Introduction to Operating System

Introduction, Components, Services, Types of OS

&

Architecture of OS

Introduction to Operating System

- Operating system − a program that acts as an **intermediary** between a user of a computer and the computer hardware.

- An operating system is **software** that **manages** a computer's hardware.
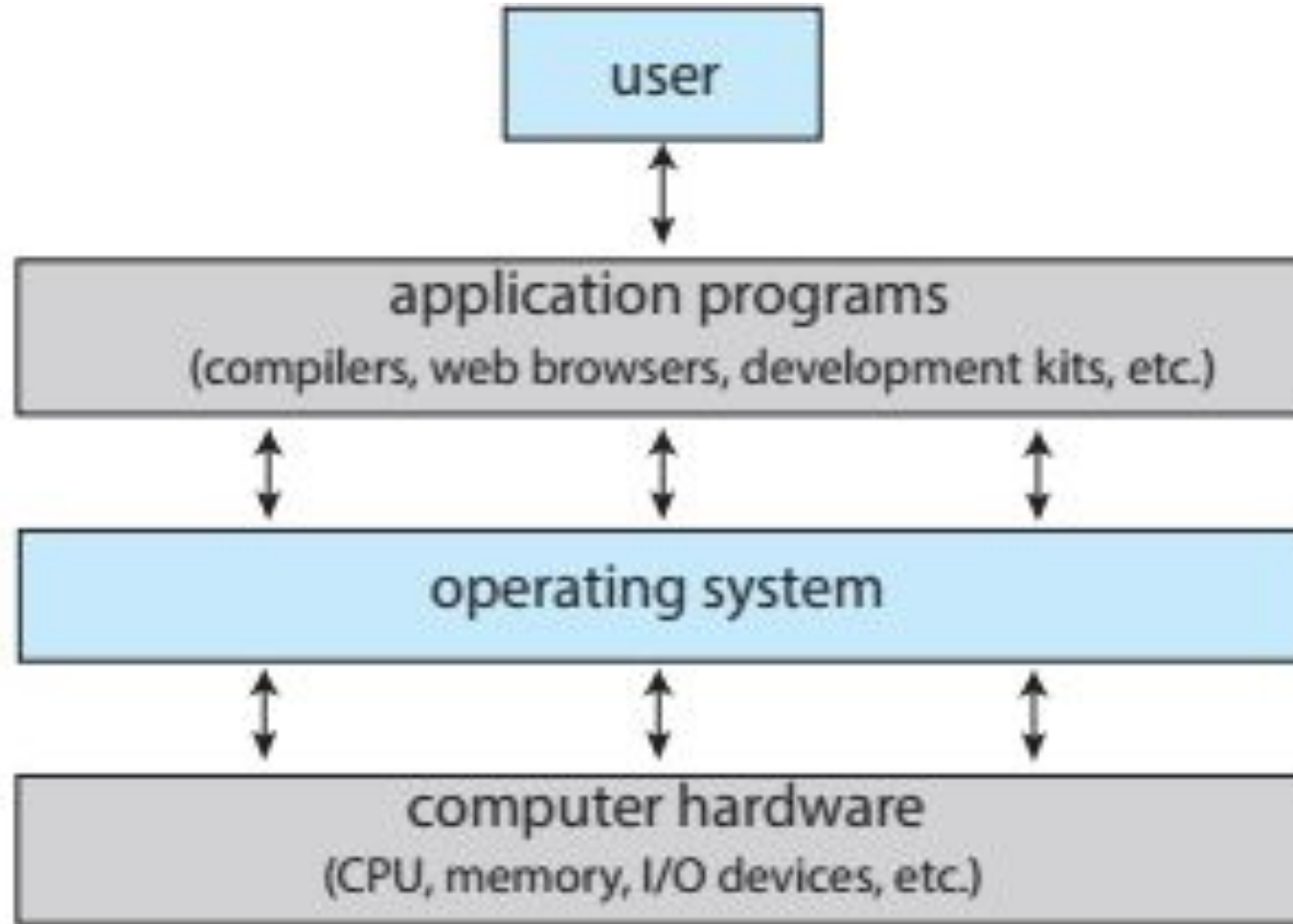
Operating system goals:

- Execute user programs and make solving user problems easier.

- Make the computer system convenient to use.

- Use the computer hardware in an efficient manner.

Introduction to Operating System

Computer System Components:

- Hardware – provides basic computing resources

   (CPU, memory, I/O devices).

- Operating system – controls and coordinates the use of the hardware among the various application programs for the various users.

- 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).

- Users (people, machines, other computers).

✔ A fundamental responsibility of an operating system is to allocate these resources (CPU, memory, I/O devices, and storage) to programs.

Computer System

Abstract view of the components of a computer system.
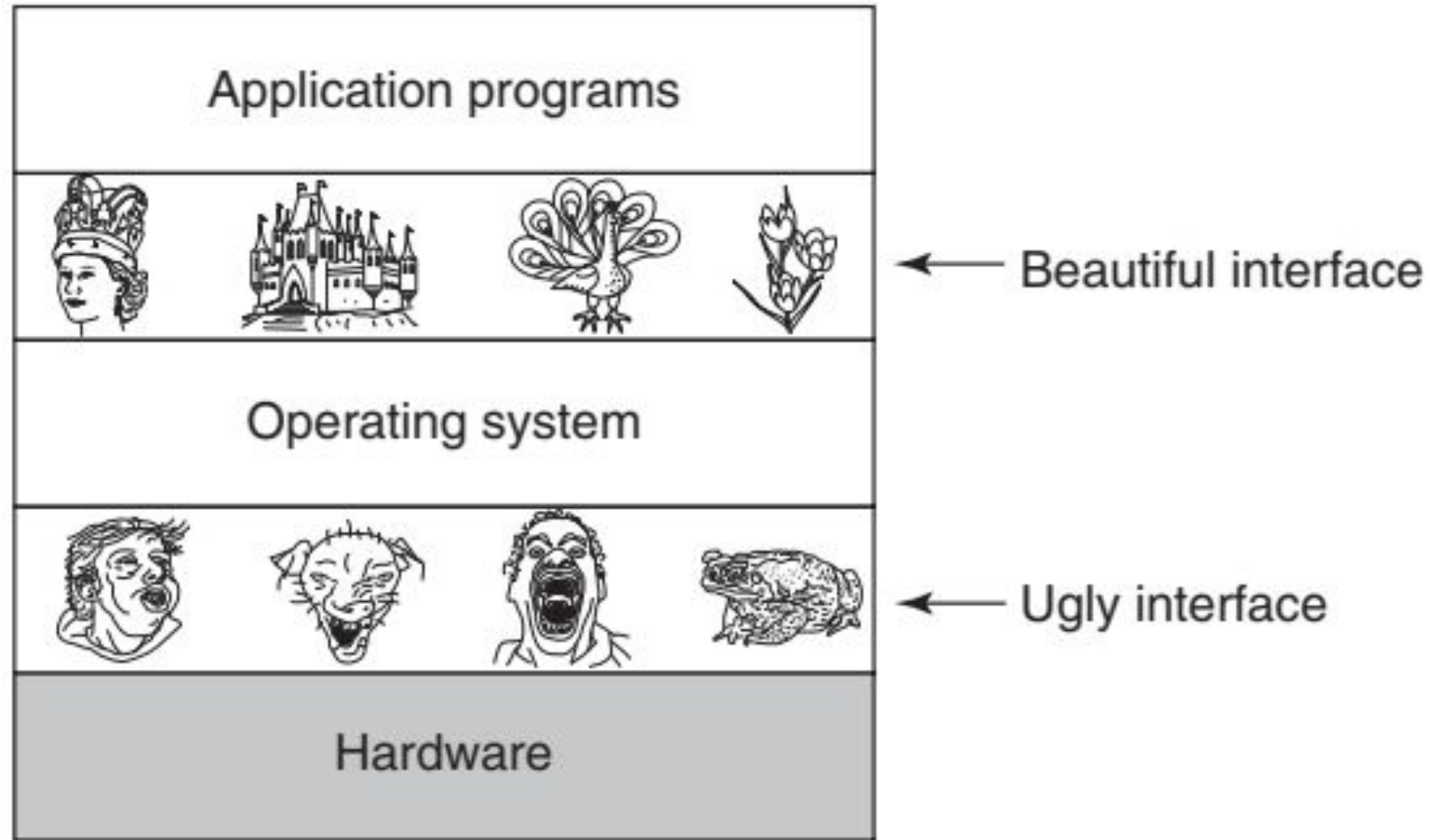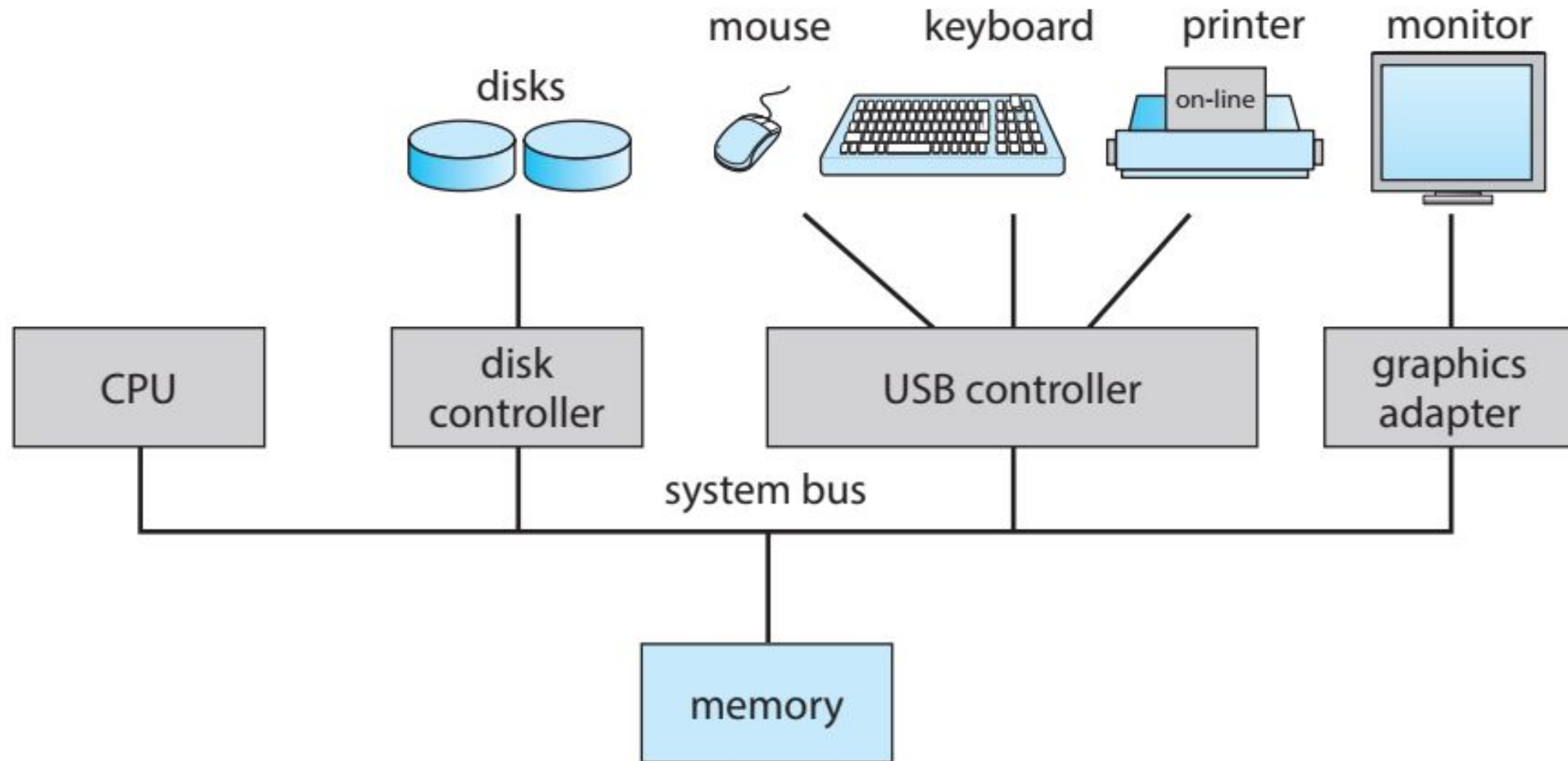
Computer System

Fig: Operating systems turn ugly hardware into beautiful abstractions.

One of the major tasks of the operating system is to hide the hardware & ugly interfaces, and present programs with nice, clean, elegant, consistent.

Computer System

A typical PC computer system

A resource manager

- The primary task of an operating system is to keep track of which programs are using which resource, to grant resource requests, to account for usage, and to mediate conflicting requests from different programs and users.

- Resource management includes multiplexing (sharing) resources in two different ways: in time and in space.

- **Time multiplexed**: different programs or users take turns to use it. First one of them gets to use the resource, then another, and so on.

- For example, with only one CPU and multiple programs that want to run on it, the operating system first allocates the CPU to one program, then, after it has run long enough, another program gets to use the CPU, then another, and then eventually the first one again.

- Determining how the resource is time multiplexed—who goes next and for how long—is the task of the operating system.

A resource manager

- **Space multiplexing**: Instead of the customers taking turns, each one gets part of the resource.
- For example, main memory is normally divided up among several running programs, so each one can be resident at the same time.
- Assuming there is enough memory to hold multiple programs, it is more efficient to hold several programs in memory at once rather than give one of them all of it, especially if it only needs a small fraction of the total.
- This raises issues of fairness, protection, and so on, and it is up to the operating system to solve them.
- Another resource that is space multiplexed is the disk. In many systems a single disk can hold files from many users at the same time. Allocating disk space and keeping track of who is using which disk blocks is a typical operating system task.

 Each program gets time with the resource.

 Each program gets space on the resource.

Operating System Components

Most operating systems support the following types of system components:

- Process Management
- Main Memory Management
- I/O System Management
- File Management
- Secondary Management
- Networking
- Protection System
- Command-Interpreter System

Process Management

- A process is **a program in execution**. A process needs certain resources, including **CPU time, memory, files**, and **I/O devices**, to accomplish its task.

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

  - Process creation and deletion.

  - Process suspension and resumption.

  - Provision of mechanisms for:
    - process synchronization
    - process communication

Main-Memory Management

- Memory is a **large array of words** or bytes, each with its own address. It is a repository of quickly accessible data shared by the CPU and I/O devices.

- Main memory is a volatile storage device. It loses its contents in the case of system failure.

- The operating system is responsible for the following activities in connection with memory management:
  - Keep track of which parts of memory are currently being used and by whom.
  - Decide which processes to load when memory space becomes available.
  - Allocate and de-allocate memory space as needed.

Secondary-Storage Management

- Since main memory (primary storage) is volatile and too small to accommodate all data and programs permanently, the computer system must provide secondary storage to back up main memory.

- Most modern computer systems use disks as the principle on-line storage medium, for both programs and data.

- The operating system is responsible for the following activities in connection with disk management:
  - Free-space management
  - Storage allocation
  - Disk scheduling

I/O System Management

- The I/O system consists of:
  - A buffer-caching system
  - A general device-driver interface
  - Drivers for specific hardware devices

- The operating system is responsible for the following activities in connection with I/O management:
  - It keeps track of data, status, location, uses, etc.
  - It monitors the status of every device, including printers, storage drivers, and other devices.
  - It allocates and effectively deallocates the device.

File Management

- A file is a collection of related information defined by its creator. Commonly, files represent programs (both source and object forms) and data.

- The operating system is responsible for the following activities in connection with file management:
  - File creation and deletion.
  - Directory creation and deletion.
  - Support of primitives for manipulating files and directories.
  - Mapping files onto secondary storage.
  - File backup on stable (nonvolatile) storage media.

Protection System

- Protection refers to a mechanism for controlling access by programs, processes, or users to both system and user resources.

- The protection mechanism must:
  - distinguish between authorized and unauthorized usage.
  - specify the controls to be imposed.
  - provide a means of enforcement.

# Networking (Distributed Systems)

- A distributed system is a collection of processors that do not share memory or a clock. Each processor has its own local memory.

- The processors in the system are connected through a communication network.

- A distributed system provides user access to various system resources.

- Access to a shared resource allows:
  - Computation speed-up
  - Increased data availability
  - Enhanced reliability

Command-Interpreter System

- Many commands are given to the operating system by control statements which deal with:
  - process creation and management
  - I/O handling
  - secondary-storage management
  - main-memory management
  - file-system access
  - protection
  - networking
- The program that reads and interprets control statements is called variously:
  - control-card interpreter
  - command-line interpreter
  - shell (in UNIX)

  Its function is to get and execute the next command statement

Operating System Services
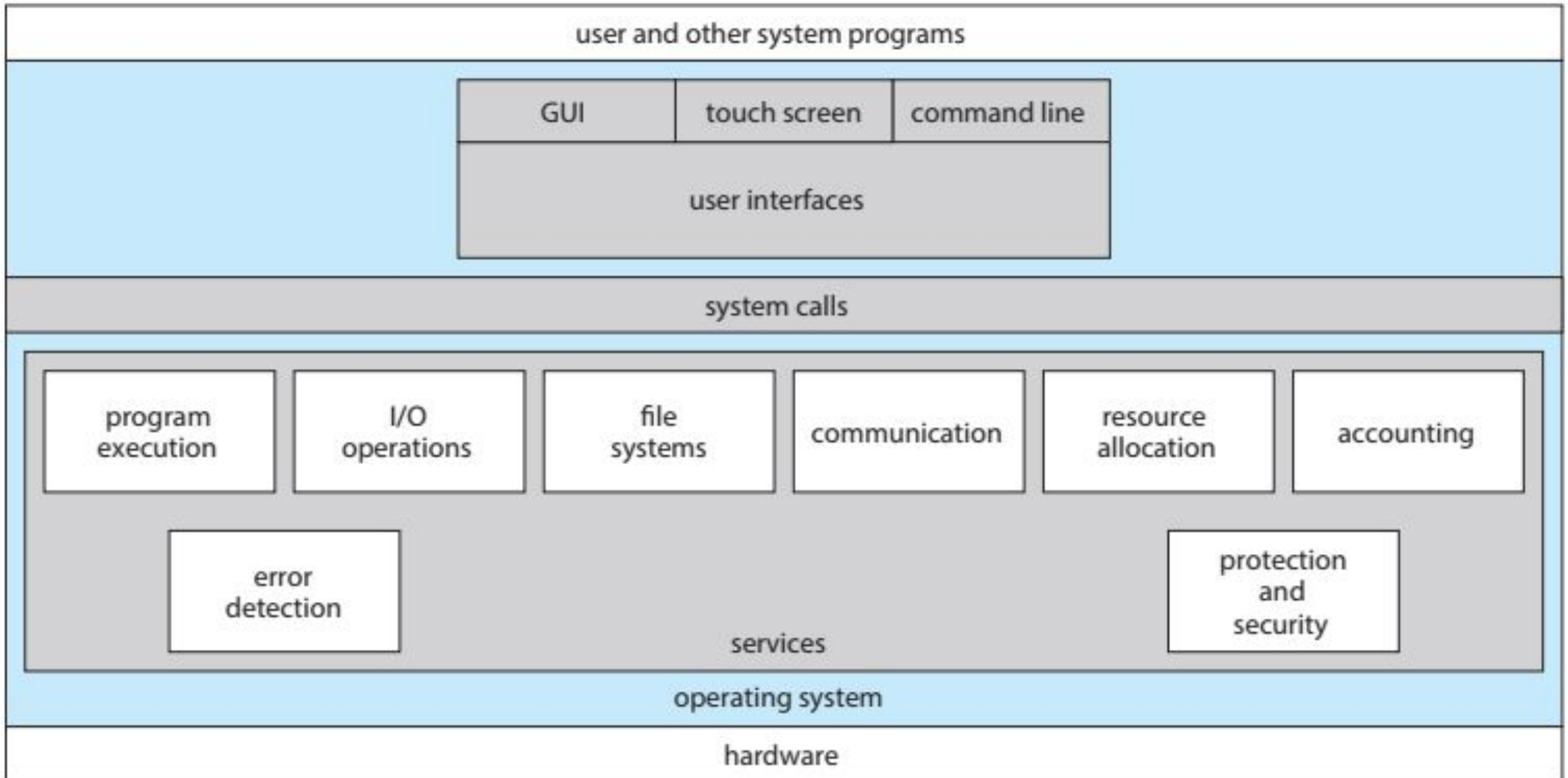
A view of operating system services.

Operating-System Services

- An operating system provides an environment for the execution of programs.

- It makes certain services available to programs and to the users of those programs.

- The specific services provided, of course, differ from one operating system to another, but we can identify common classes.
  - ✔ Program execution
  - ✔ I/O operations
  - ✔ File-system manipulation
  - ✔ Communications
  - ✔ Error detection

Operating-System Services

Program execution:

- system capability to load a program into memory, run the program, and terminate the program, either normally or abnormally.

I/O operations:

- since user programs cannot execute I/O operations directly, the operating system must provide some means to perform I/O.
- The OS is responsible for transferring data to and from I/O devices, including keyboards, terminals, printers, and storage devices.

File-system manipulation:

- In addition to raw data storage, the OS is also responsible for maintaining directory and subdirectory structures, mapping file names to specific blocks of data storage, and providing tools for navigating and utilizing the file system.
- program capability to read, write, create, and delete files.

Operating-System Services

Communications:

- Inter-process communications, IPC, either between processes running on the same processor, or between processes running on separate processors or separate machines.

- exchange of information between processes executing either on the same computer or on different systems tied together by a network. Implemented via shared memory or message passing.

Error detection:

- ensure correct computing by detecting errors in the **CPU & memory hardware**, in **I/O devices**, or in **user programs**.
  - ✔ Memory error or a power failure
  - ✔ Parity error on tape, a connection failure on a network, or lack of paper in the printer
  - ✔ Arithmetic overflow, an attempt to access an illegal memory location, or a too-great use of CPU time

Operating-System Services

Additional operating-system functions exist not for helping the user, but rather for ensuring **efficient** system operation.

Resource allocation:

- allocating resources to multiple users or multiple jobs running at the same time.
- Many different types of resources are managed by OS such as CPU cycles, main memory, and file storage.
- best to use the CPU

Accounting:

- keep track of and record which users use how much and what kinds of computer resources.
- Usage statistics may be a valuable tool for researchers who wish to reconfigure the system to improve computing services.

Operating-System Services

Additional operating-system functions exist not for helping the user, but rather for ensuring **efficient** system operation.

Protection and security:

- Preventing harm to the system and to resources
- **Protection** involves ensuring that all access to system resources is controlled.
- **Security** of the system from outsiders is also important.

User Operating-System Interface

There are two fundamental approaches for users to interface with the operating system.

- Command-line Interface Or Command Interpreter
  - ✔ command interpreter allows users to directly enter commands that are to be performed by the operating system.
- Graphical user interface or GUI.

System Calls

- System calls provide the interface between a running program and the operating system.

  - Generally available as assembly-language instructions.

- System calls provide a means for user or application programs to call upon the services of the operating system.

- Generally written in C or C++, although some are written in assembly for optimal performance.

System Calls

- Example: how the operating system handles a user application invoking the open() system call.
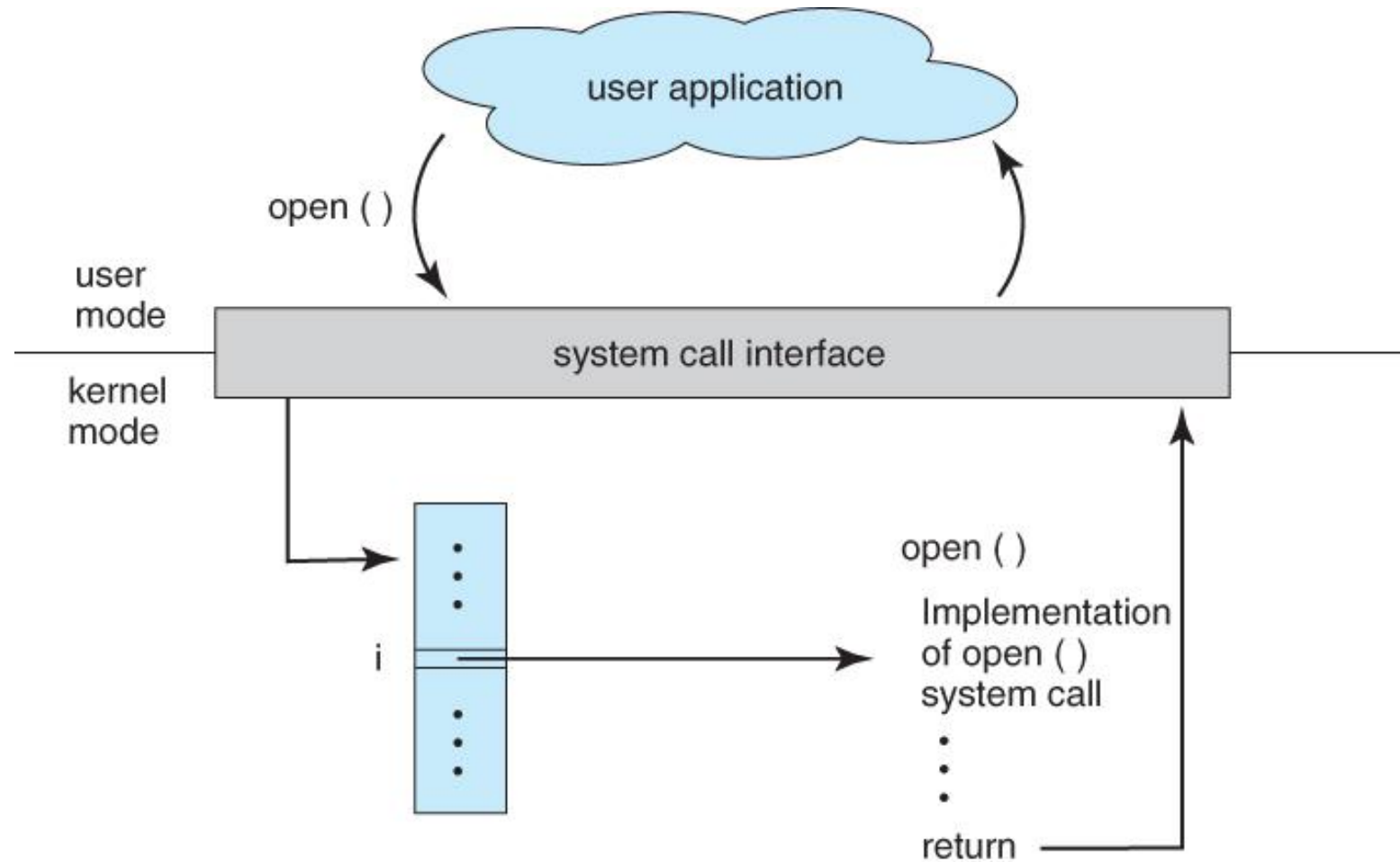
Types of System Calls

- System calls can be grouped roughly into six major categories:
    - Process Control
    - File Manipulation
    - Device Manipulation
    - Information maintenance, and
    - Communications
    - Protection

Types of System Calls

Process control:

- A running program needs to be able to halt its execution either normally (end) or abnormally (abort).

- If a system call is made to terminate the currently running program abnormally, or if the program runs into a problem and causes an error trap, a dump of memory is sometimes taken and an error message generated.

Types of System Calls

Process control:

- Process control

  - create process, terminate process

  - load, execute

  - get process attributes, set process attributes

  - wait event, signal event

  - allocate and free memory

	Windows	Unix
Process control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()

Types of System Calls

File manipulation:

• We identify several common system calls dealing with files.

• We first need to be able to create() and delete() files.

• Either system call requires the name of the file and perhaps some of the file's attributes.

Types of System Calls

File manipulation:

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

File management	CreateFile()	open()
	ReadFile()	read()
	WriteFile()	write()
	CloseHandle()	close()

Types of System Calls

- **Device manipulation:**
- A process may need several resources to execute—main memory, disk drives, access to files, and so on.
- If the resources are available, they can be granted, and control can be returned to the user process.
- Otherwise, the process will have to wait until sufficient resources are available.
- The various resources controlled by the operating system can be thought of as devices.
- Some of these devices are physical devices (e.g. disk drives), while others can be thought of as abstract or virtual devices (e.g. files).

Types of System Calls

- **Device manipulation:**

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

Device management	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()

Types of System Calls

Information maintenance:

- Many system calls exist simply for the purpose of transferring information between the user program and the operating system.

- For example, most systems have a system call to return the current time() and date().

- Other system calls may return information about the system, such as the **version** number of the operating system, the amount of free memory or disk space, and so on

Types of System Calls

Information maintenance:

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

| Information maintenance | GetCurrentProcessID()
SetTimer()
Sleep() | getpid()
alarm()
sleep() |

Types of System Calls

Communications:

- There are two common models of inter-process communication:
  - ✔ Message-passing model
  - ✔ Shared-memory model.

- In the message-passing model, the communicating processes exchange messages with one another to transfer information.

- Messages can be exchanged between the processes either directly or indirectly through a common mailbox.

- Before communication can take place, a connection must be opened.

Types of System Calls

Communications:

- Communications
  - create, delete communication connection
  - send, receive messages
  - transfer status information
  - attach or detach remote devices

Communications	CreatePipe()	pipe()
	CreateFileMapping()	shm_open()
	MapViewOfFile()	mmap()

Types of System Calls

Protection:

- Protection provides a mechanism for controlling access to the resources provided by a computer system.

- Historically, protection was a concern only on multiprogrammed computer systems with several users.

- However, with the advent of networking and the Internet, all computer systems, from servers to mobile handheld devices, must be concerned with protection.

Types of System Calls

Protection:

- Protection
  - get file permissions
  - set file permissions

Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

Types of Operating System

Operating system classifications:

- Batch Operating System

- Time-Sharing Systems

- Desktop Systems

- Parallel Systems

- Distributed Systems

- Clustered Systems

- Real-Time Systems

Batch Operating System

- Reduce setup time by batching similar jobs
  - ✔ There is an operator which takes similar jobs having the same requirement and group them into batches.
  - ✔ It is the responsibility of the operator to sort jobs with similar needs.
- Automatic job sequencing – automatically transfers control from one job to another.
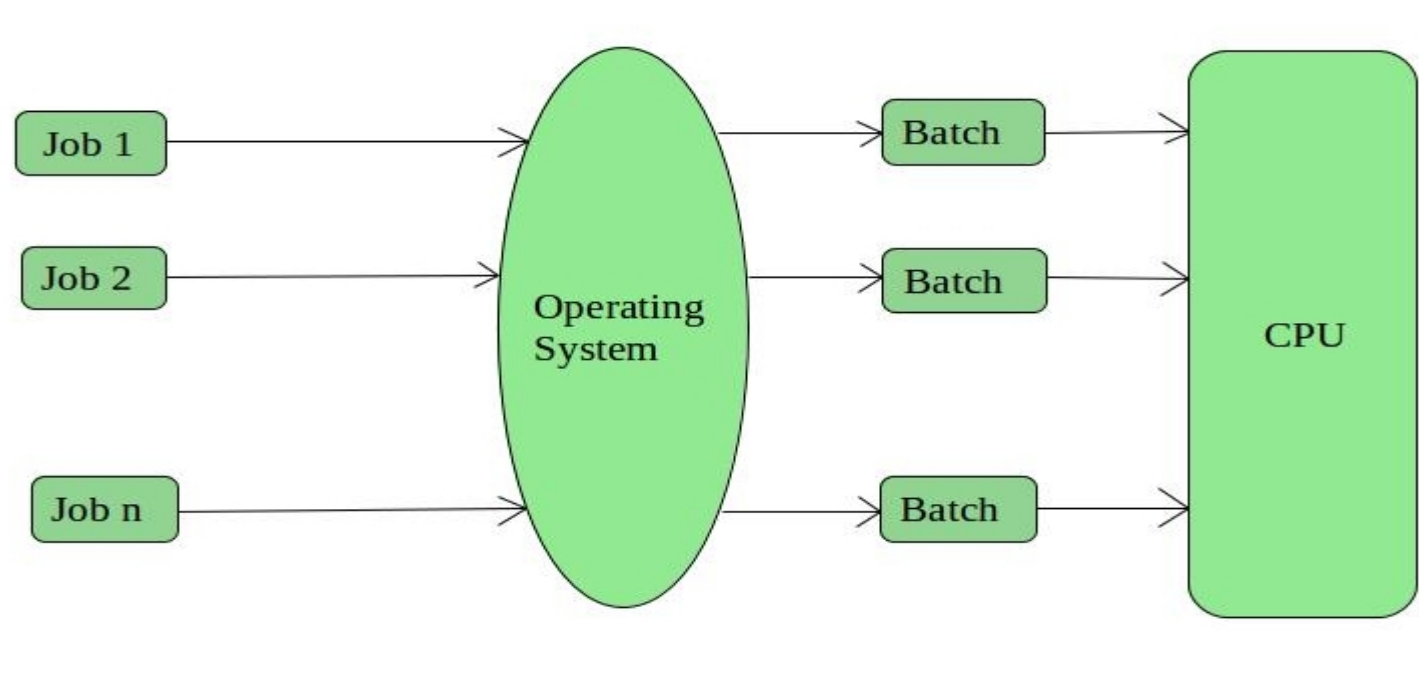
Batch Operating System

- Multiple users can share the batch systems

- The idle time for the batch system is very less

- It is easy to manage large work repeatedly in batch systems.

- Batch systems are hard to debug

- It is sometimes costly

- The other jobs will have to wait for an unknown time if any job fails

OS Features Needed for Multiprogramming

- I/O routine supplied by the system.
- Memory management – the system must allocate the memory to several jobs.
- CPU scheduling – the system must choose among several jobs ready to run.
- Allocation of devices.

Time Sharing Systems

- Each task is given some time to execute so that all the tasks work smoothly.

- Each user gets the time of CPU as they use a single system.

- These systems are also known as Multitasking Systems.

- The task can be from a single user or different users also.
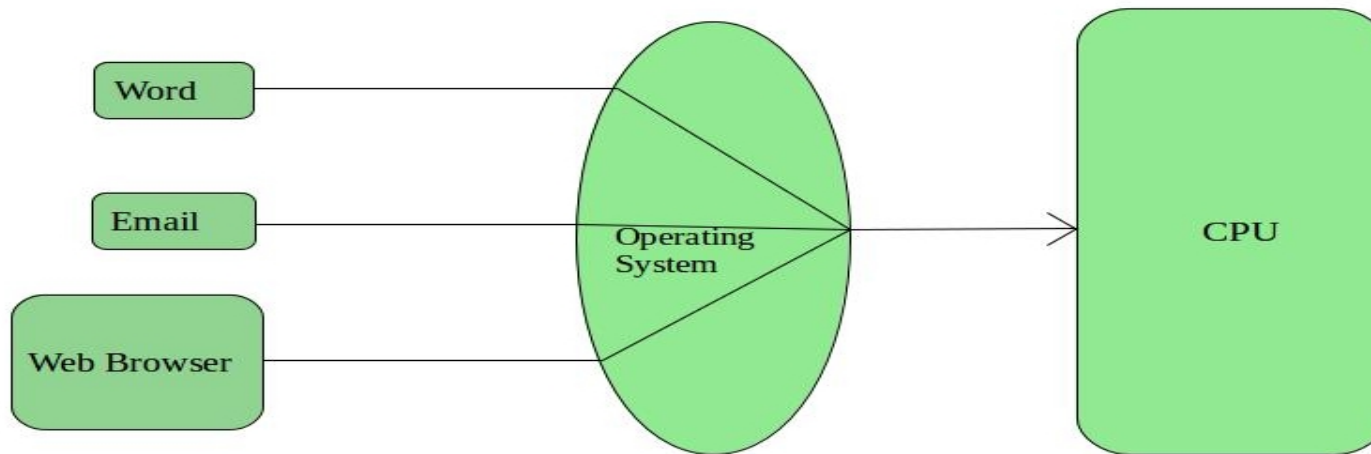
Time Sharing Systems

- The time that each task gets to execute is called **quantum**. After this time interval is over OS switches over to the next task.

- Each task gets an equal opportunity

- Fewer chances of duplication of software

- CPU idle time can be reduced
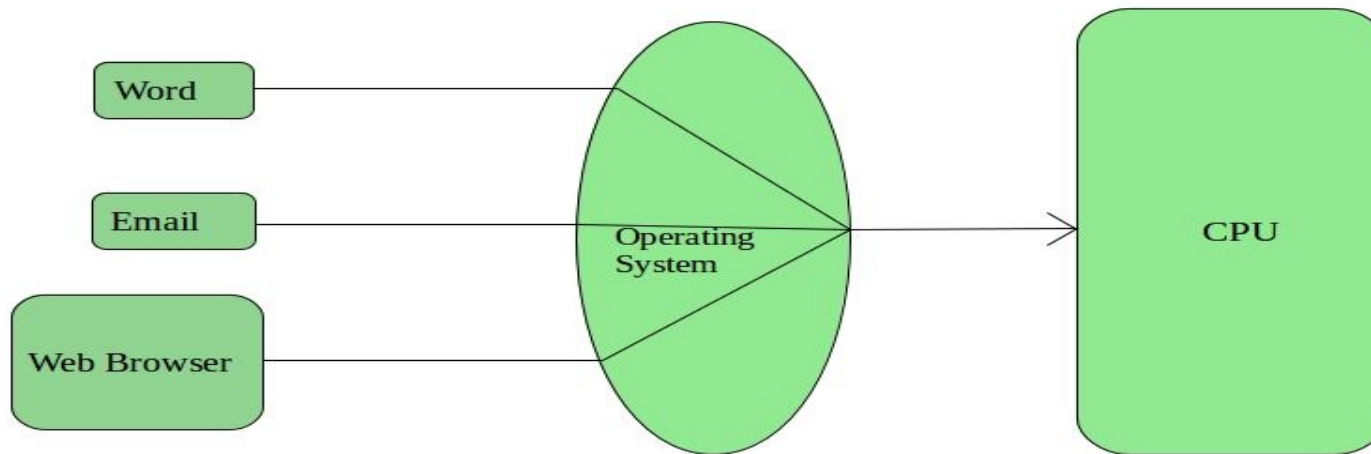
Desktop Systems

- Personal computers – computer system dedicated to a single user.

- I/O devices – keyboards, mice, display screens, small printers.

- User convenience and responsiveness.

- Can adopt technology developed for larger operating system' often individuals have sole use of computer and do not need advanced CPU utilization of protection features.

- May run several different types of operating systems (Windows, MacOS, UNIX, Linux)

Parallel Systems

- Multiprocessor systems with more than on CPU in close communication.
- Tightly coupled system – processors share memory and a clock; communication usually takes place through the shared memory.
- Advantages of parallel system:
  - Increased throughput
  - Economical
  - Increased reliability
   - graceful degradation
    - fail-soft systems

Parallel Systems

- Symmetric multiprocessing (SMP)
  - Each processor runs and identical copy of the operating system.
  - Many processes can run at once without performance deterioration.
  - Most modern operating systems support SMP
- Asymmetric multiprocessing
  - Each processor is assigned a specific task; master processor schedules and allocated work to slave processors.
  - More common in extremely large systems

Distributed Systems

- Distribute the computation among several physical processors.
- Loosely coupled system – each processor has its own local memory; processors communicate with one another through various communications lines, such as high-speed buses or telephone lines.
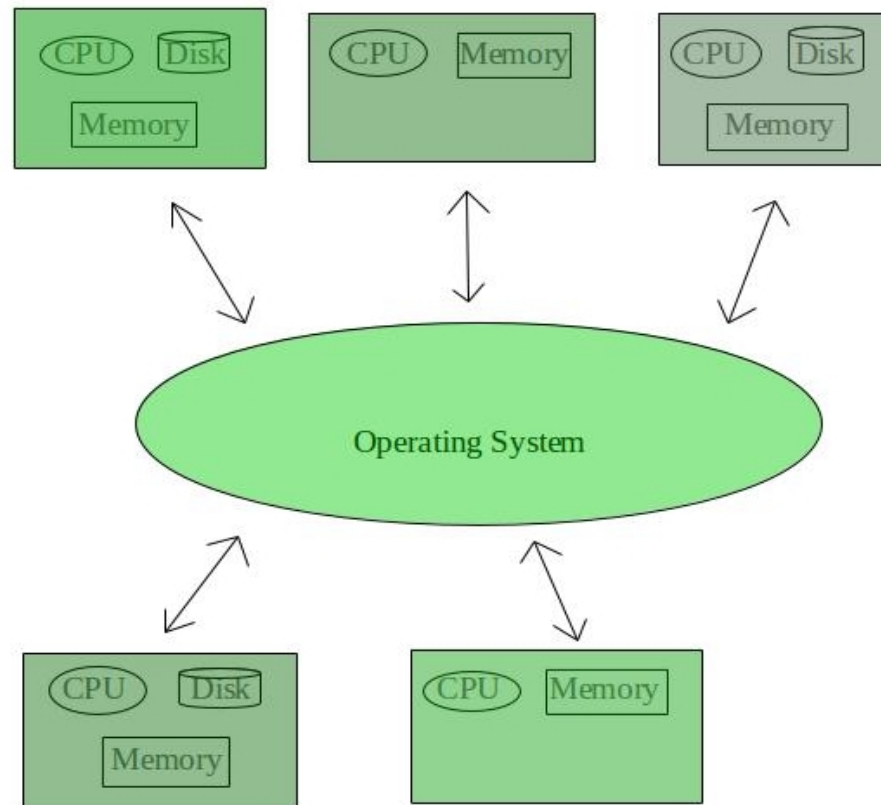
Distributed Systems

- Advantages of distributed systems.
  - Resources Sharing
  - Computation speed up – load sharing
  - Reliability
  - Communications

- Requires networking infrastructure.

- Local area networks (LAN) or Wide area networks (WAN)

- May be either client-server or peer-to-peer systems.

Clustered Systems

- Clustering allows two or more systems to share storage.

- Provides high reliability.

- Asymmetric clustering: one server runs the application while other servers standby.

- Symmetric clustering: all N hosts are running the application.

Real-Time Systems

- Often used as a control device in a dedicated application such as controlling scientific experiments, medical imaging systems, industrial control systems, and some display systems.

- Well-defined fixed-time constraints.

- Real-Time systems may be either hard or soft real-time.

Real-Time Systems

- Hard real-time:
  - Secondary storage limited or absent, data stored in short term memory, or read-only memory (ROM)
  - Conflicts with time-sharing systems, not supported by general-purpose operating systems.

- Soft real-time
  - Limited utility in industrial control of robotics
  - Useful in applications (multimedia, virtual reality) requiring advanced operating-system features.
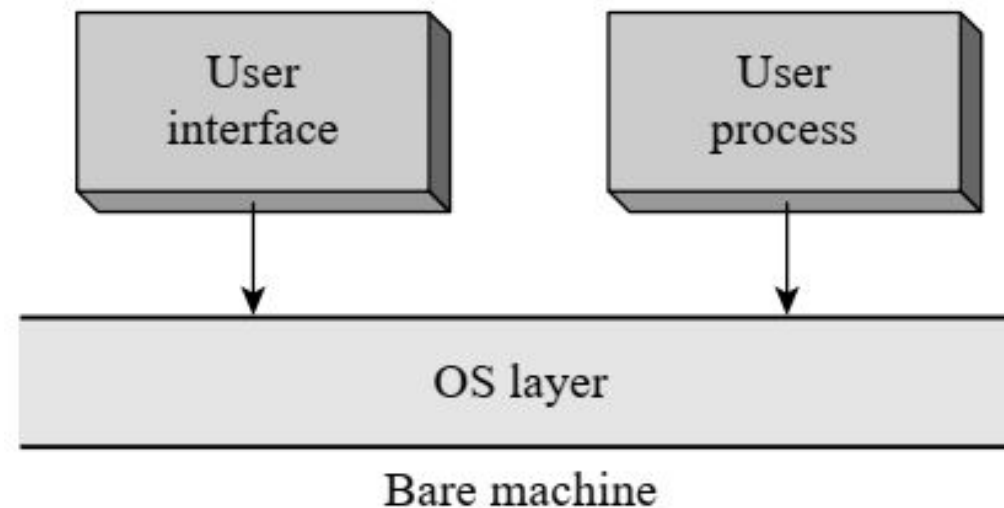
Operating System Structure

- For efficient performance and implementation an OS should be partitioned into separate **subsystems**, each with carefully defined tasks, inputs, outputs, and performance characteristics.

- You may use a similar approach when you structure your programs: rather than placing all of your code in the **main()** function, you instead separate logic into a number of **functions**, clearly articulate parameters and return values, and then call those functions from main().

- These subsystems can then be arranged in various architectural configurations:
  - Monolithic Systems
  - Layered Systems
  - Microkernels
  - Client-server Systems
  - Virtual Machines

Operating System Structure: Monolithic Systems

- An OS is a complex software that has a large number of functionalities and may contain millions of instructions.

- It is designed to consist of a **set of software modules**, where each module has a well-defined interface that must be used to access any of its functions or data.

- The simplest structure for organizing an operating system is **no structure** at all.

- That is, place *all of the functionality of the kernel into a single*, static binary file that runs in a single address space.

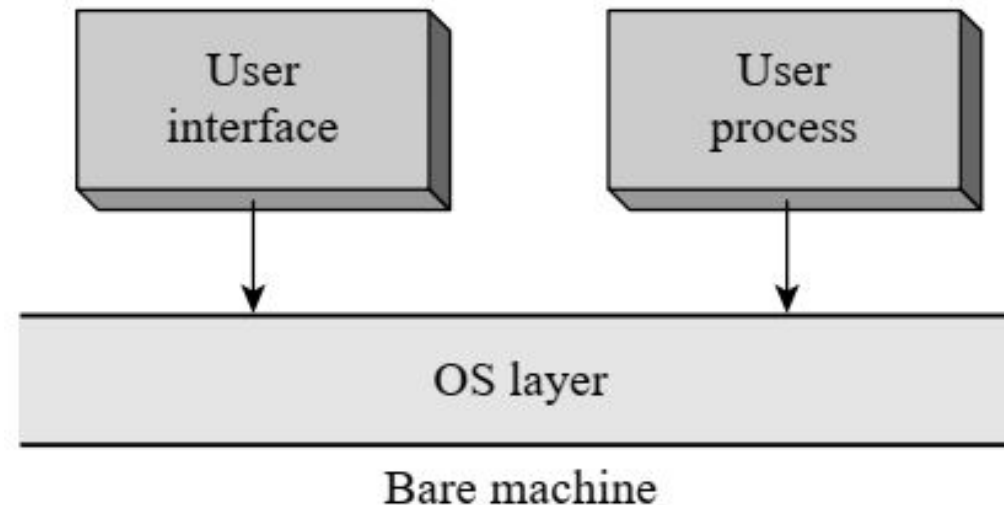# Operating System Structure: Monolithic Systems

- Early operating systems had a monolithic structure, whereby the OS formed a single software layer between the user and the bare machine.

- The user interface was provided by a command interpreter. The command interpreter organized creation of user processes.
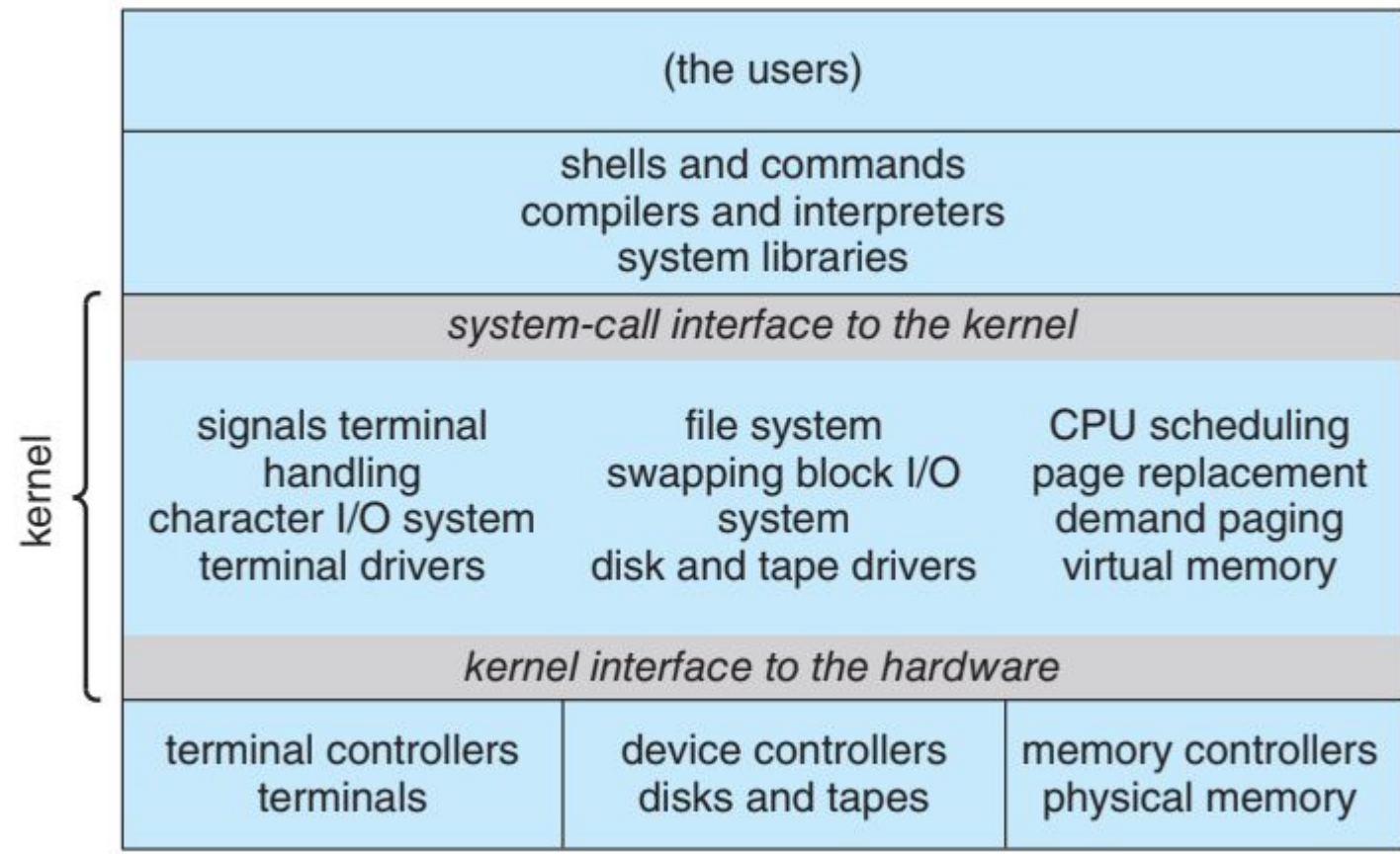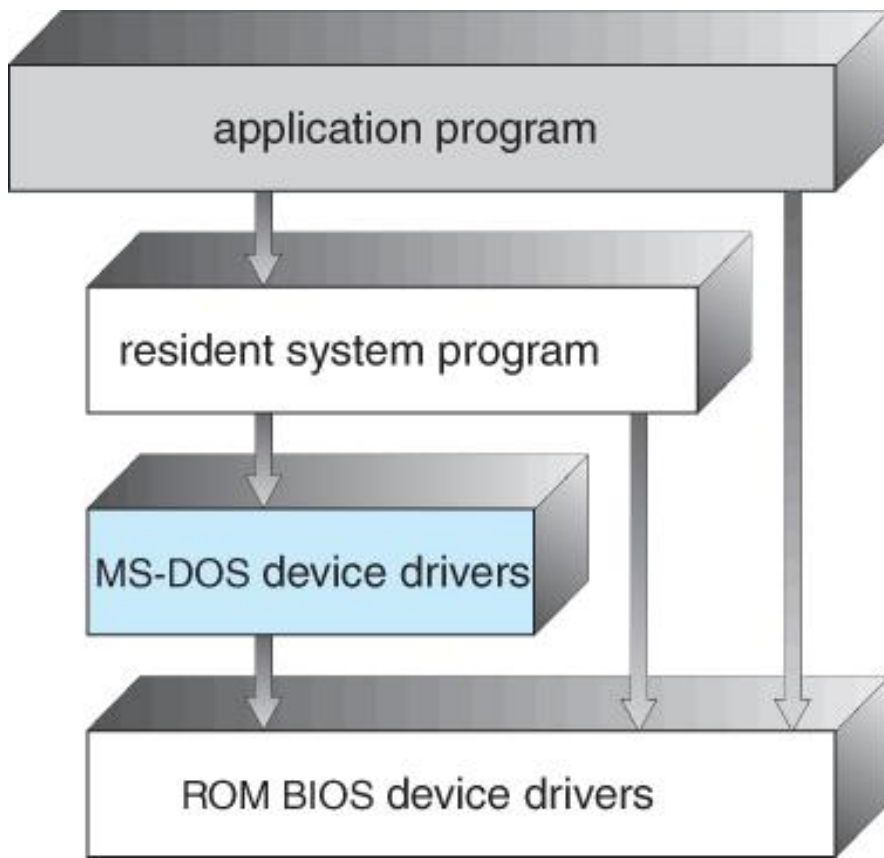
Operating System Structure: Monolithic Systems

- Both the command interpreter and user processes invoked OS functionalities and services through **system calls**.

- Two kinds of problems with the monolithic structure were realized over a period of time. The sole OS layer had an interface with the bare machine. Hence architecture-dependent code was spread throughout the OS, and so there was poor portability.

- It also made testing and debugging difficult, leading to high costs of maintenance and enhancement.

Operating System Structure: Monolithic Systems

- These problems led to the search for alternative ways to structure an OS.
- Solutions to these problems: Layered structure, Kernel-based structure and Microkernel-based OS structure.

application program

resident system program

MS-DOS device drivers

ROM BIOS device drivers

| (the users)		
shells and commands compilers and interpreters system libraries		
system-call interface to the kernel		
signals terminal handling character I/O system terminal drivers	file system swapping block I/O system disk and tape drivers	CPU scheduling page replacement demand paging virtual memory
kernel interface to the hardware		
terminal controllers terminals	device controllers disks and tapes	memory controllers physical memory

kernel

Operating System Structure: Layered Systems

- The monolithic approach is often known as a tightly coupled system because changes to one part of the system can have wide-ranging effects on other parts.

- Alternatively, we could design a loosely coupled system. Such a system is divided into separate, smaller components that have specific and limited functionality.

- All these components together comprise the kernel.

- The advantage of this modular approach is that changes in one component affect only that component, and no others, allowing system implementers more freedom in creating and changing the inner workings of the system.

Operating System Structure: Layered Systems

- The monolithic OS structure suffered from the problem that all OS components had to be able to work with the bare machine.

- This feature increased the cost and effort in developing an OS because of the large semantic gap between the operating system and the bare machine.

- **Semantic Gap:** The mismatch between the nature of operations needed in the application and the nature of operations provided in the machine.
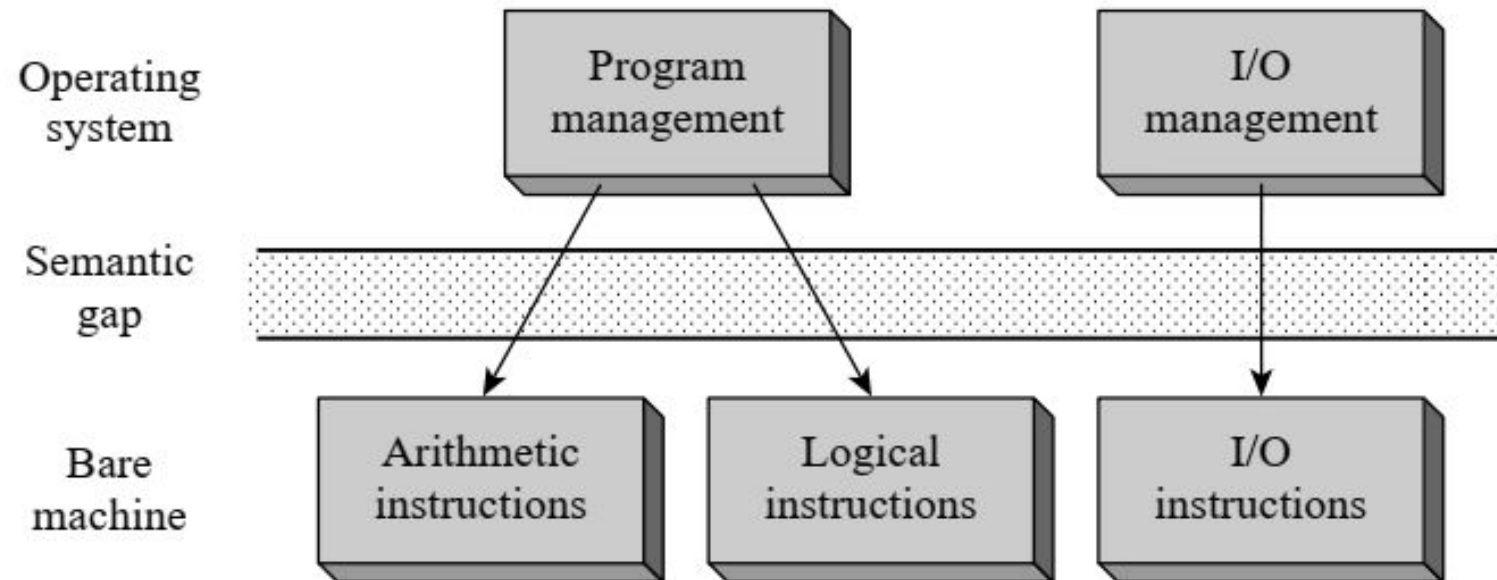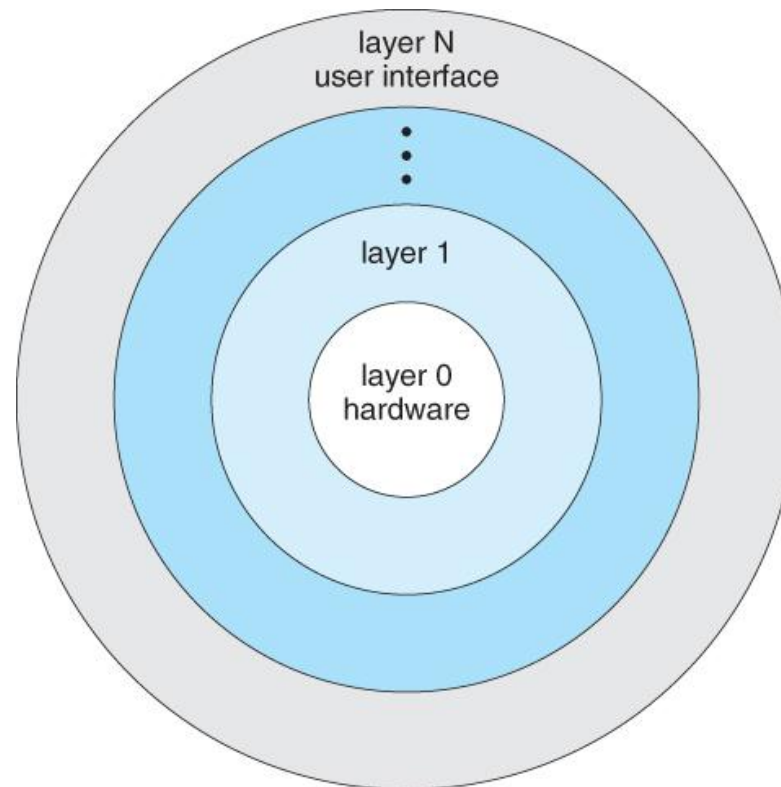
Operating System Structure: Layered Systems

- The extended machine provides operations like context save, dispatching, swapping, and I/O initiation.

- The operating system layer is located on top of the extended machine layer. This arrangement considerably simplifies the coding and testing of OS modules by separating the algorithm of a function from the implementation of its primitive operations.
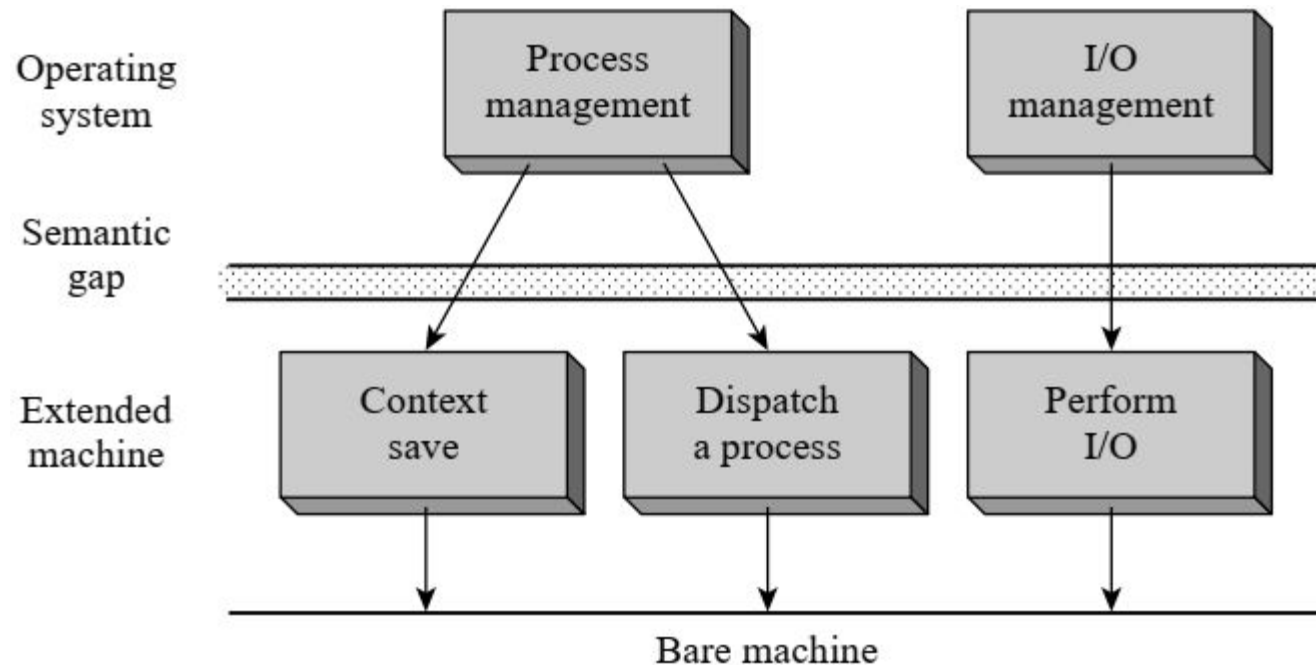
Operating System Structure: Layered Systems

- It is now easier to test, debug, and modify an OS module than in a monolithic OS.

- We say that the lower layer provides an abstraction that is the extended machine. We call the operating system layer the top layer of the OS.
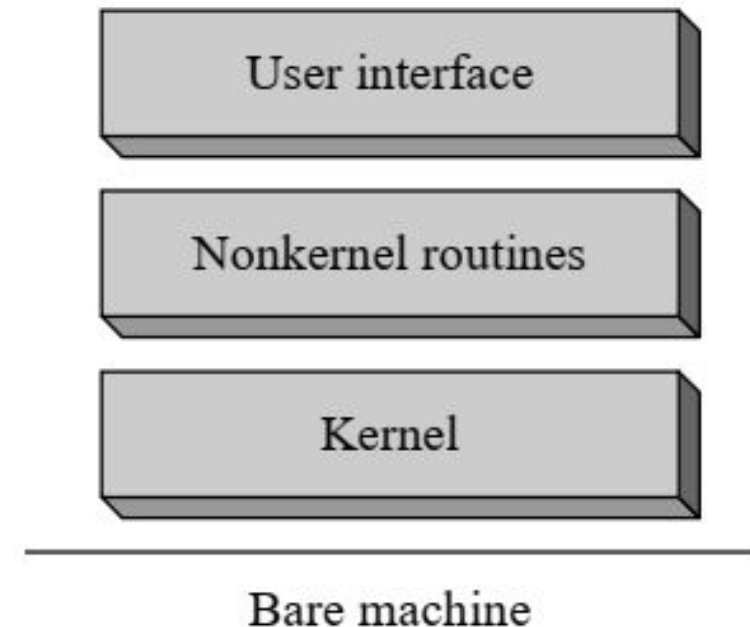
Operating System Structure: Layered Systems

- The layered approach to OS design suffers from two main problems:

1) The operation of a system may be slowed down by the layered structure. Recall that each layer can interact only with adjoining layers. It implies that a request for OS service made by a user process must move down from the highest numbered layer to the lowest numbered layer before the required action is performed by the bare machine. This feature leads to **high overhead**.

2) The second problem concerns difficulties in developing a layered design. Since a layer can access only the immediately lower layer, all features and facilities needed by it must be available in lower layers. This requirement poses a problem in the ordering of layers that require each other's services.

Operating System Structure: Kernel-based

- The kernel is the core of the OS.

- It provides a set of functions and services to support various OS functionalities. The rest of the OS is organized as a set of non-kernel routines, which implement operations on processes and resources that are of interest to users, and a user interface.

- We all know the operation of the kernel is interrupt-driven.

- The kernel gets control when an interrupt such as a timer interrupt or an I/O completion interrupt notifies occurrence of an event to it, or when the software-interrupt instruction is executed to make a system call.

- When the interrupt occurs, an interrupt servicing routine performs the context save function and invokes an appropriate event handler, which is a nonkernel routine of the OS.

User interface

Nonkernel routines

Kernel

Bare machine

Operating System Structure: Kernel-based

Functions and Services Offered by the Kernel

| OS functionality	Examples of kernel functions and services
Process management	Save context of the interrupted program, dispatch a process, manipulate scheduling lists
Process communication	Send and receive interprocess messages
Memory management	Set memory protection information, swap-in/swap-out, handle page fault (that is, "missing from memory" interrupt of Section 1.4)
I/O management	Initiate I/O, process I/O completion interrupt, recover from I/O errors
File management	Open a file, read/write data
Security and protection	Add authentication information for a new user, maintain information for file protection
Network management	Send/receive data through a message

• Kernel-based operating systems have poor extensibility because addition of a new functionality to the OS may require changes in the functions and services offered by the kernel.

Operating System Structure: Microkernels

- The microkernel was developed in the early 1990s to overcome the problems concerning portability, extensibility, and reliability of kernels.

- A microkernel is an **essential core of OS** code, thus it contains only a subset of the mechanisms typically included in a kernel and supports only a small number of system calls. This feature enhances portability and reliability of the microkernel.

- **Less essential parts** of OS code are **outside the microkernel** and use its services, hence these parts could be modified without affecting the kernel

Operating System Structure: Microkernels

- The microkernel includes mechanisms for process scheduling and memory management, etc., but does not include a scheduler or memory handler.

- These functions are implemented as servers, which are simply processes that never terminate. The servers and user processes operate on top of the microkernel, which merely performs interrupt handling and provides communication between the servers and user processes.
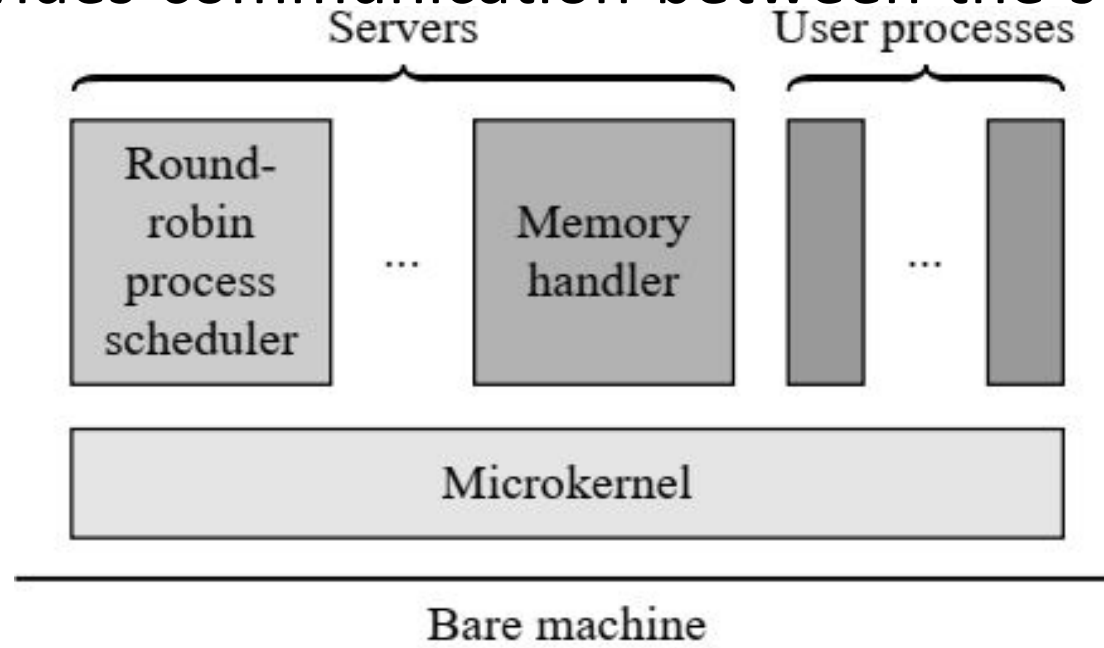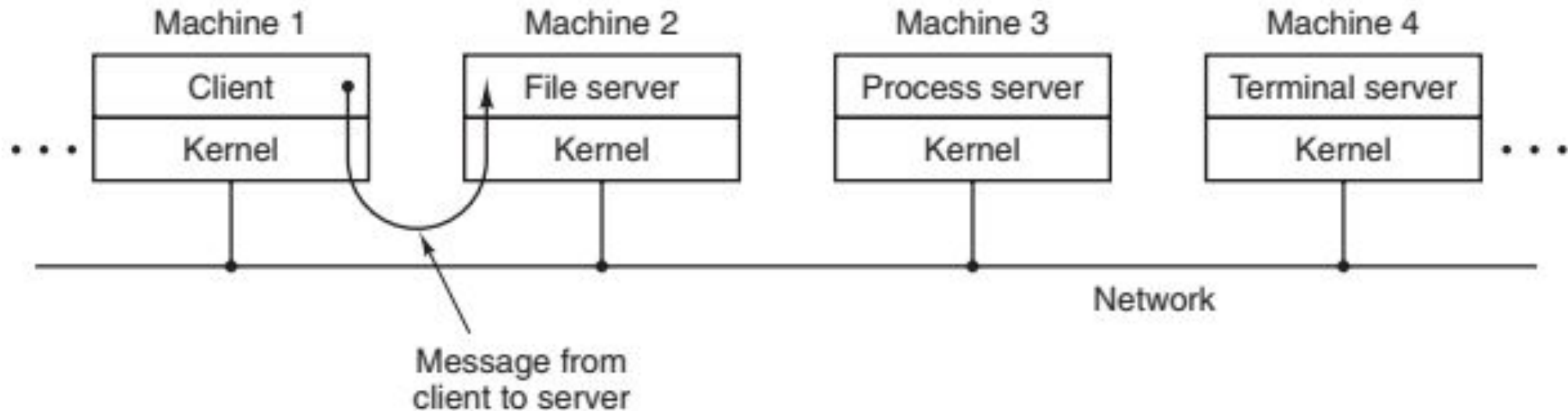
Operating System Structure: Microkernels

- The small size and extensibility of microkernels are valuable properties for the embedded systems environment, because operating systems need to be both small and fine-tuned to the requirements of an embedded application.
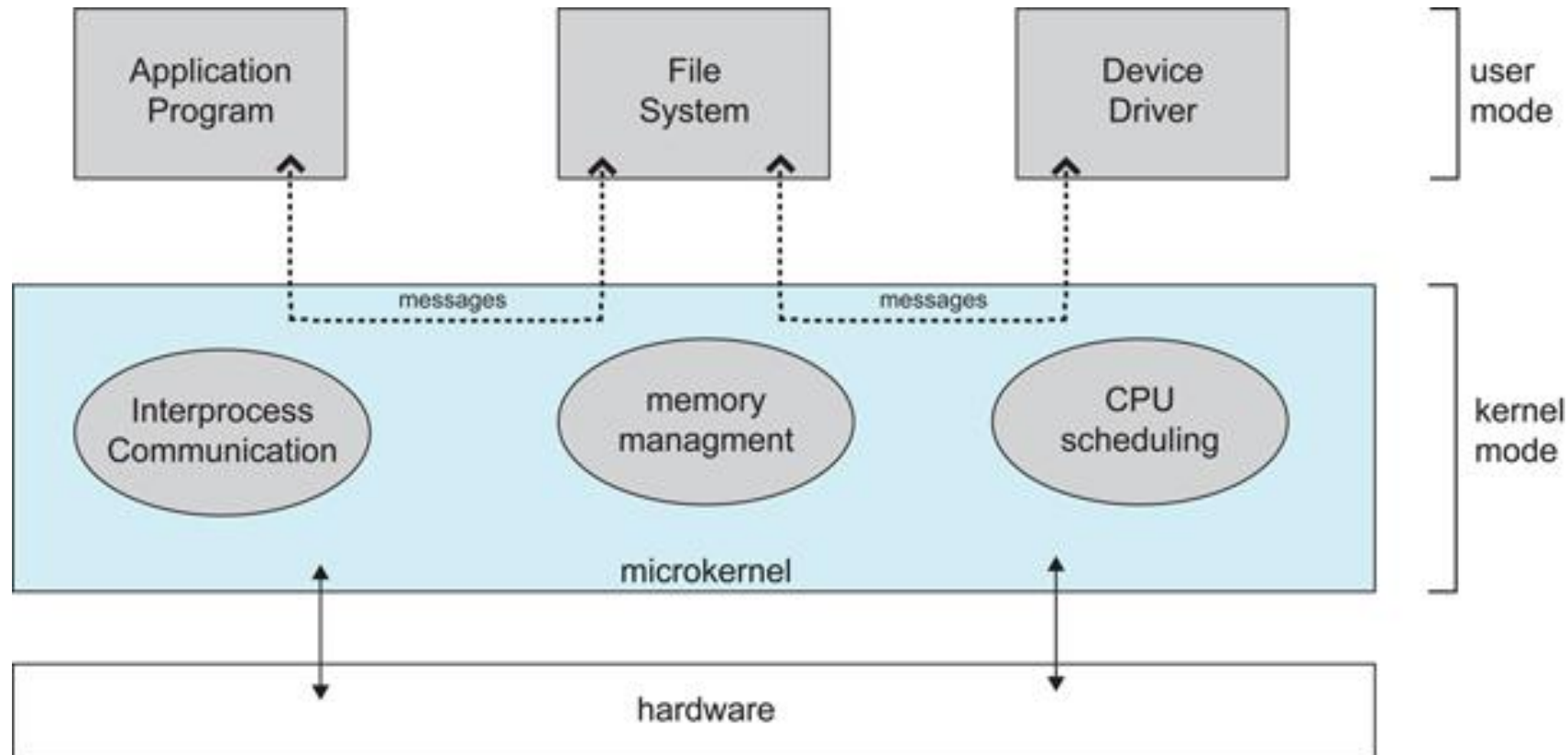
Operating System Structure: Client-Server Systems

- A slight variation of the microkernel idea is to distinguish two classes of processes, Client & Server.

- The servers, each of which provides some service, and

- The clients, which use these services.

- This model is known as the client-server model. Often the lowest layer is a microkernel, but that is not required. The essence is the presence of client processes and server processes.

Machine 1		Machine 2		Machine 3		Machine 4	
Client		File server		Process server		Terminal server	
Kernel		Kernel		Kernel		Kernel	

Network

Message from client to server

# Operating System Structure: Virtual Machines

- Different classes of users need different kinds of user service. Hence running a single OS on a computer system can disappoint many users.

- Operating the computer under different OSs during different periods is not a satisfactory solution because it would make accessible services offered under only one of the operating systems at any time.

- This problem is solved by using a virtual machine operating system (VM OS) to control the computer system.

- The VM OS creates several virtual machines. Each virtual machine is allocated to one user, who can use any OS of his own choice on the virtual machine and run his programs under this OS.

- This way users of the computer system can use different operating systems at the same time

Modern Operating System

- Architecture of some modern operating systems:
    - Architecture of Unix
    - The Kernel of Linux
    - The Kernel of Solaris
    - Architecture of Windows

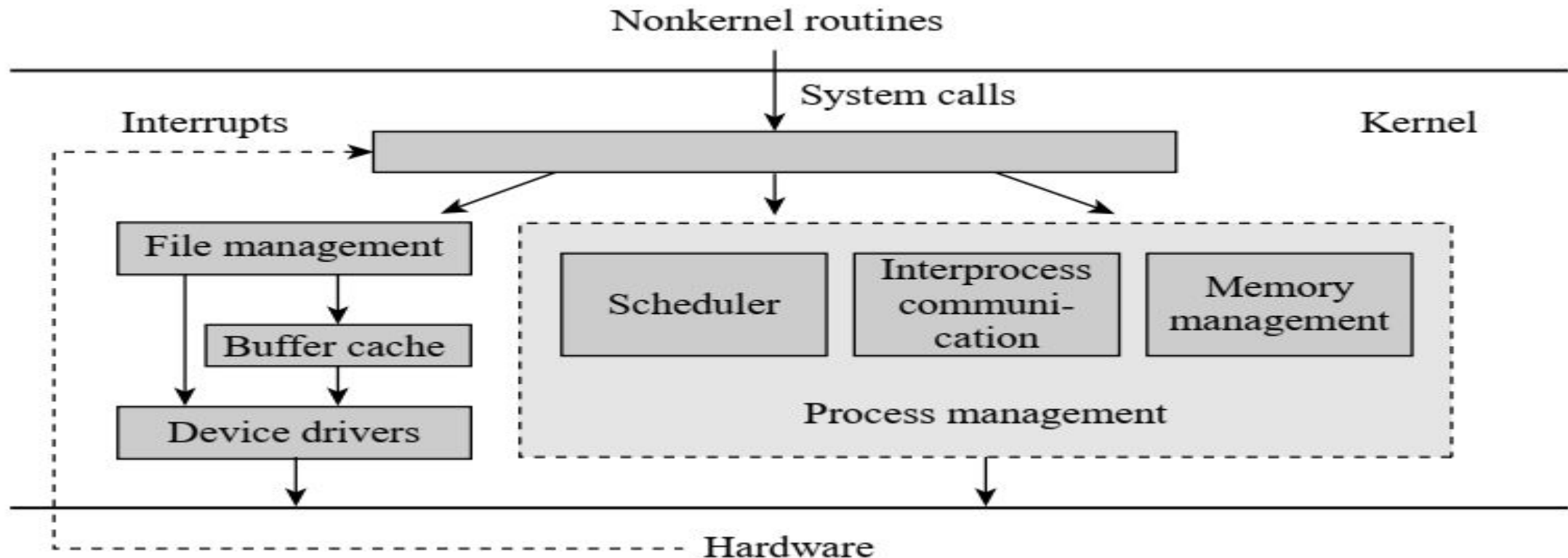# Modern Operating System- Unix

- Unix is a kernel-based operating system.
- It consists of two main components—process management and file management.
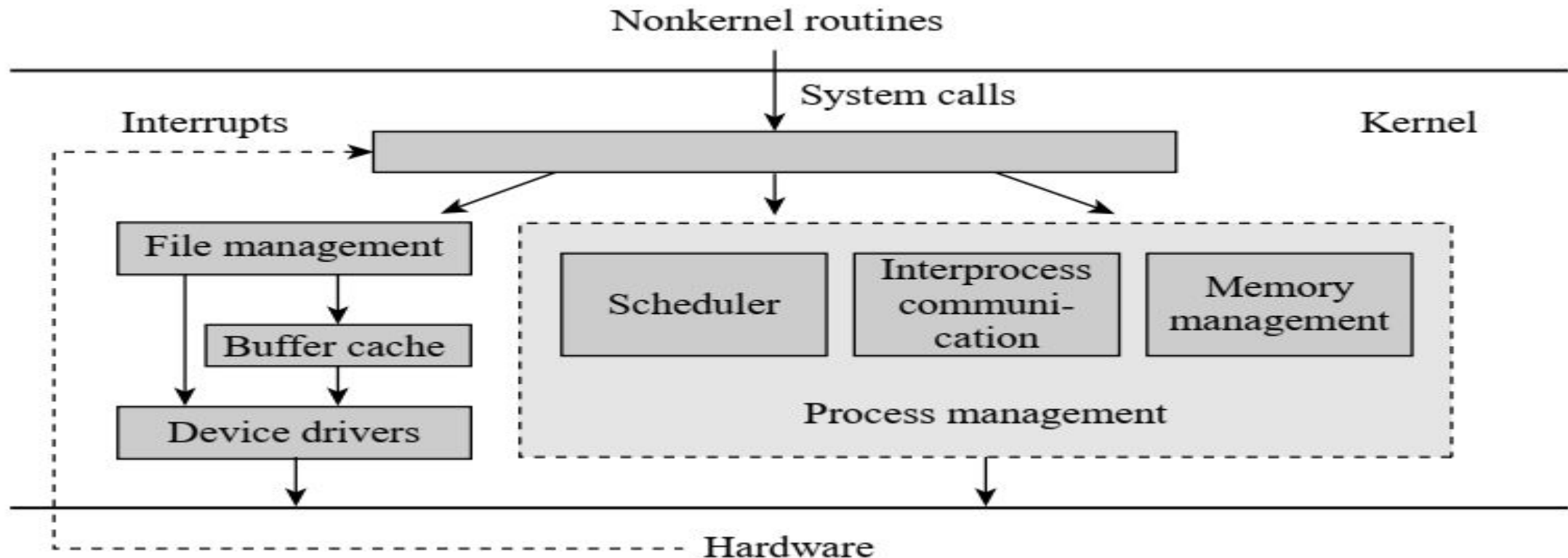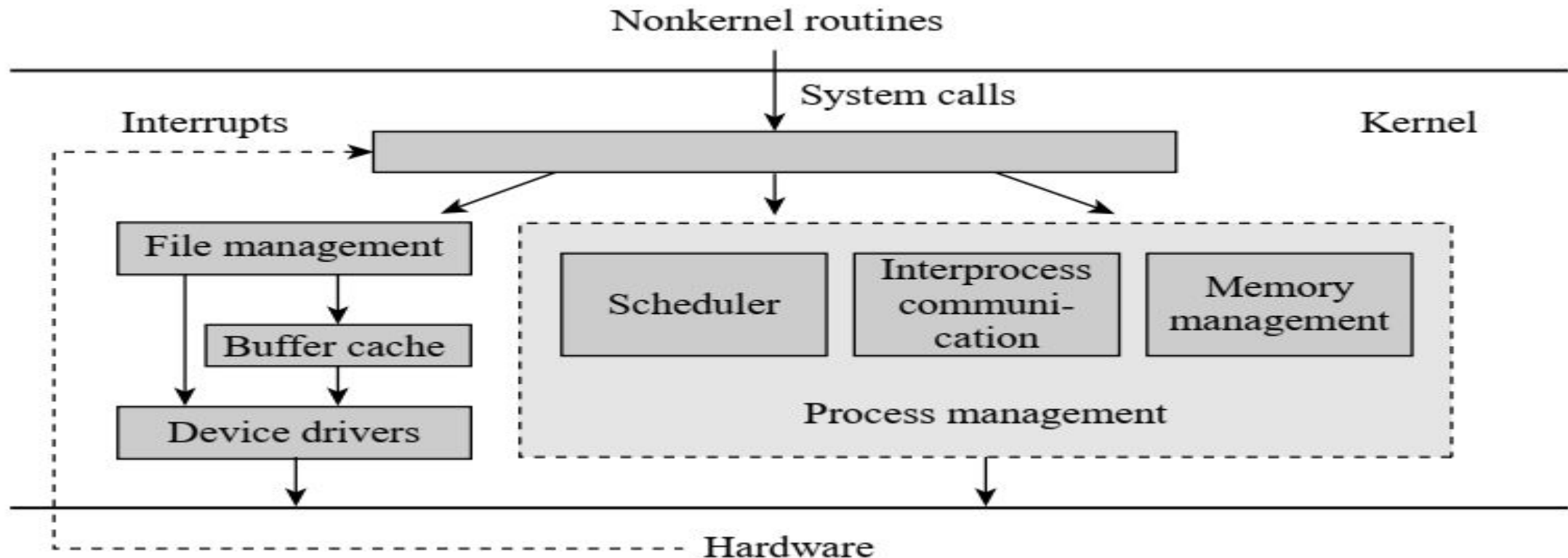
Modern Operating System- Unix

- The process management component consists of a module for inter-process communication, which implements communication and synchronization between processes, and the memory management and scheduling modules.
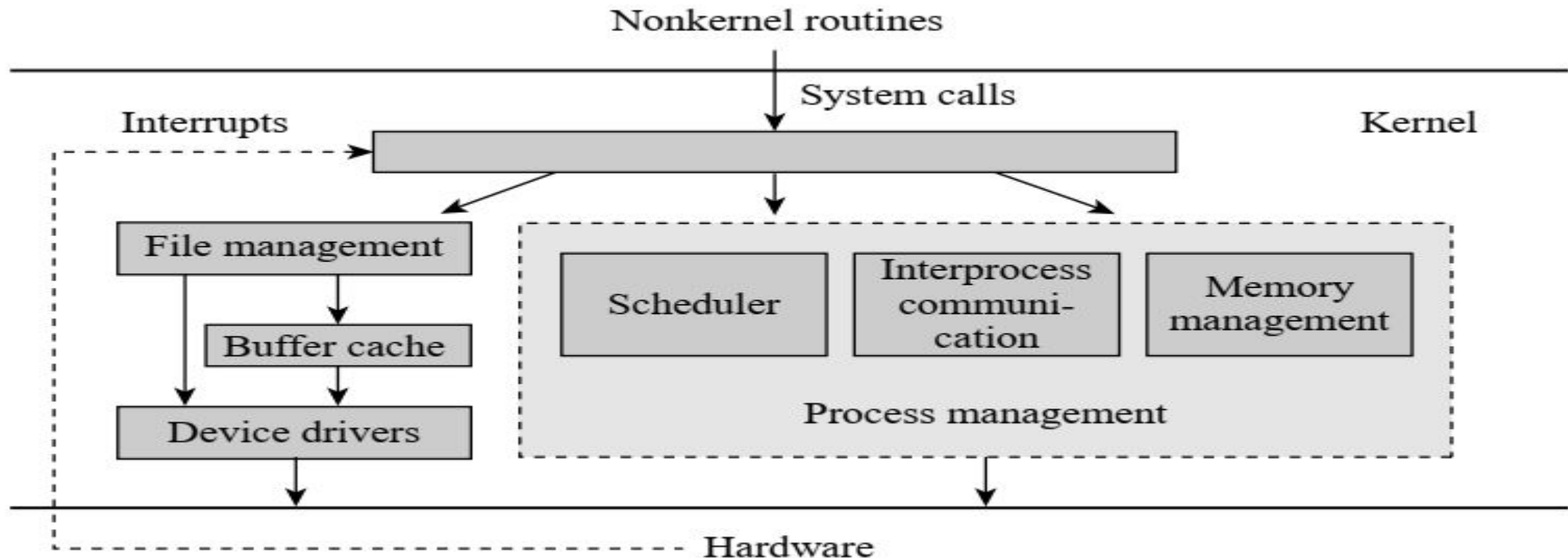
Modern Operating System- Unix

- The file management component performs I/O through device drivers.

- Each device driver handles a specific class of I/O devices and uses techniques like disk scheduling to ensure good throughput of an I/O device.

Modern Operating System- Unix

- The buffer cache is used to reduce both the time required to implement a data transfer between a process and an I/O device, and the number of I/O operations performed on devices like disks.

Modern Operating System- Unix

- The process management and file management components of the kernel are activated through interrupts raised in the hardware, and system calls made by processes and non-kernel routines of the OS.

- The user interface of the OS is a command interpreter, called a shell, that runs as a user process.

- The Unix kernel cannot be interrupted at any arbitrary moment of time.

- It can be interrupted only when a process executing kernel code exits, or when its execution reaches a point at which it can be safely interrupted.

- This feature ensures that the kernel data structures are not in an inconsistent state when an interrupt occurs and another process starts executing the kernel code.

Modern Operating System- Unix

- The Unix kernel has a long history of over four decades. The original kernel was small and simple.

- It provided a small set of abstractions, simple but powerful features like the pipe mechanism, which enabled users to execute several programs concurrently, and a small file system that supported only one file organization called the byte stream organization.

- All devices were represented as files, which unified the management of I/O devices and files.

- The kernel was written in the C language and had a size of less than 100 KB. Hence it was easily portable.

Modern Operating System- Unix

- Unix kernel was monolithic and not very extensible. So it had to be modified as new computing environments, like the client–server environment.

- A major strength of Unix was its use of open standards. It enabled a large number of organizations ranging from the academia to the industry to participate in its development, which led to widespread use of Unix.

- But also led to the development of a large number of variants because of concurrent and uncoordinated development.

- The kernel became bulky, growing to a few million bytes in size, which affected its portability.

Modern Operating System- Unix

- Around this time, a feature was added to dynamically load kernel modules in memory. It enabled kernel modules to be loaded only when needed.

- This feature reduced the memory requirement of the kernel, but not its code size. Hence it did not enhance its portability
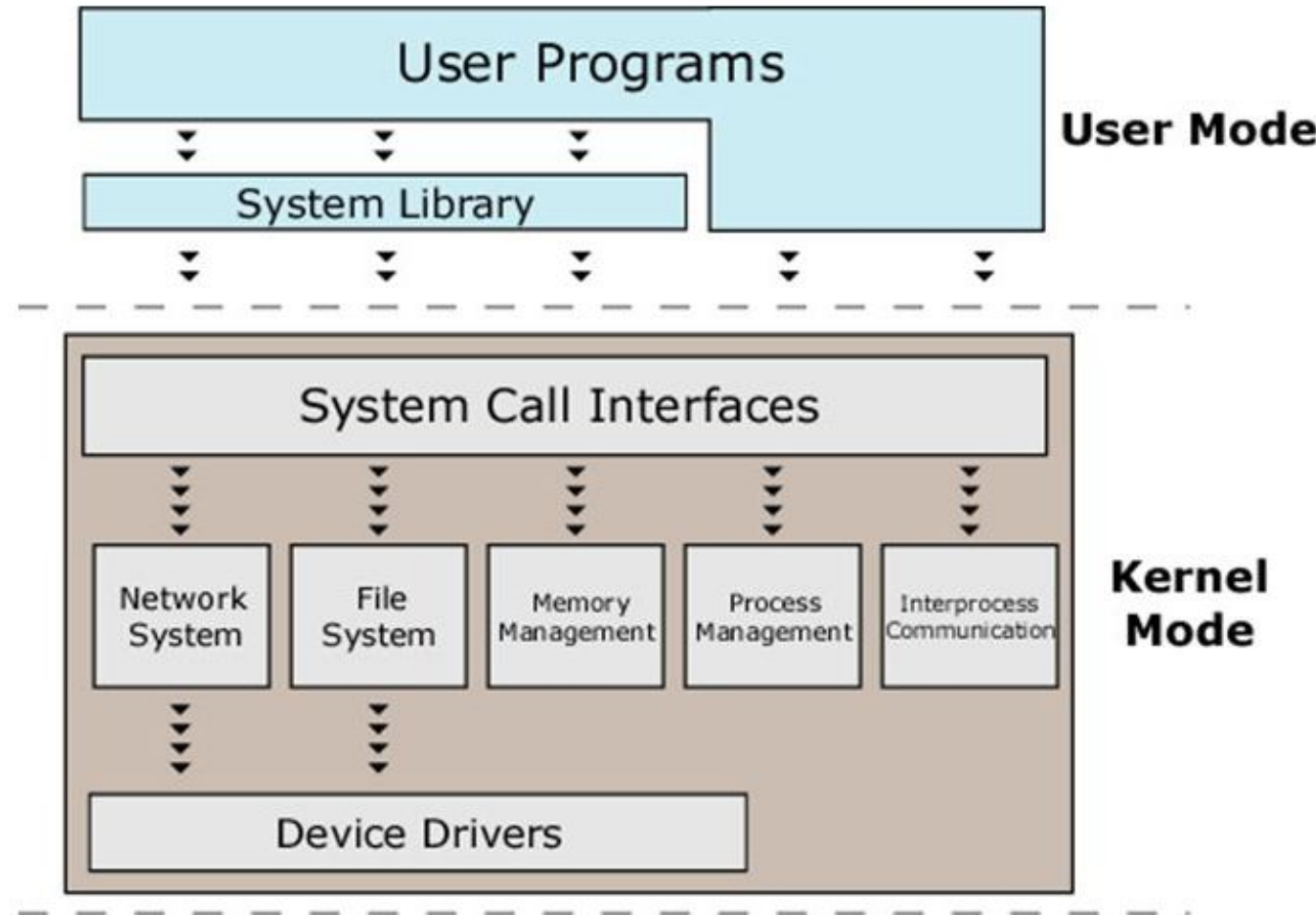
Modern Operating System- Linux

- Linux operating system provides the functionalities of Unix System V and Unix BSD; it is also compliant with the POSIX standard.

- It was initially implemented on the Intel 80386 and has since been implemented on later Intel processors and several other architectures.

- Linux has a monolithic kernel.

Modern Operating System- Linux

- Linux kernel is one layer in the architecture of the entire Linux system. The kernel is conceptually composed of five major modules: the process scheduler, the memory manager, the virtual file system, the network interface, and the interprocess communication interface.

- These modules interact with each other using function calls and shared data structures.

Modern Operating System- Linux

- Each module can be individually loaded into memory, or removed from it, depending on whether it is likely to be used in near future.

- In principle, any component of the kernel can be structured as a loadable module, but typically device drivers become separate modules.
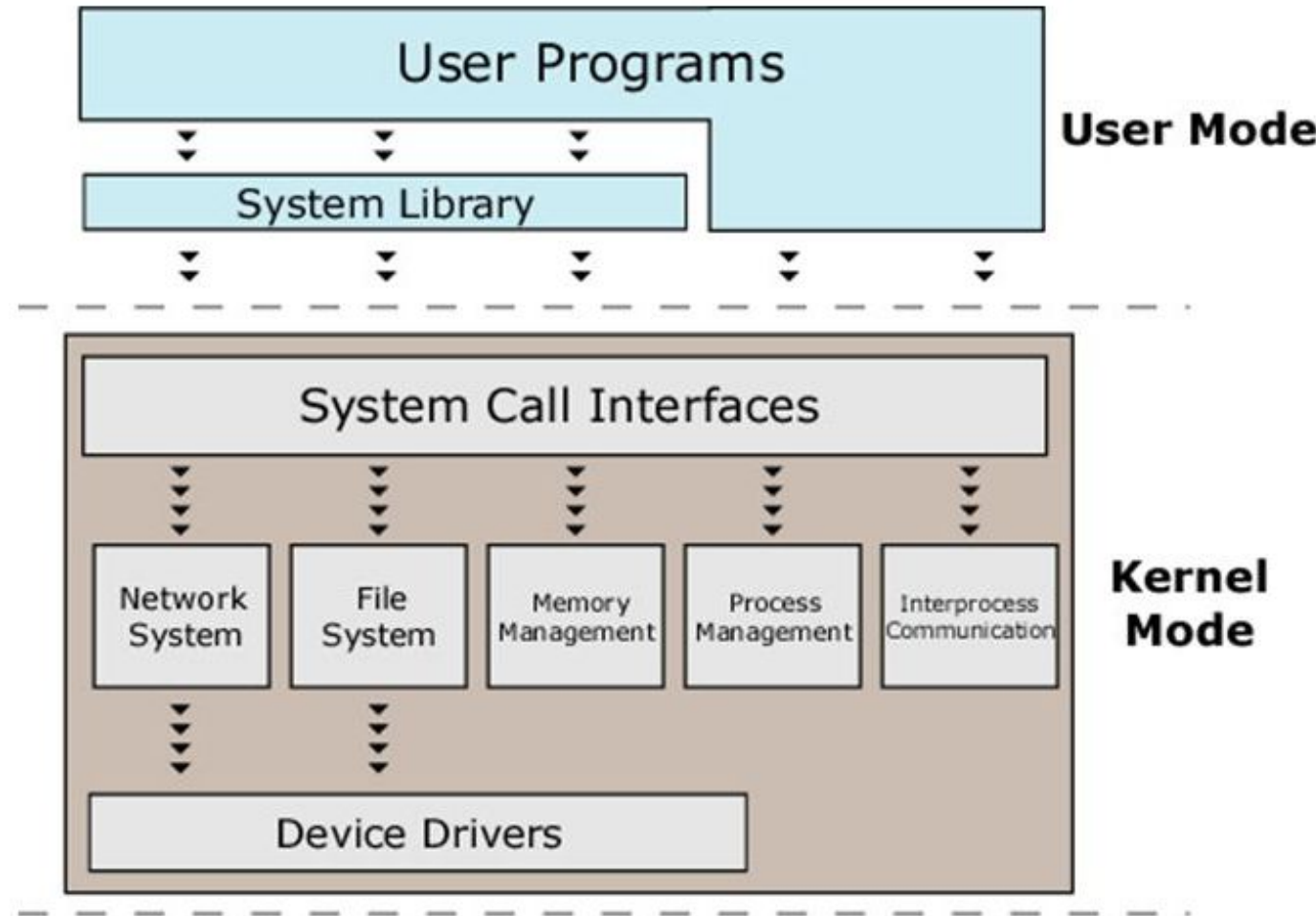
Modern Operating System- Linux

- A few kernel modules are loaded when the system is booted.

- A new kernel module is loaded dynamically when needed

- However, it has to be integrated with the kernel modules that already existed in memory so that the modules can collectively function as a monolithic kernel.
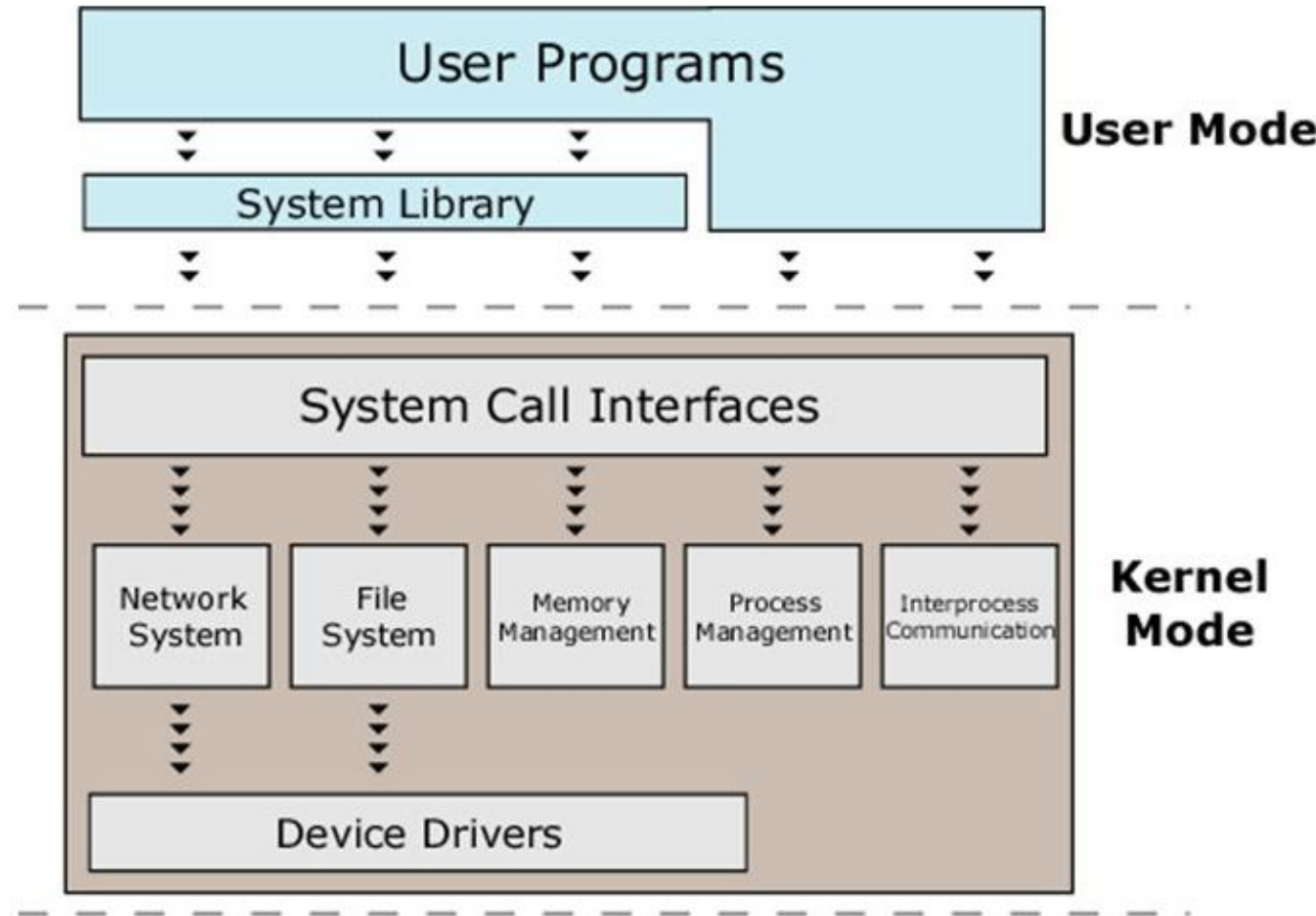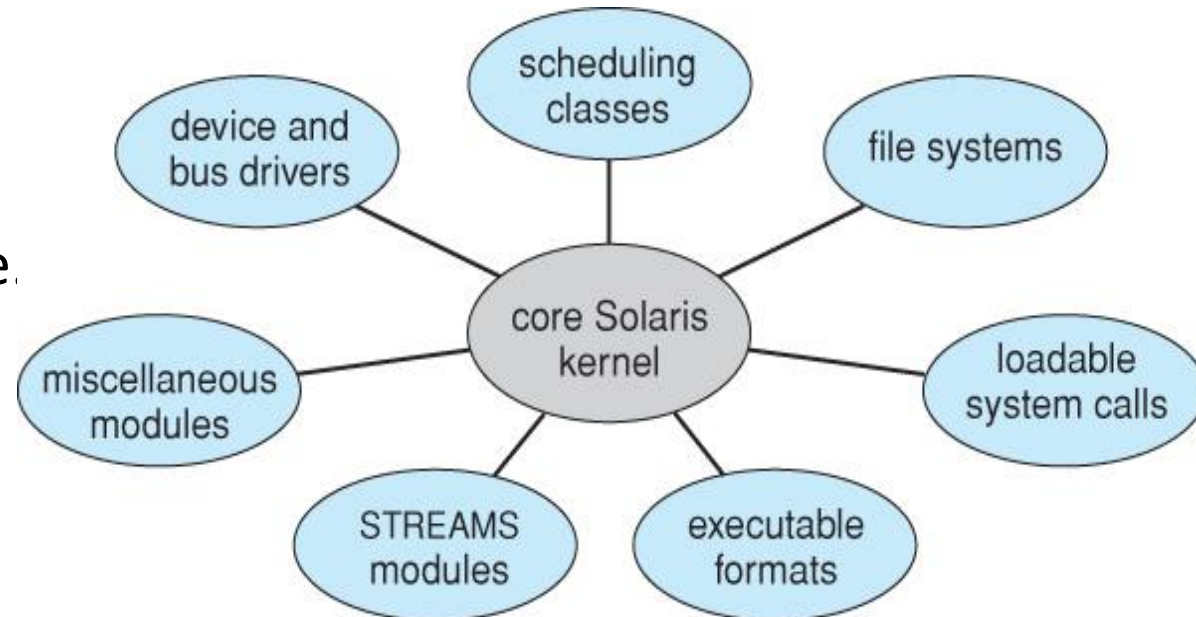
Modern Operating System- Solaris

- The Solaris kernel has an abstract machine layer that supports a wide range of processor architectures of the SPARC and Intel 80x86 family, including multiprocessor architectures.

- The pre-SVR4 versions of the OS are called SunOS, while the SVR4-based and later versions are called Solaris.

- The kernel is fully preemptible and provides real-time capabilities.

- Solaris 7 employs the kernel-design methodology of dynamically loadable kernel modules.

- The kernel has a core module that is always loaded; it contains interrupt servicing routines, system calls, process and memory management, and a virtual file system framework that can support different file systems concurrently.
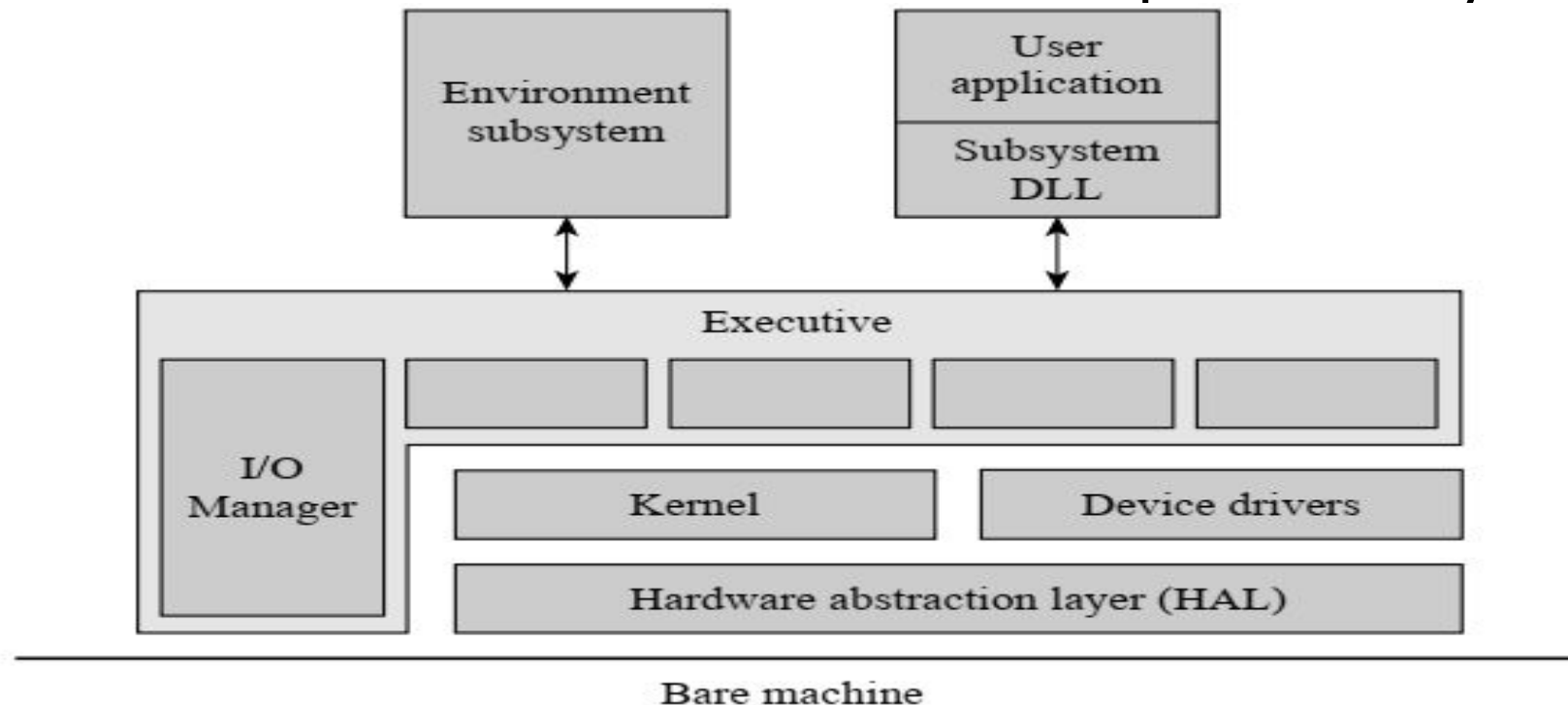
Modern Operating System- Solaris

- The kernel maintains a symbol table containing information about symbols defined in currently loaded kernel modules. This information is used while loading and linking a new module. New information is added to the symbol table after a module is loaded and some information is deleted after a module is deleted.

- The Solaris kernel supports seven types of loadable modules:
  - ✔ Scheduler classes
  - ✔ File systems
  - ✔ Loadable system calls
  - ✔ Loaders for different formats of exe. file.
  - ✔ Streams modules
  - ✔ Bus controllers and device drivers
 - ✔ Miscellaneous modules

Modern Operating System- Solaris

- The hardware abstraction layer (HAL) interfaces with the bare machine and provides abstractions of the I/O interfaces, interrupt controllers, and interprocessor communication mechanisms in a multiprocessor system.

- The HAL and the kernel are together equivalent to a conventional kernel. A device driver also uses the abstractions provided by the HAL to manage I/

Environment subsystem

User application

Subsystem DLL

Executive

I/O Manager

Kernel

Device drivers

Hardware abstraction layer (HAL)

Bare machine

Modern Operating System- Solaris

- The kernel performs the process synchronization and scheduling functions.

- The executive comprises nonkernel routines of the OS; its code uses facilities in the kernel to provide services such as process creation and termination, virtual memory management, an interprocess message passing facility for client–server communication.

Bare machine
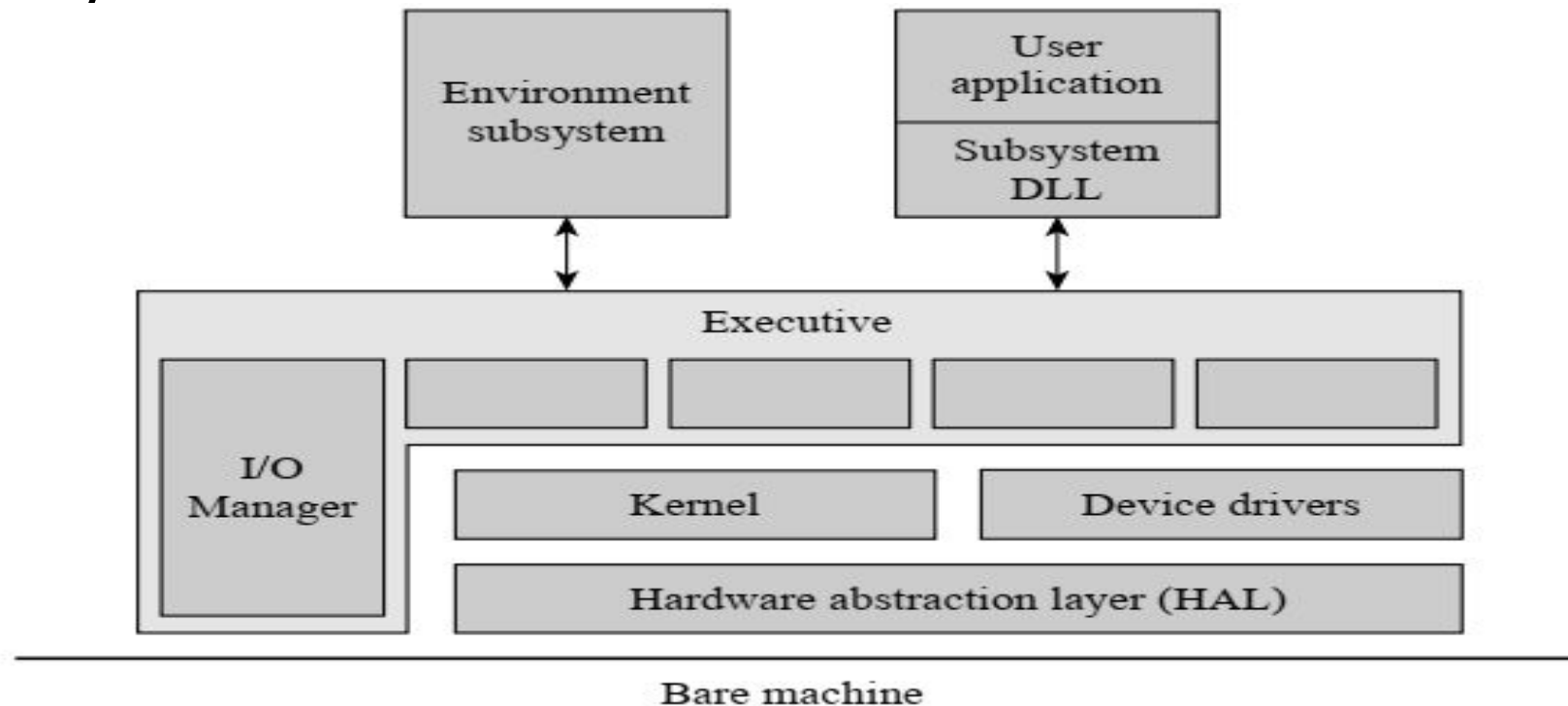
Modern Operating System- Solaris

- The I/O manager uses device drivers, which are loaded dynamically when needed.

- Many functions of the executive operate in the kernel mode, thus avoiding frequent context switches when the executive interacts with the kernel; it has obvious performance benefits.