Unit- I
Introduction to OS and services

Mr. S. C. Sagare

Lecture-1.1

Mr. S. C. Sagare

Part 1: Introduction to OS

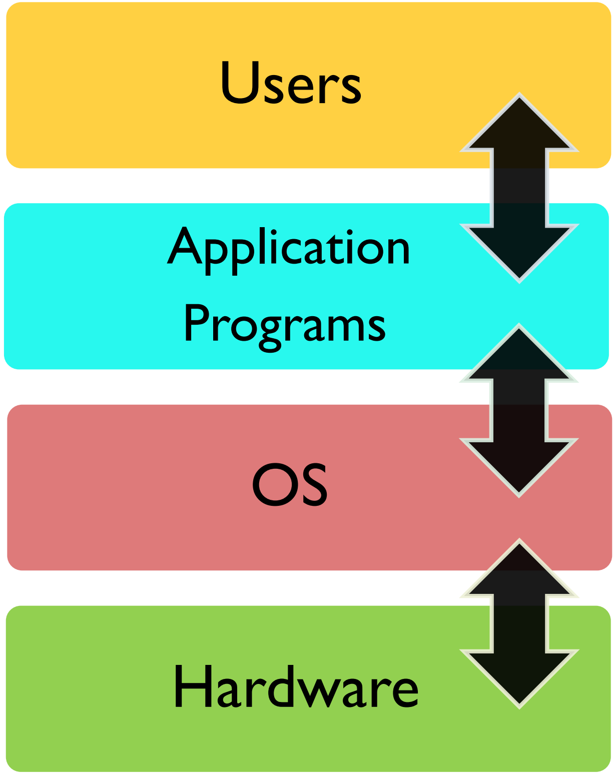
Mr. S. C. Sagare

CONTENTS

- Operating System fundamentals
- Computer System Organization
- Computer System Architecture
- Operating System Structure
- Operating System Operations
- Process Management
- Memory Management
- Storage Management
- Computing Environments

What is an Operating System?

- An **operating system** is a program that manages a computer's hardware. It also provides a basis for application programs and acts as an intermediary between the computer user and the computer hardware.



Operating system goals

- Some operating systems are designed to be ***convenient,*** others to be ***efficient,*** and others to be some combination of the two.
  - Personal computer (PC) operating systems support complex games, business applications, and everything in between.
  - Operating systems for mobile computers provide an environment in which a user can easily interface with the computer to execute programs.
  - Mainframe operating systems are designed primarily to optimize utilization of hardware. i.e. to use the computer hardware in an efficient manner.

What Operating Systems Do?

- A Computer system can be divided into four components
  - **Hardware** – provides basic computing resources for the system
    - Central Processing Unit (CPU),
    - Memory,
    - Input/Output (I/O) devices
  - **Operating system**
    - Controls the hardware and coordinates its use among the various application programs for the various users.
  - **Application programs** –
    - Define the ways in which the system resources are used to solve the computing problems of the users
    - Word processors, web browsers, database systems, video games
  - **Users**
    - People, other computers or machines.

Components of a Computer System

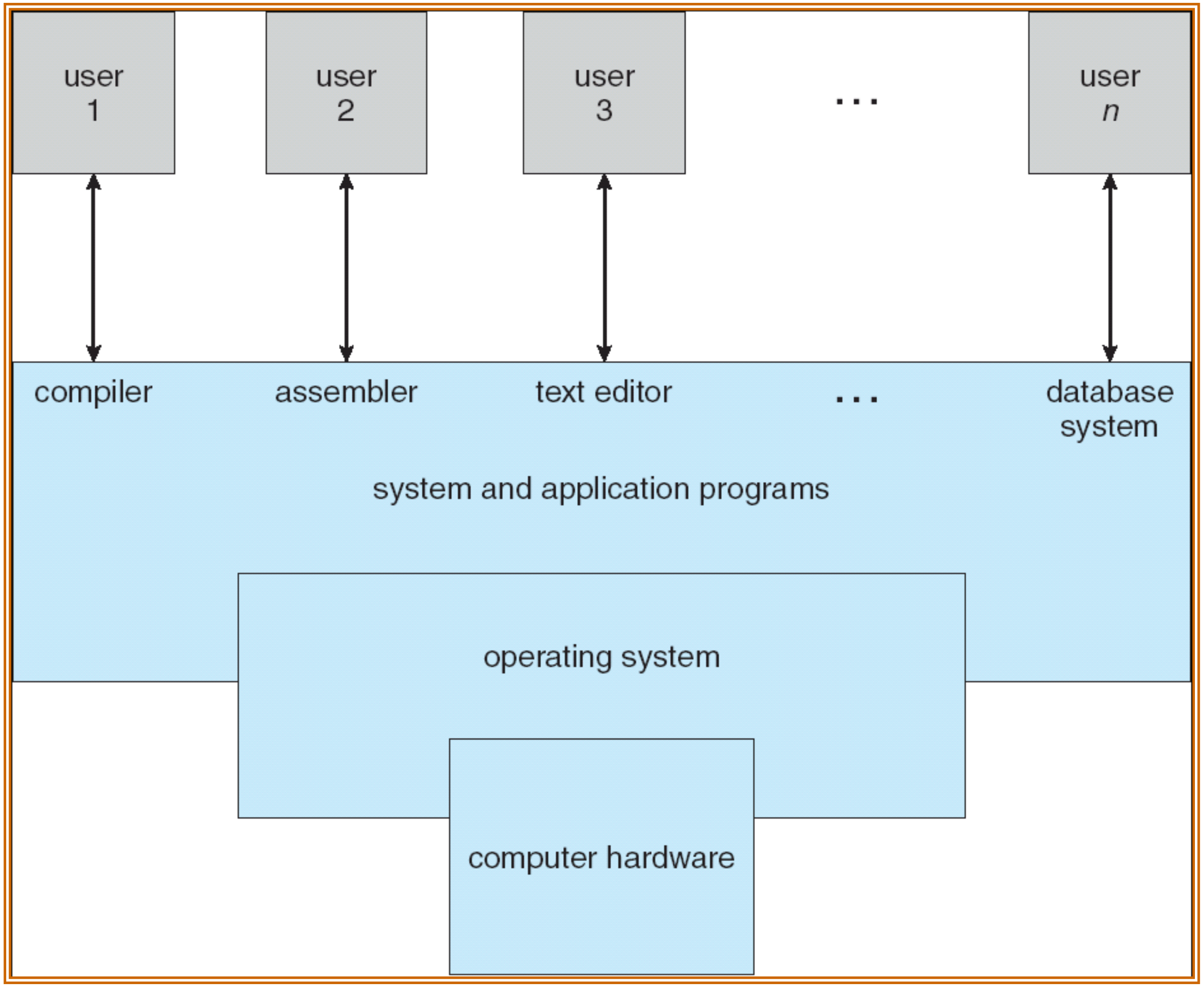


Figure 1 Abstract view of the components of a computer system.

Operating Systems Viewpoints

- **User View:**
- The user’s view of the computer varies according to the interface being used.
  - Most computer users sit in front of a Personal computers.
    - Such a system is designed for one user to control its resources.
    - The goal is to maximize the work that the user is performing.
    - In this case, the operating system is designed mostly for **ease of use**, with some attention paid to **performance** and no attention paid to **resource utilization** (how hardware and s/w resources are shared)
    - Performance is, of course, important to the user; but such systems are optimized for the single-user experience rather than the requirements of multiple users.

- In other cases, a user sits at a terminal connected to mainframe or a minicomputer.
  - Here, other users are accessing the same computer through other terminals. These users share resources and may exchange information.
  - The operating system in such cases is designed to maximize resource utilization i.e. to maximize hardware and software resource sharing.

- In still other cases, users sit at workstations connected to networks of other workstations and servers.
  - These users have dedicated resources at their site, but they also share resources such as networking and servers, including file, compute, and print servers.
  - Therefore, their operating system is designed to compromise between individual usability and resource utilization.

Operating Systems Viewpoints

- **System View:**
  - From the computer's point of view, the operating system is the program most closely involved with the hardware.
  - In this context, we can view an operating system as a **resource allocator**.
    - A computer system has many resources that may be required to solve a problem: CPU time, memory space, file-storage space, I/O devices, and so on
    - The operating system acts as the manager of these resources.
    - Facing numerous and possibly conflicting requests for resources, the operating system must decide how to allocate them to specific programs and users so that it can operate the computer system efficiently and fairly.
    - As we have seen, resource allocation is especially important where many users access the same mainframe or minicomputer.
  - An operating system is a control program.
    - A **control program manages the execution of user programs** to prevent errors and improper use of the computer.
    - It Is especially concerned with the operation and control of I/O devices.

Lecture-1.2

Mr. S. C. Sagare

Part 1: Introduction to OS

Mr. S. C. Sagare

CONTENTS

- Operating System fundamentals
- Computer System Organization
- Computer System Architecture
- Operating System Structure
- Operating System Operations
- Process Management
- Memory Management
- Storage Management
- Computing Environments

Operating System Definition

- No universally accepted definition

- A more common definition is:

- “Operating system is the one program running at all times on the computer—usually called the **kernel**.

  Along with the kernel, there are two other types of programs:

- **System programs**, which are associated with the operating system but are not necessarily part of the kernel, and
- **Application programs**, which include all programs not associated with the operation of the system.”

Operating System Definition

- The kernel is the main part of the operating system and is responsible for translating the command into something that can be understood by the computer.

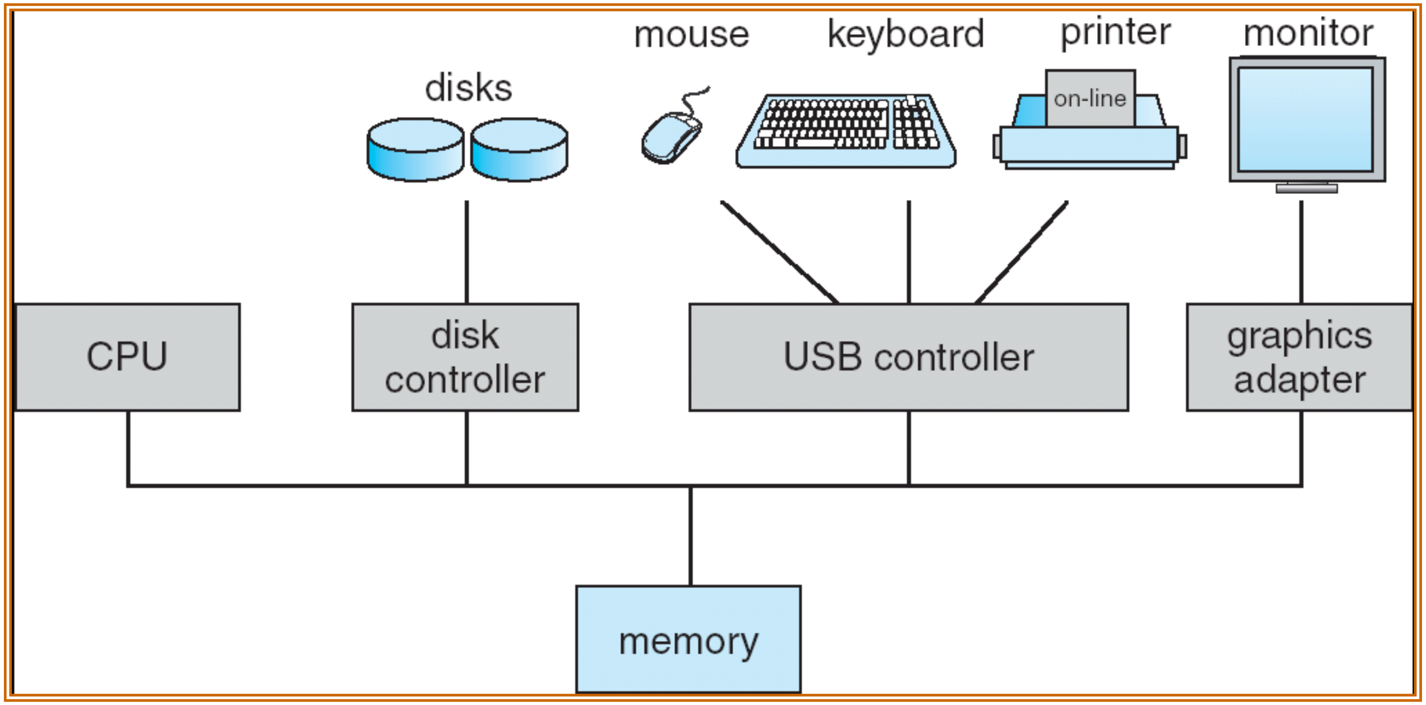
- The kernel is the lowest level of the operating system.

- The main functions of the kernel are:
  - Memory management
  - Network management
  - Device driver
  - File management
  - Process management

- User Problems → Hardware
  - E. g. typing & printing a document.
- User Problems → APs → Hardware
  - E. g. MS word
- User Problems → APs → OS → Hardware
  - E. g. Windows os providing device control & resource allocation
- OS= Common functions + Various Additional features
- **Common functions** [providing device control , resource allocation, etc.]
- **Additional features** [middleware that supports databases, multimedia, and graphics]
- "Operating system is the one program running at all times on the computer—usually called the **kernel**.
- OS → Kernel
- → System programs[SP]
- → Application programs[AP]

Computer System Organization

- Computer-system operation
  - One or more CPUs, device controllers connect through common bus providing access to shared memory
  - Concurrent execution of CPUs and devices competing for memory cycles



Computer-System Operation

- For a computer to start running—for instance, when it is powered up or rebooted, it **needs to have an initial program** to run.

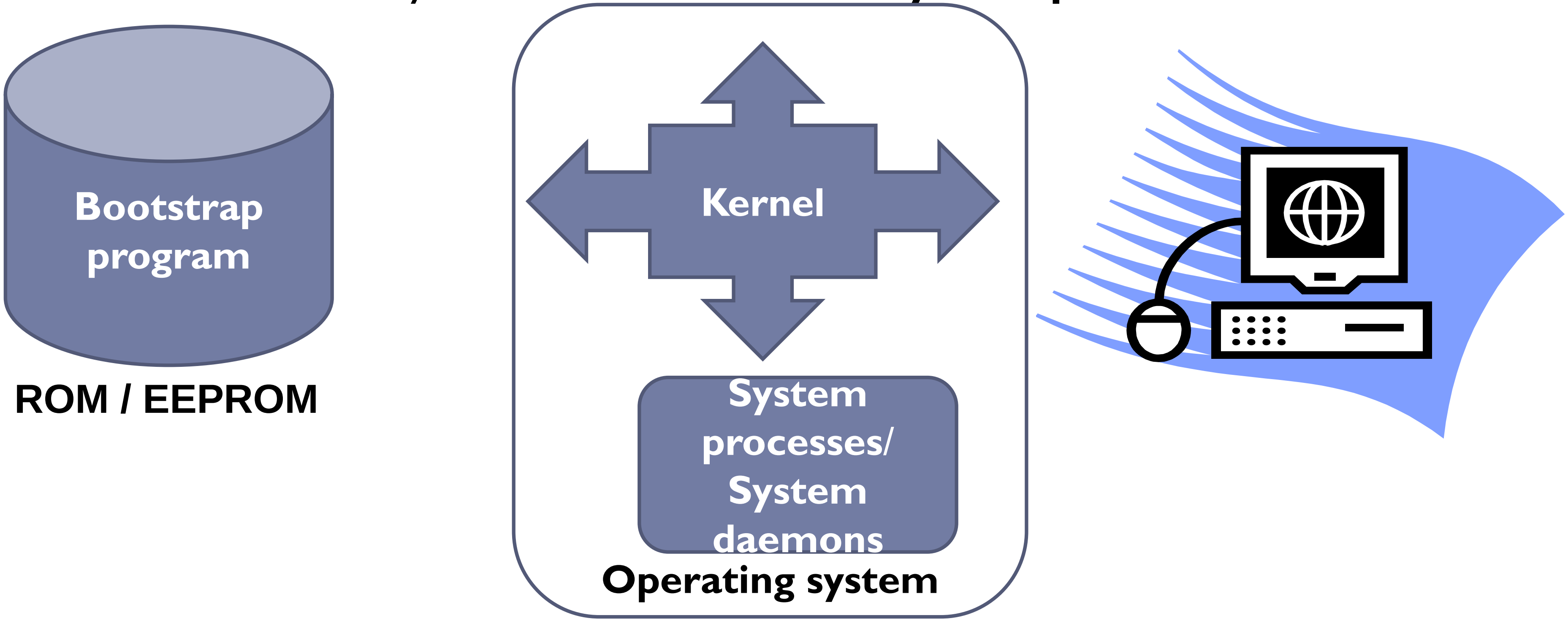
- This initial program, or **bootstrap program**, tends to be simple.

- Typically, it is stored within the computer hardware in read-only memory (**ROM) or Electrically Erasable Programmable Read-Only Memory** (**EEPROM).**

- It initializes all aspects of the system, from CPU registers to device controllers to memory contents.

- The bootstrap program must know how to **load the operating system** and how to start executing that system.

- To accomplish this goal, the bootstrap program must **locate the operating-system kernel and load it into memory.**
- Once the kernel is loaded and executing, it can start providing services to the system and its users.
- Some services are provided outside of the kernel, by system programs that are loaded into memory at boot time to become **system processes/ daemons**



Bootstrap program

ROM / EEPROM

Kernel

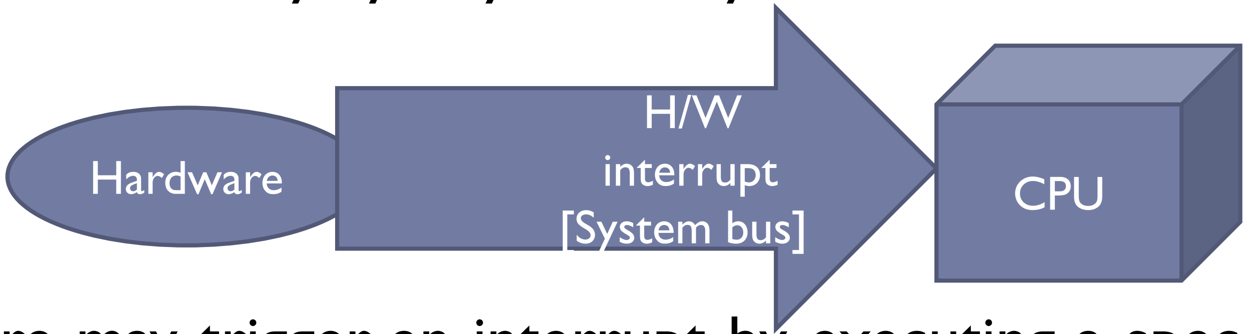
System processes/ System daemons

Operating system

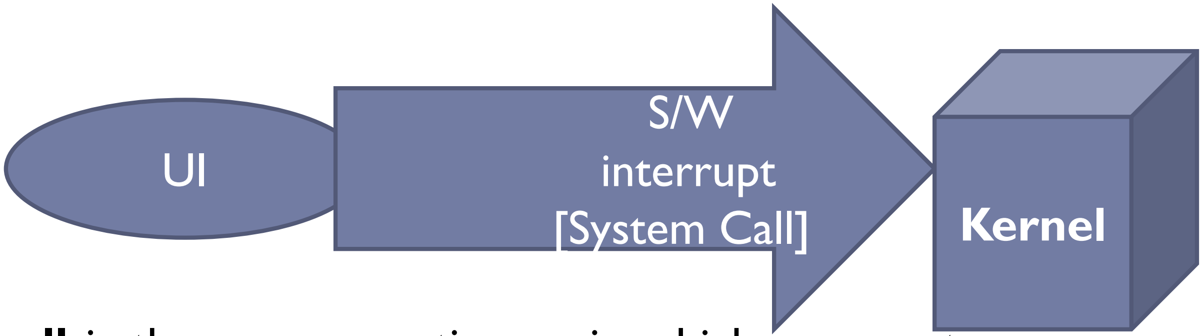
- Once this phase is complete, then the system is fully booted, and it waits for some event to occur.

- 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.

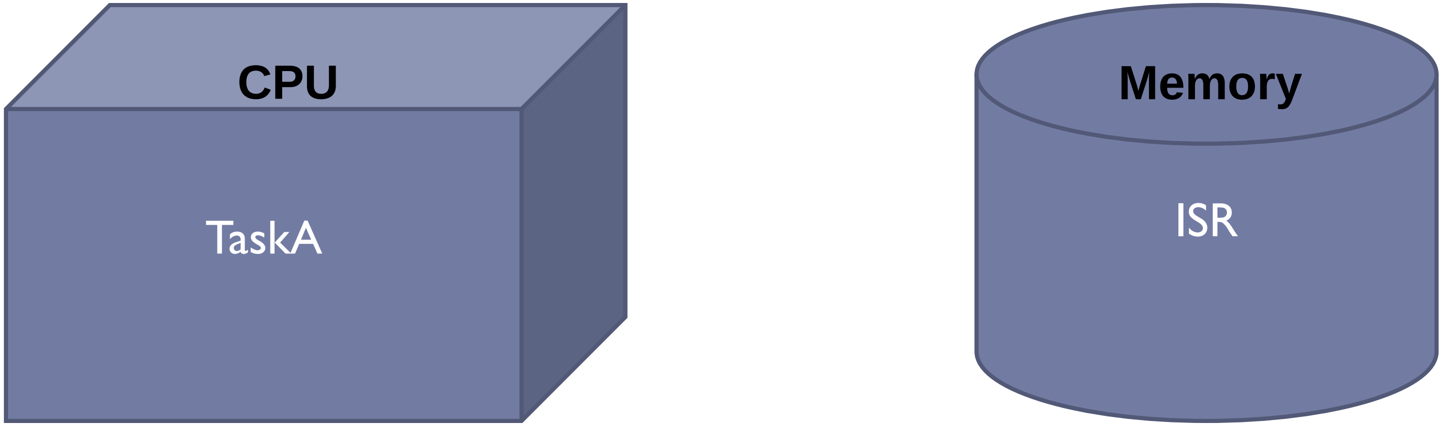


- **System call** is the programmatic way in which a computer program requests a service from the kernel of the **operating system** it is executed on.

- When the CPU is interrupted, it stops what it is doing and immediately transfers execution to a fixed location.

- The fixed location usually contains the starting address where the service routine for the interrupt is located.

- The interrupt service routine executes; on completion, the CPU resumes the interrupted computation.



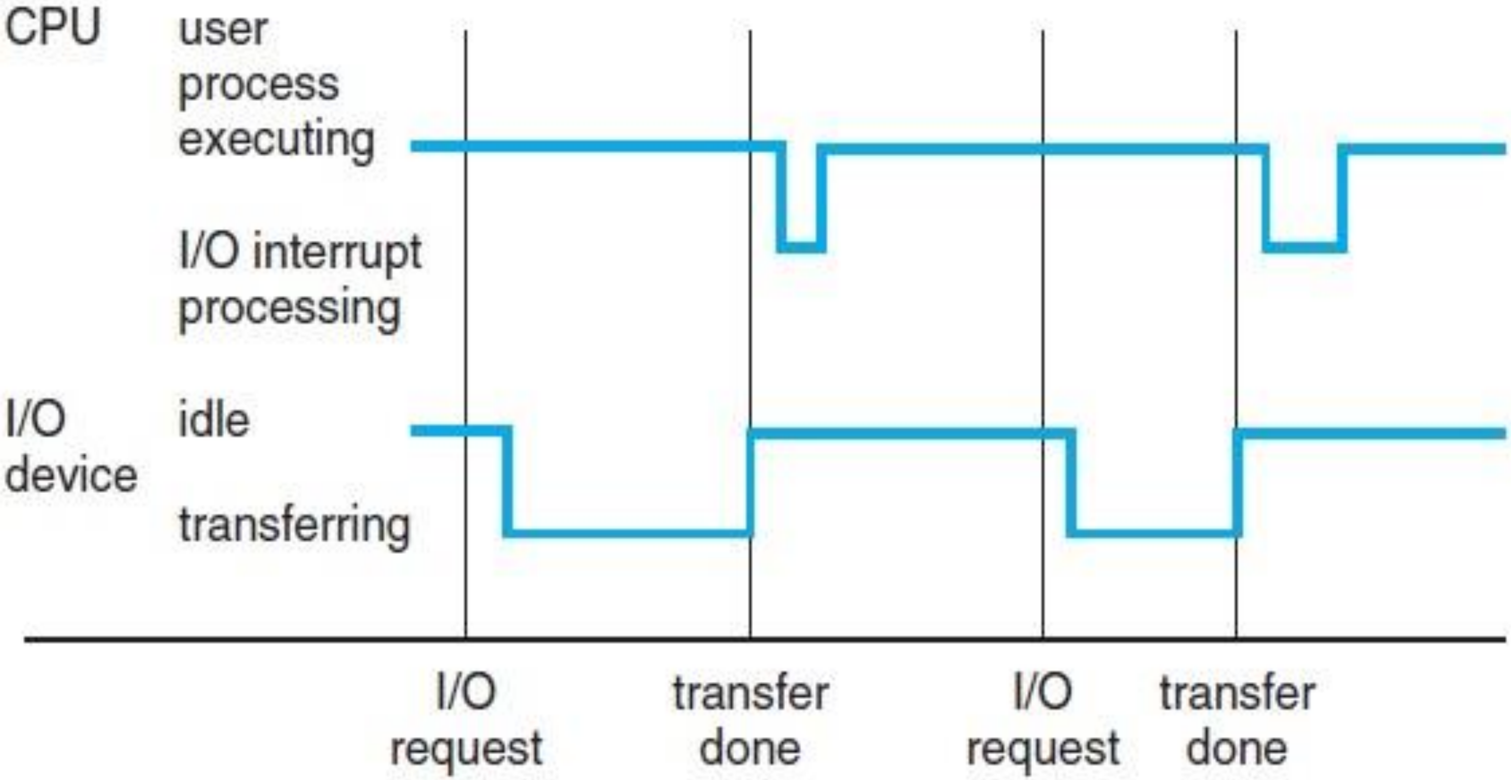


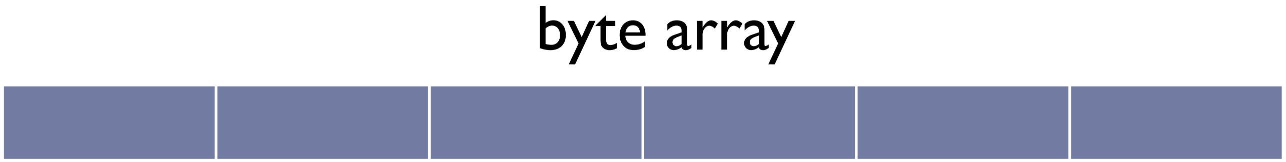
Figure 1.3 Interrupt timeline for a single process doing output.

Storage Structure

- The CPU can load instructions only from **main memory** (or RAM). So any programs to run must be stored there.
- Main memory is implemented commonly in a semiconductor technology called **dynamic random-access memory** (DRAM).
- Computers use other forms of memory as well. read-only memory, ROM) and electrically erasable programmable read-only memory, EEPROM),etc.
- Because ROM **cannot be changed**, only static programs, such as the **bootstrap program**, are stored there.
- EEPROM can be changed, but not frequently and so contains mostly static programs. For example,
  - Smartphones have EEPROM to store their factory-installed programs

Storage Structure

- All forms of memory provide an **array of bytes**. Each byte has its own address.



- Interaction is achieved through a sequence of **load or store** instructions to specific memory addresses.
  - The **load** instruction moves a byte or word from:

    Main memory to → internal register within the CPU
  - The **store** instruction moves :

    Content of a register to →Main memory

- Ideally, we want the programs and data to reside in main memory permanently.

- This is not possible for the following two reasons:
  - Main memory is usually too small to store all needed programs and data permanently.

  - Main memory is a volatile storage device that loses its contents when power is turned off or otherwise lost.

- Most systems provide **secondary storage** as an extension of main memory which can hold large quantities of data permanently.

- The most common secondary-storage device is a **magnetic disk,** which provides storage for both programs and data.

- Most programs (system and application) are stored on a disk until they are loaded into main memory.

- Many programs then use the disk as both the source and the destination of their processing.

The main differences among the various storage systems lie in speed, cost, size, and volatility.

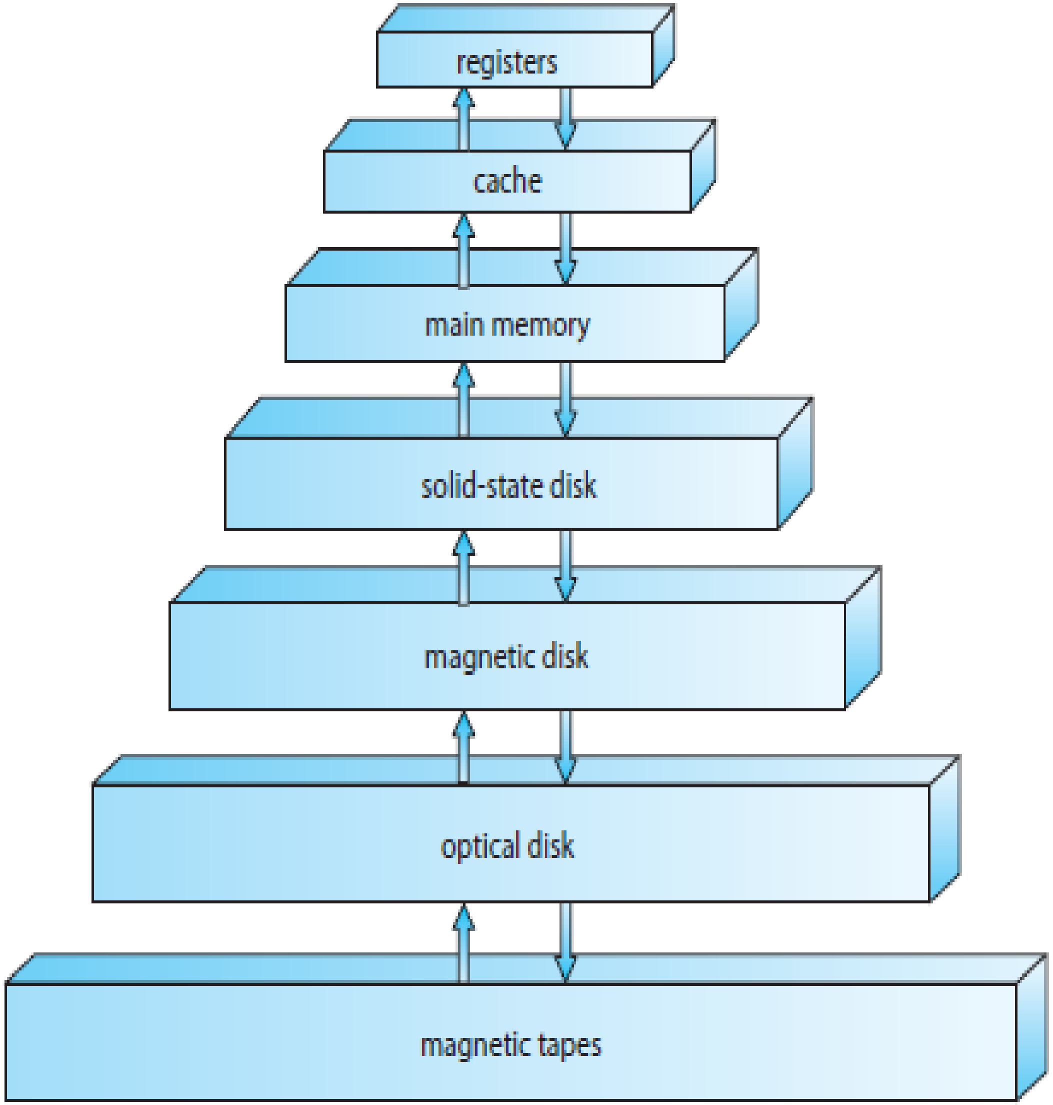


Figure 1.4 Storage-device hierarchy.

Lecture-1.3

Mr. S. C. Sagare

Part 1: Introduction to OS

Mr. S. C. Sagare

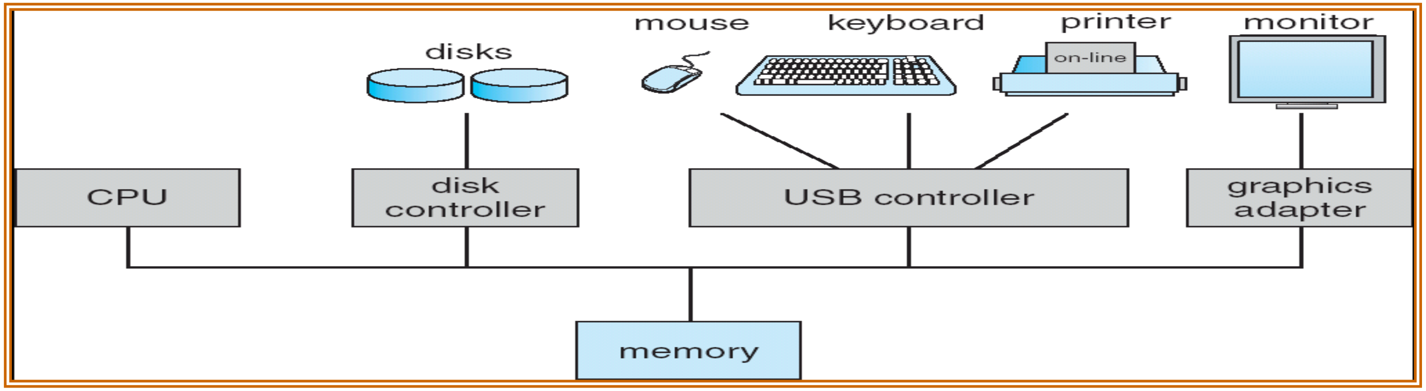
CONTENTS

- Operating System fundamentals
- Computer System Organization
- Computer System Architecture
- Operating System Structure
- Operating System Operations
- Process Management
- Memory Management
- Storage Management
- Computing Environments

I/O Structure

- Storage is only one of many types of I/O devices within a computer.

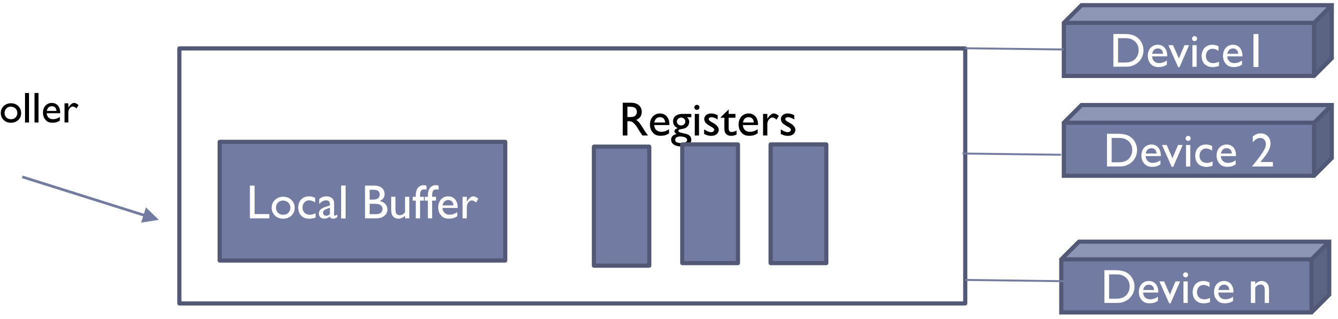
- A large portion of operating system code is dedicated to managing I/O.
- A general-purpose computer system consists of CPUs and multiple device controllers that are connected through a common bus.



- Each device controller is in charge of a specific type of device. Depending on the controller, more than one device may be attached.

- A device controller maintains some local buffer storage and a set of special-purpose registers.

- Device controller

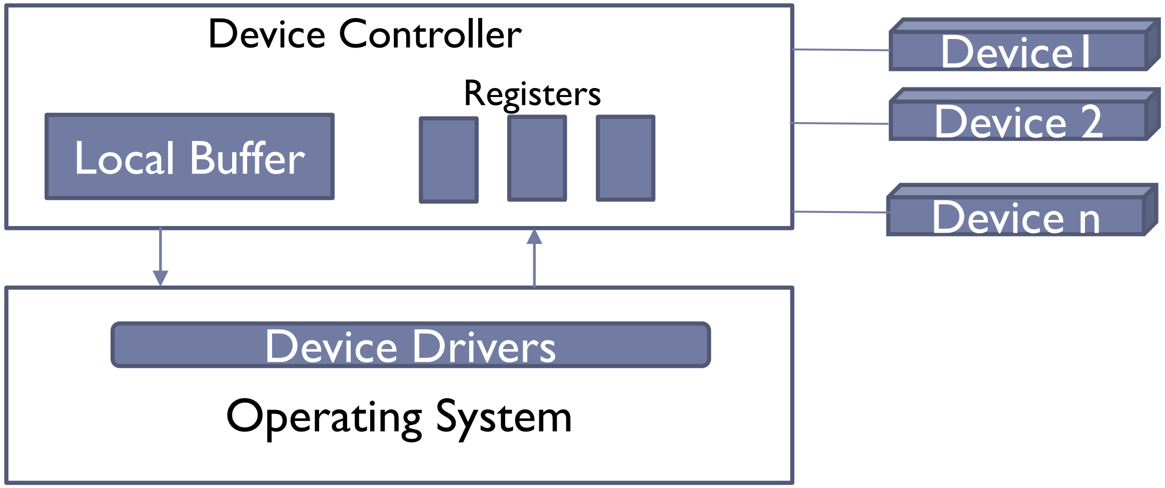


- The device controller is responsible for moving the data between the peripheral devices that it controls and its local buffer storage.

- Typically, operating systems have a **device driver for each device controller.**

- This device driver understands the device controller and provides the rest of the operating system with a uniform interface to the device.

- To start an I/O operation, the device driver loads the appropriate registers within the device controller.
- The device controller, in turn, examines the contents of these registers to determine what action to take.
- The controller starts the transfer of data from the device to its local buffer.



- Once the transfer of data is complete, the device controller informs the device driver via an interrupt that it has finished its operation.
- The device driver then returns control to the operating system:
  - Returns the data or a pointer to the data if its read operation.
  - For other operations, returns status information.

- This form of interrupt-driven I/O is fine for moving small amounts of data, but can produce high overhead when used for bulk data movement such as disk I/O.
- To solve this problem, **direct memory access (DMA)** is used.

- Device controller transfers an entire block of data directly to or from its own buffer storage to memory, with no intervention by the CPU.

- Only one interrupt is generated per block, to tell the device driver that the operation has completed, rather than the one interrupt per byte generated for low-speed devices

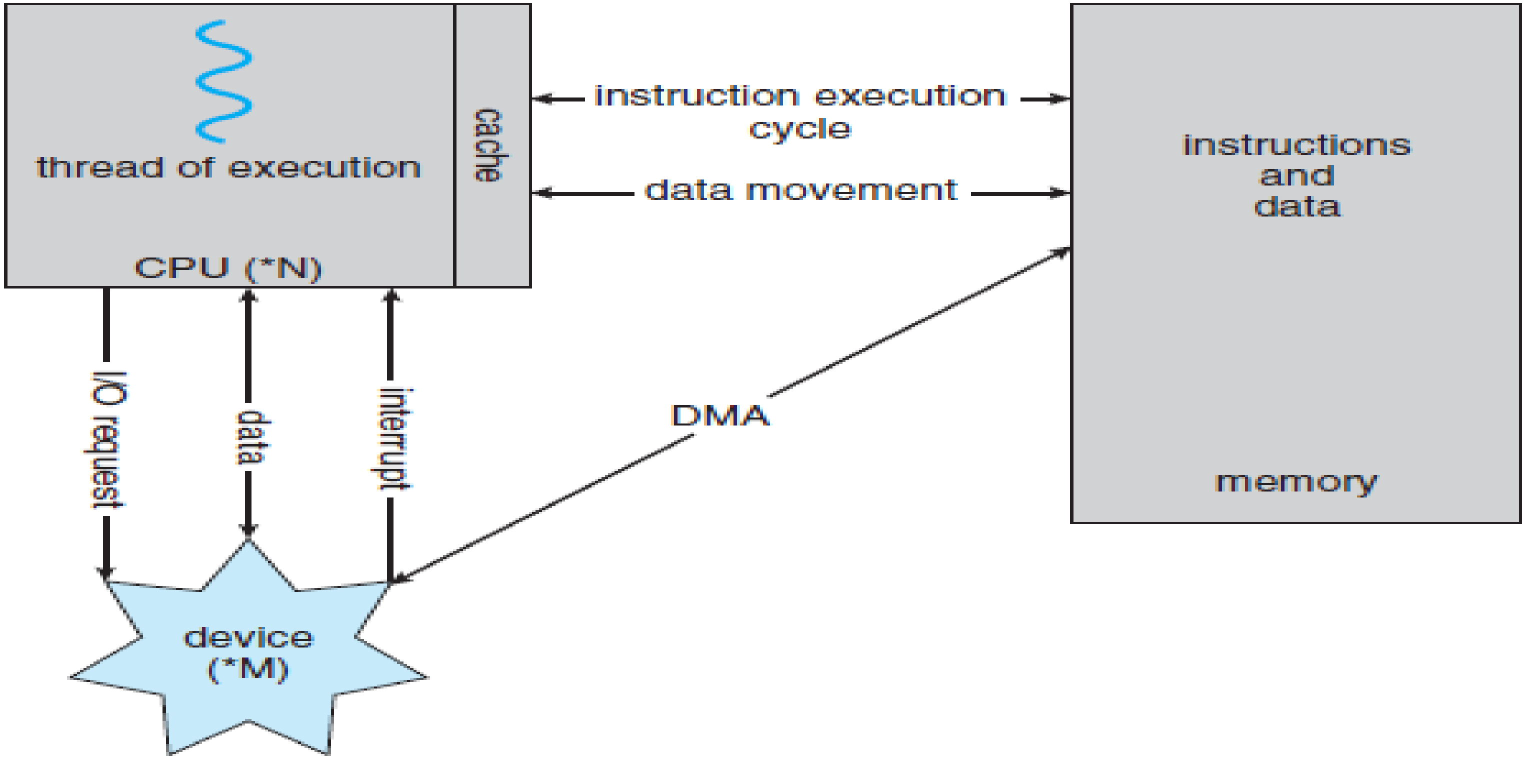


Figure 1.5 How a modern computer system works.

Computer-System Architecture

- Single-Processor operating Systems:-
  - They performs only one process at a given time.
  - And it carries out next process in the queue only after the current process is completed.
  - OS monitors the status and also sends them next executable instruction.

- Multiprocessor operating Systems:-
  - This refers to the use of two or more central processing units within a single computer system.
  - These multiple CPU's are in close communication sharing the computer bus, memory and other peripheral devices.
  - These types of systems are used when very high speed is required to process a large volume of data.

- Multiprocessor systems have three main advantages:

  - **Increased throughput:-**
  - By increasing the number of processors, we expect to get more work done in less time.

  - **Economy of scale:--**
  - Multiprocessor systems can cost less than equivalent multiple single-processor systems, because they can share peripherals, mass storage, and power supplies.

  - **Increased reliability:--**
  - If functions can be distributed properly among several processors, then the failure of one processor will not halt the system.

- The multiple-processor systems in use today are of two types:-
- **Asymmetric multiprocessing:-** in which each processor is assigned a specific task.
  - A **boss** processor controls the system; the other processors either look to the boss for instruction or have predefined tasks.
  - This scheme defines a **boss–worker relationship.**
  - The boss processor schedules and allocates work to the worker processors.

- **Symmetric multiprocessing:-** in which each processor performs all tasks within the operating system.
  - SMP means that all processors are peers; no boss–worker relationship exists between processors.
  - Notice that each processor has its own set of registers, as well as a private or local cache.
  - However, all processors share physical memory

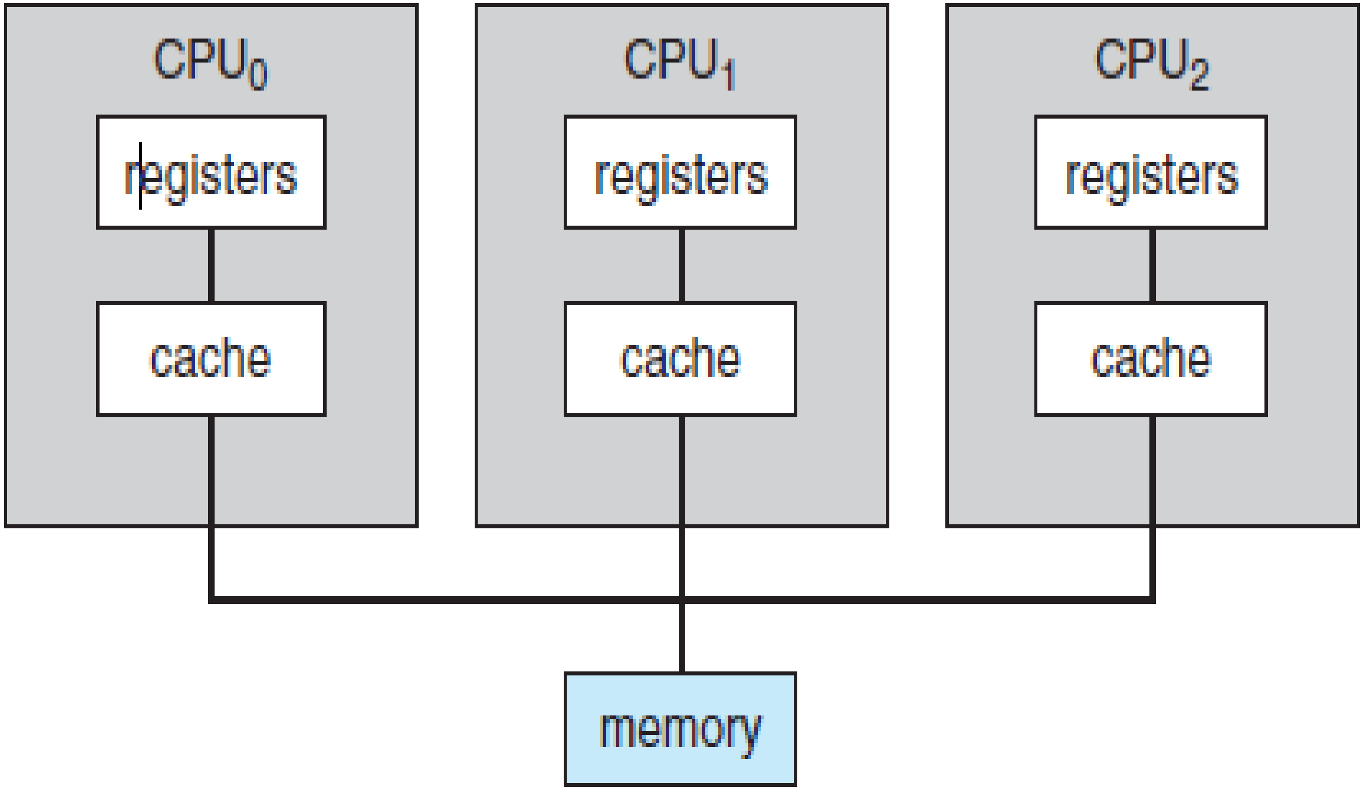
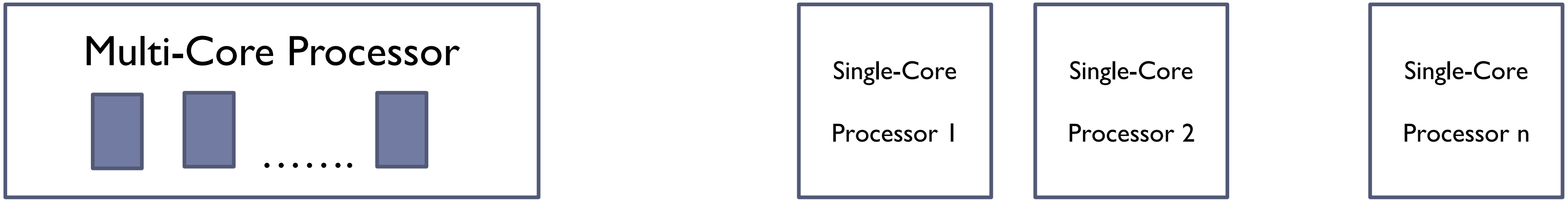


Figure 1.6 Symmetric multiprocessing architecture.

- **A multi-core processor** is a single computing component with two or more independent processing units called **cores**, which read and execute program instructions
- They can be more efficient than multiple chips with single cores because on-chip communication is faster than between-chip communication.



- In addition, one chip with multiple cores uses significantly less power than multiple single-core chips.
- It is important to note that while multicore systems are multiprocessor systems, not all multiprocessor systems are multicore.

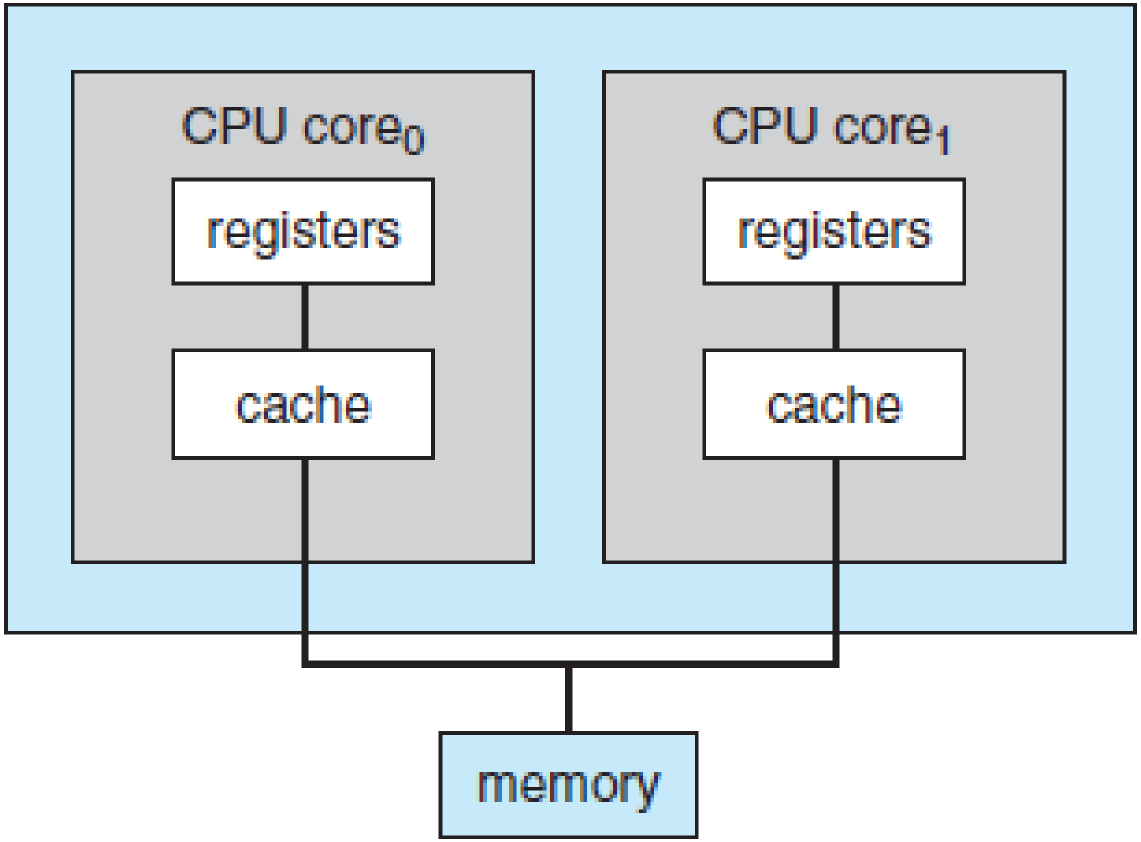


Figure 1.7 A dual-core design with two cores placed on the same chip.

- **Clustered Systems:-**
  - Another type of multiprocessor system is a clustered system, which gathers together multiple CPUs.
  - Clustered systems differ from the multiprocessor systems in that they are composed of two or more individual systems or nodes joined together.
  - Such systems are considered loosely coupled.
  - Each node may be a single processor system or a multi-core system.
  - Clustering is usually used to provide high-availability service that is, service will continue even if one or more systems in the cluster fail.

- Clustering can be structured asymmetrically or symmetrically.

- In **asymmetric clustering, one machine is in hot-standby mode while the other is** running the applications.

- The hot-standby host machine does nothing but monitor the active server.

- If that server fails, the hot-standby host becomes the active server.

- In **symmetric clustering, two or more hosts are running** applications and are monitoring each other.

- This structure is obviously more efficient, as it uses all of the available hardware

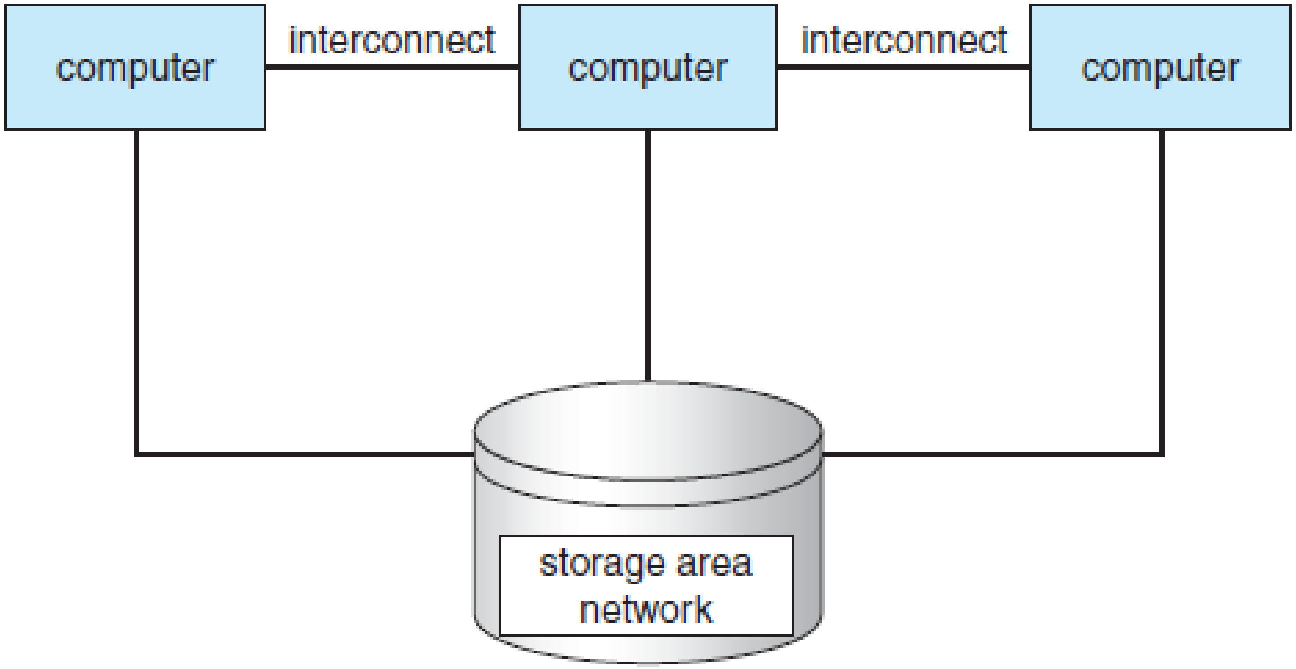


Figure 1.8 General structure of a clustered system.

Lecture-1.4

Mr. S. C. Sagare

Part 1: Introduction to OS

Mr. S. C. Sagare

CONTENTS

- Operating System fundamentals
- Computer System Organization
- Computer System Architecture
- Operating System Structure
- Operating System Operations
- Process Management
- Memory Management
- Storage Management
- Computing Environments

Operating System Structure

- **Multiprogramming:** One of the most Important aspects of OS
  - Single program cannot keep CPU and/or I/O devices busy at all times
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute
  - Jobs are kept initially on the disk in the job pool.
  - Job pool consists of all processes residing on disk awaiting allocation of main memory.
  - One job selected and run via **job scheduling,** When it has to wait (for I/O for example), OS switches to another job.

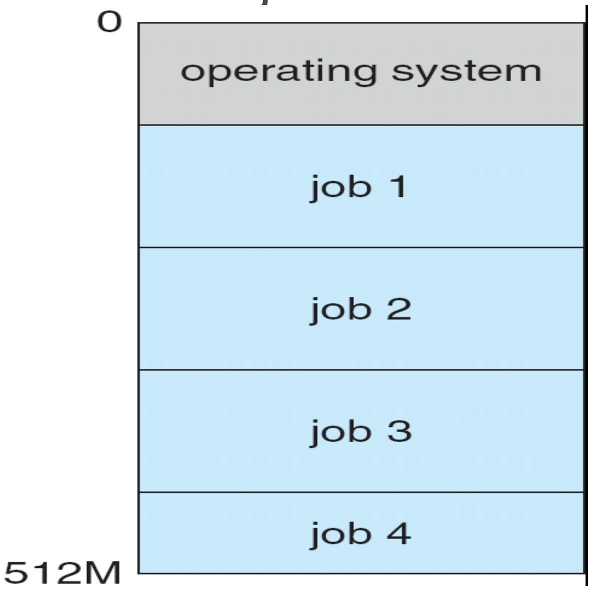
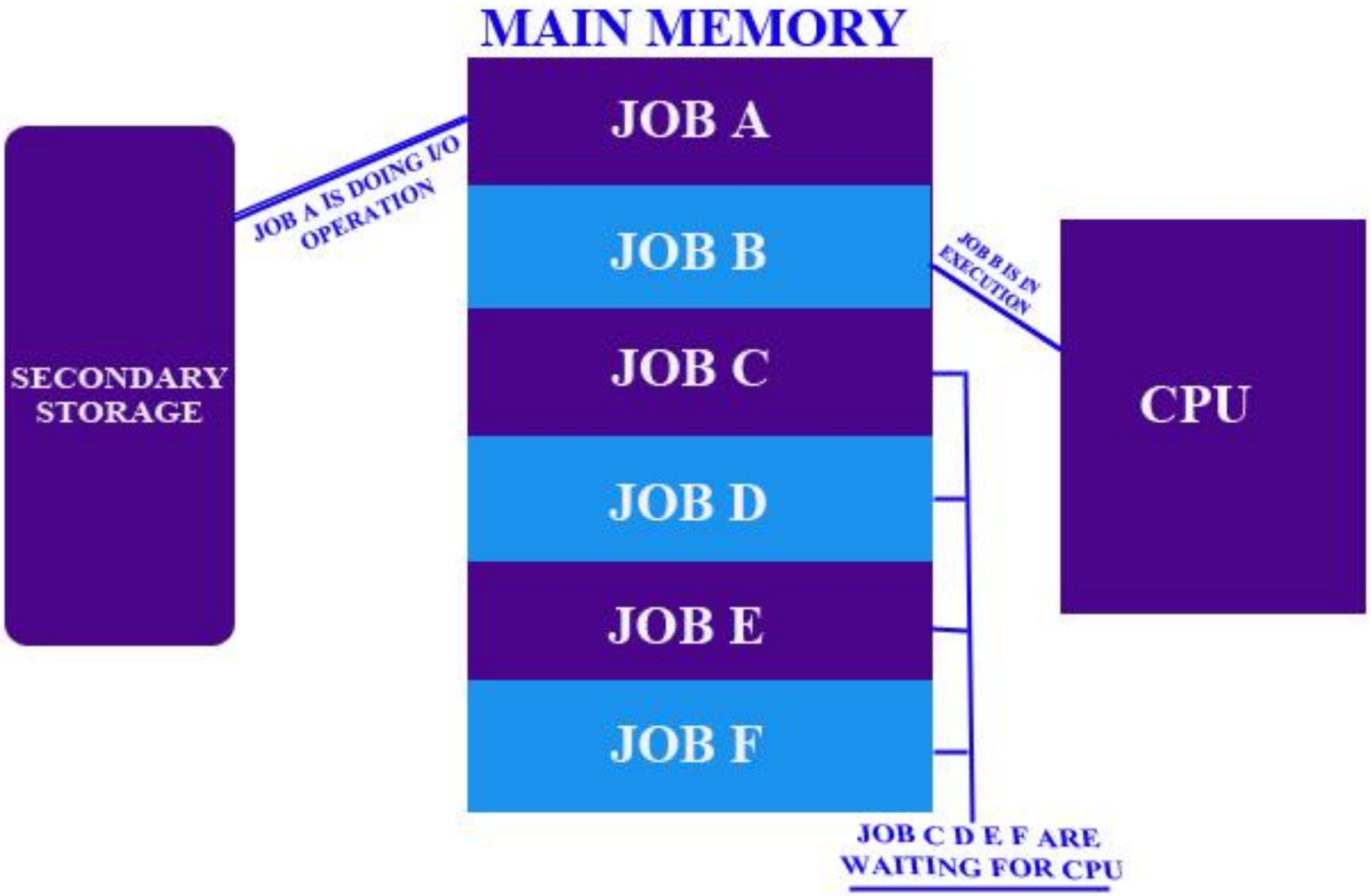


Fig. Memory layout for a multiprogramming system



Multiprogramming Operating System

Operating System Structure

- **Timesharing (multitasking)** is logical extension of multiprogramming in which CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing
  - **Response time** should be < 1 second
  - Each user has at least one program executing in memory ⇨ **Process**
  - If several jobs ready to run at the same time, system uses ⇨ **CPU scheduling**
  - If processes don't fit in memory, **swapping** moves them in and out to run
  - **Virtual memory** allows execution of processes that are not completely in memory
    - It enables users to run programs that are larger than actual physical memory.
  - A time-sharing system must also provide a **file system** that resides on a collection of disks; hence, **disk management** must be provided.
  - Also, time-sharing system must provide mechanisms for **job synchronization and communication**, ensure that jobs do not get stuck in a deadlock, forever waiting for one another.

Operating-System Operations

- **Modern operating systems are interrupt driven.**
  - If there are no processes to execute, no I/O devices to service, and no users to whom to respond, an operating system will sit quietly, waiting for something to happen.

  - **Events** are almost always signaled by the occurrence of an interrupt or a trap.
  - **Trap** is a software-generated interrupt caused
    - either by an error (e.g., division by zero or invalid memory access)
    - or by a specific request from a user program that an operating-system service be performed.

- **Interrupt Service Routines:** For each type of interrupt, these are the separate segments of code in the operating system that determine what action should be taken.

- A properly designed operating system must ensure that an incorrect (or malicious) program cannot cause other programs to execute incorrectly.

  - Since the operating system and the users share the hardware and software resources of the computer system, we need to make sure that an error in a user program could cause problems only for the one program running.

Dual-Mode and Multimode Operation

- In order to ensure the proper execution of the operating system, we must be able to distinguish between the execution of operating-system code and user defined code.

- The approach taken by most computer systems is to provide hardware support that allows us to differentiate among various modes of execution.

- At the very least, we need two separate ***modes of operation: user mode*** and **kernel mode (also called supervisor mode, system mode, or privileged mode).**

- A bit, called the **mode bit, is added to the hardware of the computer** to indicate the current mode: kernel (0) or user (1).

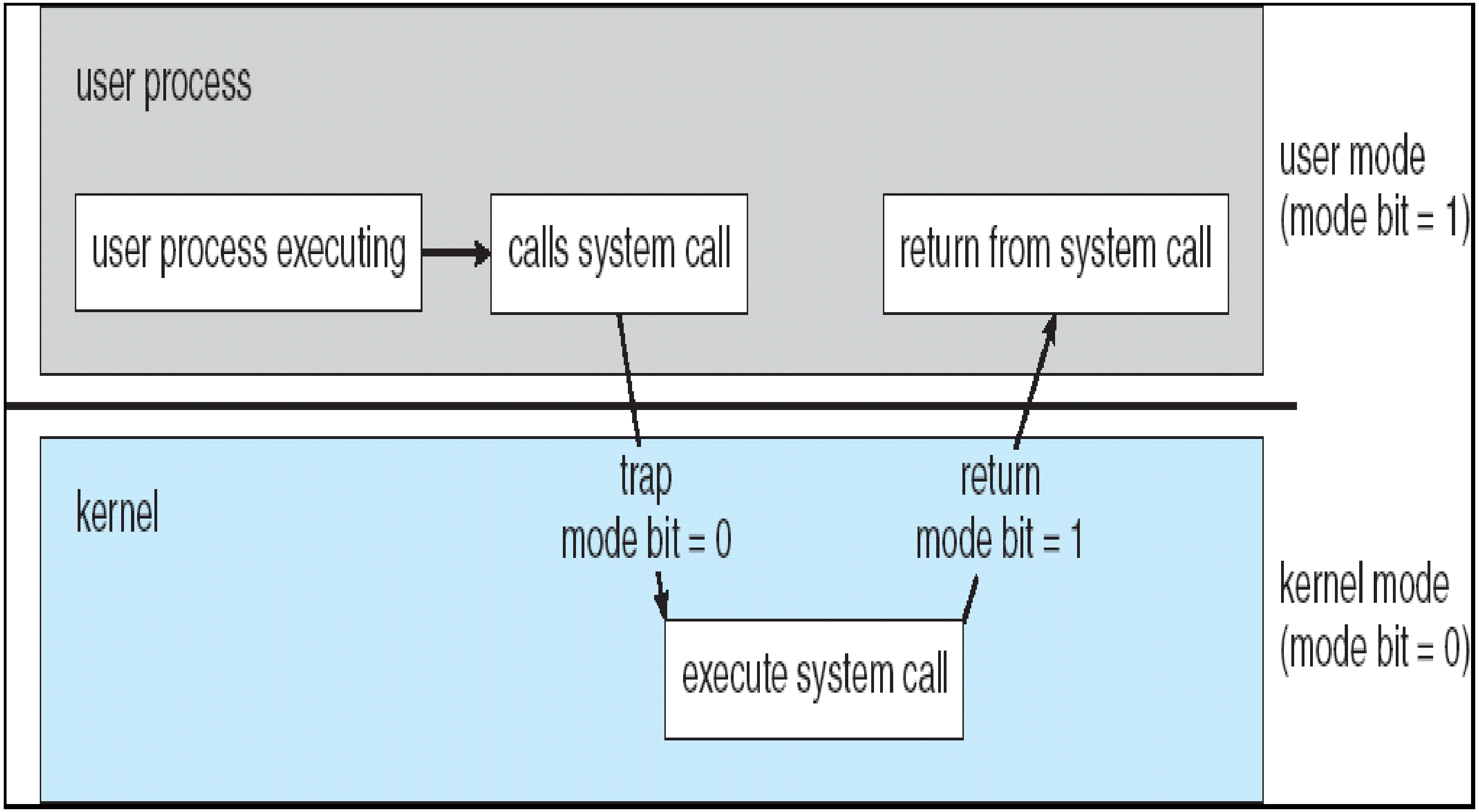
- With the mode bit, we can distinguish between a task that is executed on behalf of the operating system and one that is executed on behalf of the user.
- When the computer system is executing on behalf of a user application, the system is in **user mode**.

- However, when a user application requests a service from the operating system (via a system call), the system must transition from user to **kernel mode** to fulfill the request.

- At system boot time, the hardware starts in kernel mode.
- The operating system is then loaded and starts user applications in user mode.

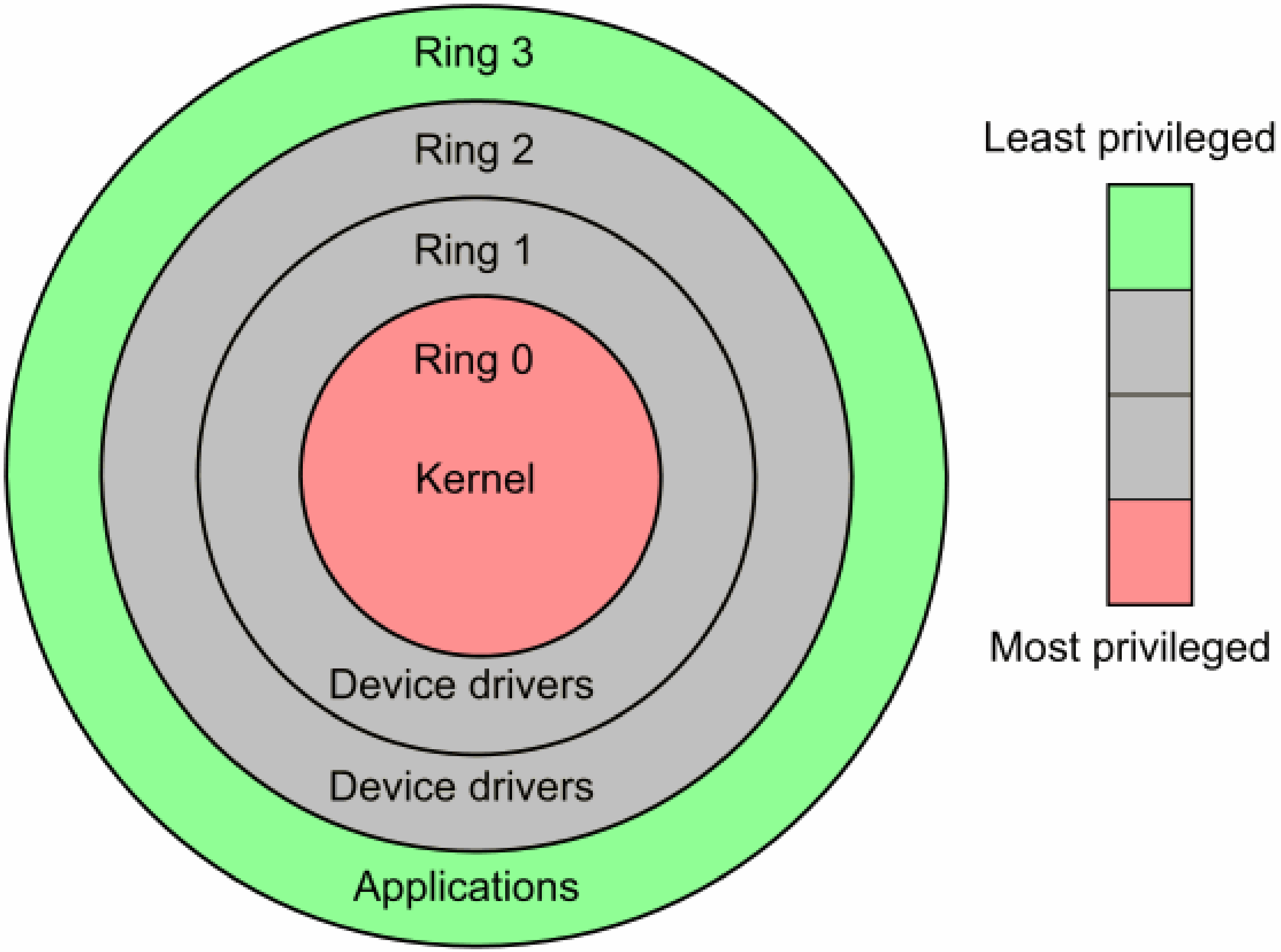
- Whenever a trap or interrupt occurs, the hardware switches from user mode to kernel mode.

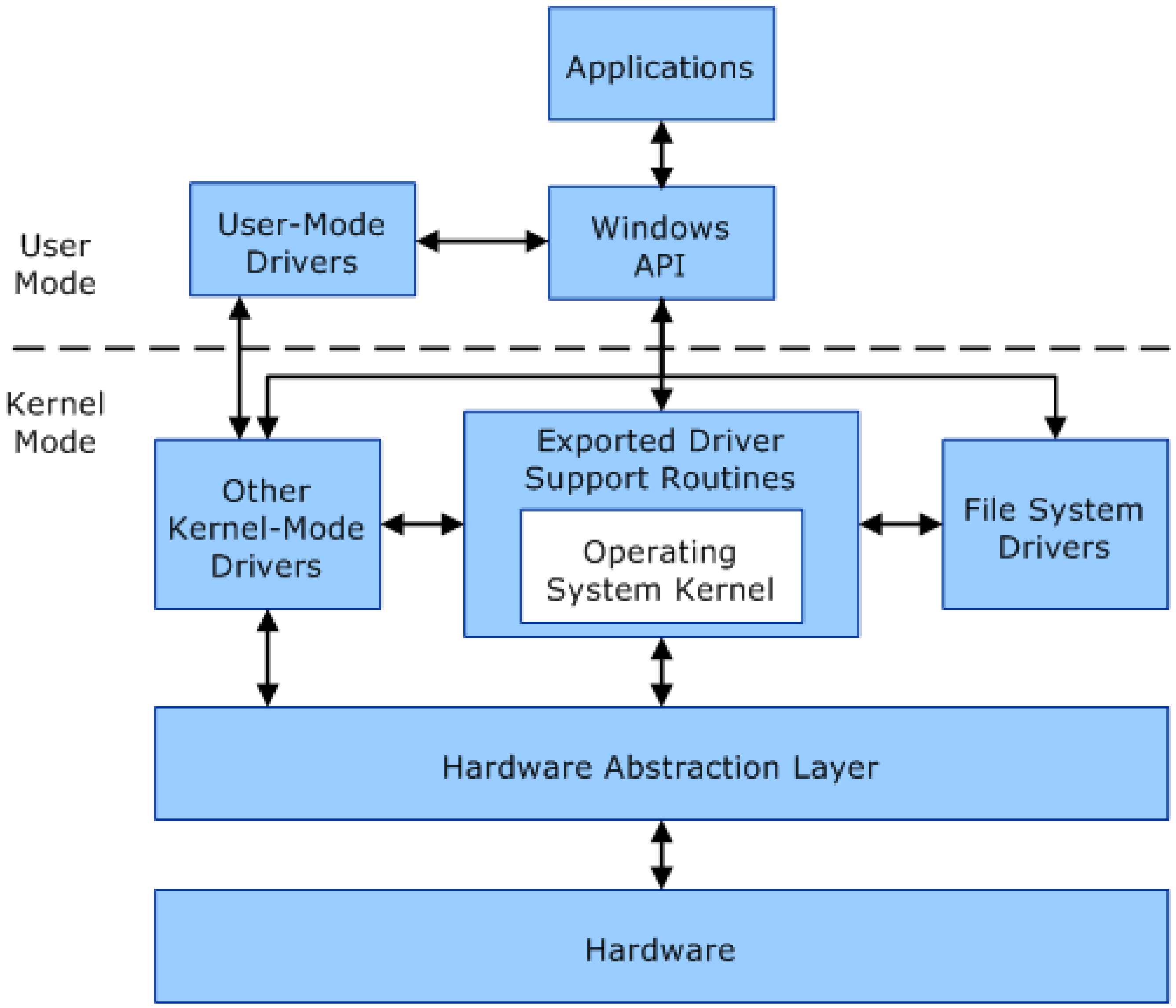
Transition from User to Kernel Mode



Privileged Instructions

- The instruction to switch to kernel mode is an example of a privileged instruction
- The dual mode of operation provides us with the means for protecting the operating system from errant users—and errant users from one another.
- We accomplish this protection by designating some of the machine instructions that may cause harm as privileged instructions.
- The hardware allows privileged instructions to be executed only in kernel mode.
- The concept of modes can be extended beyond two modes (in which case the CPU uses more than one bit to set and test the mode).
- CPUs that support virtualization frequently have a separate mode to indicate when the virtual machine manager (VMM)—and the virtualization management software— is in control of the system.
- In this mode, the VMM has more privileges than user processes but fewer than the kernel.


Ring 3
Ring 2
Ring 1
Ring 0
Kernel
Device drivers
Device drivers
Applications
Least privileged
Most privileged


Applications
User Mode
User-Mode Drivers
Windows API
Kernel Mode
Other Kernel-Mode Drivers
Exported Driver Support Routines
Operating System Kernel
File System Drivers
Hardware Abstraction Layer
Hardware

Lecture-1.5

Mr. S. C. Sagare

Part 1: Introduction to OS

Mr. S. C. Sagare

CONTENTS

- Operating System fundamentals
- Computer System Organization
- Computer System Architecture
- Operating System Structure
- Operating System Operations
- Process Management
- Memory Management
- Storage Management
- Computing Environments

Process Management

- A program does nothing unless its instructions are executed by a CPU.
  - I.e. A program is a passive entity, like the contents of a file stored on disk, whereas a process is an active entity.

- A process is a program in execution.
  - A time-shared user program such as a compiler is a process.
  - A word-processing program being run by an individual user on a PC is a process.
  - A system task, such as sending output to a printer, can also be a process.

- Process needs resources to accomplish its task.
  - E.g. CPU time, memory, files, I/O devices
- These resources are either given to the process when it is created or allocated to it while it is running.

- Various initialization data (input) may also be passed along with the various physical and logical resources that a process obtains when it is created,

Process Management

- A single-threaded process has one program counter specifying the next instruction to execute.

- The execution of such a process must be sequential. i.e. the CPU executes one instruction of the process after another, until the process completes.

- Further, at any time, one instruction at most is executed on behalf of the process.

- Thus, although two processes may be associated with the same program, they are nevertheless considered two separate execution sequences.

- A process is the unit of work in a system. A system consists of a collection of processes, some of which are operating-system processes (those that execute system code) and the rest of which are user processes.

Process Management

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

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

Memory Management

- Main memory is central to the operation of a modern computer system.
- The central processor **reads instructions** from main memory during the **instruction-fetch cycle** and both **reads and writes data** from main memory during the **data-fetch cycle**.
- Main memory is a repository of quickly accessible data shared by the CPU and I/O devices.
- It is a **large array of bytes**, ranging in size from hundreds of thousands to billions. Each byte has its own address.
- For a program to be executed, it must be mapped to **absolute addresses** and loaded into memory. As the program executes, it accesses program instructions and data from memory by generating these absolute addresses
- After the program is terminated, then its memory space is declared available, and next program can be loaded and executed.

Memory Management

- To improve both the **utilization of the CPU** and the speed of the computer's **response** to its users, general-purpose computers must keep several programs in memory, creating a need for memory management.
- We must take into account many factors while selecting a memory-management scheme for a specific system—
  - Especially the **hardware design** of the system
- The operating system is responsible for the following activities in connection with memory management.
  - Keeping track of which parts of memory are currently being used and who is using them
  - Deciding which processes (or parts of processes) and data to move into and out of memory
  - Allocating and de-allocating memory space as needed.

Storage Management

- To make the computer system convenient for users, the operating system provides a uniform, logical view of information storage.
- **File-System Management**
- The operating system abstracts from the physical properties of its storage devices to define a logical storage unit, the **file.**
- A file is a collection of related information defined by its creator.
- File are stored on several different types of physical media whose properties include access speed, capacity, data-transfer rate, and access method
- Commonly, files represent programs (both source and object forms) and data. Data files may be numeric, alphabetic, alphanumeric, or binary.
- Files may be free-form (for example, text files), or they may be formatted rigidly (for example, fixed fields).
- Clearly, File management is one of the most visible components of an operating system.

- The operating system implements the abstract concept of a file by managing mass-storage media, such as tapes and disks, and the devices that control them.
- In addition, files are normally organized into directories to make them easier to use.
- Finally, when multiple users have access to files, it may be desirable to control which user may access a file and how that user may access it (for example, read, write, append).

- The operating system is responsible for the following activities in connection with file management:
  - **Creating and deleting files**
  - **Creating and deleting directories to organize files**
  - **Supporting primitives for manipulating files and directories**
  - **Mapping files onto secondary storage**
  - **Backing up files on stable (nonvolatile) storage media**

Storage Management

- **Mass-Storage Management**
- Most modern computer systems use disks as the principal on-line storage medium for both programs and data.
- Most programs—including compilers, assemblers, word processors, editors, and formatters—are stored on a disk until loaded into memory. They then use the disk as both the source and destination of their processing.
- Proper management is of central importance
- Entire speed of computer operation depends on disk subsystem and its algorithms
- OS activities for disk management are
  - Free-space management
  - Storage allocation
  - Disk scheduling
- Some storage like **tertiary storage** devices need not be fast
  - Tertiary storage includes optical storage, magnetic tape
  - not crucial to system performance, still must be managed
  - Varies between WORM (write-once, read-many-times) and RW (read-write)

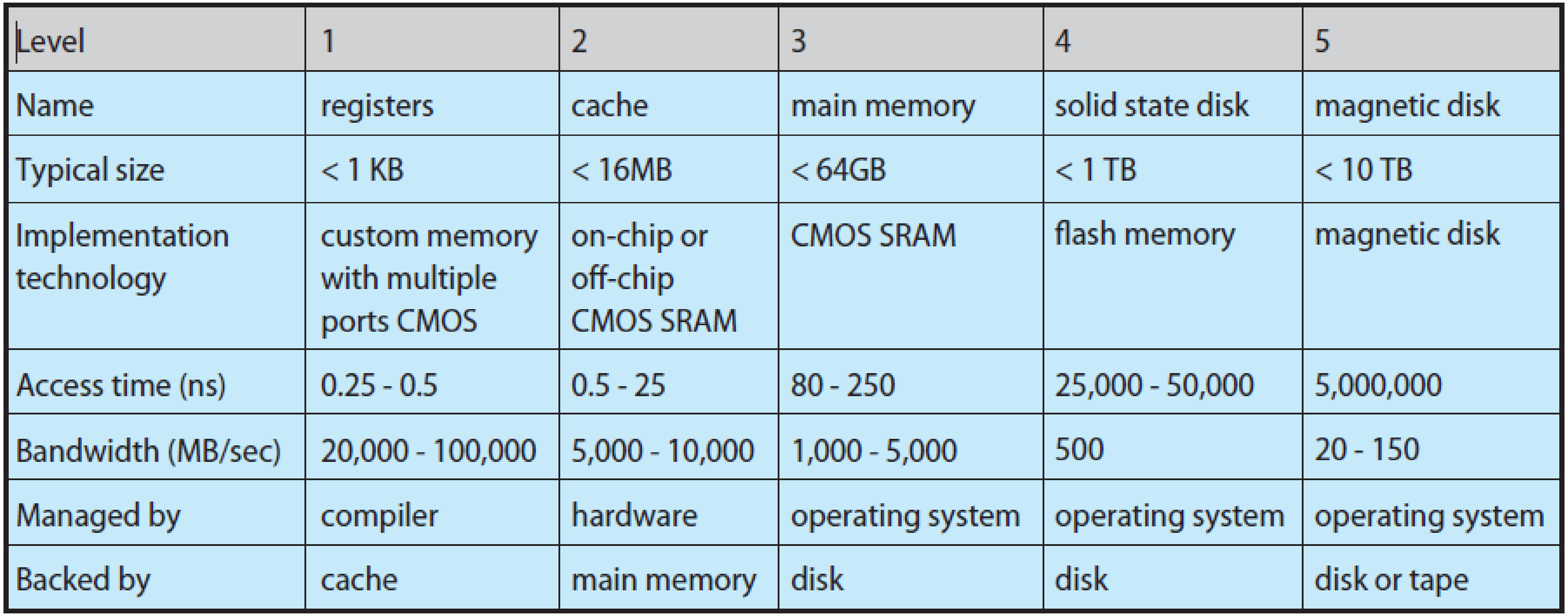
Storage Management

- **Caching-**
- Caching is an important principle of computer systems.
- Information is normally kept in some storage system (such as main memory), as it is used, it is copied into a faster storage system—the cache—on a temporary basis.
- When we need a particular piece of information, we first check whether it is in the cache. If it is, we use the information directly from the cache.
- If it is not, we use the information from the source, putting a copy in the cache under the assumption that we will need it again soon.

Storage Management

- **Caching-**
- Internal programmable **registers**, such as **index registers**, provide a high-speed cache for main memory.
- The programmer (or compiler) implements the **register-allocation** and **register-replacement** algorithms to decide which information to keep in registers and which to keep in main memory.
- Other caches are implemented totally in **hardware**. For instance, most systems have an instruction cache to hold the instructions expected to be executed next.
- Careful selection of the cache size and of a replacement policy can result in greatly increased performance
- Therefore **cache management** is an important design problem.

Storage Management



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

Figure 1.11 Performance of various levels of storage.

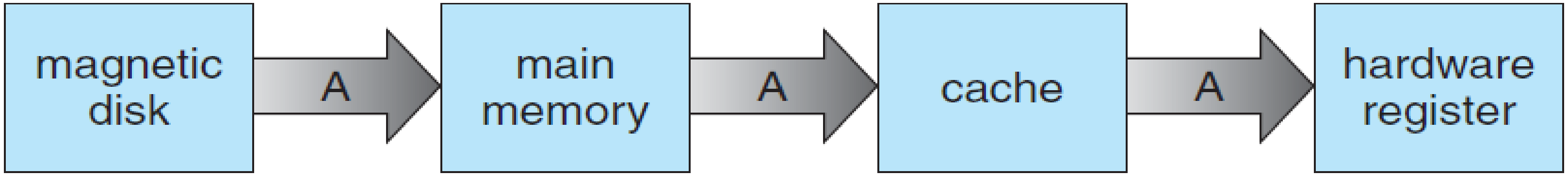


Figure 1.12 Migration of integer A from disk to register.

I/O Systems

- One purpose of OS is to hide some strange properties of hardware devices from the user

- I/O subsystem responsible for
  - Memory management of I/O including buffering (storing data temporarily while it is being transferred),
  - Caching (storing parts of data in faster storage for performance),
  - Spooling (the overlapping of output of one job with input of other jobs)
  - General device-driver interface
  - Drivers for specific hardware devices

Lecture-1.6

Mr. S. C. Sagare

Part 1: Introduction to OS

Mr. S. C. Sagare

CONTENTS

- Operating System fundamentals
- Computer System Organization
- Computer System Architecture
- Operating System Structure
- Operating System Operations
- Process Management
- Memory Management
- Storage Management
- Computing Environments

Computing Environments

- Its important to study how operating systems are used in a variety of **computing environments**.
- 1. **Traditional Computing:**
  - Today, traditional time-sharing systems are uncommon which used **timer** and scheduling algorithms to cycle processes rapidly through the CPU, giving each user a share of the resources.
  - The current trend is toward providing more ways to access these computing environments.
  - User processes, and system processes that provide services to the user, are managed so that each frequently gets a slice of computer time
  - Web technologies and increasing WAN bandwidth are stretching the boundaries of traditional computing.
  - Companies establish portals, which provide Web accessibility to their internal servers

Computing Environments

- 2. **Mobile Computing:**
  - Refers to computing on handheld smartphones and tablet computers. These devices share the distinguishing physical features of being portable and lightweight.

  - In fact, we might argue that the features of a contemporary mobile device allow it to provide functionality that is either unavailable or impractical on a desktop or laptop computer.

  - In several computer games that employ accelerometers, players interface with the system not by using a mouse or a keyboard but rather by tilting, rotating, and shaking the mobile device!

  - Perhaps more a practical use of these features is found in augmented-reality applications, which overlay information on a display of the current environment

  - The memory capacity and processing speed of mobile devices, however, are more limited than those of PCs. Whereas a smartphone or tablet may have 64 GB in storage, it is not uncommon to find 1 TB in storage on a desktop computer
  - Two operating systems currently dominate mobile computing: **Apple iOS** and **Google Android**. iOS was designed to run on Apple iPhone and iPad mobile devices. Android powers smartphones and tablet computers available from many manufacturers

Computing Environments

- 3. **Distributed Systems:**
  - A distributed system is a collection of physically separate, possibly heterogeneous, computer systems that are networked to provide users with access to the various resources that the system maintains.
  - Access to a shared resource increases computation speed, functionality, data availability, and reliability
  - Generally, systems contain a mix of the two modes—for example FTP and NFS.
  - Distributed systems depend on networking for their functionality.
- 4. **Client –Server Computing:**
  - Terminals connected to centralized systems are now being supplanted by PCs and mobile devices.
  - many of today's systems act as server systems to satisfy requests generated by client systems. This form of specialized distributed system, is called a **client–server system**,
  - It has the general structure as shown in fig.

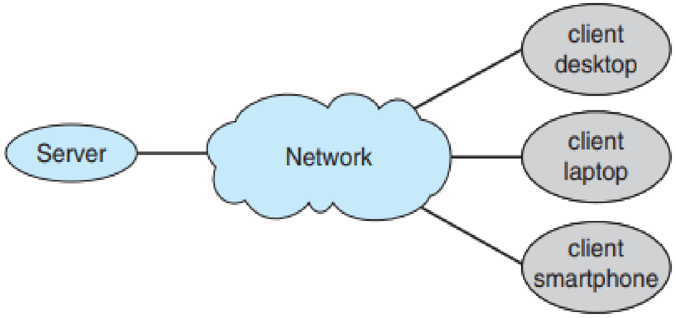


Figure 1.18 General structure of a client–server system.

Computing Environments

- 5. **Peer-to-Peer Computing:**
  - In a client-server system, the server is a bottleneck; but in a peer-to-peer system, services can be provided by several nodes distributed throughout the network
  - In the peer-to-peer (P2P) system model, clients and servers are not distinguished from one another.
  - Instead, all nodes within the system are considered peers, and each may act as either a client or a server, depending on whether it is requesting or providing a service

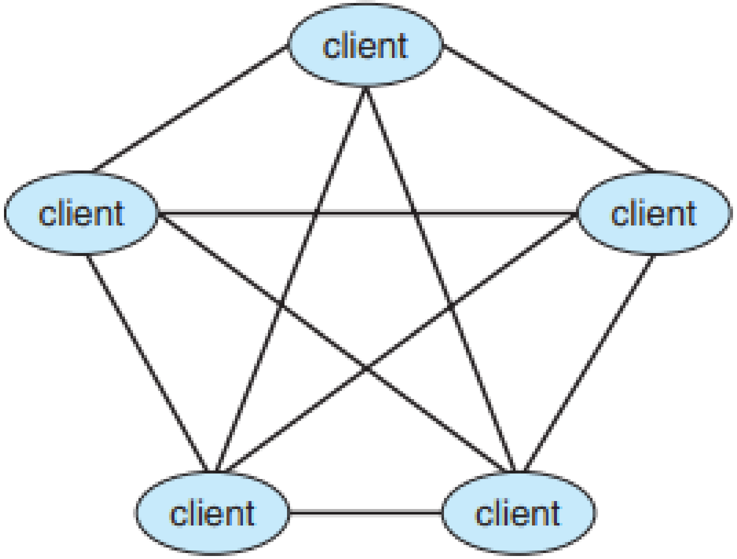


Figure 1.19 Peer-to-peer system with no centralized service.

Computing Environments

- 6. **Virtualization:**
  - Allows operating systems to run as applications within other operating systems using **Emulation.**
  - Here, every machine-level instruction that runs natively on the source system must be translated to the equivalent function on the target system, frequently resulting in several target instructions.

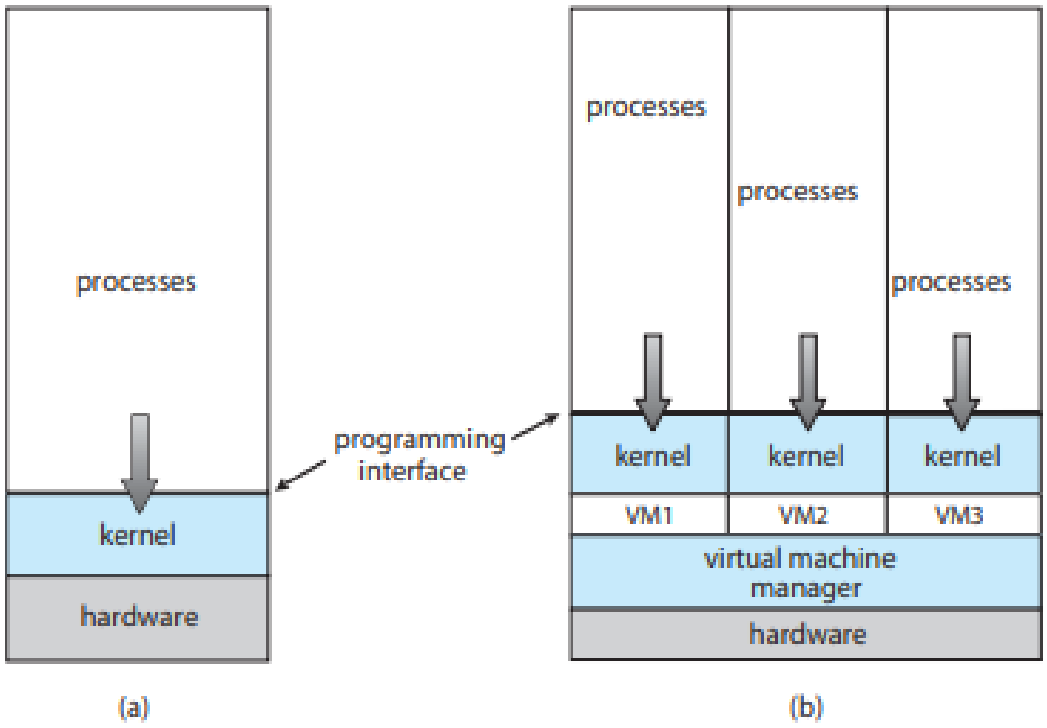


Figure 1.20 VMware.

Computing Environments

- 7. **Cloud Computing:**
  - Computing that delivers computing, storage, and even applications as a service across a network.
  - In some ways, it's a logical extension of virtualization, because it uses virtualization as a base for its functionality

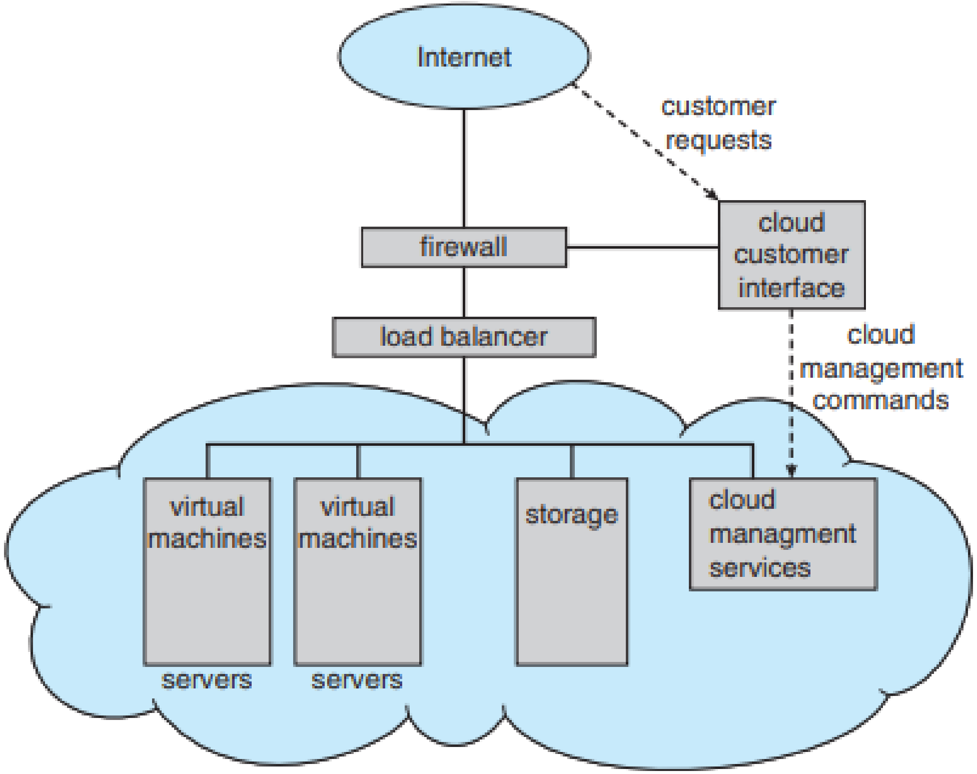


Figure 1.21 Cloud computing.

Part 2: Operating System Services

Mr. S. C. Sagare

CONTENTS

- Operating System Services
- User and OS interface
- System calls
- Types of system calls
- OS structure
- System boot

Operating-System Services

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

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

- The specific services provided, of course, differ from one operating system to another.

- These operating system services are provided for the convenience of the programmer, to make the programming task easier.

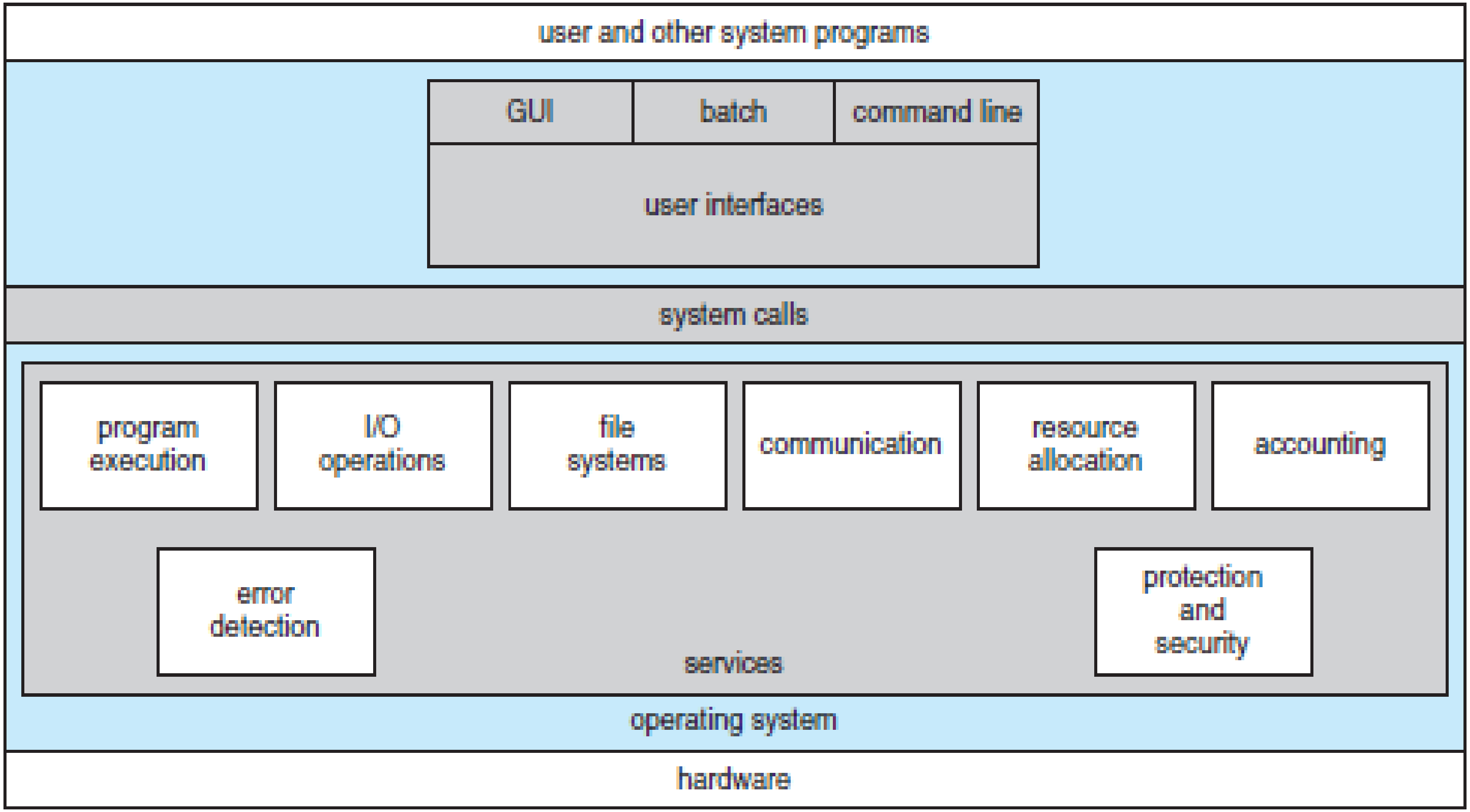


Figure 2.1 A view of operating system services.

- **User interface:--** Almost all operating systems have a user interface (UI). command-line interface, batch interface and graphical user interface (GUI).

- **Program execution**:-- The system must be able to load a program into memory and to run that program.

- **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. Obviously programs need to read and write files and directories. They also need to create and delete them by name, search for a given file, and list file information.

- **Communications**:-- process to process communication.

- **Error detection:--** The operating system needs to be detecting and correcting errors constantly.
  - Errors may occur in the CPU and memory hardware (such as a memory error or a power failure).
  - in I/O devices (such as a parity error on disk, a connection failure on a network, or lack of paper in the printer).
  - in the user program (such as an arithmetic overflow, an attempt to access an illegal memory location, or a too-great use of CPU time).

- **Resource allocation:--** When there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them.
- **Accounting:--** We want to keep track of which users use how much and what kinds of computer resources. This record keeping may be used for accounting.
- **Protection and security:--** The owners of information stored in a multiuser or networked computer system may want to control use of that information.

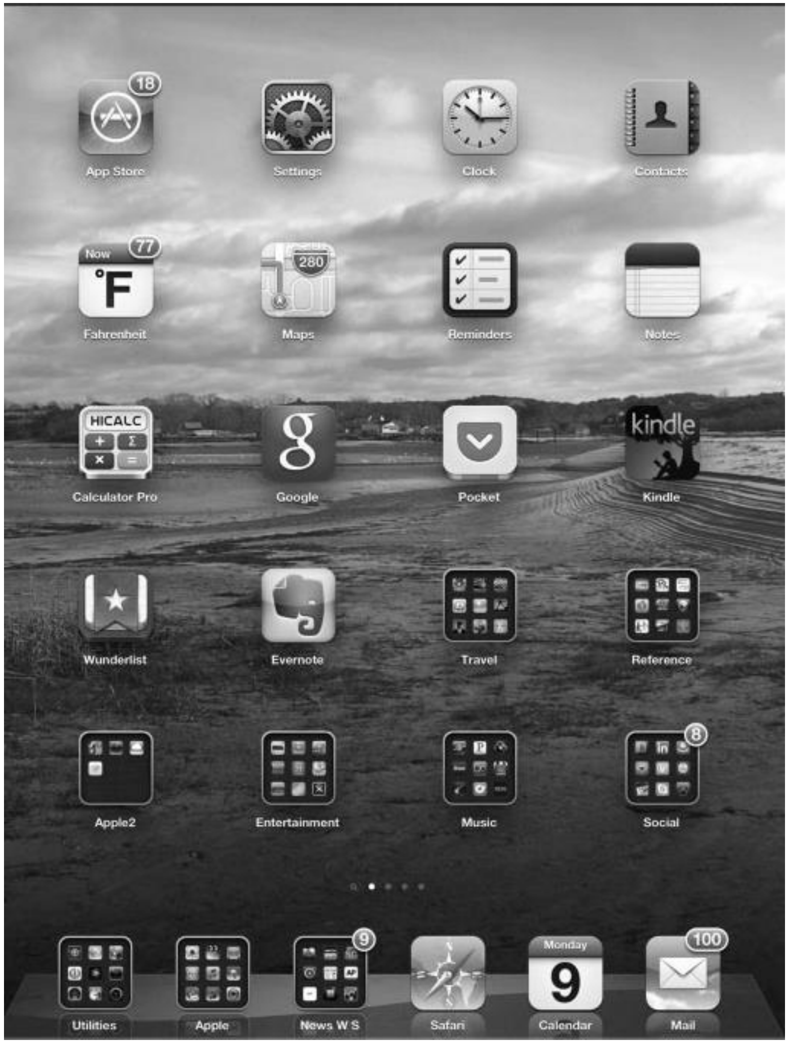
User and Operating-System Interface

- **Command Interpreters**:-
  - Some operating systems include the command interpreter in the kernel.
  - Others, such as Windows and UNIX, treat the command interpreter as a special program that is running when a job is initiated or when a user first logs on.
  - On systems with multiple command interpreters to choose from, the interpreters are known as shells.
- The main function of the command interpreter is to get and execute the next user-specified command.

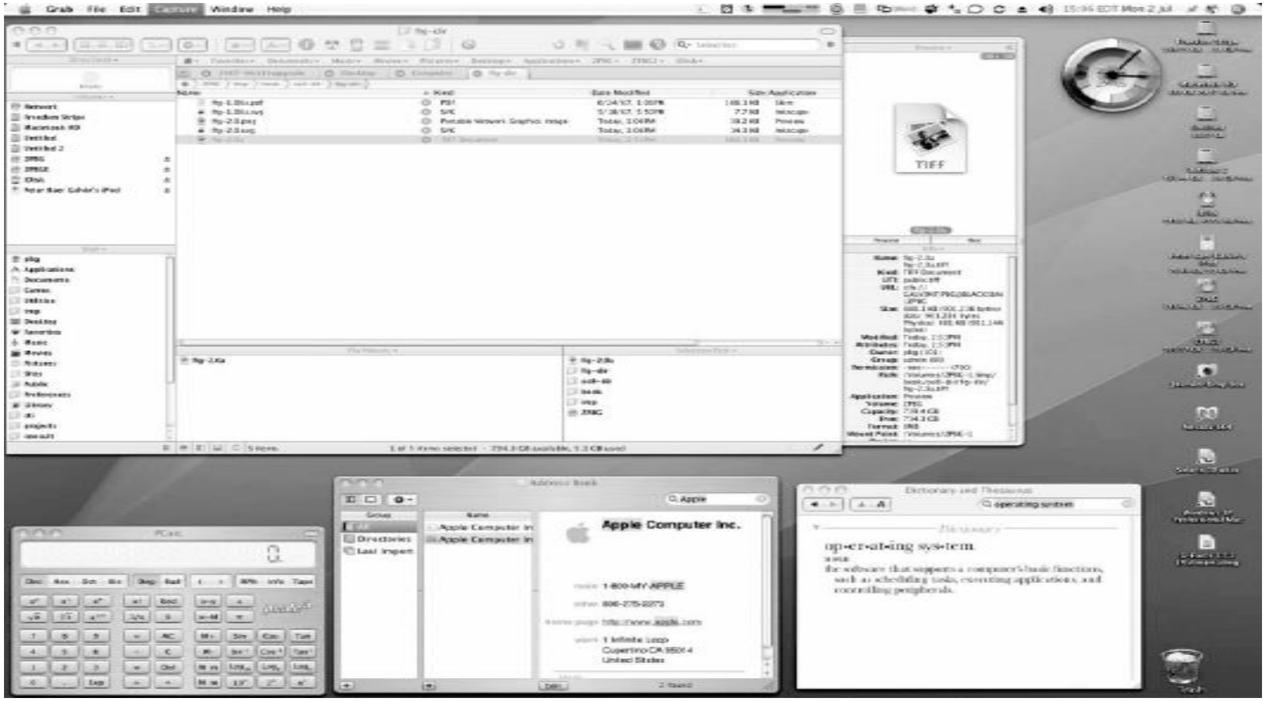
- Many of the commands given at this level manipulate files: create, delete, list, print, copy, execute, and so on.

- The MS-DOS and UNIX shells operate in this way.

Graphical User Interfaces:-



- **Choice of Interface**
  - The choice of whether to use a command-line or GUI interface is mostly one of personal preference.
  - System administrators who manage computers and power users who have deep knowledge of a system frequently use the command-line interface.
  - on some systems, only a subset of system functions is available via the GUI, leaving the less common tasks to those who are command-line knowledgeable.



Apple Computer Inc.

1-800-MY-APPLE

800-275-2273

http://www.apple.com

1 Infinite Loop
Cupertino CA 95014
United States

op·er·at·ing sys·tem
noun
the software that supports a computer's basic functions, such as scheduling tasks, executing applications, and controlling peripherals.

- In contrast, most Windows users are happy to use the Windows GUI environment and almost never use the MS-DOS shell interface.

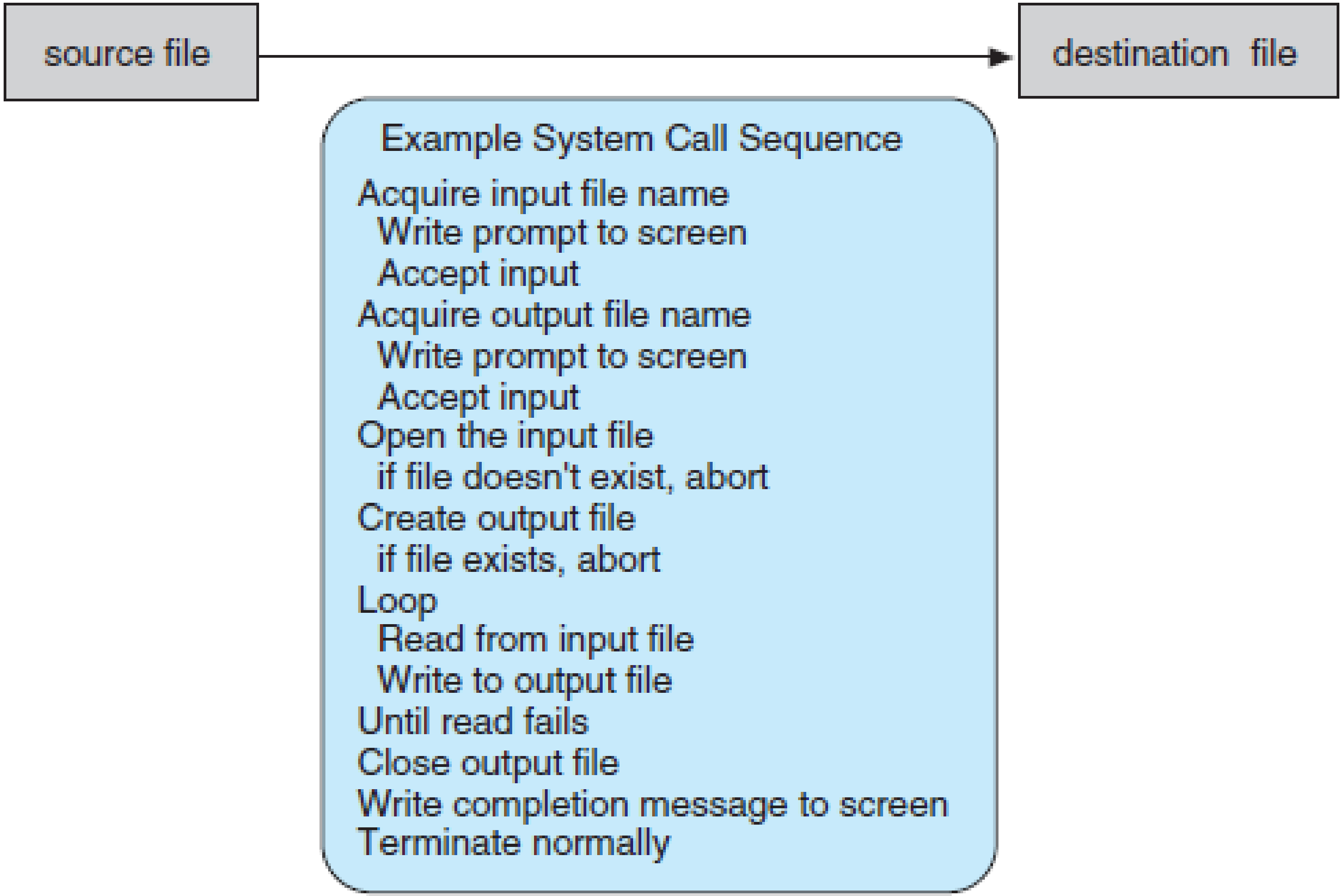
- Historically, Mac OS has not provided a command-line interface, always requiring its users to interface with the operating system using its GUI.

- However, with the release of Mac OS X (which is in part implemented using a UNIX kernel), the operating system now provides both a Aqua interface and a command-line interface.

System Calls

- In computing, a **system call** is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on.
- A system call is a way for programs to interact with the operating system.

- System call provides the services of the operating system to the user programs via Application Program Interface(API)
- An example to illustrate how system calls are used:
  - writing a simple program to read data from one file and copy them to another file.
  - The first input that the program will need is the names of the two files: the input file and the output file.
  - These names can be specified in many ways, depending on the operating-system design.
  - Once the two file names have been obtained, the program must open the input file and create the output file.
  - Each of these operations requires another system call.



```
Example System Call Sequence
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
```

Figure 2.5 Example of how system calls are used.

- Possible error conditions for each operation can require additional system calls.
- When the program tries to open the input file, for example, it may find that there is no file of that name or that the file is protected against access
- these cases, the program should print a message on the console (another sequence of system calls.

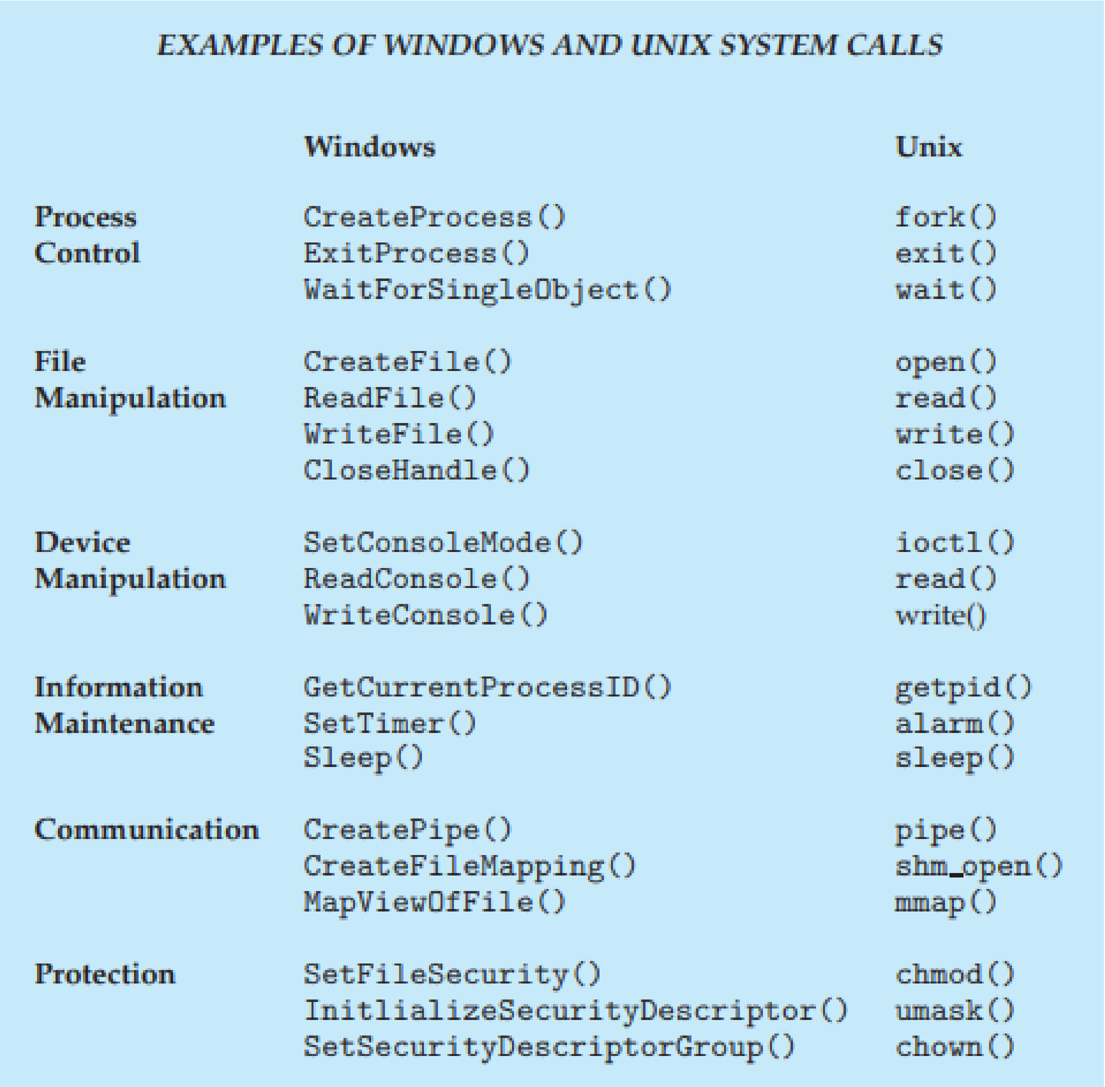
- simple programs may make heavy use of the operating system. Frequently, systems execute thousands of system calls per second.

- Typically, application developers design programs according to an application programming interface (API).

- API is a **set of functions and procedures** that allow the creation of applications which access the features or data of an operating system, application, or other services.

Services Provided by System Calls

- **Process creation and management**
- **Main memory management**
- **File Access, Directory and File system management**
- **Device handling(I/O)**
- **Protection**
- **Networking, etc.**



EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

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

Lecture-1.7

Mr. S. C. Sagare

Part 2: Operating System Services

Mr. S. C. Sagare

CONTENTS

- Operating System Services
- User and OS interface
- System calls
- Types of system calls
- OS structure
- System boot

System call interface

- For most programming languages, the run-time support system (a set of functions built onto libraries included with a compiler) provides a **system call interface** that serves as the link to system calls made available by the operating system.

- The system-call interface intercepts **function calls** in the API and invokes the necessary system calls within the operating system.

- Typically, a number is associated with each system call, and the system-call interface maintains a table indexed according to these numbers.

- The system call interface then invokes the intended system call in the operating-system kernel and returns the status of the system call and any return values.

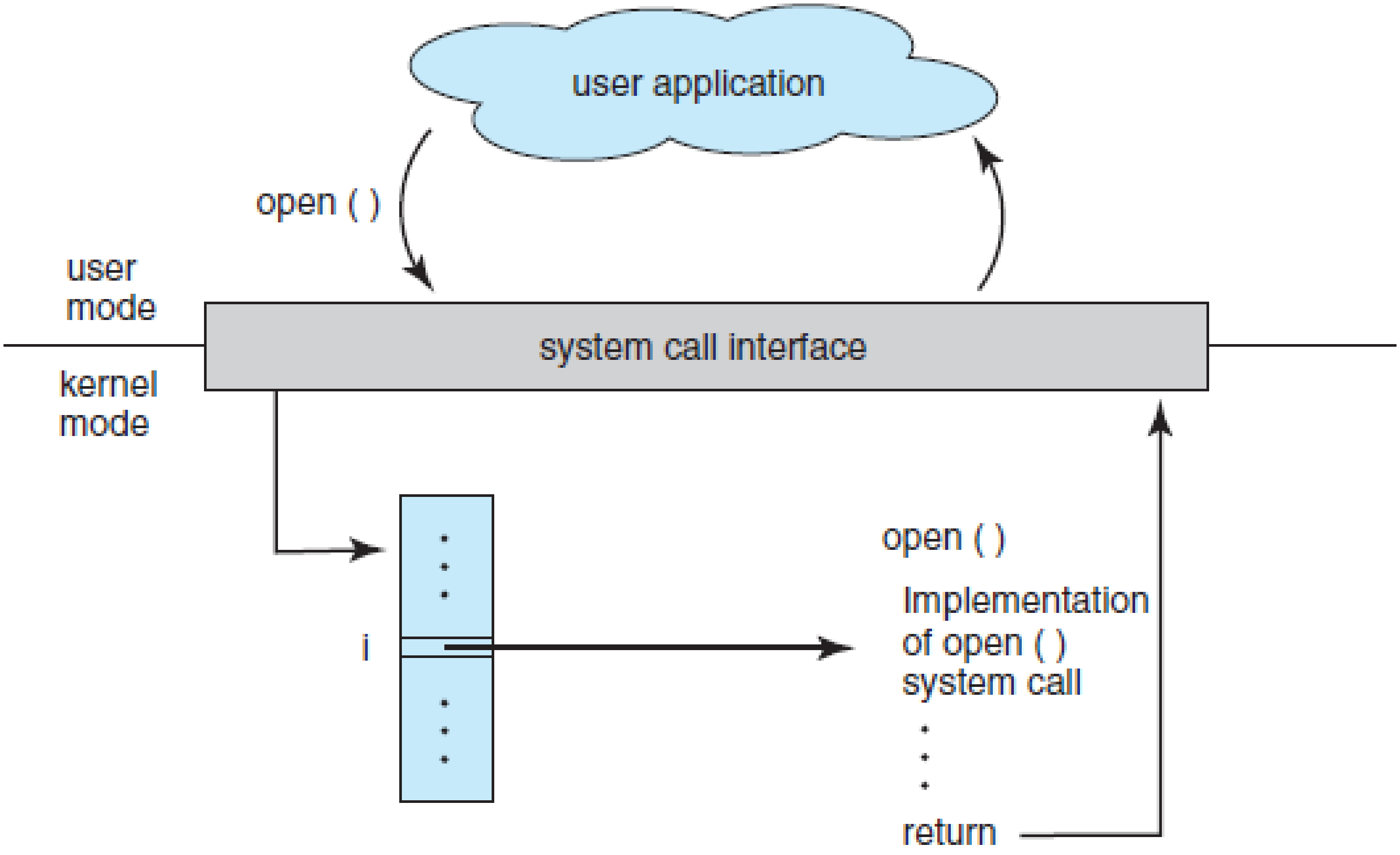
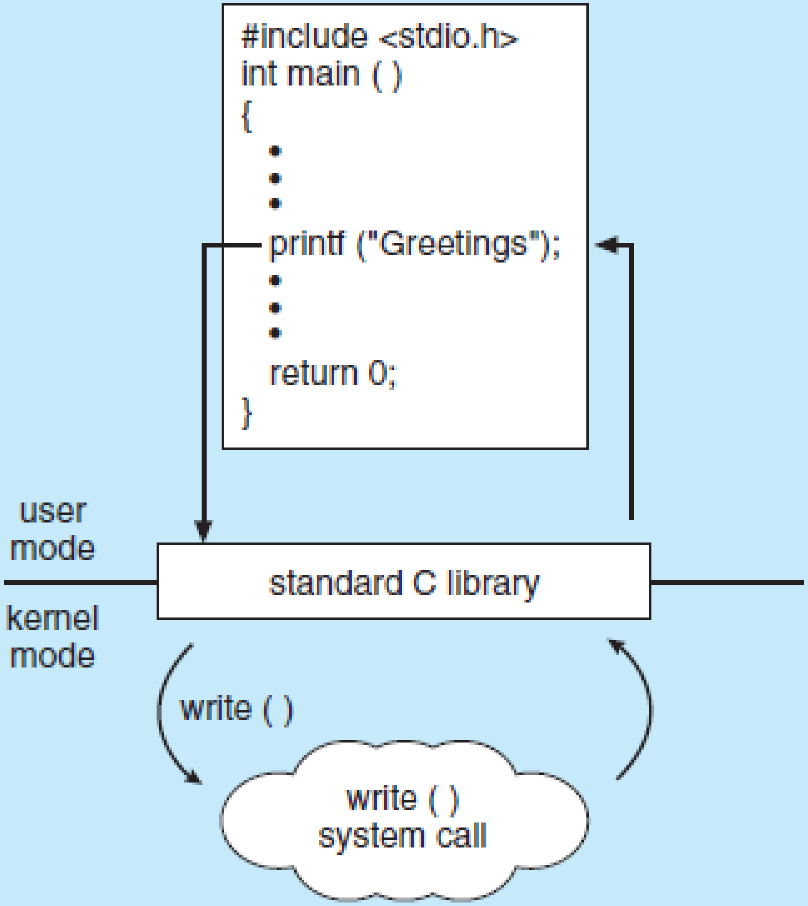


Figure 2.6 The handling of a user application invoking the `open()` system call.

EXAMPLE OF STANDARD C LIBRARY

The standard C library provides a portion of the system-call interface for many versions of UNIX and Linux. As an example, let's assume a C program invokes the `printf()` statement. The C library intercepts this call and invokes the necessary system call (or calls) in the operating system—in this instance, the `write()` system call. The C library takes the value returned by `write()` and passes it back to the user program. This is shown below:



```
#include <stdio.h>
int main ( )
{
   •
   •
   •
   printf ("Greetings");
   •
   •
   •
   return 0;
}
```

- Three methods are used to pass parameters to the operating system.

- The simplest approach is to pass the parameters in registers.
- In some cases, however, there may be more parameters than registers. In these cases, the parameters are generally stored in a block, or table, in memory, and the address of the block is passed as a parameter in a register.
- Parameters also can be placed, or pushed, onto the stack by the program and popped off the stack by the operating system.

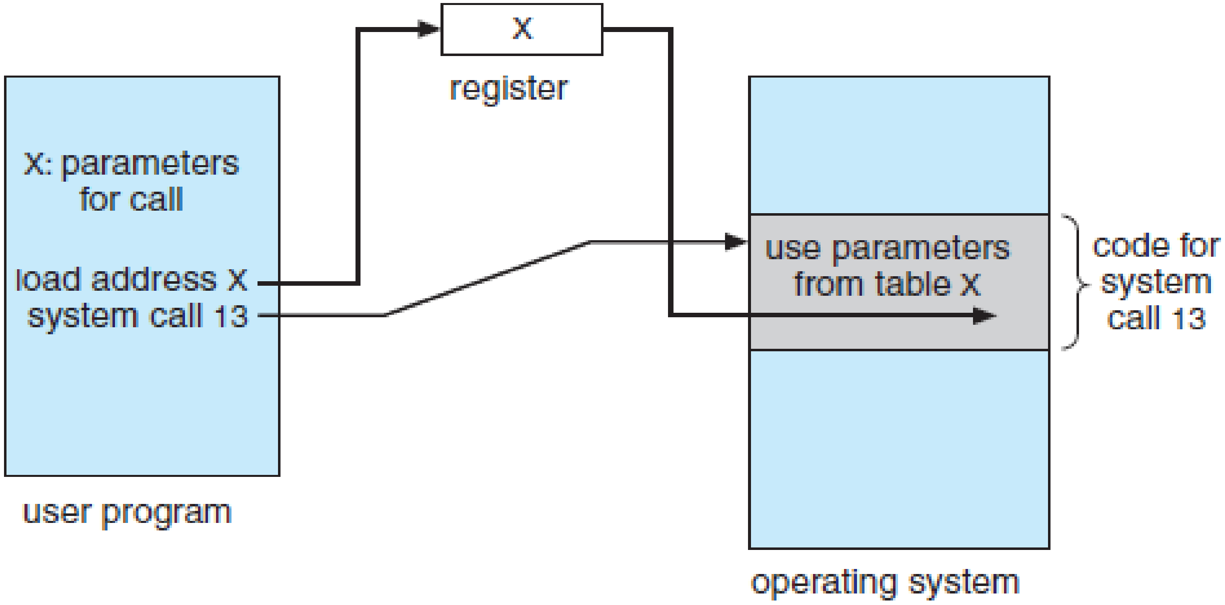


Figure 2.7 Passing of parameters as a table.

Types of System Calls

- **System calls can be grouped roughly into six major categories:**
  - Process control
  - File manipulation
  - Device manipulation
  - Information maintenance
  - communications
  - protection.

- Process control
  - 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
- File management
  - create file, delete file
  - open, close
  - read, write, reposition
  - get file attributes, set file attributes
- Device management
  - 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 process, file, or device attributes
  - set process, file, or device attributes
- Communications
  - create, delete communication connection
  - send, receive messages
  - transfer status information
  - attach or detach remote devices

Figure 2.8 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
  - The dump is written to disk and may be examined by a debugger—a system program designed to aid the programmer in finding and correcting errors, or bugs —to determine the cause of the problem.
  - Under either normal or abnormal circumstances, the operating system must transfer control to the invoking **command interpreter**. The command interpreter then reads the next command.
  - In an interactive system, the command interpreter simply continues with the next command; it is assumed that the user will issue an appropriate command to respond to any error
  - In a GUI system, a pop-up window might alert the user to the error and ask for guidance.
  - In a batch system, the command interpreter usually terminates the entire job and continues with the next job

- If we create a new job or process, or perhaps even a set of jobs or processes, we should be able to control its execution.

- This control requires the ability to determine and reset the attributes of a job or process, including the job's priority, Its maximum allowable execution time, and so on (get process attributes() and set process attributes()).

- We may also want to terminate a job or process that we created (terminate process()) if we find that it is incorrect or is no longer needed.

- Having created new jobs or processes, we may need to wait for them to finish their execution.

- We may want to wait for a certain amount of time to pass (wait time()). More probably, we will want to wait for a specific event to occur (wait event()).

- The jobs or processes should then signal when that event has occurred (signal event()).

- Quite often, two or more processes may share data.
- To ensure the integrity of the data being shared, operating systems often provide system calls allowing a process to **lock shared data.**
- **Then, no other process can access the data until** the lock is released.
- Typically, such system calls include acquire lock() and release lock()

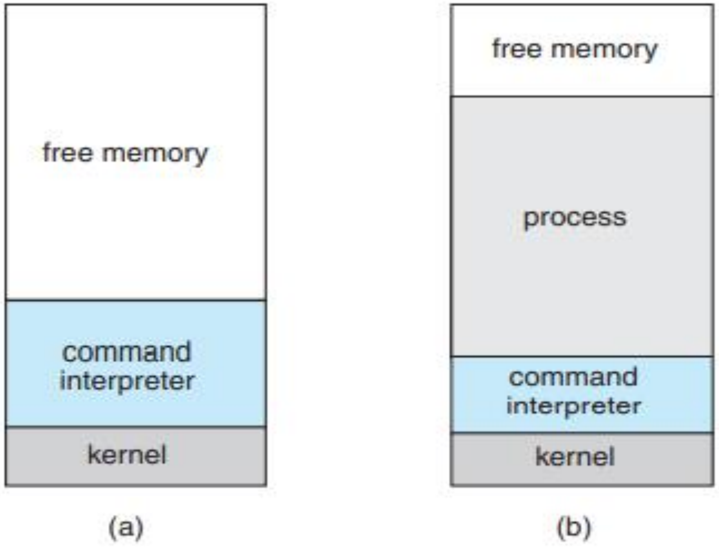


Figure 2.9 MS-DOS execution. (a) At system startup. (b) Running a program.

File management

- 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

- 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.

- Finally, we need to close() the file, indicating that we are no longer using it.

File management

E.g.

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

Device Management

- 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.

- As system with multiple users may require us to first request() a device.

- The device has been requested (and allocated to us), we can read(),write(), reposition

Device Management

e.g.

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

Information Maintenance

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

- Another set of system calls is helpful in debugging a program. Many systems provide system calls to dump() memory

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

Communication

- There are two common models of interprocess communication: the message passing model and the shared-memory model.

- Each computer in a network has a **host name by which it is commonly known. A host also has a** network identifier, such as an IP address.

- Similarly, each process has a **process name, and this name is translated into an identifier by which the operating** system can refer to the process.

- The get hostid() and get processid() system calls do this translation.

- The identifiers are then passed to the general purpose open() and close() calls provided by the file system or to specific open connection() and close connection() system calls, depending on the system's model of communication.

Communications

- **E.g.**
  - create, delete communication connection
  - send, receive messages
  - transfer status information
  - attach or detach remote devices

- **Protection**
  - Protection provides a mechanism for controlling access to the resources provided by a computer system
  - System calls providing protection include set permission() and get permission(), which manipulate the permission settings of resources such as files and disks
  - The allow user() and deny user() system calls specify whether particular users can—or cannot—be allowed access to certain resources

System Programs

- One important aspect of a modern system is its collection of system programs.
- System programming leads to development of computer system software that manages and controls the computer operations.

- They can be divided into these categories:

  - **File management**. These programs create, delete, copy, rename, print**,** dump, list, and generally manipulate files and directories

  - **Status information.** Some programs simply ask the system for the date**,** time, amount of available memory or disk space, number of users, or similar status information.

  - **File modification.** Several text editors may be available to create and modify the content of files stored on disk or other storage devices.

  - **Programming-language support. Compilers, assemblers, debuggers, and** interpreters for common programming languages (such as C, C++, Java, and PERL) are often provided with the operating system or available as a separate download.

System Programs

- **Program loading and execution**. Once a program is assembled or compiled, it must be loaded into memory to be executed.

- **Communications.** These programs provide the mechanism for creating virtual connections among processes, users, and 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.

- **Background services**. All general-purpose systems have methods for launching certain system-program processes at boot time

Operating-System Structure

- **Simple Structure:-**
- Many operating systems do not have well-defined structures.
- Frequently, such systems started as small, simple, and limited systems and then grew beyond their original scope.
- MS-DOS is an example of such a system. It was originally designed and implemented by a few people who had no idea that it would become so popular.
- It was written to provide the most functionality in the least space, so it was not carefully divided into modules.
- In MS-DOS, the interfaces and levels of functionality are not well separated. So, application programs are able to access the basic I/O routines to write directly to the display and disk drives
- This leaves MS-DOS vulnerable to errant (or malicious) programs, causing entire system crashes when user programs fail

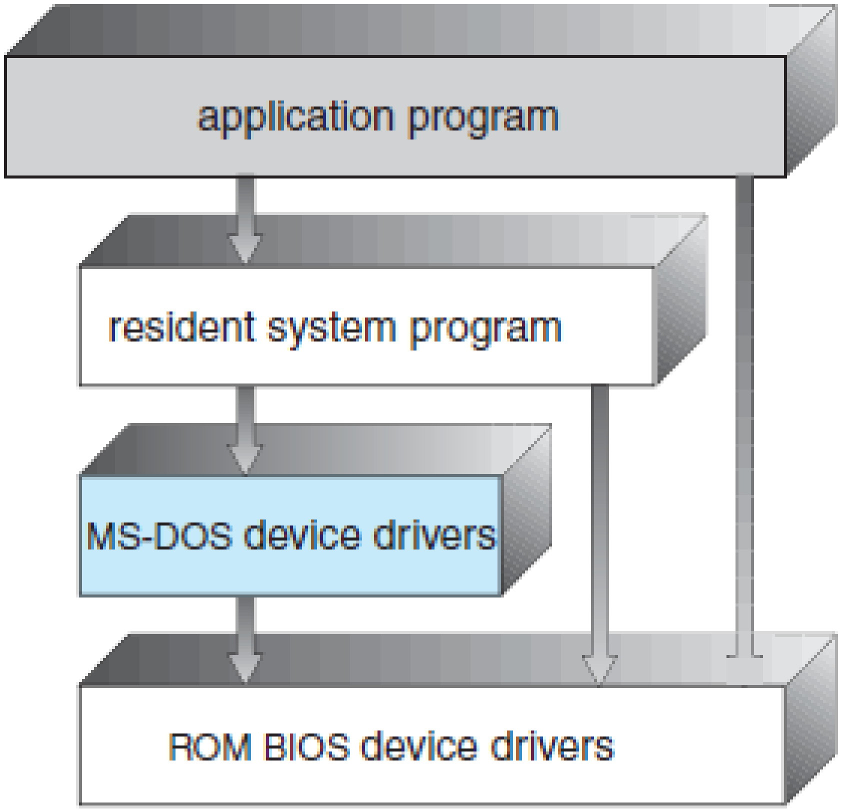


Figure 2.11 MS-DOS layer structure.

- Another example of limited structuring is the original UNIX operating system.
- Like MS-DOS, UNIX initially was limited by hardware functionality.
- It consists of two separable parts: the kernel and the system programs.

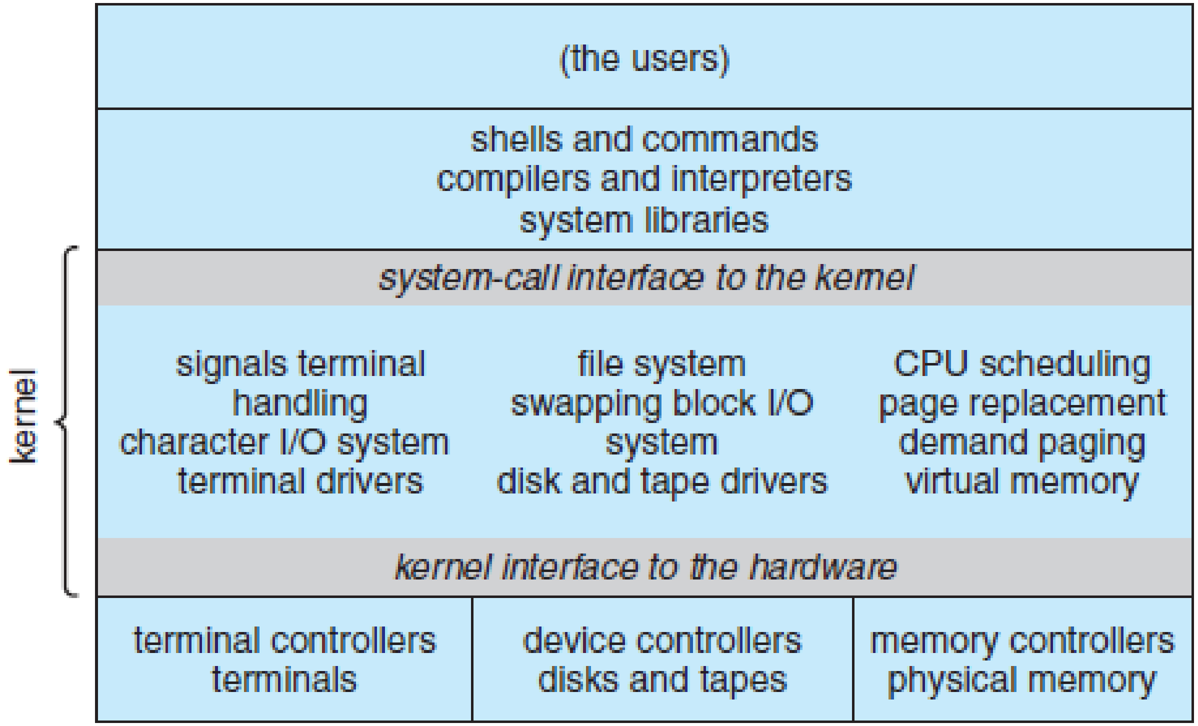


Figure 2.12 Traditional UNIX system structure.

Lecture-1.8

Mr. S. C. Sagare

Part 2: Operating System Services

Mr. S. C. Sagare

CONTENTS

- Operating System Services
- User and OS interface
- System calls
- Types of system calls
- OS structure
- System boot

Layered Approach

- With proper hardware support, operating systems can be broken into pieces that are **smaller and more appropriate** than those allowed by the original MS-DOS and UNIX systems.

- Layering provides a distinct advantage in an operating system. All the layers can be defined separately and interact with each other as required.

- Also, it is easier to **create, maintain and update the system** if it is done in the form of layers.

- Change in one layer specification **does not affect** the rest of the layers.

- Each of the layers in the operating system can only interact with the layers that are above and below it.

- The lowest layer handles the hardware and the uppermost layer deals with the user applications.

- A typical operating-system layer—say, layer M—consists of data structures and a set of routines that can be invoked by higher-level layers.
- Layer M, in turn, can invoke operations on lower-level layers

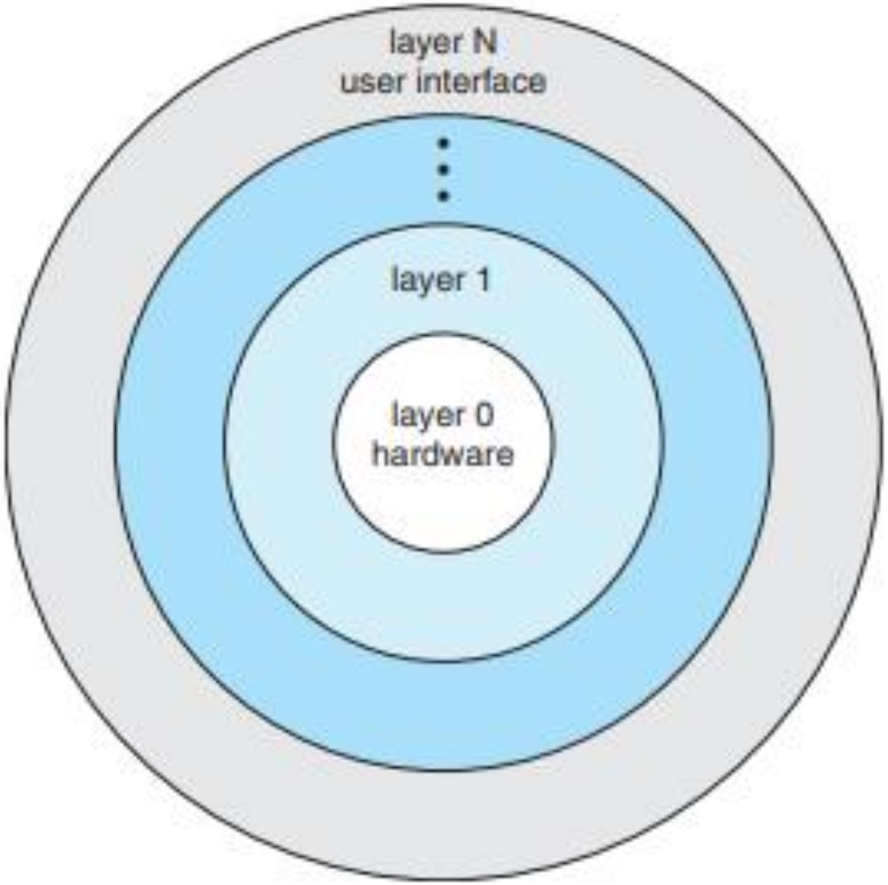


Figure 2.13 A layered operating system.

Microkernels

- In UNIX expanded OS, the kernel became large and difficult to manage.

- This method structures the operating system by removing all nonessential components from the kernel and implementing them as system and user-level programs.

- The result is a smaller kernel.

- Typically, however, micro kernels provide minimal process and memory management, in addition to a communication facility

- The main function of the microkernel is to provide communication between the client program and the various services that are also running in user space.

- Communication is provided through **message passing.**

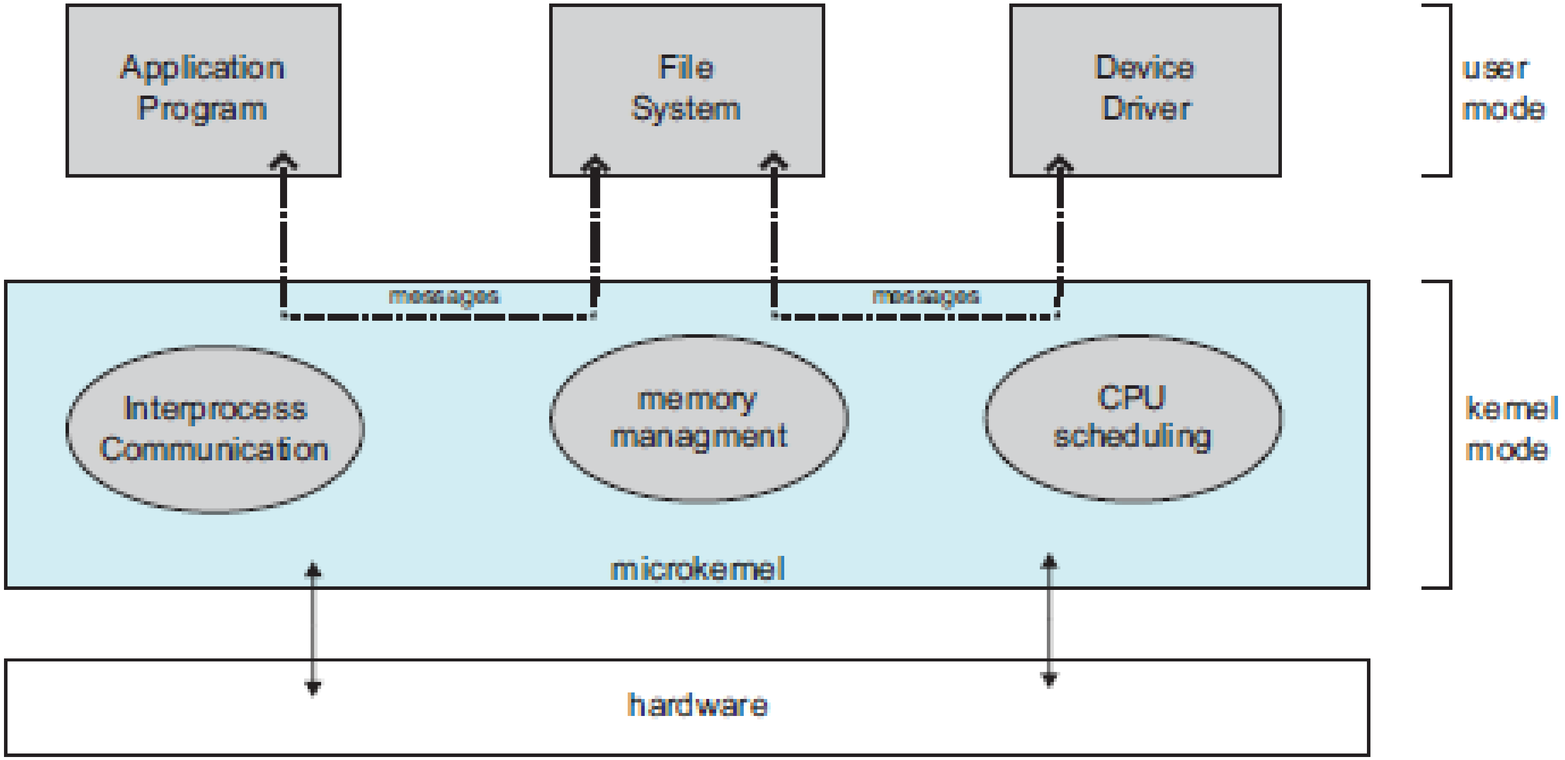


Figure 2.14 Architecture of a typical microkernel.

Modules

- the best current methodology for operating-system design involves using **loadable kernel modules.**

- Here, the kernel has a set of core components and links in additional services via modules, either at boot time or during run time.

- This type of design is common in modern implementations of UNIX, such as Solaris, Linux, and Mac OS X, as well as Windows.

- The idea of the design is for the kernel to provide core services while other services are implemented dynamically, as the kernel is running.

- Linking services dynamically is preferable **to adding new features directly** to the kernel, which would require recompiling the kernel every time a change was made

- Modules are **similar to layers** in that each subsystem has clearly defined tasks and interfaces, but any **module is free to contact any other module**,

- The kernel is relatively small in this architecture, similar to microkernels, but the kernel does not have to implement message passing since modules are free to contact each other directly.

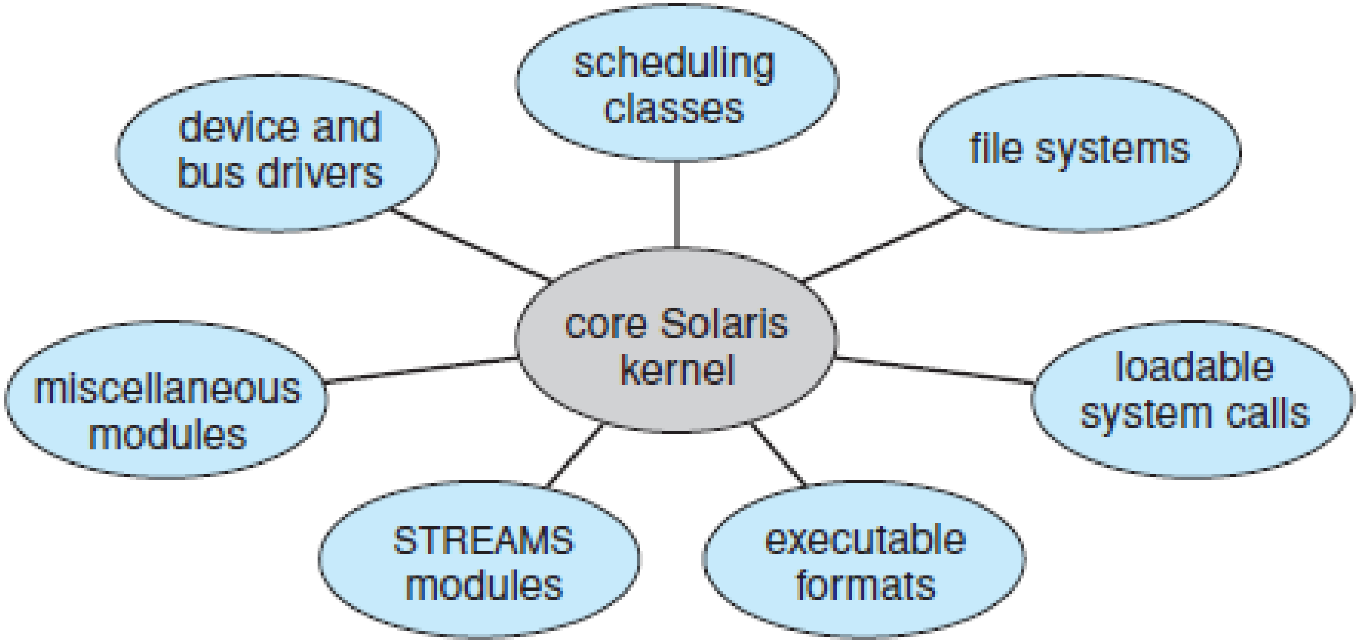


Figure 2.15 Solaris loadable modules.

Hybrid Systems

- In practice, very few operating systems adopt a single, strictly defined structure. Instead, they combine different structures, resulting in hybrid systems that address performance, security, and usability issues.

- The Apple Mac OS X operating system uses a hybrid structure.

- The top layers include the ***Aqua user interface*** and a set of application environments and services.
- Notably, the **Cocoa environment specifies an API for the Objective-C programming** language, which is used for writing Mac OS X applications

- Below these layers is the ***kernel environment, which consists primarily of the Mach*** microkernel and the BSD UNIX kernel.

- Mach provides memory management; support for remote procedure calls (RPCs) and interprocess communication (IPC) facilities, including message passing; and thread scheduling

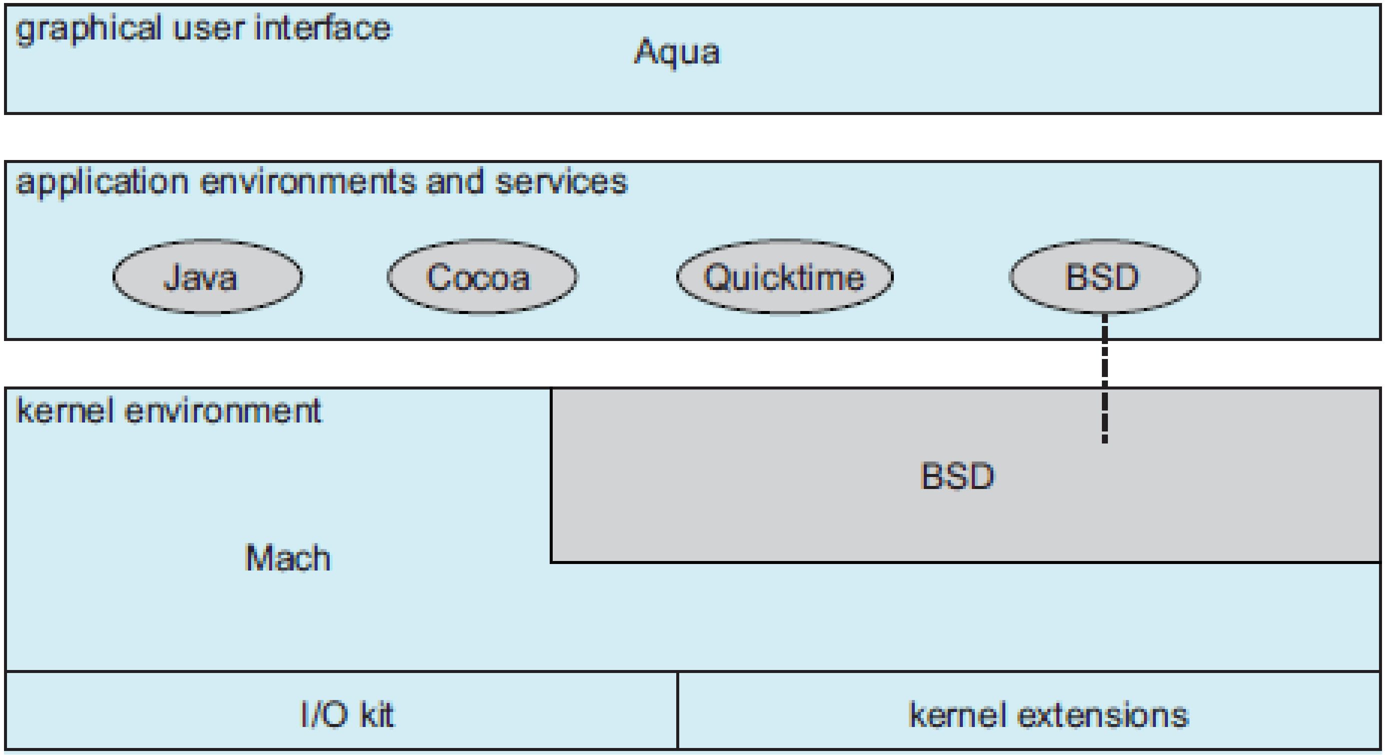


Figure 2.16 The Mac OS X structure.

iOS

- iOS is a mobile operating system designed by Apple to run its smart phone, the ***iPhone, as well as its tablet computer, the iPad.***

- iOS is structured on the Mac OS X operating system, with added functionality applicable to mobile devices, but does not directly run Mac OS X applications.

- **Cocoa Touch is an API for Objective-C that provides several frameworks for** developing applications that run on iOS devices.

- The fundamental difference between Cocoa, mentioned earlier, and Cocoa Touch is that the latter provides support for hardware features unique to mobile devices, such as touch screens

- The **media services layer provides services for graphics, audio, and video**

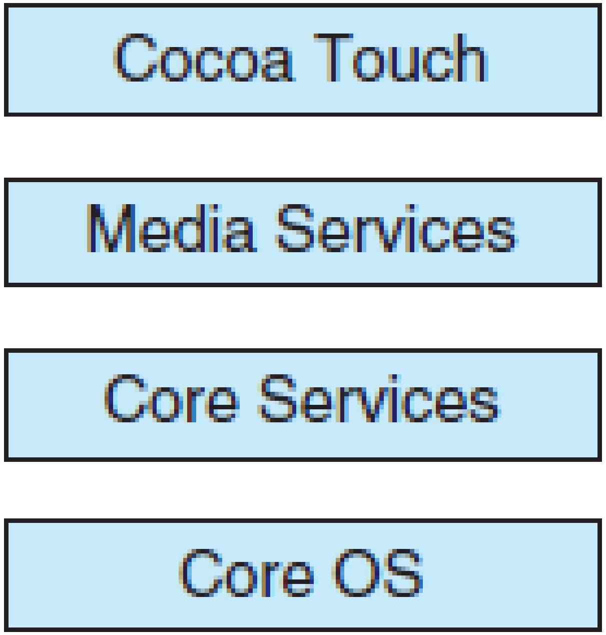


Figure 2.17 Architecture of Apple's iOS.

Android

- The Android operating system was designed by the Open Handset Alliance and was developed for Android smart phones and tablet computers.

- Whereas iOS is designed to run on Apple mobile devices and is close-sourced.

- Android runs on a variety of mobile platforms and is open-sourced.

- Android is similar to iOS in that it is a layered stack of software that provides a rich set of frameworks for developing mobile applications.

- At the bottom of this software stack is the Linux kernel.

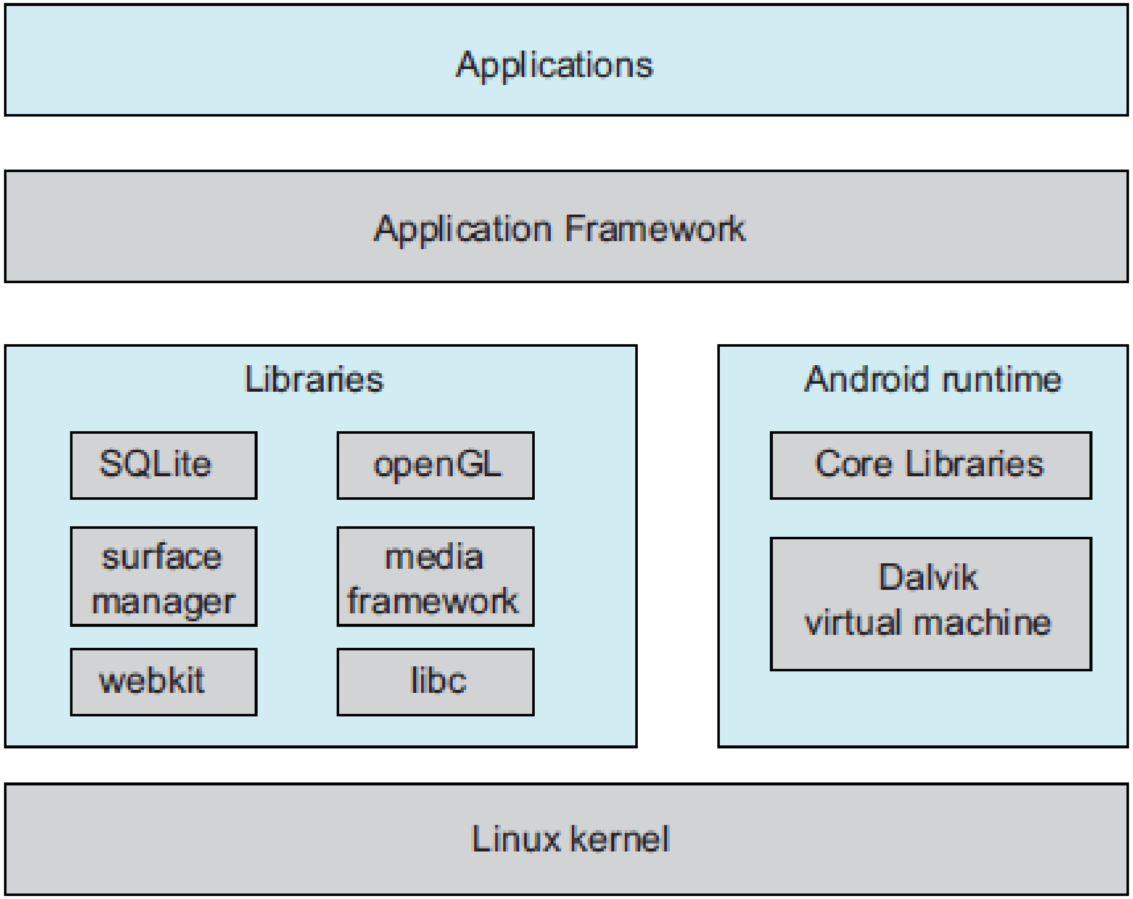


Figure 2.18 Architecture of Google's Android.

System Boot

- How does the hardware know where the kernel is or how to load that kernel?
- The procedure of starting a computer by loading the kernel is known as booting the system.
- In most computer systems, a small piece of code known as the **bootstrap program** or **bootstrap loader** locates the kernel, loads it into main memory, and starts its execution
- In some computer systems, such as PCs, use a **two-step process** in which a simple bootstrap loader fetches a more complex boot program from disk, which in turn loads the kernel

- When a CPU receives a reset event— i.e., when it is powered up or rebooted—the instruction register is loaded with a predefined memory location, and execution starts there. At that location is the initial bootstrap program.
- This program is in the form of read-only memory (**ROM**) because the RAM is in an unknown state at system startup.
- ROM is convenient because it needs no initialization and cannot easily be infected by a computer virus

System Boot

- The bootstrap program can perform a variety of tasks.
  - Usually, one task is to run diagnostics to determine the state of the machine. If the diagnostics pass, the program can continue with the booting steps.
  - It can also **initialize** all aspects of the system, from **CPU registers** to **device controllers** and the contents of **main memory**. Sooner or later, it starts the operating system
- A problem with the approach to store the entire operating system in ROM is that changing the bootstrap code requires changing the ROM hardware chips.
- Some systems resolve this problem by using erasable programmable read-only memory (**EPROM**), which is read-only except when explicitly given a command to become writable.
- All forms of ROM are also known as firmware, since their characteristics fall somewhere between those of hardware and those of software.
- A problem with firmware in general is that executing code there is slower than executing code in RAM.
- For large operating systems (including most general-purpose operating systems like Windows, Mac OS X, and UNIX) or for systems that change frequently, the bootstrap loader is stored in **firmware**, and the operating system is on disk.

CONTENTS

- Operating System Services
- User and OS interface
- System calls
- Types of system calls
- OS structure
- System boot

THANK YOU