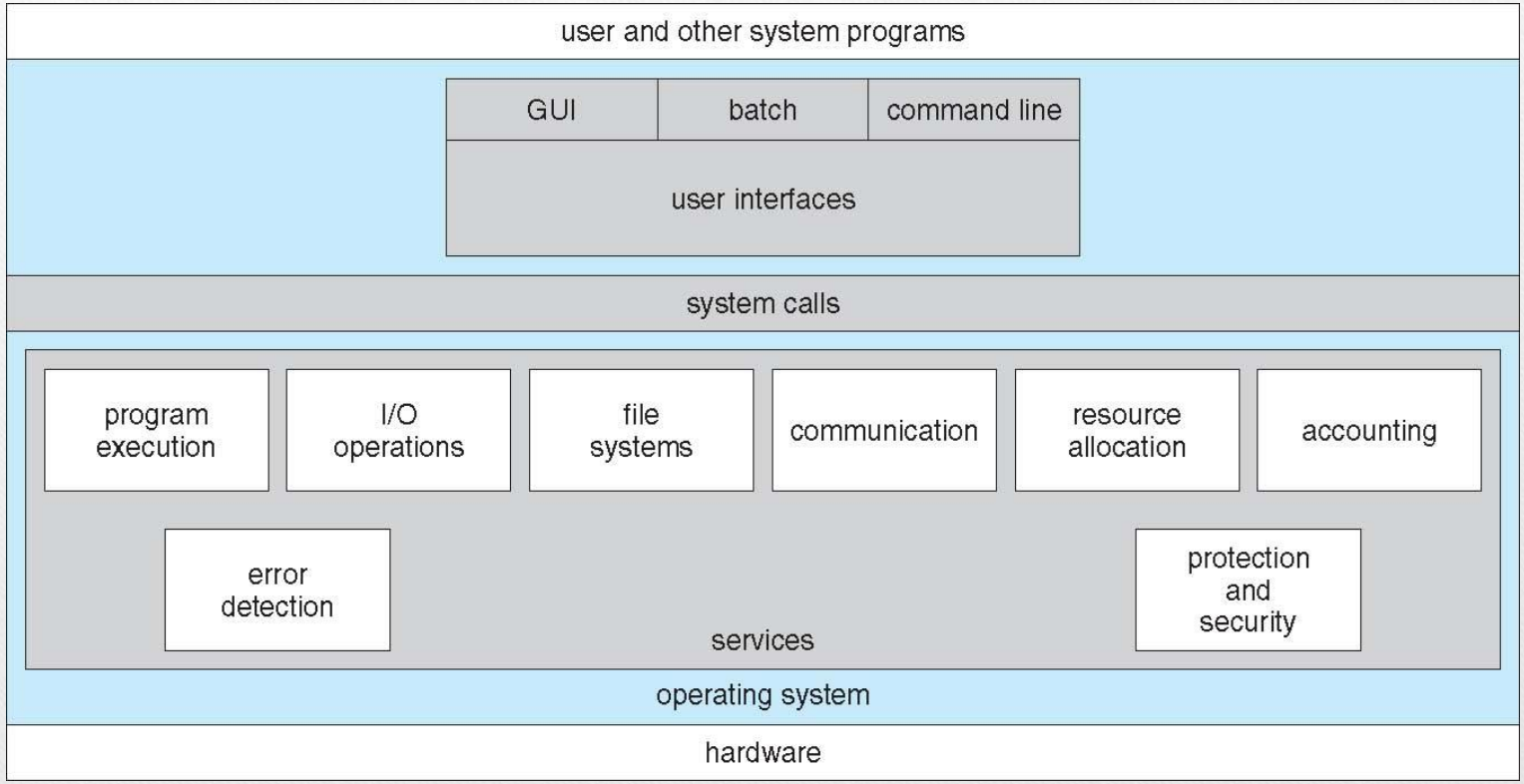
OPERATING SYSTEM SERVICES

- Operating systems provide an environment for execution of programs and services to programs and users
- One set of operating-system services provides functions 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
- **File-system manipulation** - The file system is of particular interest. Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.
- **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)
- **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 can greatly enhance the user's and programmer's abilities to efficiently use the system

- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing
  - **Resource allocation -** When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
    - Many types of resources - Some (such as CPU cycles, main memory, and file storage) may have special allocation code, others (such as I/O devices) may have general request and release code
  - **Accounting -** To keep track of which users use how much and what kinds of computer resources
  - **Protection and security -** The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
    - **Protection** involves ensuring that all access to system resources is controlled
    - **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts
    - If a system is to be protected and secure, precautions must be instituted throughout it. A chain is only as strong as its weakest link.

A View of Operating System Services



User Operating System Interface - CLI

- **Command line interface(CLI)**: Command Line Interface (CLI) or command interpreter that allows user to directly enter command that are to performed by OS from command line . In this process the user has to remember all the commands that he needs for performing certain task.

- **Command line interface(CLI):**
  In some OS the CI is included in the kernel(heart of OS)l whereas in others like Windows XP and UNIX the CI is treated as special program.

➔ On some system with multiple CI to choose from the interpreters are called shells. For example:

- Bourne shell
- C shell
- Bourne-Again shell(BASH)
- Korn shell

User Operating System Interface - GUI

Graphical User Interface(GUI) : The most commonly used UI is the GUI. It is a interface where you have a windows system with a pointing device like your mouse and menus and you can click those menus or provide input using your keyboards. It is the most user friendly user interface that we have.

SYSTEM CALLS

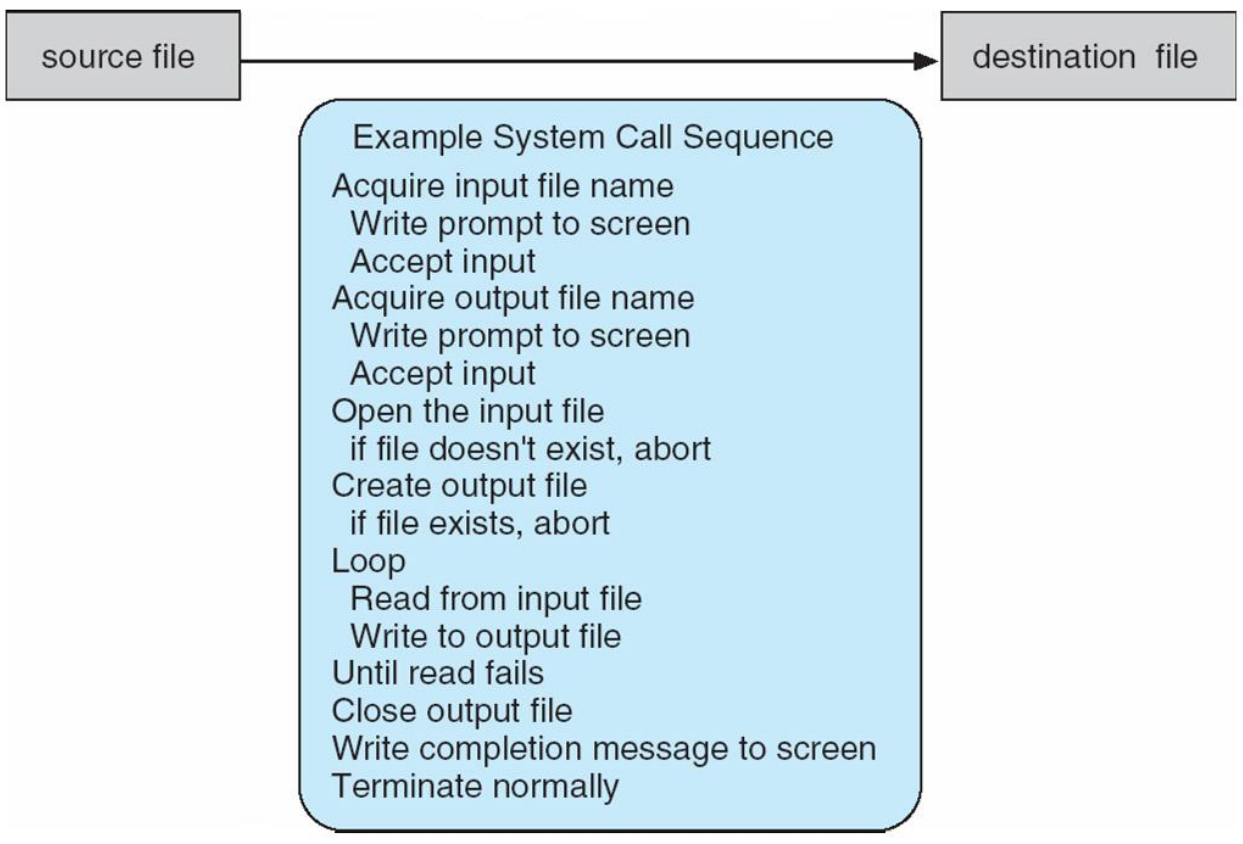
- **System call is the programmatic way in which a computer program request a service from the kernel of the operating system it is executed on.**

- Typically written in a high-level language (C or C++)

- Mostly accessed by programs via a high-level **Application Programming Interface (API)** rather than direct system call use. It provides an interface between a process and operating system to allow user-level processes to request services of the operating system.

Example of System Calls

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



- API Calls: An API is a set of functions or protocols that allow different software applications or components to communicate with each other. When you make an API call, you're typically interacting with higher-level functionality provided by libraries or services.

- API calls are used by applications to interact with other services, libraries, or software, without directly dealing with low-level operations. For example, a web application calling an API to fetch data from a database or interacting with a cloud service.

- Scope: API calls can be made within an application or across different systems via HTTP (e.g., RESTful APIs). Example: A call to a weather service API to get the current weather.

A system call is a low-level function that provides an interface for programs to interact directly with the operating system's kernel.

- System calls are essential for performing tasks that require access to hardware or kernel-level resources, such as reading/writing files, managing processes, or interacting with memory.

- System calls are used for essential OS operations like memory management, process control, and file I/O, which applications cannot access directly.

- Scope: System calls are typically part of the OS kernel and interact with the underlying hardware or system resources. Example: A read() system call to read data from a file in a Unix-based system.

Key Differences:

- Level of Abstraction: API calls are higher-level and may invoke system calls behind the scenes. System calls are low-level and interact directly with the kernel.
- Purpose: API calls are for inter-application or library communication, while system calls handle OS-level interactions.
- Usage: API calls can be made by application-level code, while system calls are typically made by the OS or applications needing direct access to hardware or system resources.

System Call Implementation

The occurrence of an event is usually signaled by an interrupt from either the hardware or the software.

Hardware may trigger an interrupt at any time by sending a signal to the CPU, usually by way of the system bus.

Software may trigger an interrupt by executing a special operation called a system call (also called a monitor call).

Interrupts are an important part of a computer architecture.

Each computer design has its own interrupt mechanism, but several functions are common. The interrupt must transfer control to the appropriate interrupt service routine.

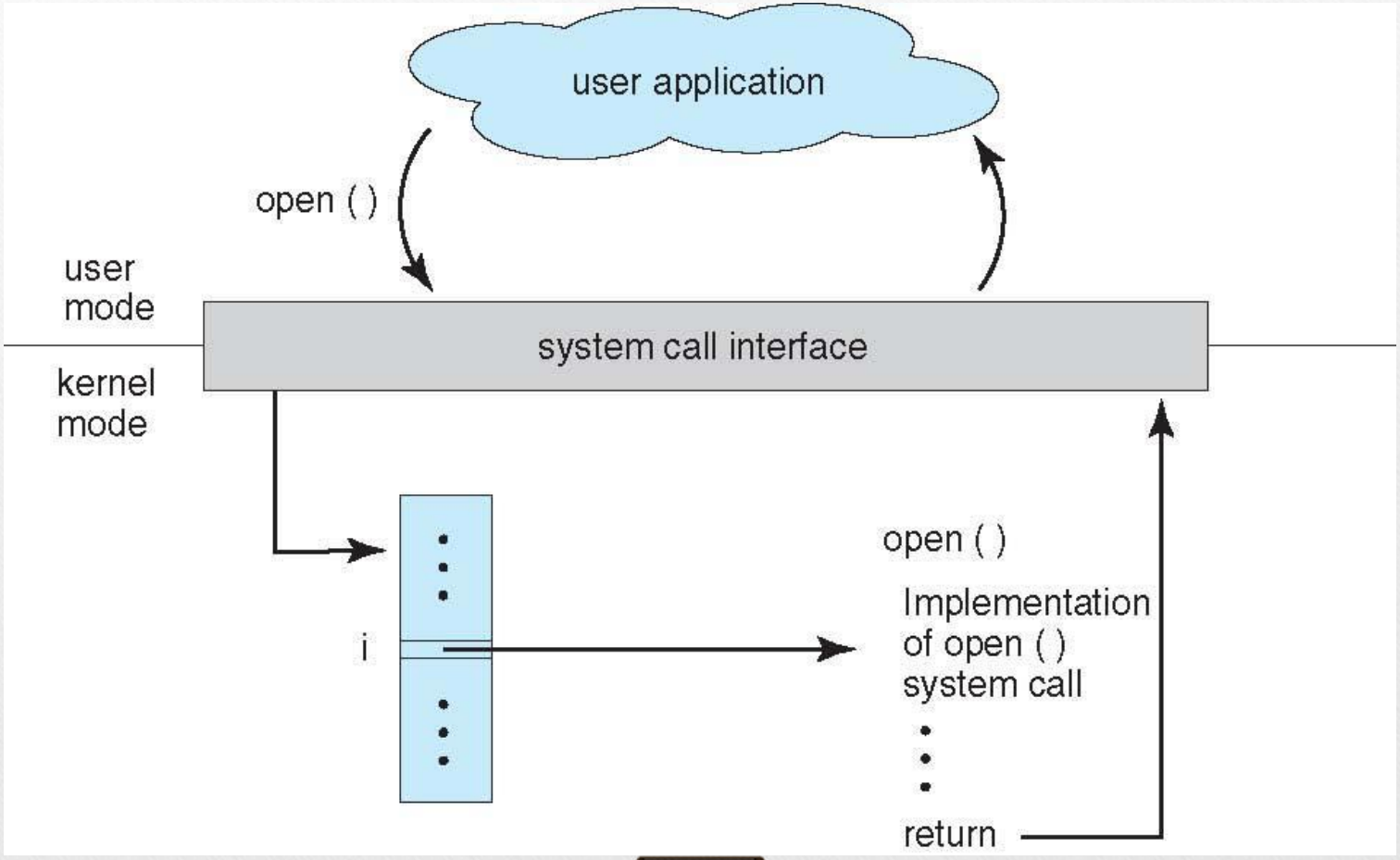
System Call Implementation

The straightforward method for handling this transfer would be to invoke a generic routine to examine the interrupt information. The routine, in turn, would call the interrupt-specific handler.

However, interrupts must be handled quickly. Since only a predefined number of interrupts is possible, a table of pointers to interrupt routines can be used instead to provide the necessary speed. The interrupt routine is called indirectly through the table, with no intermediate routine needed.

Generally, the table of pointers is stored in low memory (the first hundred or so locations). These locations hold the addresses of the interrupt service routines for the various devices. This array, or interrupt vector, of addresses is then indexed by a unique device number, given with the interrupt

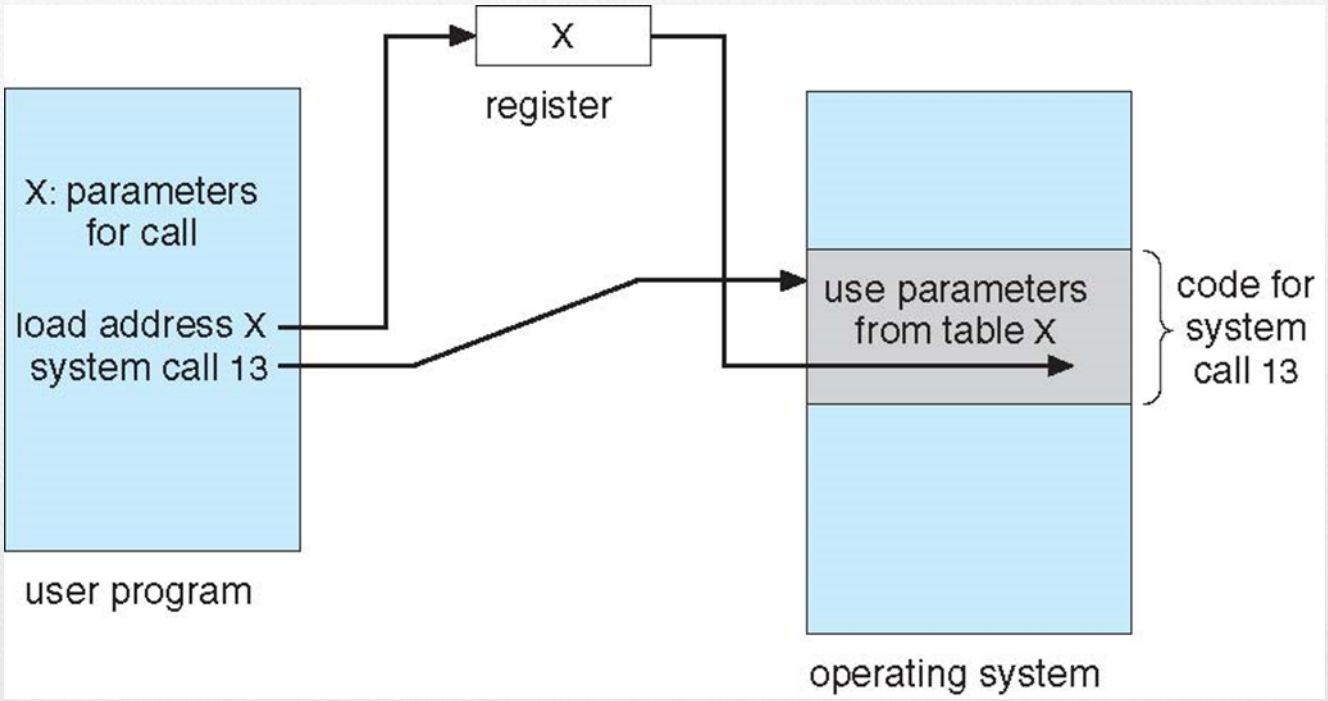
API – System Call – OS Relationship



System Call Parameter Passing

- Often, more information is required than simply identity of desired system call Exact type and amount of information vary according to OS and call
- 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

- **Process control**
- **File management**
- **Device management**
- **Information maintenance**
- **Communication**

- **Process control**

- Process creation and management and Main memory management
  - end, abort
  - load, execute
  - create process, terminate process
  - 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

File management

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

Device management

- Device handling(I/O)
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices

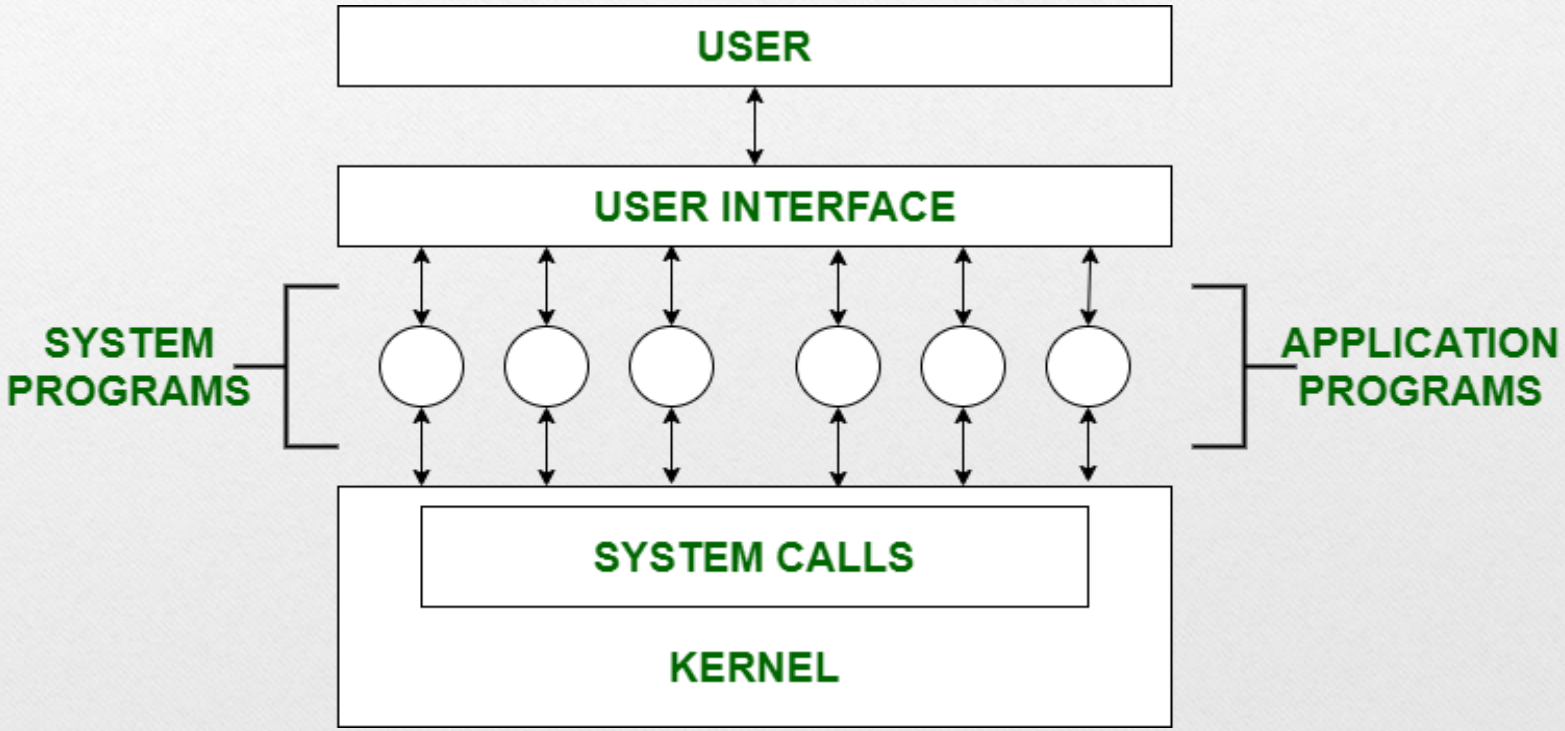
- **Information maintenance**
  - get time or date, set time or date
  - get system data, set system data
  - get and set process, file, or device attributes

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

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

System Programs

- System Programs provide a convenient environment for program development and execution. Some of them are simply user interface to system calls . Others are considerably more complex.



- System programs provide a convenient environment for program development and execution. They can be divided into:
  - File manipulation
  - Status information sometimes stored in a File modification
  - Programming language support
  - Program loading and execution
  - Communications
  - Background services
  - Application programs
- Most users' view of the operation system is defined by system programs, not the actual system calls

- Provide a convenient environment for program development and execution
  - Some of them are simply user interfaces to system calls; others are considerably more complex
- **File management** - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
- **Status information**
  - Some ask the system for info - date, time, amount of available memory, disk space, number of users
  - Others provide detailed performance, logging, and debugging information
  - Typically, these programs format and print the output to the terminal or other output devices
  - Some systems implement a **registry** - used to store and retrieve configuration information

- **File modification**
  - Text editors to create and modify files
  - Special commands to search contents of files or perform transformations of the text
- **Programming-language support** - Compilers, assemblers, debuggers and interpreters sometimes provided
- **Program loading and execution**- Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
- **Communications** - Provide the mechanism for creating virtual connections among processes, users, and computer systems
  - Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

- Some examples of system programs in O.S. are –
- Windows 10
- Mac OS X
- Ubuntu
- Linux
- Unix
- Android
- Anti-virus
- Disk formatting
- Computer language translators

Operating System Structure

- The OS offers a virtual machine to the user by creating a user friendly environment in which other system or application programs can run. The necessary resources such as CPU, memory, I/O devices are provided to the programs by OS. Additionally the OS has to provide a large number of services like creation and maintenance of files and directories.

- It is therefore not surprising that structure of modern OS is quite complex. In fact it is a very large software consisting of many components. How these components are organized depends on the underlying OS architecture model.

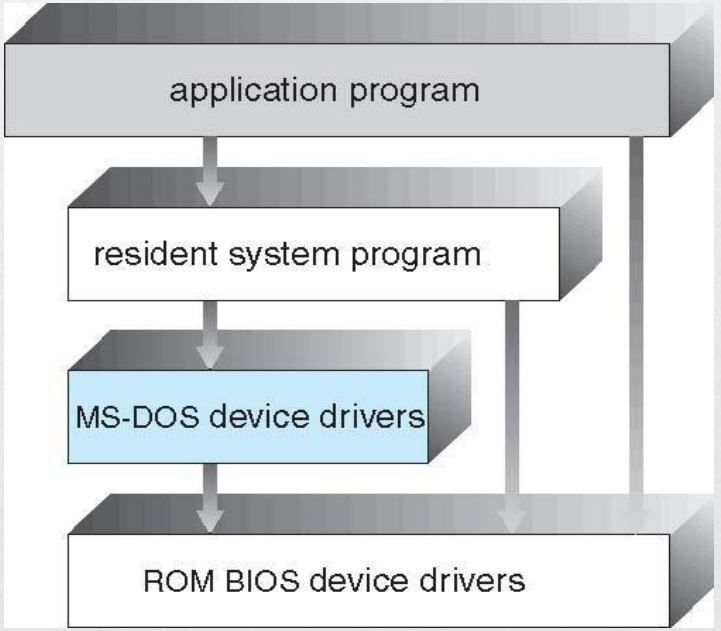
- Operating system can be implemented with the help of various structures. The structure of the OS depends mainly on how the various common components of the operating system are interconnected and melded into the kernel.

Simple structure

- Such operating systems do not have well defined structure and are small, simple and limited systems.
- The interfaces and levels of functionality are not well separated
- In MS-DOS application programs are able to access the basic I/O routines. These types of operating system cause the entire system to crash if one of the user programs fails.

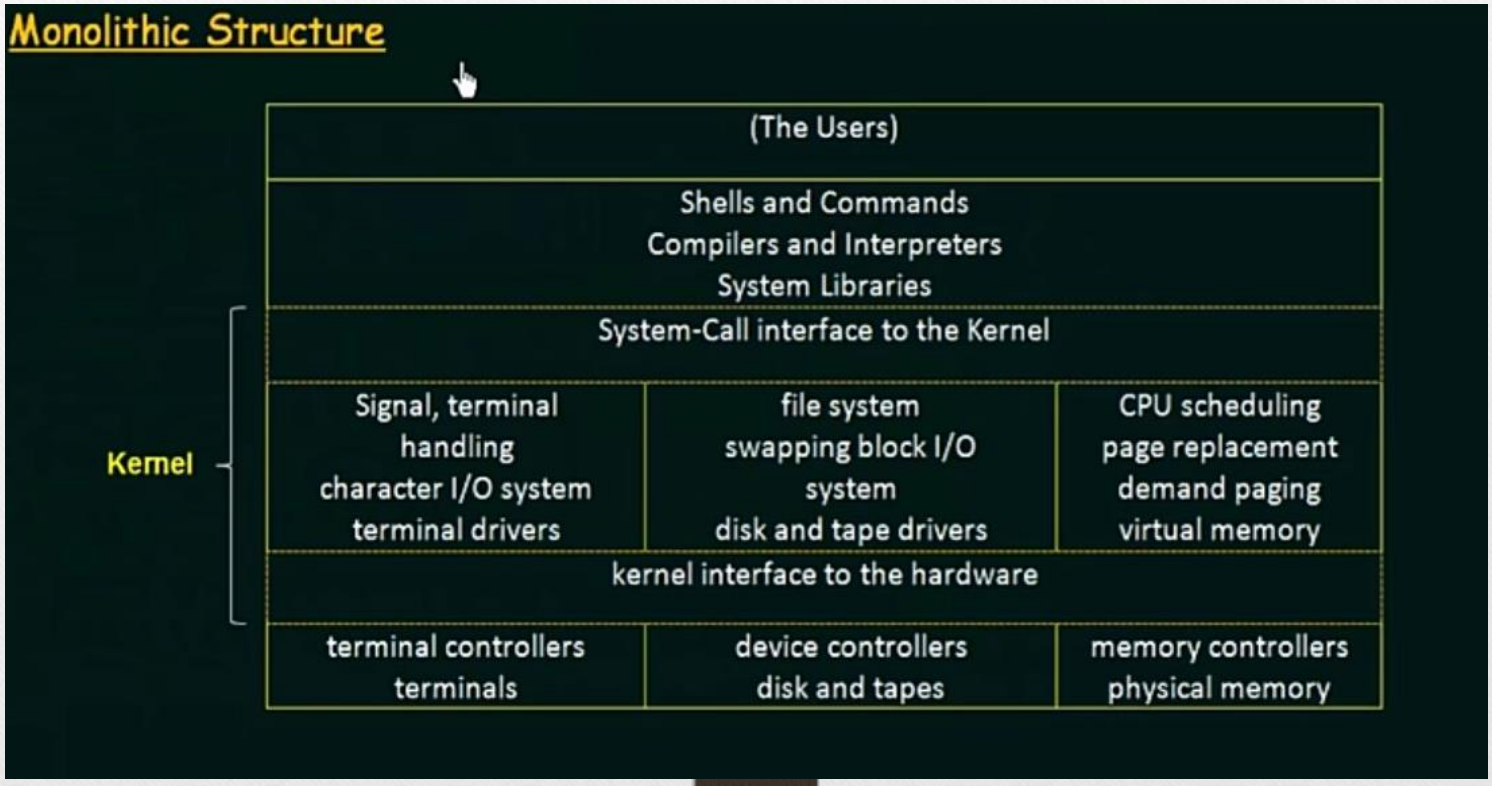
example:

- I.e. 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



Monolithic System Architecture

The entire operating system works in the kernel space in the monolithic system. This increases the size of the kernel as well as the operating system. This is different than the microkernel system where the minimum software that is required to correctly implement an operating system is kept in the kernel.



Monolithic System Architecture

The kernel provides various services such as memory management, file management, process scheduling etc. using function calls. This makes the execution of the operating system quite fast as the services are implemented under the same address space.

Advantages :

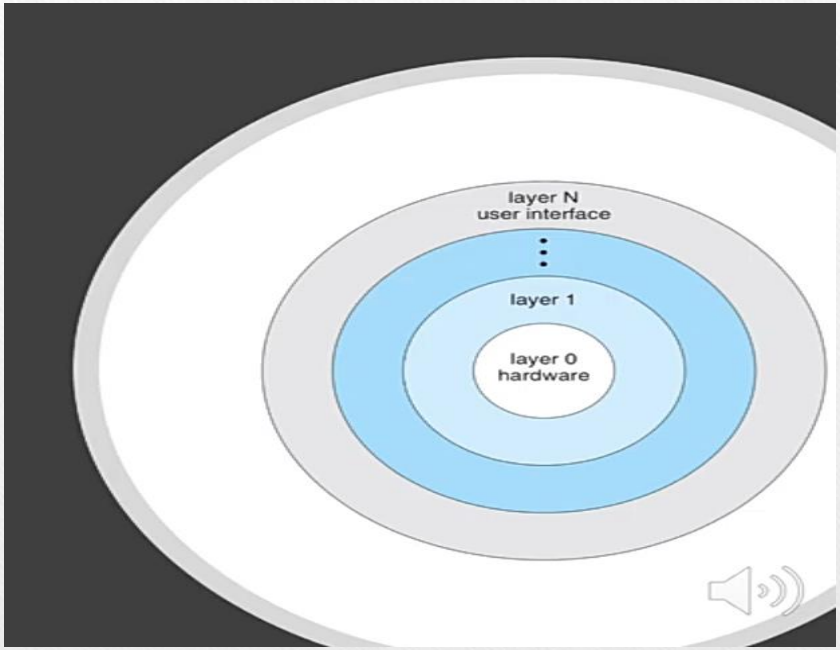
- The execution of the monolithic kernel is quite fast as the services such as memory management, file management, process scheduling etc.are implemented under the same address space.
- A process runs completely in a single address space in the monolithic kernel.
- The monolithic kernel is a static single binary file.

Disadvantages :

- If any service fails in the monolithic kernel, it leads to the failure of the entire system.
- To add any new service, the entire operating system needs to be modified by the user.

Layered structure

- Another approach is to break the OS into a number of smaller layers, each of which rests on the layer below it, and relies solely on the services provided by the next lower layer.
- The bottom layer (layer 0) is the hardware and the topmost layer (layer N) is the user interface.

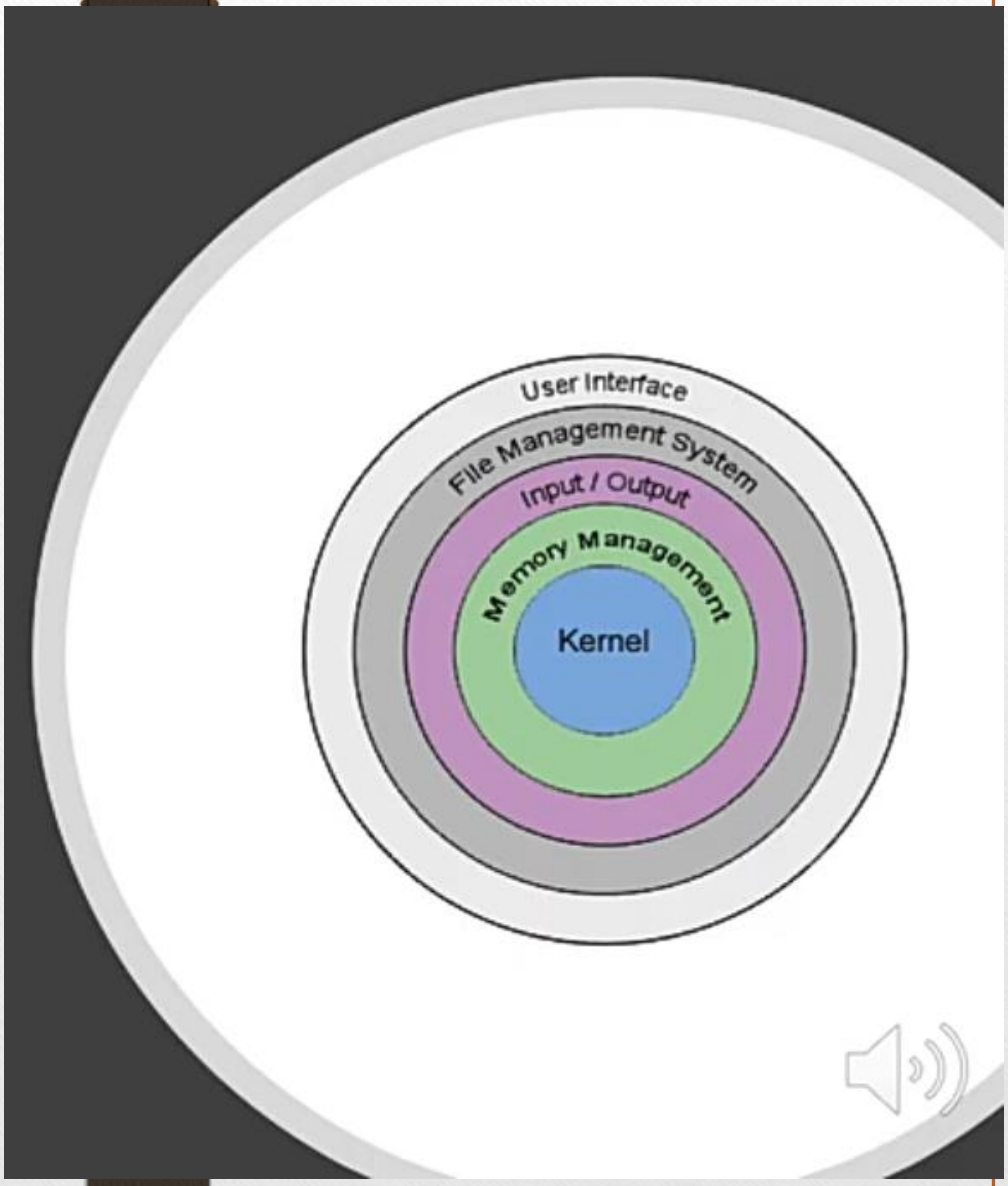


- These layers are so designed that each layer uses the functions of the lower level layers only. This simplifies the debugging process as if lower level layers are debugged and an error occurs during debugging then the error must be on that layer only as the lower level layers have already been debugged.

Disadvantages:

- The problem is deciding what order in which to place the layers, as no layer can call upon the services of any higher layer.

- Layered approaches can also be less efficient, as a request for service from a higher layer has to filter through all lower layers before it reaches the HW, possibly with significant processing at each step.



Layered structure

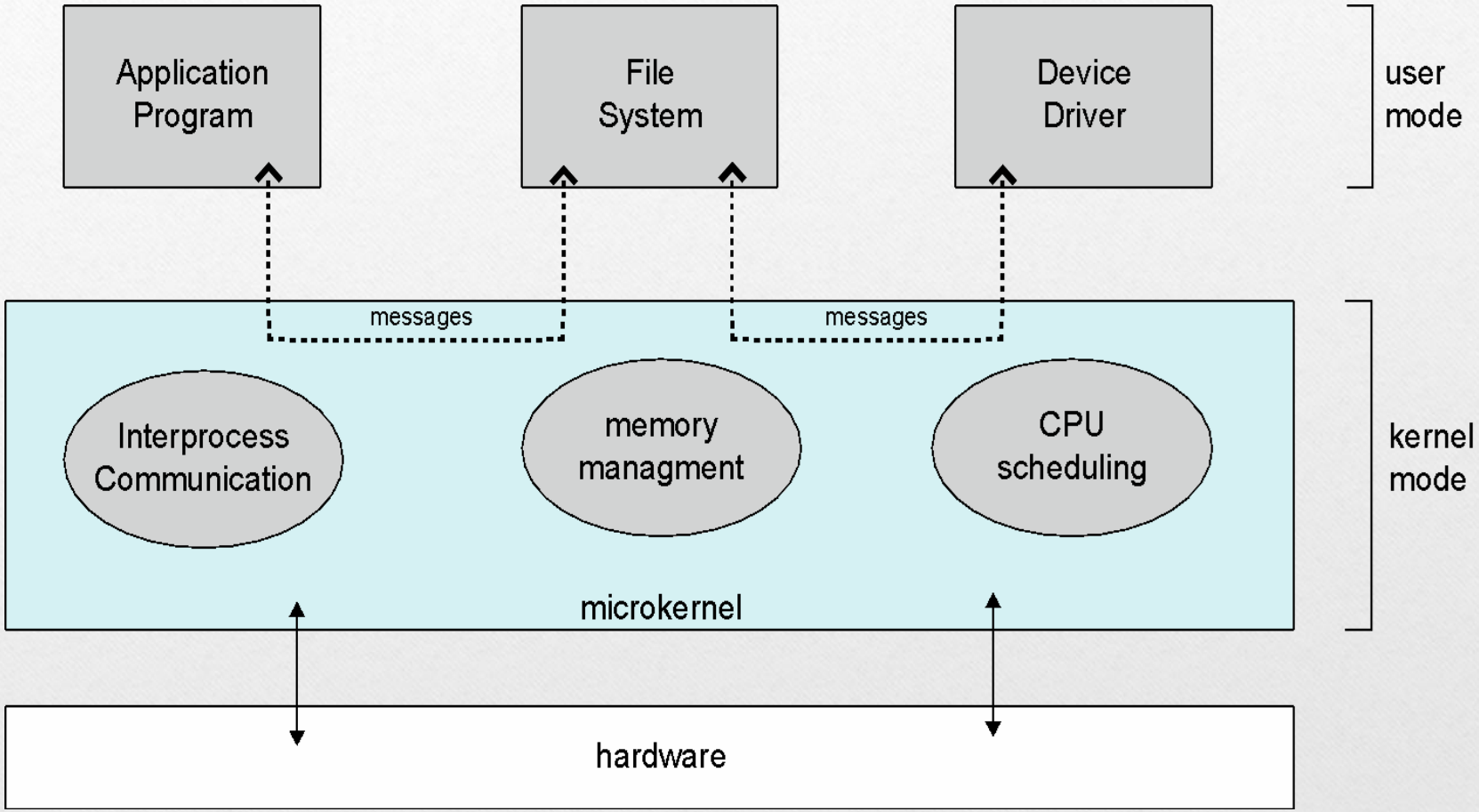
Advantages:

- **Easy debugging and system verification.**
- **Allows good maintenance**

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

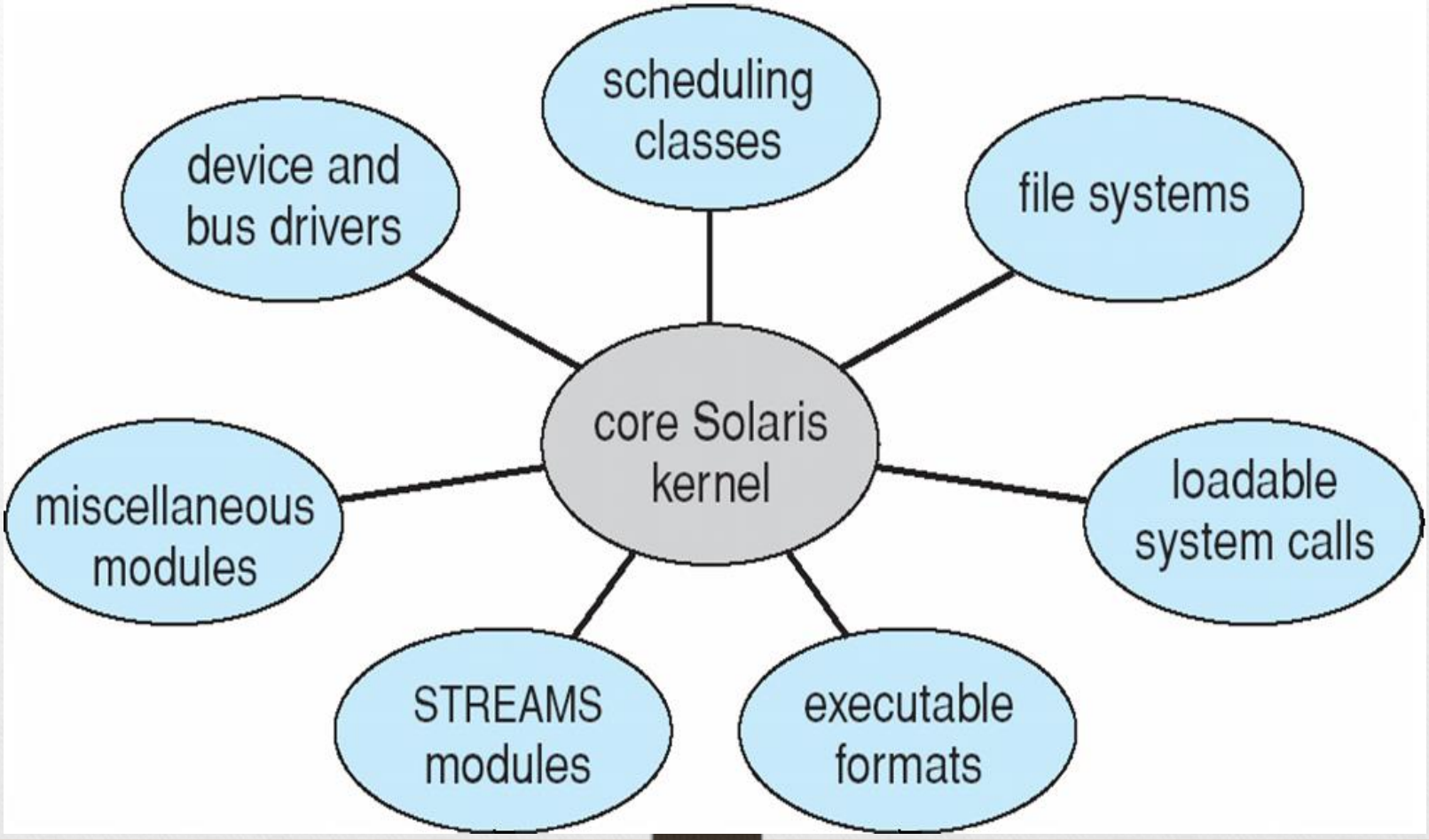
Microkernel System Architecture



MODULES

- Most 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

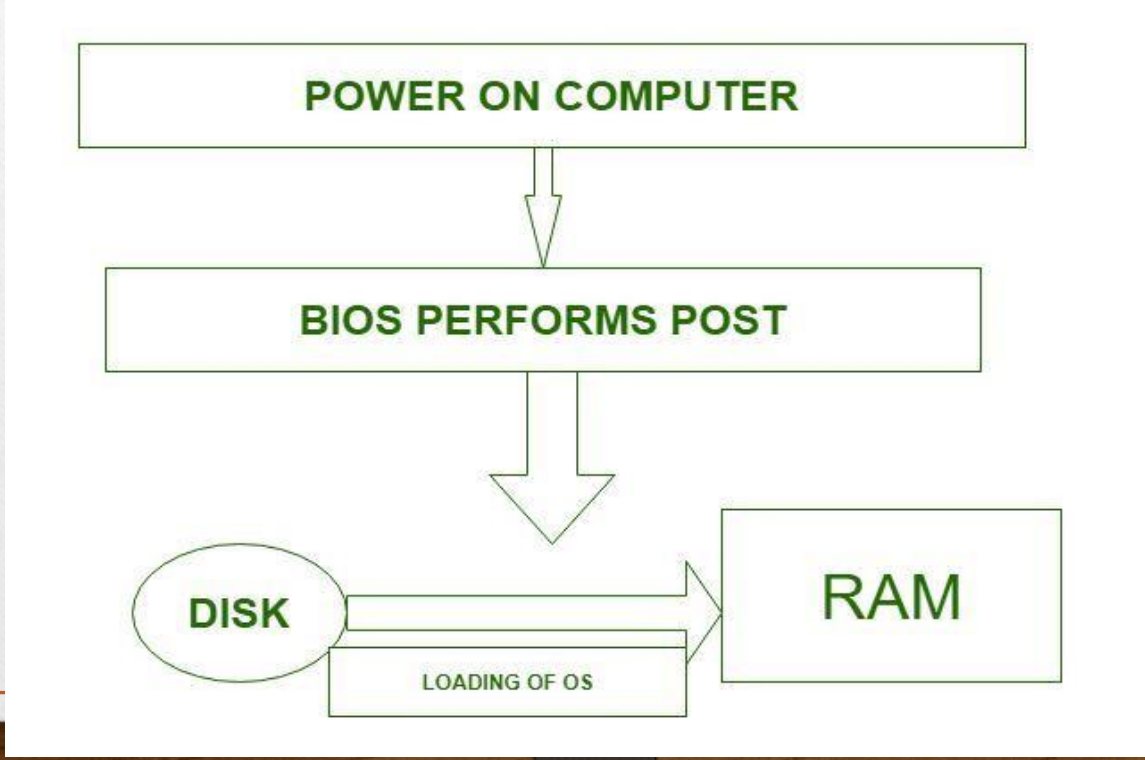
Solaris Modular Approach



System Boot

- Every time the CS is switched on, a copy of an essential part of OS is brought into the main memory from the hard disk. This essential part is called as monitor program. The monitor starts executing. **Hence the process of bringing the monitor program into the main memory and executing is known as booting.**

Steps Involved In System Boot Process:



System Boot Process

Booting the system is done by loading the kernel into main memory, and starting its execution.

- A small program called the BIOS is permanently stored in the ROM, capable of performing input and output operations on the I/O devices .
- When the computer is switched on, the CPU generates the address of the memory locations where the BIOS is stored(in the ROM) and the BIOS starts executing.
- The BIOS conducts a series of self-diagnostic tests called Power On Self Test(POST). These built-in tests verify whether the components connected to the CS are correctly configured and whether they are operational.

- After the POST, the BIOS looks for the monitor program(also called boot program) of the OS, which resides on a particular location of the disk(sector on which boot program is located is called boot sector)

- The BIOS picks up a copy of boot program from the boot sector of the disk and loads it into RAM. Thereafter, the BIOS transfers control to the boot program, and execution begins.

- The boot program takes control of CS and provides a user interface and on user's request it loads the application programs and required components of OS an the RAM on need-felt basis.