UNIT-I

Introduction to OS

1.Operating System

An operating system is the most important software that runs on a computer. It manages the computer's memory and processes, as well as all of its software and hardware. It also allows you to communicate with the computer without knowing how to speak the computer's language.

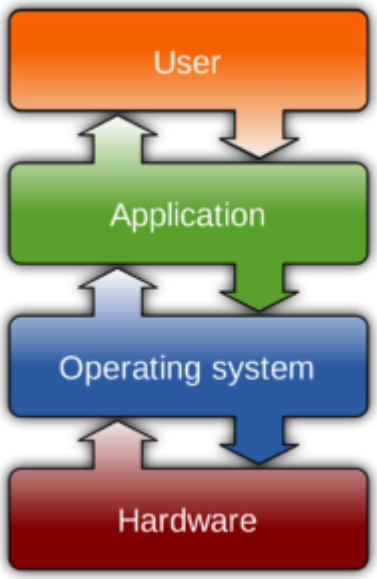
The operating system's job

Your computer's **operating system** (**OS**) manages all of the **software** and **hardware** on the computer. Most of the time, there are several different computer programs running at the same time, and they all need to access your computer's **central processing unit (CPU)**, **memory**, and **storage**. The operating system coordinates all of this to make sure each program gets what it needs.

Types of operating systems

Operating systems usually come **pre-loaded** on any computer you buy. Most people use the operating system that comes with their computer, but it's possible to upgrade or even change operating systems. The three most common operating systems for personal computers are **Microsoft Windows**, **macOS**, and **Linux**.

An operating system (OS) is a program that acts as an interface between the system hardware and the user. Moreover, it handles all the interactions between the software and the hardware. All the working of a computer system depends on the OS at the base level. Further, it performs all the functions like handling memory, processes, the interaction between hardware and software, etc. Now, let us look at the functions of operating system.



Operating System

Abstract view of the components of the computer

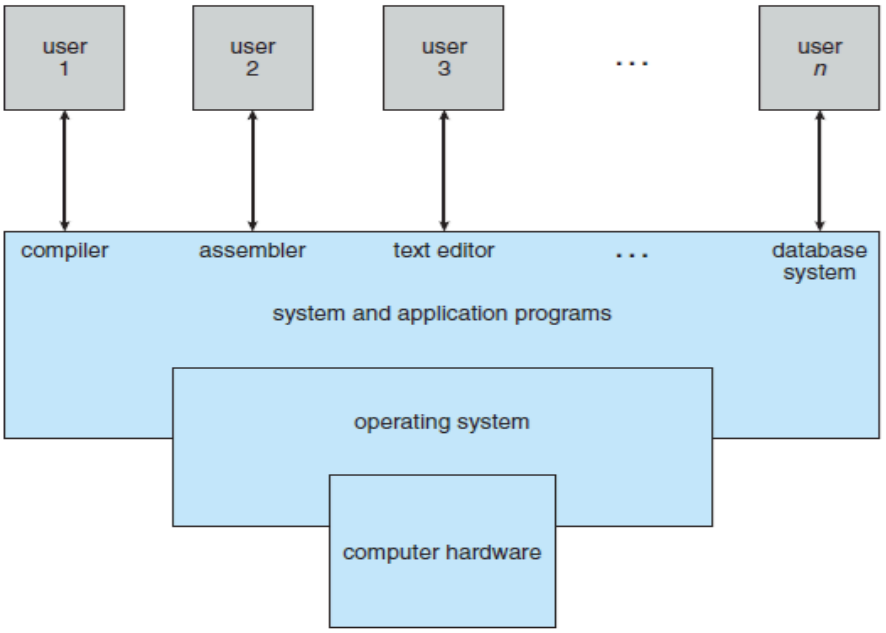


Figure 1.1 Abstract view of the components of a computer system.

2.Functions of Operating System

- **Security –** For security, modern operating systems employ a firewall. A firewall is a type of security system that monitors all computer activity and blocks it if it detects a threat.
- **Job Accounting –** As the operating system keeps track of all the functions of a computer system. Hence, it makes a record of all the activities taking place on the system. It has an account of all the information about the memory, resources, errors, etc. Therefore, this information can be used as and when required.
- **Control over system performance –** The operating system will collect consumption statistics for various resources and monitor performance indicators such as reaction time, which is the time between requesting a service and receiving a response from the system.
- **Error detecting aids –** While a computer system is running, a variety of errors might occur. Error detection guarantees that data is delivered reliably across susceptible networks. The operating system continuously monitors the system to locate or recognize problems and protects the system from them.
- **Coordination between other software and users –** The operating system (OS) allows hardware components to be coordinated and directs and allocates

assemblers, interpreters, compilers, and other software to different users of the computer system.

- **Booting process –** The process of starting or restarting a computer is referred to as Booting. Cold booting occurs when a computer is totally turned off and then turned back on. Warm booting occurs when the computer is restarted. The operating system (OS) is in charge of booting the computer.

3.Process Management

- A program does nothing unless its instructions are executed by a CPU
- A process needs certain resources—including CPU time, memory, files, and I/O devices—to accomplish its task. These resources are either given to the process when it is created or allocated to it while it is running.
- The process will be given the name of the file as an input and will execute the appropriate instructions and system calls to obtain and display the desired information on the terminal. When the process terminates, the operating system will reclaim any reusable resources.
- We emphasize that a program by itself is not a process. A program is a passive entity, like the contents of a file stored on disk, whereas a process is an active entity. A single-threaded process has one program counter specifying the next instruction to execute.
- The execution of such a process must be sequential. The CPU executes one instruction of the process after another, until the process completes.
- 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 (those that execute user code).
- The operating system is responsible for the following activities in connection with process management:
  - Scheduling processes and threads on the CPUs
  - Creating and deleting both user and system processes
  - Suspending and resuming processes
  - Providing mechanisms for process synchronization
  - Providing mechanisms for process communication

4.Memory Management

- The 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
- 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. Eventually, the program terminates, its memory space is declared available, and the next program can be loaded and executed.
- 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.
- 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 deallocating memory space as needed

5.Storage Management

The operating system provides a uniform, logical view of information storage. The operating system abstracts from the physical properties of its storage devices to define a logical storage unit, the file. The operating system maps files onto physical media and accesses these files via the storage devices.

File-System Management

- File management is one of the most visible components of an operating system. Computers can store information on several different types of physical media. Magnetic disk, optical disk, and magnetic tape are the most common.
- Each medium is controlled by a device, such as a disk drive or tape drive, that also has its own unique characteristics. These properties include access speed, capacity, data-transfer rate, and access method (sequential or random).
- A file is a collection of related information defined by its creator. 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, the concept of a file is an extremely general one.
- 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 directories to organize files
  - Supporting primitives for manipulating files and directories
  - Mapping files onto secondary storage
  - Backing up files on stable (non-volatile) storage media

Mass-Storage Management

Because main memory is too small to accommodate all data and programs, and because the data that it holds are lost when power is lost, the computer system must provide secondary storage to back up main memory.

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. Hence, the proper management of disk storage is of central importance to a computer system.

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

Because secondary storage is used frequently, it must be used efficiently. The entire speed of operation of a computer may hinge on the speeds of the disk subsystem and the algorithms that manipulate that subsystem.

There are, however, many uses for storage that is slower and lower in cost (and sometimes of higher capacity) than secondary storage. Backups of disk data, storage of seldom-used data, and long-term archival storage are some examples. Magnetic tape drives and their tapes and CD and DVD drives and platters are typical tertiary storage devices. The media (tapes and optical platters) vary between WORM (write-once, read-many-times) and RW (read– write) formats

Caching

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

In addition, 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

Because caches have limited size, cache management is an important design problem. Careful selection of the cache size and of a replacement policy can result in greatly increased performance.

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.

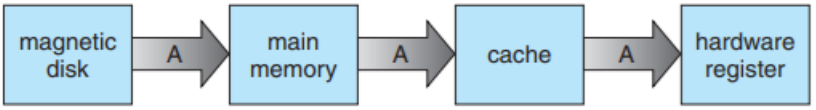


Figure 1.12 Migration of integer A from disk to register.

The movement of information between levels of a storage hierarchy may be either explicit or implicit, depending on the hardware design and the controlling operating-system software

For instance, data transfer from cache to CPU and registers is usually a hardware function, with no operating-system intervention. In contrast, transfer of data from disk to memory is usually controlled by the operating system

In a hierarchical storage structure, the same data may appear in different levels of the storage system. For example, suppose that an integer A that is to be incremented by 1 is located in file B, and file B resides on magnetic disk. The increment operation proceeds by first issuing an I/O operation to copy the disk block on which A resides to main memory. This operation is followed by copying A to the cache and to an internal register. Thus, the copy of A appears in several places: on the magnetic disk, in main memory, in the cache, and in an internal register (see Figure 1.12). Once the increment takes place in the internal register, the value of A differs in the various storage systems. The value of A becomes the same only after the new value of A is written from the internal register back to the magnetic disk.

we must make sure that an update to the value of A in one cache is immediately reflected in all other caches where A resides. This situation is called cache coherency, and it is usually a hardware issue

I/O Systems

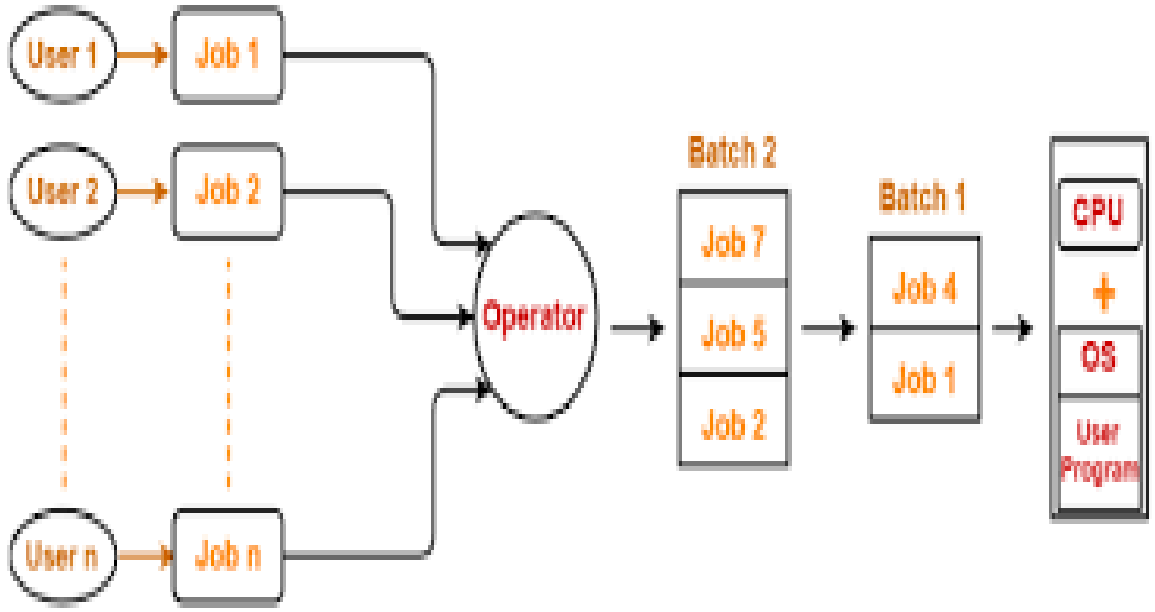
The peculiarities of I/O devices are hidden from the bulk of the operating system itself by the I/O subsystem. The I/O subsystem consists of several components: • A memory-management component that includes buffering, caching, and spooling • A general device-driver interface • Drivers for specific hardware devices Only the device driver knows the peculiarities of the specific device to which it is assigned.

6.TYPES OF OPERATING SYSTEM

1. Batch operating system
2. Multi Programming operating system
3. Multi-Tasking/Time Sharing operating system
4. Multi-Processing operating system
5. Network operating system
6. Distributed operating system
7. Real Time operating system
8. Mobile/Handheld operating system
9. Clustered operating system

1.Batch operating system:

- In this type of system, there is **no direct interaction between user and the computer**.
- The user has to submit a job (written on cards or tape) to a computer operator.
- Then computer operator places **a batch of several jobs on an input device.**
- Jobs are batched together by type of languages and requirement.
- Then a special program, the **monitor, manages the execution of each program in the**
- batch.
- The monitor is always in the main memory and available for execution.



Advantages and Disadvantage

- 1. No interaction between user and computer.
- 2. No mechanism to prioritize the processes.

Disadvantage:

Waiting Time of CPU from executing one job to other job.

2.Multiprogramming:

- In this the operating system picks up and begins to execute one of the jobs from memory.
- Once this job needs an I/O operation operating system switch to another job (CPU
- and OS always busy).

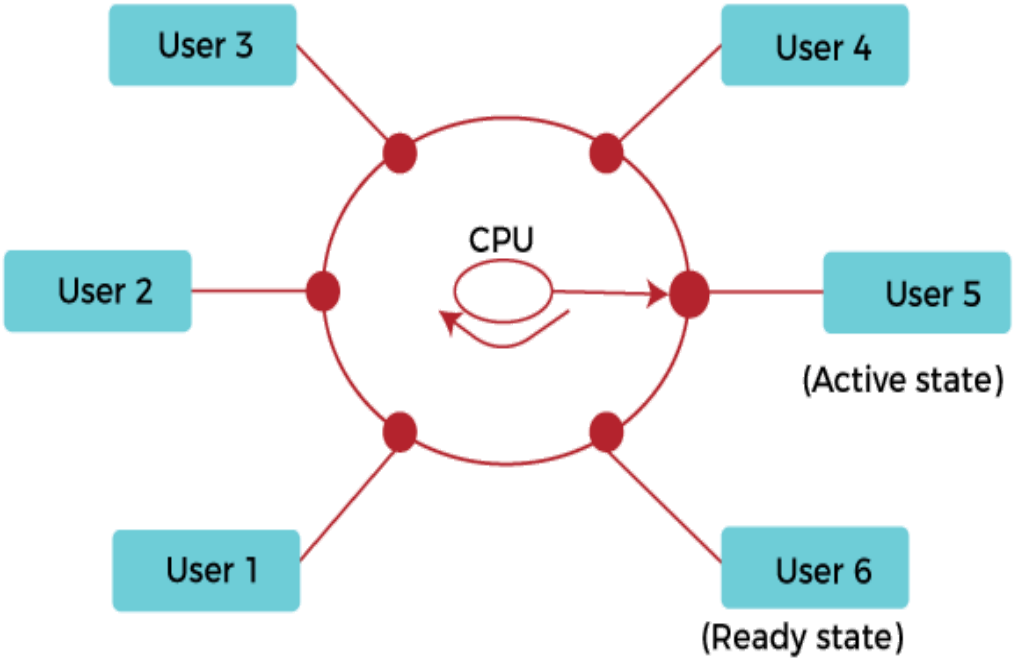
If several jobs are ready to run at the same time, then the system chooses which one to run through the process of CPU Scheduling.

- In Multiprogramming system, CPU will never be idle and keeps on processing

3.Time Sharing Systems

- Time Sharing Systems are very similar to Multiprogramming batch systems. In fact time sharing systems **are an extension of multiprogramming systems**.

In Time sharing systems the prime focus is on minimizing the **response time, while in multiprogramming the prime focus is to maximize the CPU usage**



Advantages and disadvantages

1. Enhanced performance

2. Execution of several tasks by different processors concurrently, increases the system's throughput without speeding up the execution of a single task

Disadvantage is hardware overhead

4.Multiprocessor Systems:

- A Multiprocessor system consists of several processors that share a common physical memory. Multiprocessor system provides higher computing power and speed.
- In multiprocessor system all processors operate under single operating system.
- Multiplicity of the processors and how they do act together are transparent to the others

5.Network operating system:

- Network operating system that run on a server and managing all the networking functions.
- Allow sharing various files , applications, printers , security and other networking functions over a small network of computers like LAN or any other private network

6.Distributed Operating System:

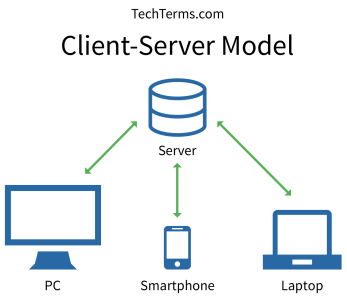
- The motivation behind developing distributed operating systems is the availability of powerful and inexpensive microprocessors and advances in communication technology.
- These advancements in technology have made it possible to design and develop distributed systems comprising of many computers that are inter connected by communication networks.
- The main benefit of distributed systems is its low price/performance ratio.

Types of Distributed Operating Systems

- Following are the two types of distributed operating systems used:
- 1. Client-Server Systems
- 2. Peer-to-Peer Systems

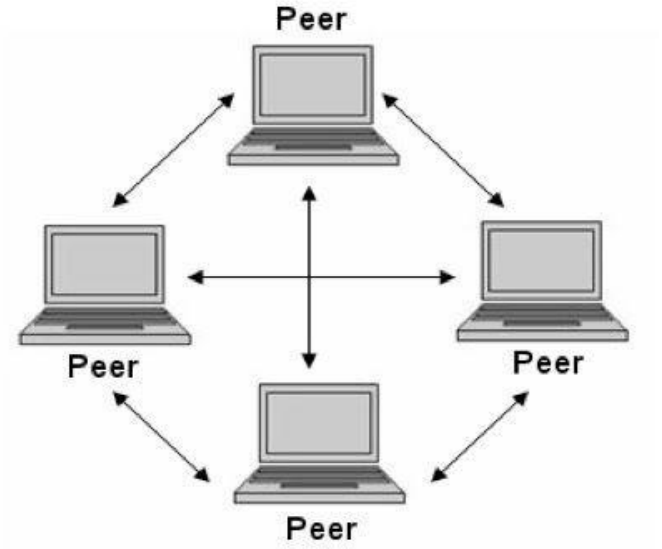
Client-Server Systems

- **Centralized systems** today act as **server systems** to satisfy requests generated by **client systems**.



Peer-to-Peer Systems

- A peer-to-peer network is a technology that allows you to connect two or more computers to one system.
- This connection allows you to easily share data without having to use a separate server for your file-sharing.
- Each end-computer that connects to this network becomes a '**peer**' and is allowed to receive or send files to other computers in its network.



7.Real Time Operating System:

- It is defined as an operating system known to give maximum time for each of the critical operations that it performs, like OS calls and interrupt handling.
- The Real-Time Operating system which guarantees the maximum time for critical operations and complete them on time are referred to as **Hard Real-Time Operating Systems.**

Soft Real-Time Operating Systems.

- While the real-time operating systems that can only guarantee a maximum of the time, critical task will get priority over other tasks, but no assurity of completeing it in a defined time. These systems are referred to as Soft Real-Time Operating Systems.

8.Mobile /Handheld operating Systems:

- **Handheld systems include Personal Digital Assistants(PDAs)**, such as Palm_x0002_Pilots or Cellular Telephones with connectivity to a network such as the Internet.
- They are usually of **limited size due to which most handheld devices have a small amount of memory**, include slow processors, and feature small display screens.
- Currently, many handheld devices do not use virtual memory techniques, thus forcing program developers to work within the confines of limited physical memory.

9.Clustered Systems:

- Like parallel systems, clustered systems gather together multiple CPUs to accomplish computational work. ·
- Clustered systems differ from parallel systems, however, in that they are composed of two or more individual systems coupled together.

Categories of clustering

- **Asymmetric Clustering** - In this, 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.

- **Symmetric Clustering** - In this, two or more hosts are running applications, and they
are monitoring each other. This mode is obviously more efficient, as it uses all of the available hardware. ·
- **Parallel Clustering** - Parallel clusters allow multiple hosts to access the same data on
the shared storage. Because most operating systems lack support for this simultaneous data access by multiple hosts, parallel clusters are usually accomplished by special versions of software and special releases of applications

7.operating system structures

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.

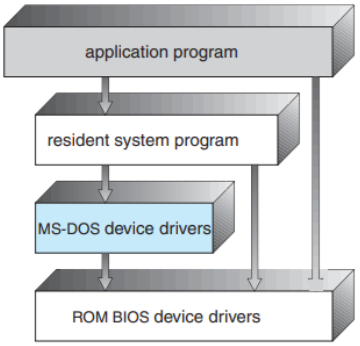


Figure 2.11 MS-DOS layer structure.

In MS-DOS, the interfaces and levels of functionality are not well separated. For instance, application programs are able to access the basic I/O routines to write directly to the display and disk drives. Such freedom leaves MS-DOS vulnerable to errant (or malicious) programs, causing entire system crashes when user programs fail. Of course, MS-DOS was also limited by the hardware of its era. Because the Intel 8088 for which it was written provides no dual mode and no hardware protection, the designers of MS-DOS had no choice but to leave the base hardware accessible.

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. The kernel is further separated into a series of interfaces and device drivers, which have been added and expanded over the years as UNIX has evolved. We can view the traditional UNIX operating system as being layered to some extent, as shown in Figure

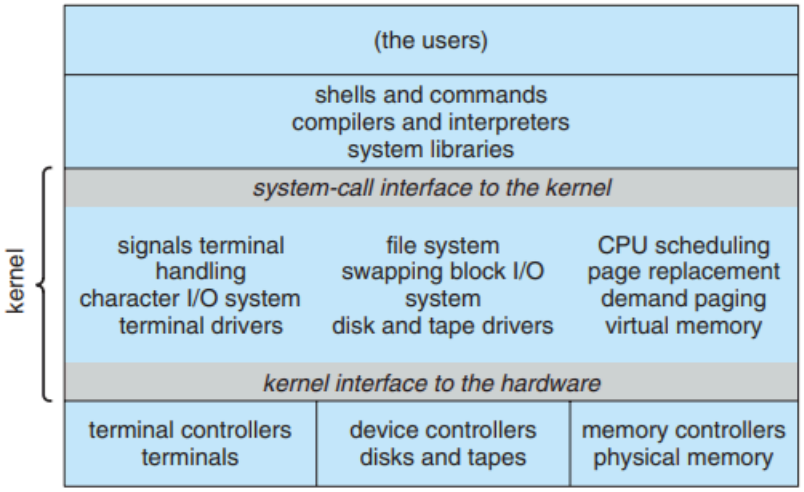


Figure 2.12 Traditional UNIX system structure.

Layered Approach

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. This layering structure is depicted in Figure

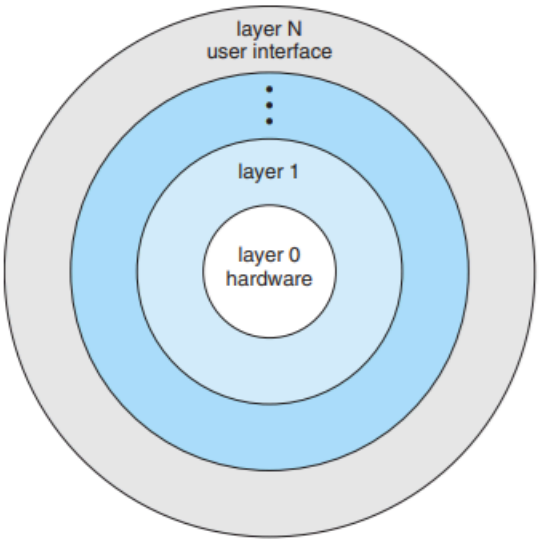


Figure 2.13 A layered operating system.

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. 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. This approach simplifies debugging and system verification. The first layer can be debugged without

any concern for the rest of the system, because, by definition, it uses only the basic hardware (which is assumed correct) to implement its functions. Once the first layer is debugged, its correct functioning can be assumed while the second layer is debugged, and so on. If an error is found during the debugging of a particular layer, the error must be on that layer, because the layers below it are already debugged. Thus, the design and implementation of the system are simplified

The major difficulty with the layered approach involves appropriately defining the various layers. Because a layer can use only lower-level layers, careful planning is necessary. For example, the device driver for the backing store (disk space used by virtual-memory algorithms) must be at a lower level than the memory-management routines, because memory management requires the ability to use the backing store.

The backing-store driver would normally be above the CPU scheduler, because the driver may need to wait for I/O and the CPU can be rescheduled during this time. However, on a large system, the CPU scheduler may have more information about all the active processes than can fit in memory. Therefore, this information may need to be swapped in and out of memory, requiring the backing-store driver routine to be below the CPU scheduler.

A final problem with layered implementations is that they tend to be less efficient than other types. For instance, when a user program executes an I/O operation, it executes a system call that is trapped to the I/O layer, which calls the memory-management layer, which in turn calls the CPU-scheduling layer, which is then passed to the hardware.

Microkernels

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. 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. Typically, however, microkernels 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 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.

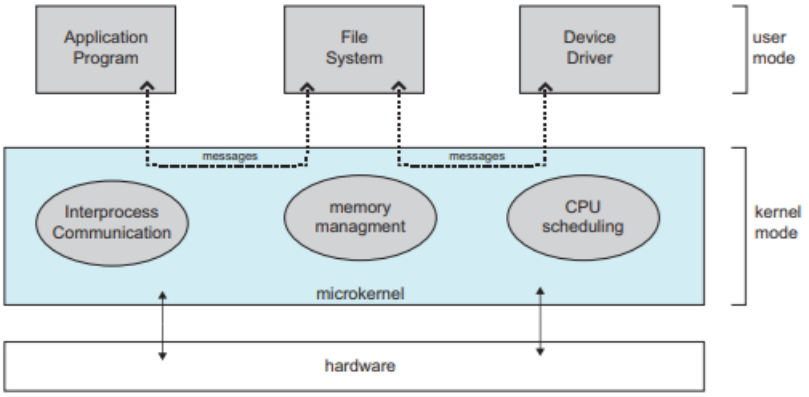


Figure 2.14 Architecture of a typical microkernel.

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 resulting operating system is easier to port from one hardware design to another

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 untouched.

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. 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.

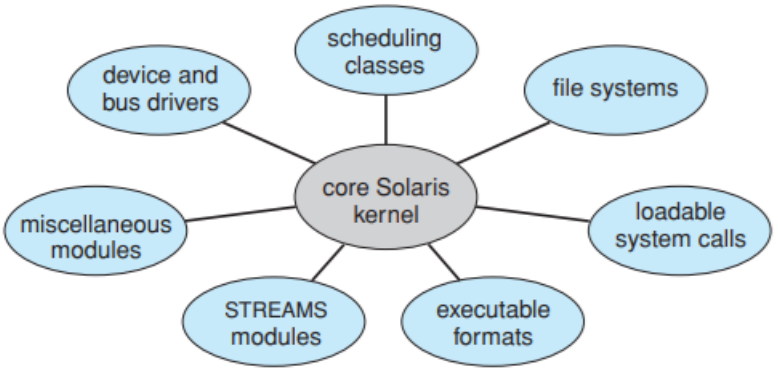


Figure 2.15 Solaris loadable modules.

The approach is also similar to the microkernel approach in that the primary module has only core functions and knowledge of how to load and communicate with other modules; but it is more efficient, because modules do not need to invoke message passing in order to communicate

, is organized around a core kernel with seven types of loadable kernel modules: 1. Scheduling classes 2. File systems 3. Loadable system calls 4. Executable formats 5. STREAMS modules 6. Miscellaneous 7. Device and bus drivers

Hybrid Systems

three hybrid systems: the Apple Mac OS X operating system and the two most prominent mobile operating systems—iOS and Android

Mac OS X The Apple Mac OS X operating system uses a hybrid structure. As shown in Figure 2.16, it is a layered system. The top layers include the Aqua user interface (Figure 2.4) 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. The BSD component provides a BSD command-line interface, support for networking and file systems, and an implementation of POSIX APIs, including Pthreads. In addition to Mach and BSD, the kernel environment provides an I/O kit for development of device drivers and dynamically loadable modules (which Mac OS X refers to as kernel extensions). As shown in Figure 2.16, the BSD application environment can make use of BSD facilities directly.

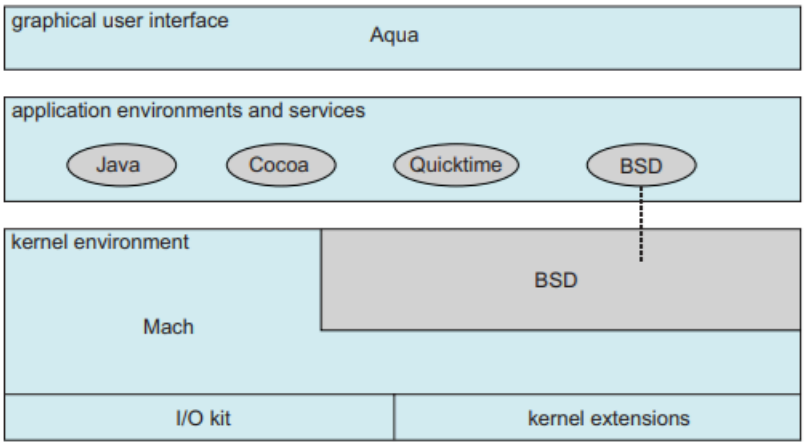


Figure 2.16 The Mac OS X structure.

IOS

iOS is a mobile operating system designed by Apple to run its smartphone, the iPhone, as well as its tablet computer, the iPad. iOS is structured on the Mac OS X operating system, with added functionality pertinent to mobile devices, but does not directly run Mac OS X applications. The structure of iOS appears in Figure 2.17. 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

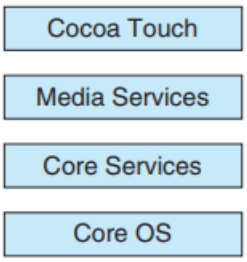


Figure 2.17 Architecture of Apple's iOS.

Android

The Android operating system was designed by the Open Handset Alliance (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, partly explaining its rapid rise in popularity. The structure of Android appears in Figure 2.18. 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, although it has been modified by Google and is currently outside the normal distribution of Linux releases

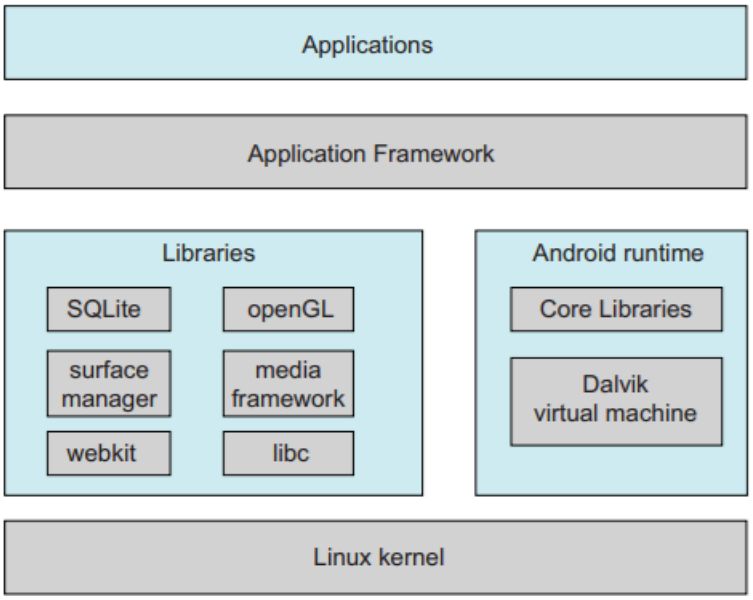


Figure 2.18 Architecture of Google's Android.

8.system calls

A system call is a method for a computer program to request a service from the kernel of the operating system on which it is running. A system call is a method of interacting with the operating system via programs. A system call is a request from computer software to an operating systems kernel.

- When a computer software needs to access the operating systems kernel, it makes a system call. The system call uses an API to expose the operating systems services to user programs. It is the only method to access the kernel system. All programs or processes that require resources for execution must use system calls, as they serve as an interface between the operating system and user programs.
- Below are some examples of how a system call varies from a user function.
- A system call function may create and use kernel processes to execute the asynchronous processing.
- A system call has greater authority than a standard subroutine. A system call with kernel-mode privilege executes in the kernel protection domain.
- System calls are not permitted to use shared libraries or any symbols that are not present in the kernel protection domain.
- The code and data for system calls are stored in global kernel memory

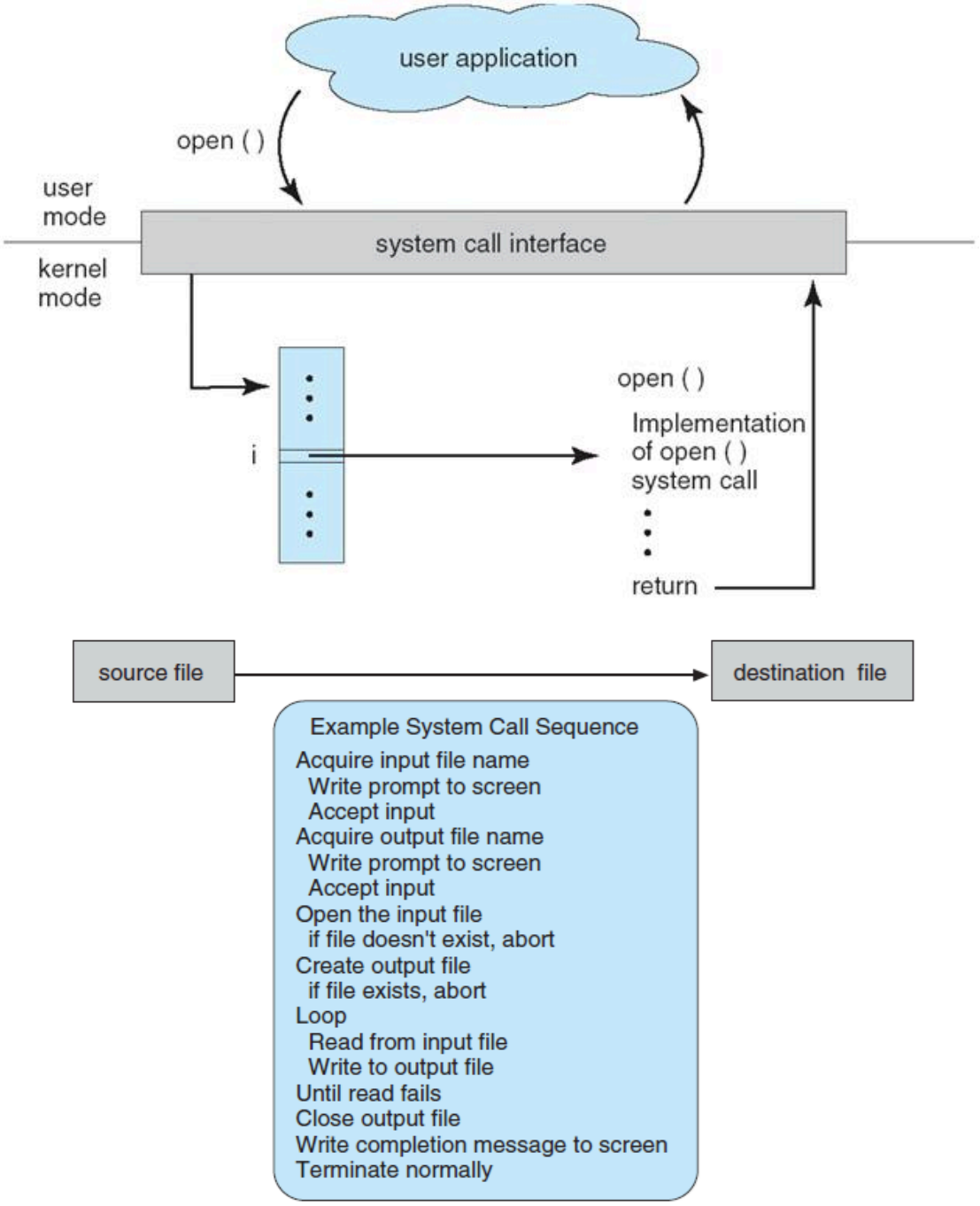


Figure 2.5 Example of how system calls are used.

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

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.

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()

File Management

We can,however, identify several common system calls dealing with files.We first need to be able to create() and delete() files. Either system call requires the name of the file and perhaps some of the file's attributes. Once the file is created, we need to open() it and to use it. We may also read(),
write(), or reposition() (rewind or skip to the end of the file, for example). Finally, we need to close() the file, indicating that we are no longer using it.
File attributes include the file name, file type, protection codes, accounting information, and so on. At least two system calls, get file attributes() and set file attributes(), are required for this function. Some operating systems provide many more calls, such as calls for file move() and copy().

Device Management

The various resources controlled by the operating system can be thought of as devices. Some of these devices are physical devices (for example, disk drives), while others can be thought of as abstract or virtual devices (for example, files).

Information Maintenance
Many system calls exist simply for the purpose of transferring information between the user program and the operating system. For example, most systems have a system call to return the current time() and date(). Other system calls may return information about the system, such as the number of current users, the version number of the operating system, the amount of free memory or disk space, and so on. Another set of system calls is helpful in debugging a program. Many systems provide system calls to dump() memory. This provision is useful for debugging. A program trace lists each system call as it is executed. Even microprocessors provide a CPU mode known as **single step**, in which a trap is executed by the CPU after every instruction. The trap is usually caught by a debugger.

Communication
There are two common models of interprocess communication: the messagepassing model and the shared-memory model. In the **message-passing model**, the communicating processes exchange messages with one another to transfer information. Messages can be exchanged between the processes either directly or indirectly through a common mailbox. Before communication can take place, a connection must be opened.
The name of the other communicator must be known, be it another process on the same system or a process on another computer connected by a communications network In the **shared-memory model**, processes use shared memory create() and shared memory attach() system calls to create and gain access to regions of memory owned by other processes. Recall that, normally, the operating system tries to prevent one process from accessing another process's memory.
Shared memory requires that two or more processes agree to remove this restriction. They can then exchange information by reading and writing data in the shared areas. The form of the data is determined by the processes and is not under the operating system's control. The processes are also responsible for ensuring that they are not writing to the same location simultaneously

Protection
Protection provides a mechanism for controlling access to the resources provided by a computer system. Historically, protection was a concern only on multi-programmed computer systems with several users. However, with the advent of networking and the Internet, all computer systems, from servers to mobile handheld devices, must be concerned with protection.
Typically, 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.

9.System Programs

System programs, also known as **system utilities**, provide a convenient environment for program development and execution. Some of them are simply user interfaces to system calls. Others are considerably more complex. 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. Others are

more complex, providing detailed performance, logging, and debugging information. Typically, these programs format and print the output to the terminal or other output devices or files or display it in a window of the GUI. Some systems also support a **registry**, which is used to store and retrieve configuration information.

• **File modification**. Several text editors may be available to create and modify the content of files stored on disk or other storage devices. There may also be special commands to search contents of files or perform transformations of the text.

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

• **Program loading and execution**. Once a program is assembled or compiled, it must be loaded into memory to be executed. The system may provide absolute loaders, relocatable loaders, linkage editors, and overlay loaders. Debugging systems for either higher-level languages or machine language are needed as well.

• **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. Some of these processes terminate after completing their tasks, while others continue to run until the system is halted. Constantly running system-program processes are known as **services**, **subsystems**,