OPERATING SYSTEMS

MODULE I

Introduction:

- What Operating Systems Do?
- Computer-System Organization
- Computer-System Architecture
- Operating-System Operations
- Resource Management
- Security and Protection
- Virtualization
- Kernel Data Structures
- Computing Environments

What Operating systems do?

- An operating system is software 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.
- A computer system can be divided roughly into four components: th[illegible]stem, the application prog

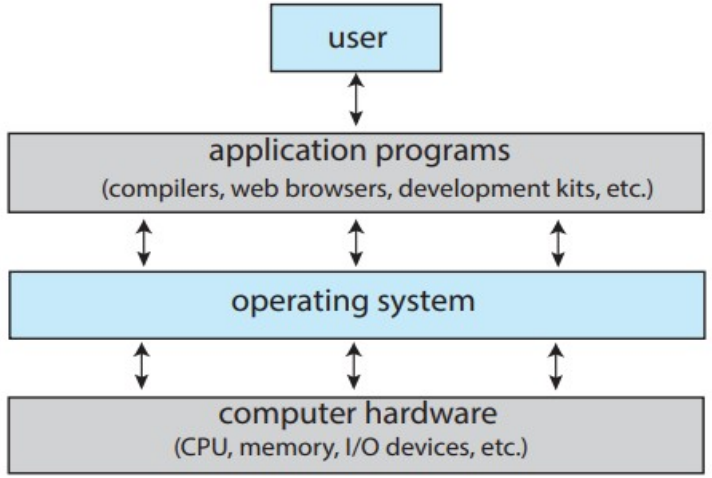


Figure 1.1 Abstract view of the components of a computer system.

What Operating systems do?

- Hardware

  provides the basic computing resources for the system.

  the central processing unit (CPU), the memory, and the input/output (I/O) devices.

- Application programs

  define the ways in which these resources are used to solve users' computing problems.

  word processors, spreadsheets, compilers, and web browsers

- Operating system

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

- Users

  People, machines, other computers

What Operating systems do?

- Operating systems can be explored from two viewpoints:
  - User view
  - System view

User View:

The user's view of the computer varies according to the interface being used. Many computer users sit with a laptop or in front of a PC consisting of a monitor, keyboard, and mouse. Such a system is designed for one user to monopolize 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 limited attention paid to performance and security and none paid to resource utilization—how various hardware and software resources are shared.

What Operating systems do?

- Increasingly, many users interact with mobile devices such as smartphones and tablets—and these devices are typically connected to networks through cellular or other wireless technologies.
- The user interface for mobile computers features a touch screen, where the user interacts with the system by pressing and swiping fingers across the screen rather than using a physical keyboard and mouse.
- Many mobile devices also allow users to interact through a voice recognition interface, such as Apple's Siri.
- For example, embedded computers in home devices and automobiles may have numeric keypads and may turn indicator lights on or off to show status, but they and their operating systems and applications are designed primarily to run without user intervention.

What Operating systems do?

System View:

From the computer's point of view, the operating system is the program most intimately involved with the hardware.

Operating system is viewed as a resource allocator.

A computer system has many resources that may be required to solve a problem: CPU time, memory space, storage space, I/O devices. The operating system acts as the manager of these 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.

Operating system emphasizes the need to control the various I/O devices and user programs.

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.

Computer System Organization

A modern general-purpose computer system consists of one or more CPUs and a number of device controllers connected through a common bus that provides access between components and shared memory.

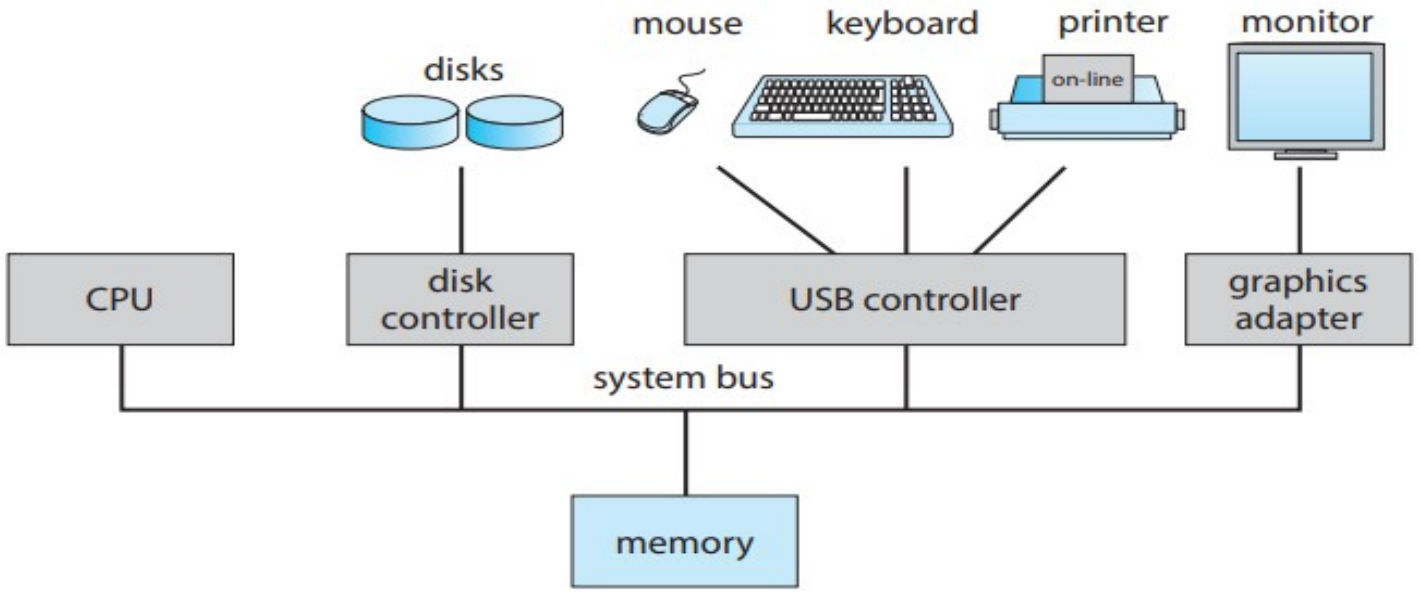


Figure 1.2 A typical PC computer system.

Computer System Organization

Each device controller is in charge of a specific type of device (example, a disk drive, audio device, or graphics display). Depending on the controller, more than one device may be attached. For instance, one system USB port can connect to a USB hub, to which several devices can connect.

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

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

CPU moves data from the main memory to the local buffers.

Operating systems have a device driver for each device controller. Device controller informs CPU that it has finished execution by causing an interrupt.

The CPU and the device controllers can execute in parallel, competing for memory cycles. To ensure orderly access to the shared memory, a memory controller synchronizes access to the memory.

Storage Definitions and Notations

The basic unit of computer storage is the bit. A bit can contain one of two values, 0 and 1.

All other storage in a computer is based on collections of bits.

A byte is 8 bits, and it is the smallest chunk of storage. For example, most computers don't have an instruction to move a bit but to move a byte.

A word is a unit of data and made up of one or more bytes. For example, a computer that has 64-bit registers and 64-bit memory addressing has 64-bit (8-byte) words.

A computer executes many operations in word size rather than a byte at a time.

Computer storage, is generally measured and manipulated in bytes and collections of bytes.

A kilobyte, or KB, is 1,024 bytes;

a megabyte, or MB, is 1,024 KB; (1,048,576 bytes)

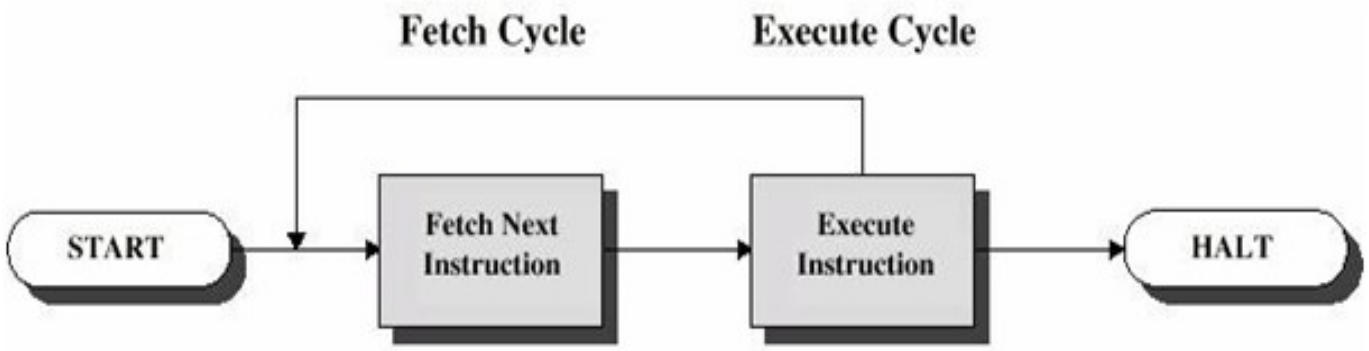
a gigabyte, or GB, is 1,024 MB; (1,073,741,824 bytes)

a terabyte, or TB, is 1,024 GB (1, 099, 511, 627, 776 bytes);

and a petabyte, or PB, is 1,024 TB (1, 125, 899, 906, 842, 624 bytes)

INSTRUCTION EXECUTION

- A program to be executed by a processor consists of a set of instructions stored in memory. The instruction processing consists of two steps.
  - The processor reads (fetches) instructions from memory one at a time **(fetch stage).**
  - Execute the instruction**(execute stage)**
- Program execution consists of repeating the process of instruction fetch and instruction execution
- The two steps are referred to as the fetch stage and the execute stage.
- The processing required for a single instruction is called an **instruction cycle.**



Instruction Fetch and Execute

- At the beginning of each instruction cycle, the processor fetches an instruction from memory.
- The instruction contains bits that specify the action the processor is to take. The processor interprets the instruction and performs the required action.
- In general, these actions fall into four categories,
- **Processor-memory:** Data may be transferred from processor to memory or from memory to processor.
- **Processor-I/O**: Data may be transferred to or from a peripheral device by transferring between the processor and an I/O module.
- **Data processing**: The processor may perform some arithmetic or logic operation on data.
- **Control**: An instruction may specify that the sequence of execution be altered.

Instruction Execution Cycle

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 an internal register within the CPU, whereas the store instruction moves the content of a register to main memory.

CPU fetches an instruction from main memory for execution from the location stored in the program counter.

It stores that instruction in the instruction register.

The instruction is then decoded and may cause operands to be fetched from memory and stored in some internal register.

After the instruction on the operands has been executed, the result may be stored back in memory.

Storage Structure

The CPU can load instructions only from memory, so any programs must first be loaded into memory to run.

General-purpose computers run most of their programs from rewritable memory, called main memory (also called random-access memory, or RAM).

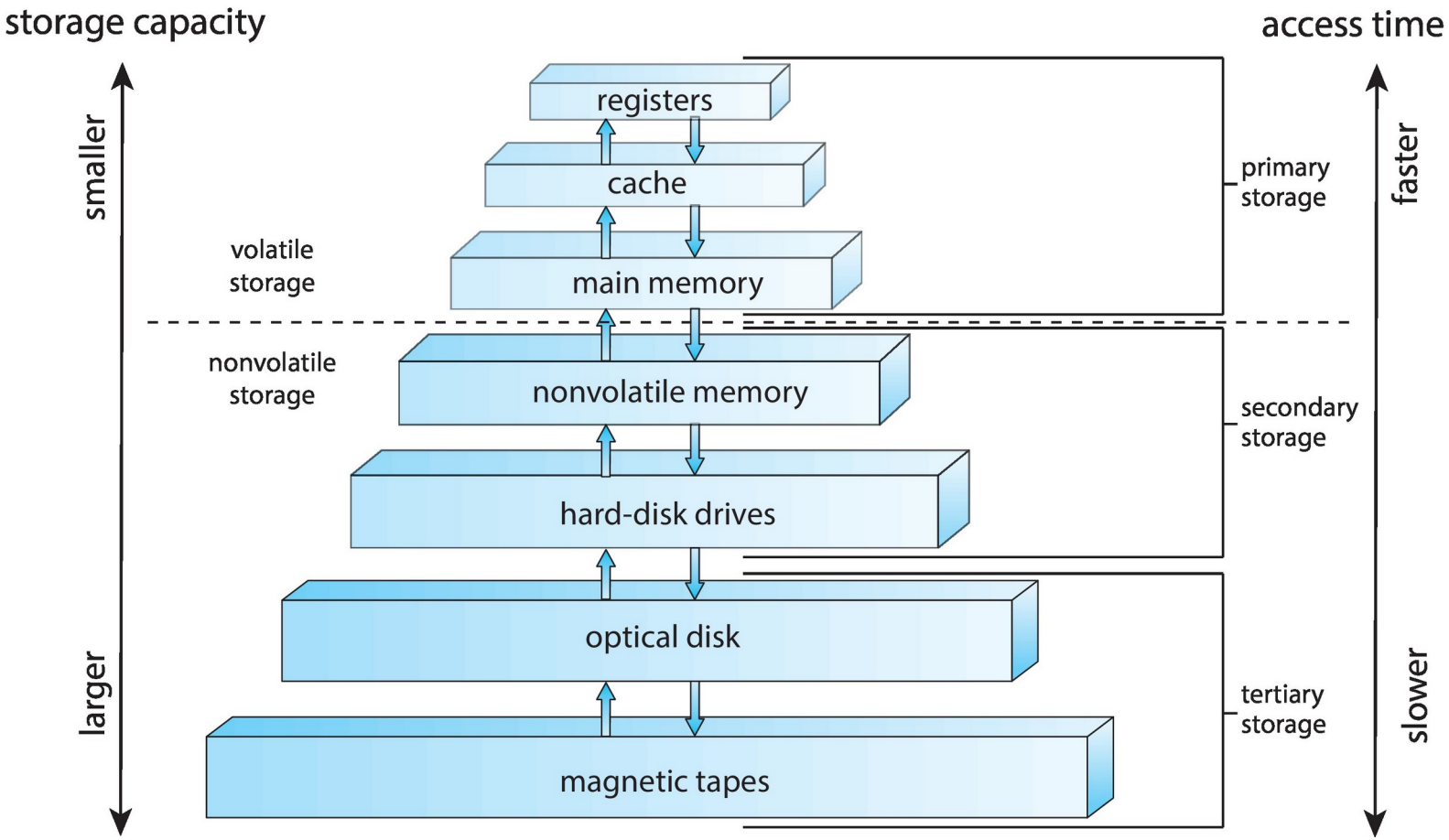
Computers use other forms of memory also. For example, the first program to run on computer power-on is a bootstrap program, which then loads the operating system.

Since RAM is volatile—loses its content when power is turned off or otherwise lost—we cannot trust it to hold the bootstrap program.

For this the computer uses electrically erasable programmable read-only memory (EEPROM) that is nonvolatile.

EEPROM can be changed but cannot be changed frequently. It is low speed, and so it contains mostly static programs and data that aren't frequently used. For example, the iPhone uses EEPROM to store serial numbers and hardware information about the device.

Storage Device Hierarchy



Storage Structure

The top levels of memory are constructed using semiconductor memory, which consists of semiconductor-based electronic circuits.

Main memory:

Too small to store all needed programs and data permanently.

Volatile—it loses its contents when power is turned off or otherwise lost.

Main memory is implemented in a semiconductor technology called dynamic random-access memory (DRAM).

Secondary storage:

extension of main memory; It should be able to hold large quantities of data permanently.

Hard Disk Drives (HDD):

Most common secondary-storage devices.

Most programs (system and application) are stored in secondary storage until they are loaded into memory. Secondary storage is also much slower than main memory.

Nonvolatile memory (NVM) devices, which provide storage for both programs and data. Faster than hard disks.

Common form of NVM device is flash memory, which is popular in mobile devices such as smartphones and tablets and is being used for long-term storage on laptops, desktops, and servers.

Computer System Architecture

Classification is based on the number of general purpose processors.

- Single-Processor Systems
- Multiprocessor Systems
- Clustered Systems

Single-Processor Systems

- Computer systems that consists of only one general-purpose CPU with a single processing core, then the system is a single-processor system.
- The core is the component that executes instructions and registers for storing data locally.
- These systems have other special-purpose processors also. They may come in the form of device-specific processors, such as disk, keyboard, and graphics controllers.
- All these special-purpose processors run a limited instruction set and do not run processes.
- Sometimes, they are managed by the operating system, in that the operating system sends them information about their next task and monitors their status. For example, a disk-controller microprocessor receives a sequence of requests from the main CPU core and implements its own disk queue and scheduling algorithm. This arrangement relieves the main CPU of the overhead of disk scheduling.
- PCs contain a microprocessor in the keyboard to convert the keystrokes into codes to be sent to the CPU.

Multi-Processor Systems

- Systems have two (or more) processors, each with a single-core CPU. They are called as parallel processing or tightly coupled systems.
- The processors share the computer bus and sometimes the clock, memory, and peripheral devices.
- The primary advantage of multiprocessor systems is increased throughput. That is, by increasing the number of processors, more work can be done in less time.
- Increased Reliability – When one processor fails the other manages the tasks.
- When multiple processors cooperate on a task, a certain amount of overhead is incurred in keeping all the parts working correctly. This overhead, and contention for shared resources, lowers the expected gain from additional processors.
- Economical to scale

Multi-Processor Systems

- The most common multiprocessor systems use symmetric multiprocessing (SMP), in which each peer CPU processor performs all tasks, including operating-system functions and user processes.

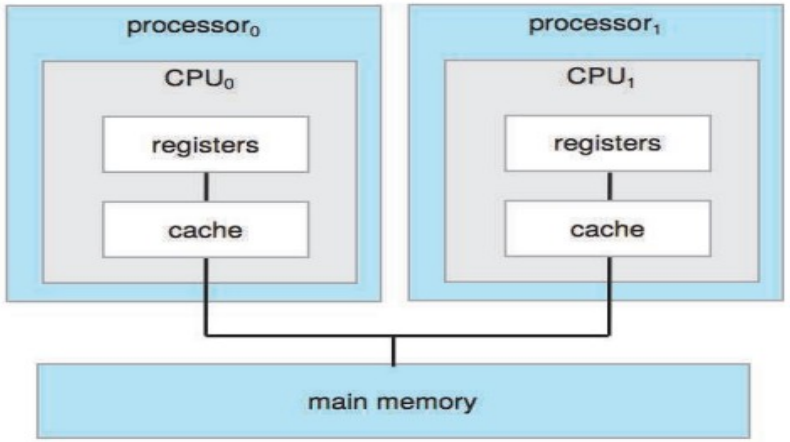


Figure 1.8 Symmetric multiprocessing architecture.

- Figure 1.8 illustrates a typical SMP architecture with two processors, each with its own CPU.
- Each CPU processor has its own set of registers, and a private—or local— cache. However, all processors share physical memory over the system bus.
- The benefit of this model is that many processes can run simultaneously —N processes can run if there are N CPUs—without causing performance to deteriorate significantly.
- However, since the CPUs are separate, one may be sitting idle while another is overloaded, resulting in inefficiencies. A multiprocessor system of this form will allow processes and resources—such as memory— to be shared dynamically among the various

Multi-Processor Systems

- Multicore systems, in which multiple computing cores reside on a single chip.
- Multicore systems 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, an important issue for mobile devices as well as laptops.

Multi-Processor Systems

- Figure 1.9, shows a dual-core design with two cores on the same processor chip.

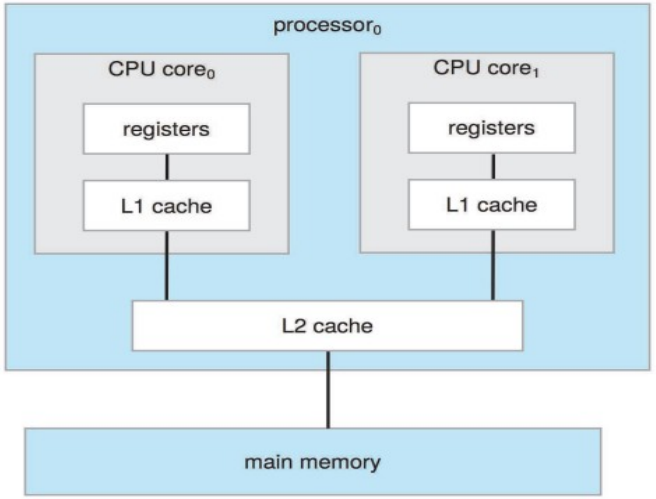


Figure 1.9 A dual-core design with two cores on the same chip.

- In this design, each core has its own register set, its own local cache, often known as a level 1, or L1, cache. A level 2 (L2) cache is local to the chip but is shared by the two processing cores.
- Most architectures adopt this approach, combining local and shared caches, where local, lower-level caches are generally smaller and faster than higher-level shared caches.
- All modern operating systems including Windows, macOS, and Linux, Android and iOS mobile systems support multicore SMP systems.

Clustered Systems

- Another type of multiprocessor system is a clustered system, which gathers together multiple CPUs.
- Clustered systems differ from the multiprocessor systems that they are composed of two or more individual systems—or nodes—joined together; each node is typically a multicore system. Such systems are considered loosely coupled.
- Clustered computers share storage and are closely linked via a local-area network LAN or a faster interconnect, such as InfiniBand.
- Clustering is used to provide high-availability service— that is, service that will continue even if one or more systems in the cluster fail.
- A layer of cluster software runs on the cluster nodes. Each node can monitor one or more of the others (over the network). If the monitored machine fails, the monitoring machine can take ownership of its storage and restart the applications that were running on the failed machine.

Operating System Operations

- 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, when a computer is powered on or rebooted.
- It is stored within the computer hardware (ROM) and 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.
- 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.
- Another form of interrupt is a trap (or an exception), which is a software-generated interrupt caused either by an error (for example, division by zero or invalid memory access) or by a specific request from a user program that an operating-system service be performed by executing a special operation called a system call.

Multiprogramming and Multitasking

- One of the most important aspects of operating systems is the ability to run multiple programs, keep either the CPU or the I/O devices busy at all times.
- Multiprogramming is the process of running more than one program at a time. It increases CPU utilization, by organizing programs so that the CPU always has one to execute.
- In a multi-programmed system, a program in execution is termed a process.
- The idea is as follows: The operating system keeps several processes in memory simultane g system picks and begins to execute one ventually, the process may have to wait f as an I/O operation, to complete.

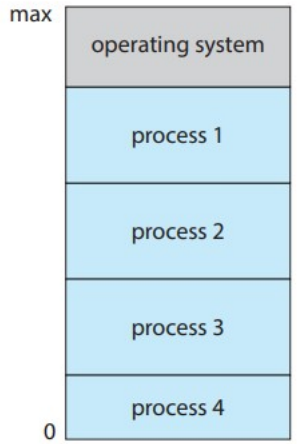
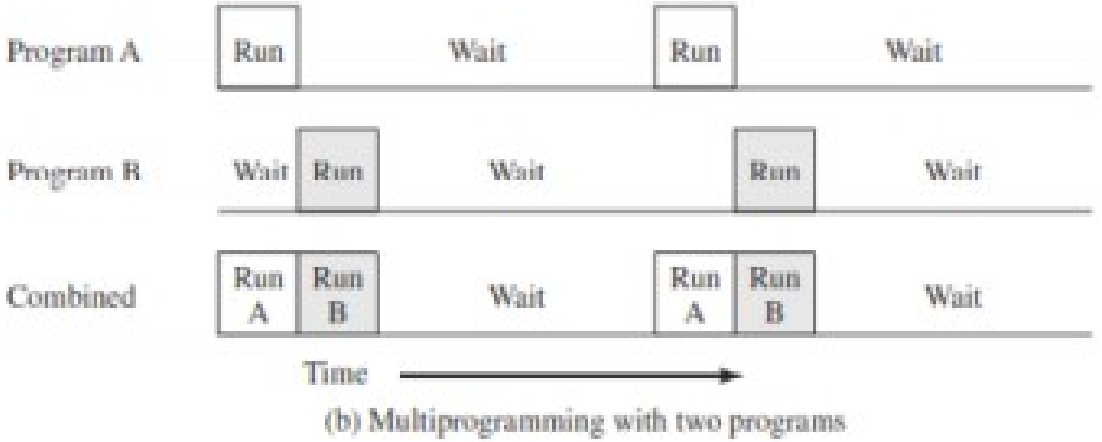


Figure 1.12 Memory layout for a multiprogramming system.

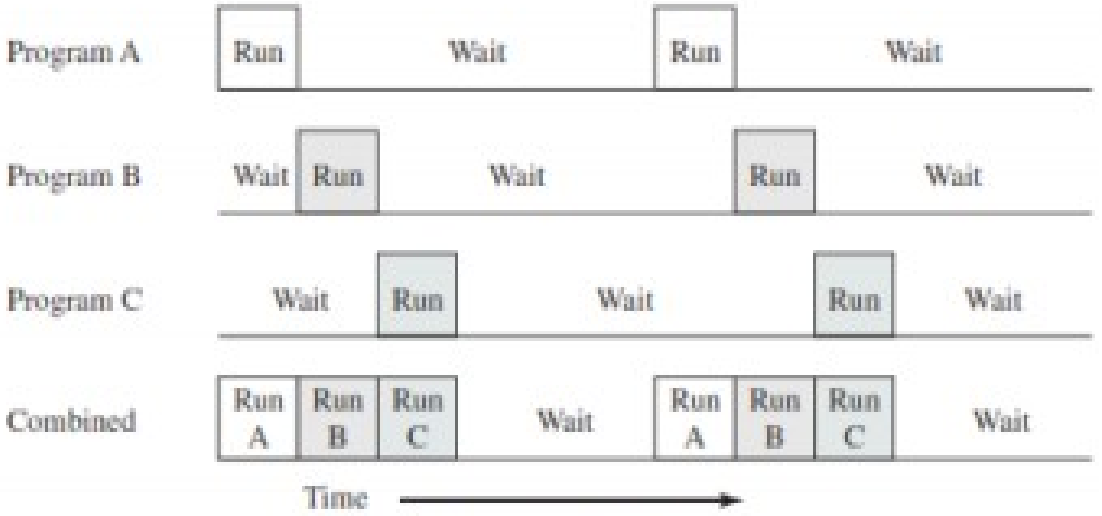
Multiprogramming and Multitasking

- In a non-multiprogrammed system, the CPU would sit idle.
- In a multi-programmed system, the operating system simply switches to, and executes, another process. When that process needs to wait, the CPU switches to another process, and so on. Eventually, the first process finishes waiting and gets the CPU back. As long as at least one process needs to execute, the CPU is never idle.
- Multitasking is a logical extension of multiprogramming.
- In multitasking systems, the CPU executes multiple processes by switching among them, but the switches occur frequently, providing the user with a fast response time.
- Consider that when a process executes, it typically executes for only a short time before it either finishes or needs to perform I/O.

Multiprogrammed Systems



(b) Multiprogramming with two programs



(c) Multiprogramming with three programs

Dual-Mode and Multimode Operation

- Since the operating system and its users share the hardware and software resources of the computer system, a properly designed operating system must ensure that an incorrect (or malicious) program cannot cause other programs —or the operating system itself— to execute incorrectly.
- In order to ensure the proper execution of the system, there are 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.

Dual-Mode and Multimode Operation

- 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.

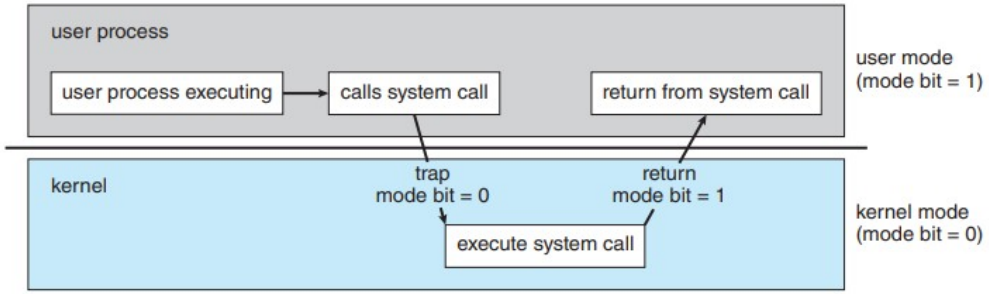


Figure 1.13 Transition from user to kernel mode.

- 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 (that is, changes the state of the mode bit to 0).
- Thus, whenever the operating system gains control of the computer, it is in kernel mode.
- The system always switches to user mode (by setting the mode bit to 1) before passing control to a user program.
- Some of the machine instructions that may cause harm are designated as privileged instructions.
- The hardware allows privileged instructions to be executed only in kernel mode. If an attempt is made to execute a privileged instruction in user mode, the hardware does not execute the instruction but rather treats it as illegal and traps it to the operating system

Timer

- It is ensured that the operating system maintains control over the CPU.
- A user program cannot be allowed to get stuck in an infinite loop or to fail to call system services and never return control to the operating system.
- To accomplish this goal, a timer is used.
- A timer can be set to interrupt the computer after a specified period. The period may be fixed (for example, 1/60 second) or variable (for example, from 1 millisecond to 1 second).
- A variable timer is implemented by a fixed-rate clock and a counter. The operating system sets the counter.
- Every time the clock ticks, the counter is decremented. When the counter reaches 0, an interrupt occurs.
- For instance, a 10-bit counter with a 1-millisecond clock allows interrupts at intervals from 1 millisecond to 1,024 milliseconds, in steps of 1 millisecond.
- Before turning over control to the user, the operating system ensures that the timer is set to interrupt. If the timer interrupts, control transfers automatically to the operating system, which may treat the interrupt as a fatal error or may give the program more time.
- Clearly, instructions that modify the content of the timer are privileged.

OPERATING SYSTEM STRUCTURE

- The operating systems are large and complex.
- A common approach for a system to function properly is to partition the task into small components, or modules, rather than have one single system.
- Each of these modules should be a well-defined portion of the system, with carefully defined interfaces and functions.
- The structure of an operating system defines the following structures.
  - Monolithic structure
  - Layered approach
  - Microkernels
  - Modules
  - Hybrid systems

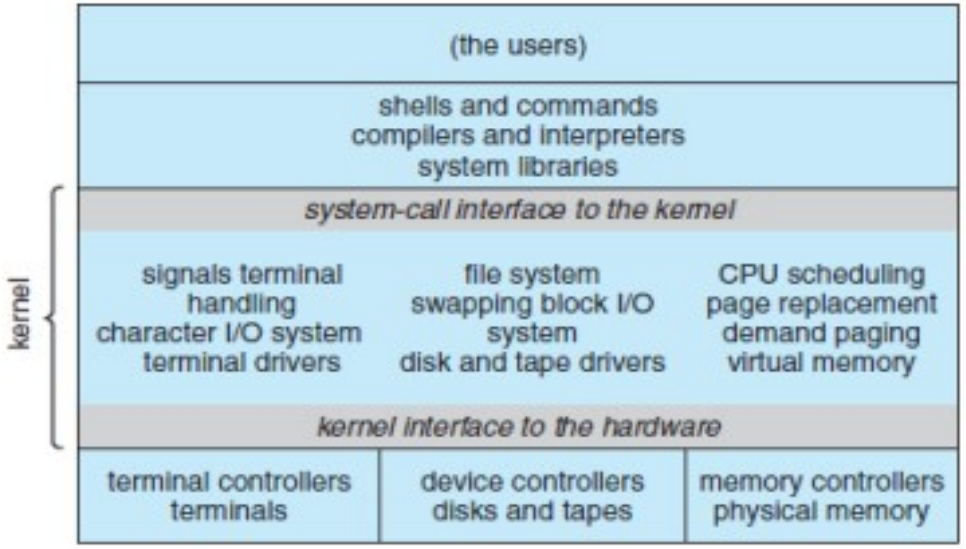
Monolithic structure

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.

Monolithic structure is a common technique for designing operating systems.

The traditional UNIX operating system can be viewed as being layered.

Example: Traditional UNIX OS



Monolithic structure

- UNIX operating system consists of two separable parts:
  the kernel and the system programs.
- The kernel is further separated into a series of interfaces and device drivers.
- Everything below the system-call interface and above the physical hardware is the kernel.
- The kernel provides the file system, CPU scheduling, memory management, and other operating system functions through system calls.
- The monolithic approach is often known as a tightly coupled system because changes to one part of the system can have effects on other parts.

Monolithic structure

- The Linux operating system is based on UNIX.
- Applications use the glibc standard C library when communicating with the system call interface to the kernel.
- The Linux kernel is monolithic in that it runs entirely in kernel mode in a single address space, but, it does have a modular design that allows the kernel to be modified during run time.
- Monolithic kernels have a distinct performance, very little overhead in the system-call interface, and communication within the kernel is fast.

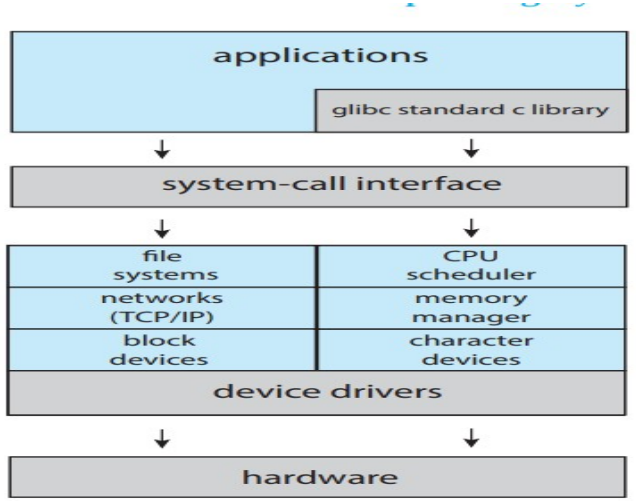
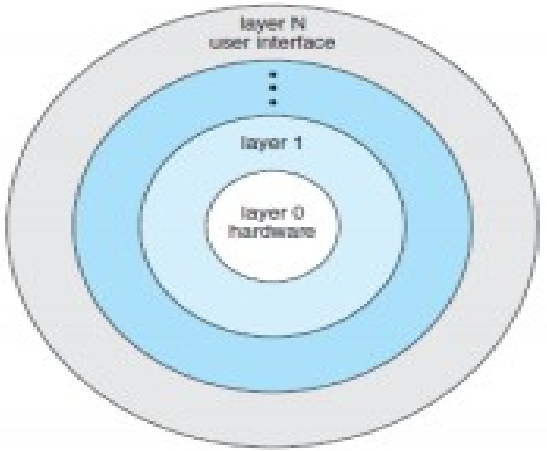


Figure 2.13 Linux system structure.

Layered approach

- Layered approach is a loosely coupled system. Such a system is divided into separate, smaller components that have specific and limited functionalities.
- The advantage of this modular approach is that changes in one component affect only that component, and no others.
- A system can be made modular in many ways. One method is the layered approach, in which the operating system is broken into a number of layers (levels). The bottom layer (layer 0) is the hardware; the highest (layer N) is the user interface.
- An operating-system layer is an implementation of an abstract object made up of data and the operations that can manipulate those data.
- A typical operating-system layer—say, layer M—consists of data structures and a set of functions that can be invoked by higher-level layers.
- Layer M, in turn can invoke operations on lower level layers.



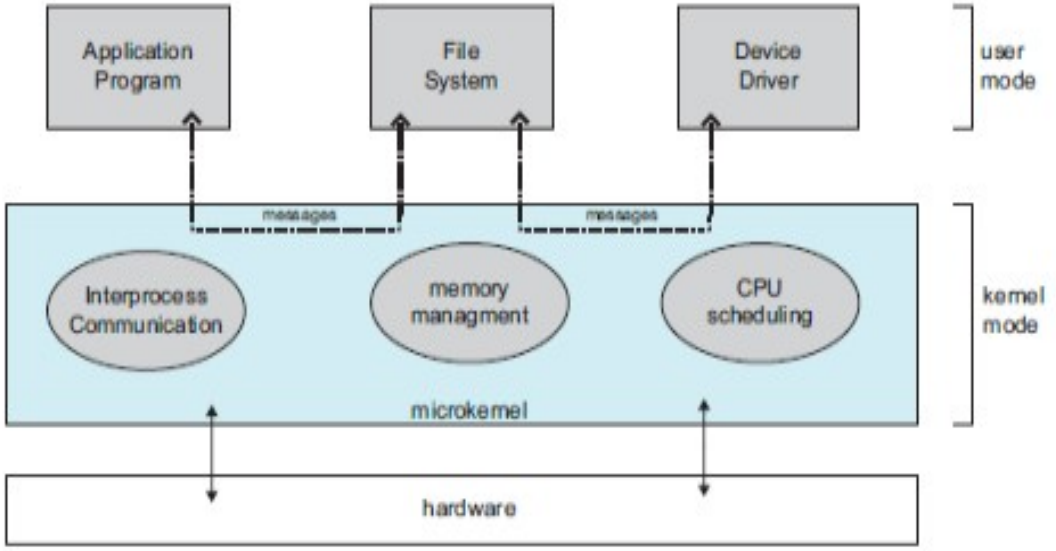
Layered approach

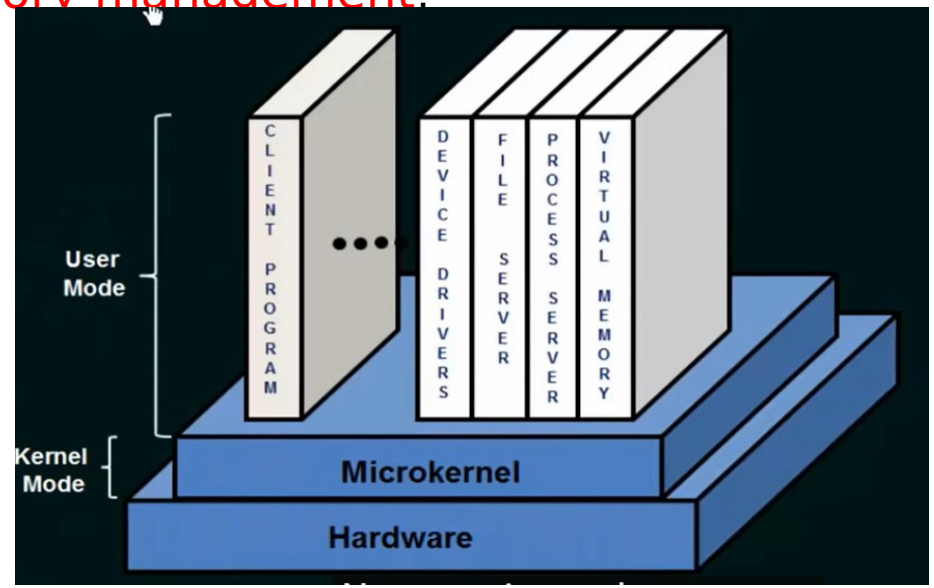
- The main advantage of the layered approach is simplicity of construction and debugging.
- The layers are selected so that each uses functions (operations) and services of only lower-level layers.
- Each layer is implemented only with operations provided by lower-level layers. A layer does not need to know how these operations are implemented; it needs to know only what these operations do.
- Hence, each layer hides the existence of certain data structures, operations, and hardware from higher level layers.
- The major difficulty with the layered approach involves designing and defining the various layers because a layer can use only lower-level layers.
- A problem with layered implementations is that they tend to be less efficient than other types.
- Layered systems are used in computer networks (such as TCP/IP) and web applications.

Microkernels

- In the mid-1980s, an operating system called Mach was developed that modularized the kernel using the microkernel approach.
- This method structures the operating system by removing all nonessential components from the kernel and implementing them as system and user-level programs that reside in separate address spaces.
- The result is a smaller kernel. There is little consensus regarding which services should remain in the kernel and which should be implemented in user space.

emory management.





Microkernels

- 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.
- For example, if the client program wishes to access a file, it must interact with the file server. The client program and service never interact directly. Rather, they communicate indirectly by exchanging messages with the microkernel.
- One benefit of the microkernel approach is that it makes extending the operating system easier. All new services are added to user space and consequently do not require modification of the kernel.
- When the kernel does have to be modified, the changes tend to be fewer, because the microkernel is a smaller kernel.
- The microkernel also provides more security and reliability, since most services are running as user rather than kernel processes. If a service fails, the rest of the operating system remains

Microkernels

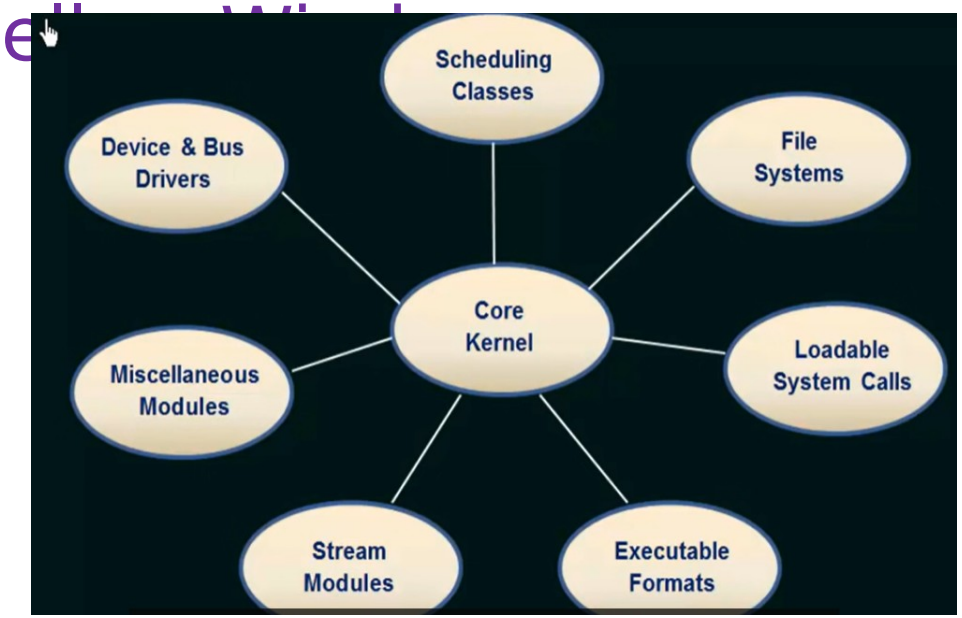
- The best-known illustration of a microkernel operating system is Darwin, the kernel component of the macOS and iOS operating systems. Darwin, consists of two kernels, one of which is the Mach microkernel.
- Another example is QNX, a real-time operating system for embedded systems.
- The QNX Neutrino microkernel provides services for message passing and process scheduling. It also handles low-level network communication and hardware interrupts.
- All other services in QNX are provided by standard processes that run outside the kernel in user mode.
- When two user-level services must communicate, messages must be copied between the services, which reside in separate address spaces. In addition, the operating system may have to switch from one process to the next to exchange the messages.
- The overhead involved in copying messages and switching between processes is the disadvantage in microkernel-based

Microkernels

- Consider the history of Windows NT:
- The first release had a layered microkernel organization.
- This version's performance was low compared with that of Windows 95.
- Windows NT 4.0 partially corrected the performance problem by moving layers from user space to kernel space and integrating them more closely.
- By the time Windows XP was designed, Windows architecture had become more monolithic than microkernel.

Modules

- The best current methodology for operating-system design involves using loadable kernel modules (LKM's).
- In this, 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 Linux, macOS, and Solaris, as we



Modules

- 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.
- Thus, for example, we might build CPU scheduling and memory management algorithms directly into the kernel and then add support for different file systems by way of loadable modules.
- Linux uses loadable kernel modules, primarily for supporting device drivers and file systems.
- LKMs can be “inserted” into the kernel as the system is started (or booted) or during run time, such as when a USB device is plugged into a running machine.
- If the Linux kernel does not have the necessary driver, it can be dynamically loaded.
- LKMs can be removed from the kernel during run time as well.

Modules

- For Linux, LKMs allow a dynamic and modular kernel, while maintaining the performance benefits of a monolithic system.
- Example: Solaris OS

  The Solaris operating system structure is organized around a core kernel with seven types of loadable kernel modules:

  - Scheduling classes
  - File systems
  - Loadable system calls
  - Executable formats
  - STREAMS modules
  - Miscellaneous
  - Device and bus drivers

Hybrid Systems

- The Operating System combines different structures, resulting in hybrid systems that address performance, security, and usability issues.
- For example, Linux is monolithic, because having the operating system in a single address space provides very efficient performance. However, it also modular, so that new functionality can be dynamically added to the kernel.
- Windows is largely monolithic, but it retains some behavior typical of microkernel systems, including providing support for separate subsystems (known as operating-system personalities) that run as user-mode processes.
- Windows systems also provide support for dynamically loadable kernel modules.
- Examples of hybrid systems are: the Apple macOS operating system and the two most prominent mobile operating systems—iOS and Android.

macOS and iOS

- Apple's macOS operating system is designed to run primarily on desktop and laptop computer systems, whereas iOS is a mobile operating system designed for the iPhone smartphone and iPad tablet computer.
- Architecturally, macOS and iOS have much in common and it has been highlighted what they share as well as how they differ from each other.
- The general architecture of these two systems is shown in Figure 2.16.

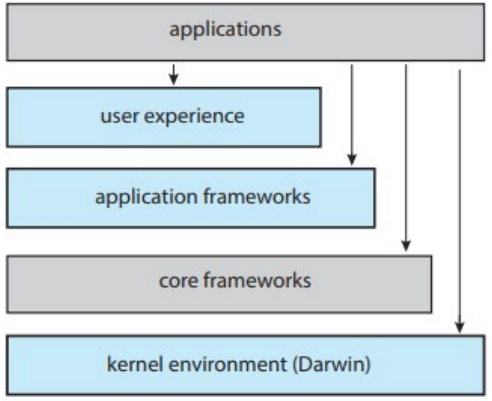


Figure 2.16 Architecture of Apple's macOS and iOS operating systems.

macOS and iOS

Highlights of the various layers include the following:

- User experience layer: This layer defines the software interface that allows users to interact with the computing devices. macOS uses the Aqua user interface, which is designed for a mouse or trackpad, whereas iOS uses the Springboard user interface, which is designed for touch devices.
- Application frameworks layer: This layer includes the Cocoa and Cocoa Touch frameworks, which provide an API for the Objective-C and Swift programming languages.
- The primary difference between Cocoa and Cocoa Touch is that the former is used for developing macOS applications, and the latter by iOS to provide support for hardware features unique to mobile devices, such as touch screens.
- Core frameworks: This layer defines frameworks that support graphics and media including, Quicktime and OpenGL.
- Kernel environment: This environment, also known as Darwin, includes the Mach microkernel and the BSD UNIX kernel.

macOS and iOS

The significant distinctions between macOS and iOS include the following:

- Because macOS is intended for desktop and laptop computer systems, it is compiled to run on Intel architectures. iOS is designed for mobile devices and thus is compiled for ARM-based architectures. Similarly, the iOS kernel has been modified to address specific features and needs of mobile systems, such as power management and aggressive memory management. Additionally, iOS has more stringent security settings than macOS.
- The iOS operating system is generally much more restricted to developers than macOS and may even be closed to developers. For example, iOS restricts access to POSIX and BSD APIs on iOS, whereas they are openly available to developers on macOS.

The structure of Darwin

- Darwin, uses a hybrid structure. Darwin is a layered system that primarily consists of the Mach microkernel and the BSD UNIX kernel.
- Most operating systems provide a single system-call interface to the kernel through the standard C library on UNIX and Linux systems.
- Darwin provides two system-call interfaces:

Mach system calls (known as traps) and BSD system calls (which provide POSIX functionality).

The interface to these system calls is a rich set of libraries that includes not only the standard C library but also libraries that provide networking, security, and programming language.

Darwin's structure is shown in Figure 2.17.

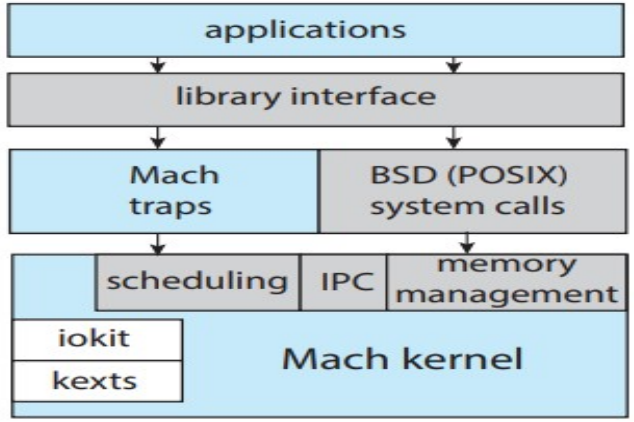


Figure 2.17 The structure of Darwin.

The structure of Darwin

- Beneath the system-call interface, Mach provides fundamental operating system services, including memory management, CPU scheduling, and interprocess communication (IPC) facilities such as message passing and remote procedure calls (RPCs).
- Much of the functionality provided by Mach is available through kernel abstractions, which include tasks (a Mach process), threads, memory objects, and ports (used for IPC).
- As an example, an application may create a new process using the BSD POSIX fork() system call.
- Mach will, in turn, use a task kernel abstraction to represent the process in the kernel.
- In addition to Mach and BSD, the kernel environment provides an I/O kit for development of device drivers and dynamically loadable modules (which macOS refers to as kernel extensions, or kexts).
- Darwin combines Mach, BSD, the I/O kit, and any kernel extensions into a single address space. Thus, Mach is not a pure microkernel in the sense that various subsystems run in user space.

Android

- The Android operating system was designed by the Open Handset Alliance (led primarily by Google) and was developed for Android smartphones 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 supporting graphics, audio, and hardware features. These features, in turn, provide a platform for developing mobile applications that run on a multitude of Android-enabled devices.
- The structure of Android appears in Figure 2.18.

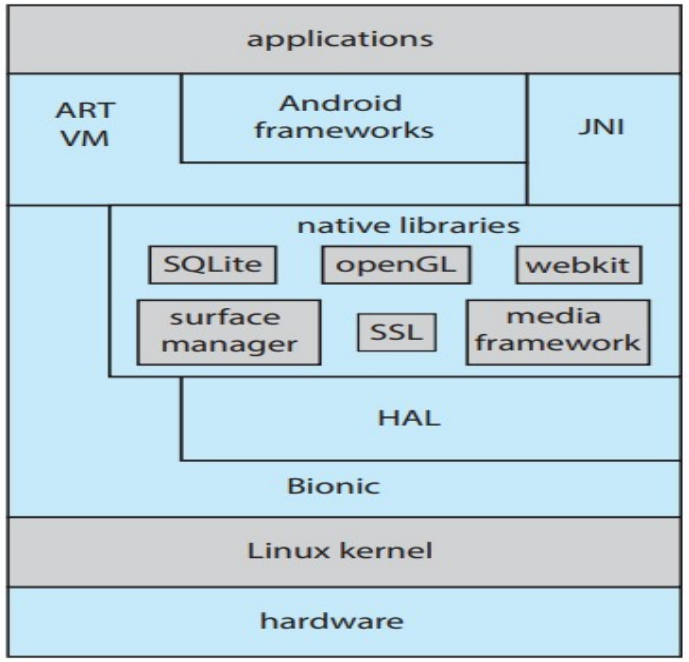


Figure 2.18 Architecture of Google's Android.

Android

- Software designers for Android devices develop applications in the Java language, but they do not generally use the standard Java API.
- Google has designed a separate Android API for Java development.
- Java applications are compiled into a form that can execute on the Android RunTime ART, a virtual machine designed for Android and optimized for mobile devices with limited memory and CPU processing capabilities.
- Java programs are first compiled to a Java bytecode .class file and then translated into an executable .dex file.
- Whereas many Java virtual machines perform just-in-time (JIT) compilation to improve application efficiency, ART performs ahead-of-time (AOT) compilation.
- Here, .dex files are compiled into native machine code when they are installed on a device, from which they can execute on the ART.
- AOT compilation allows more efficient application execution as well as reduced power consumption, features that are crucial for mobile systems.

Android

- Android developers can also write Java programs that use the Java native interface or JNI which allows developers to bypass the virtual machine and instead write Java programs that can access specific hardware features.
- Programs written using JNI are generally not portable from one hardware device to another.
- The set of native libraries available for Android applications includes frameworks for developing web browsers (webkit), database support (SQLite), and network support, such as secure sockets (SSLs).
- Because Android can run on an almost unlimited number of hardware devices, Google has chosen to abstract the physical hardware through the hardware abstraction layer, or HAL.
- By abstracting all hardware, such as the camera, GPS chip, and other sensors, the HAL provides applications with a consistent view independent of specific hardware.
- This feature, allows developers to write programs that are portable across different hardware platforms.
- The standard C library used by Linux systems is the GNU C library (glibc).
- Google instead developed the Bionic standard C library for Android. Not only does Bionic have a smaller memory footprint than glibc, but it also has been designed for the slower CPUs that characterize mobile devices. (In addition, Bionic allows Google to bypass GPL licensing of glibc.)

1. EFFICIENCY:

Throughput and Resource Utilization:

This aspect of efficiency measures how much useful work the operating system can accomplish in a given amount of time while utilizing the available resources optimally.

It focuses on maximizing the system's throughput, which is the number of tasks completed or processed per unit of time.

A highly efficient operating system can handle a significant amount of workload without wasting resources or causing unnecessary delays.

EFFICIENCY:

CPU Utilization and Response Time:

In a time-sharing system or a multi-user environment, efficiency can be measured by how well the operating system utilizes the CPU and how quickly it responds to user requests.

CPU utilization is the ratio of the time the CPU spends executing tasks compared to the total time.

A highly efficient OS aims to keep the CPU busy as much as possible, avoiding idle time.

Additionally, response time is the time it takes for the system to respond to a user's request.

Low response times indicate a more efficient system, as it means users experience less delay in getting their tasks processed

2.ROBUSTNESS:

- Robustness is the ability of a computer system to cope with errors during execution and with erroneous input.
- Formal techniques, such as fuzzy testing, are essential to show robustness since this type of testing involves invalid or unexpected inputs.
- Alternatively, fault injection can be used to test robustness.
- A distributed system may suffer from various types of hardware failure.
- The failure of a link, the failure of a site, and the loss of a message are the most common types.
- To ensure that the system is robust, we must detect any of these failures, reconfigure the system so that computation can continue, and recover when a site or a link is repaired.

2.ROBUSTNESS:

- Fault Tolerance: The software can handle unexpected errors or failures gracefully without causing a complete system crash. If one component fails, the system can continue functioning with minimal impact on other components.
- Error Handling: Robust software is designed to handle errors effectively, either by recovering from them or providing informative error messages to users for troubleshooting.
- Isolation: Failures in one application or process should not bring down the entire system. Robust systems are structured to provide isolation between different processes, ensuring that problems in one part do not affect others.
- Security: Robustness often goes hand-in-hand with security. A secure system can withstand attacks and attempts to compromise its integrity.

2.ROBUSTNESS:

- Flexibility: A flexible system can be easily extended, upgraded, or reconfigured to meet changing requirements without significant constraints or limitations.
- Portability: Portability, in the context of software and operating systems, refers to the ability of a program or the operating system itself to run on different platforms or hardware configurations with minimal changes or recompilation.
- For an operating system to be portable, it means the OS is designed and implemented in a way that allows it to be easily adapted to different hardware architectures. This includes making sure that the OS kernel and core components are hardware-independent and using proper abstraction layers to interact with the hardware.

2.ROBUSTNESS:

- In general, building robust systems that encompass every point of possible failure is difficult because of the vast quantity of possible inputs and input combinations.
- Since all inputs and input combinations would require too much time to test, developers cannot run through all cases exhaustively.
- Instead, the developer will try to generalize such cases. For example, imagine
  inputting some integer values. Some selected inputs might consist of a negative number, zero, and a positive number.
- When using these numbers to test software in this way, the developer generalizes the set of all reals into three numbers. This is a more efficient and manageable method, but more prone to failure.
- Generalizing test cases is an example of just one technique to deal with failure specifically, failure due to invalid user input.
- Systems generally may also fail due to other reasons as well, such as

3. Portability

- Portability is the ability of an application to run properly in a different platform to the one it was designed for, with little or no modification.
- Portability in high-level computer programming is the usability of the same software in different environments.
- When software with the same functionality is produced for several computing platforms, portability is the key issue for development cost reduction.

Design Issues of OS

4. Compatibility

- Compatibility is the capacity for two systems to work together without having to be altered to do so.
- Compatible software applications use the same data formats.
- For example, if word processor applications are compatible, the user should be able to open their document files in either product.
- Compatibility issues come up when users are using the same type of software for a task, such as word processors, that cannot communicate with each other.
- This could be due to a difference in their versions or because they are made by different companies.
- The huge variety of application software available and all the versions of the same software mean there are bound to be compatibility issues, even when people are using the same kind of software.

4. Compatibility

- Compatibility issues can be small, for example certain features not working properly in older versions of the same software, but they can also be problematic, such as when a newer version of the software cannot open a document created in an older version.
- In Microsoft Word for example, documents created in Word 2016 or 2013 can be opened in Word 2010 or 2007, but some of the newer features (such as collapsed headings or embedded videos) will not work in the older versions.
- If someone using Word 2016 opens a document created in Word 2010, the document will open in Compatibility Mode.
- Microsoft Office does this to make sure that documents created in older versions still work properly.

5. Flexibility

- Flexible operating systems are taken to be those whose designs have been motivated to some degree by the desire to allow the system to be tailored, either statically or dynamically, to the requirements of specific applications or application domains.

6. Scalability:

- Scalability is the measure of a system's ability to increase or decrease in performance and cost in response to changes in application and system processing demands.
- Commodity computer systems contain more and more processor cores and exhibit increasingly diverse architectural trade-offs, including memory hierarchies, interconnects, instruction sets and variants, and IO configurations.
- Previous high-performance computing systems have scaled in specific cases, but the dynamic nature of modern client and server workloads, coupled with the impossibility of statically optimizing an OS for all workloads and hardware variants pose serious challenges for operating system structures.

Influence of security, networking and multimedia

Security in Operating Systems:

- User Authentication and Access Control: Operating systems implement various authentication methods, such as passwords, biometrics, and two-factor authentication, to verify the identity of users. Access control mechanisms restrict users' permissions and privileges, ensuring that they can only access resources they are authorized to use.

- System Hardening: Operating systems employ security measures like firewalls, intrusion detection systems, and antivirus software to safeguard against external and internal threats. These measures help detect and prevent malicious activities, such as unauthorized network access, malware infections, and data breaches.

Security in Operating Systems:

- Secure Communication: Operating systems provide secure communication protocols, such as Transport Layer Security (TLS) and Virtual Private Networks (VPNs), to ensure the confidentiality, integrity, and authentication of data transmitted over networks.

- Encryption: Operating systems offer encryption mechanisms to protect sensitive data stored on disk or transmitted over networks. Encryption algorithms and protocols help prevent unauthorized access and data theft.

Networking in Operating Systems

- Internet Connectivity: Operating systems provide networking protocols and drivers to establish connections to the internet, enabling users to browse the web, access online services, and communicate over email or messaging platforms.
- Network File Sharing: Operating systems support network file sharing protocols, such as Server Message Block (SMB) or Network File System (NFS), allowing users to access shared files and folders on remote systems.
- Printing and Peripheral Devices: Operating systems include networking protocols for printing and connecting peripheral devices, such as printers and scanners, over a network. This allows users to share these devices across multiple computers.
- Remote Access: Operating systems offer remote access capabilities, such as Remote Desktop Protocol (RDP) or Secure Shell (SSH), which enable users to access and control remote systems over a network. This is particularly useful for system administration and troubleshooting purposes.

Multimedia in Operating Systems

- Streaming and Video Conferencing: Multimedia technologies enable real-time streaming of audio and video, facilitating applications like video conferencing and online media streaming.
- User Interfaces and Experience: Multimedia elements enhance user interfaces, making them more engaging and intuitive through the use of graphics, animations, and interactive elements.
- Gaming and Virtual Reality: Multimedia is foundational in gaming and virtual reality applications, creating immersive and interactive experiences.
- Entertainment Industry: Multimedia technologies have revolutionized the entertainment industry, from digital audio and video production to special effects in movies.

Abstraction

- The process of establishing the decomposition of a problem into simpler and more understood primitives is basic to science and software engineering.
- An abstraction is a model. The process of transforming one abstraction into a more detailed abstraction is called refinement. The new abstraction can be referred to as a refinement of the original one.
- Abstractions and their refinements do not coexist in the same system description.
- Composition occurs when two abstractions are used to define another higher abstraction.
- Decomposition occurs when an abstraction is split into smaller abstractions.

  Information management is one of the goals of abstraction. Complex features of one abstraction are simplified into another abstraction.

Abstraction

- Abstraction may be good or bad. Good abstractions can be very useful while bad abstractions can be very harmful. A good abstraction leads to reusable components.
- Information hiding distinguishes between public and private information.
- Only the essential information is made public while internal details are kept private. This simplifies interactions and localizes details and their operations into well-defined units.
- Abstraction, in traditional systems, forms layers representing different levels of complexity. Each layer describes a solution. These layers are then mapped onto each other.
- In this way, high level abstractions are materialized by lower-level

Abstraction

- Abstraction can be accomplished on functions, data, and processes.
- In functional abstraction, details of the algorithms to accomplish the function are not visible to the consumer of the function. The consumer of the function needs to only know the correct calling convention and have trust in the accuracy of the functional results.
- In data abstraction, details of the data container and the data elements may not be visible to the consumer of the data.
- The data container could represent a stack, a queue, a list, a tree, a graph, or many other similar data containers. The consumer of the data container is only concerned about correct behaviour of the data container and not many of the internal details.
- Also, exact details of the data elements in the data container may not

Process

- In the Operating System, a Process is a program that is currently under execution.
- So, an active program can be called a Process.
- For example, when you want to search something on web then you start a browser. This is denoted by process state.

As a process executes, it changes state

- new: The process is being created
- running: Instructions are being executed
- waiting: The process is waiting for some event to occur
- ready: The process is waiting to be assigned to a processor
- terminated: The process has finished execution

Resources

- Operating system (OS), is a program that manages a computer's resources, especially the allocation of those resources among other programs.
- Typical resources include the central processing unit (CPU), computer memory, file storage, input/output (I/O) devices, and network connections.
- Management tasks include scheduling resource use to avoid conflicts and interference between programs.
- Unlike most programs, which complete a task and terminate, an operating system runs indefinitely and terminates only when the computer is turned off.
- Modern multiprocessing operating systems allow many processes to be active, where each process is a "thread" of computation being

Resource

- Time-sharing must guard against interference between users' programs, and most systems use virtual memory, in which the memory, or "address space," used by a program may reside in secondary memory (such as on a magnetic hard disk drive) when not in immediate use, to be swapped back to occupy the faster main computer memory on demand.
- This virtual memory both increases the address space available to a program and helps to prevent programs from interfering with each other, but it requires careful control by the operating system and a set of allocation tables to keep track of memory use.
- Perhaps the most delicate and critical task for a modern operating system is allocation of the CPU; each process is allowed to use the CPU for a limited time, which may be a fraction of a second, and then must give up control and become suspended until its next turn.