Module – I
Introduction

- Introduction to Operating System
- Mainframe Systems
- Desktop Systems
- Multiprocessor Systems
- Distributed Systems
- Clustered Systems
- Real - Time Systems
- Handheld Systems
- Feature Migration
- Computing Environments
- System Components
- Operating – System Services
- System Calls
- System Programs
- System Structure

Introduction to Operating System

An operating system is an important part of almost every computer system. A computer system can be divided roughly into four components: the hardware, the operating system, the application programs, and the users (Figure 1.1).

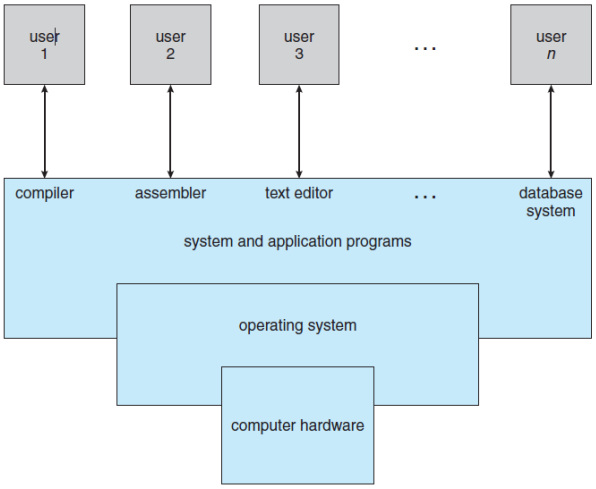

Figure 1.1 Abstract view of the components of a computer system.

The hardware-the **central processing unit (CPU),** the memory, and the **input/output (I/O)** devices-provides the basic computing resources. The application programs-such as word processors, spreadsheets, compilers, and web browsers-define the ways in which these resources are used to solve the computing problems of the users. The operating system controls and coordinates the use of the hardware among the various application programs for the various users. The components of a computer system are its hardware, software, and data. The operating system provides the means for the proper use of these resources in the operation of the computer system. An operating system is similar to a **government**. Like a government, it performs no useful function by itself. It simply provides an **environment** within which other programs can do useful work. Operating systems can be explored from **two viewpoints: the user and the system**.

The User View

The user view of the computer varies by the interface being used. Most computer users sit in front of a PC, consisting of a monitor, keyboard, mouse, and system unit. Such a system is designed for one user to monopolize its resources, to maximize the work (or play) that the user is performing. In this case, the operating system is designed mostly for ease of use, with some attention paid to performance, and none paid to resource utilization. Performance is important to the user, but it does not matter if most of the system is sitting idle, waiting for the slow 1/0 speed of the user.

Some users sit at a terminal connected to a **mainframe or minicomputer**. Other users are accessing the same computer through other **terminals.** These users share resources and may exchange information. The operating system is designed to maximize resource utilization-to assure that all available CPU time, memory, and I/O are used efficiently, and that no individual user takes more than her fair share.

Other users sit at **workstations**, connected to networks of other workstations and servers. These users have dedicated resources at their disposal, but they also share resources such as

networking and servers-file, compute and print servers. Therefore, their operating system is designed to compromise between individual usability and resource utilization.

Recently, many varieties of **mobile computers**, such as smartphones and tablets, have come into fashion. Most mobile computers are standalone units for individual users. Quite often, they are connected to networks through cellular or other wireless technologies. Increasingly, these mobile devices are replacing desktop and laptop computers for people who are primarily interested in using computers for e-mail and web browsing. The user interface for mobile computers generally 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.

Some computers have little or no user view. For example, **embedded computers** in home devices and automobiles may have a numeric keypad, and may turn indicator lights on or off to show status, but mostly they and their operating systems are designed to run without user intervention.

The System View

We can view an operating system as a **resource allocator**. A computer system has many resources-hardware and software-that may be required to solve a problem: CPU time, memory space, file-storage space, I/O devices, and so on. The operating system acts as the manager of these resources. Facing numerous and possibly conflicting requests for resources, the operating system must decide how to allocate them to specific programs and users so that it can operate the computer system efficiently and fairly.

A slightly different view of an 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.

Mainframe Systems

Mainframe computer systems were the first computers used to tackle many commercial and scientific applications. The growth of mainframe systems traces from simple batch systems, where the computer runs one -and only one-application, to time-shared systems, which allow for user interaction with the computer system.

Batch Systems

Early computers were physically enormous machines run from a console. The common input devices were card readers and tape drives. The common output devices were line printers, tape drives, and card punches. The user did not interact directly with the computer systems. Rather, the user prepared a job -which consisted of the program, the data, and some control information about the nature of the job (control cards)-and submitted it to the computer operator. The job was usually in the form of punch cards. At some later time (after minutes, hours, or days), the output appeared. The output consisted of the result of the program, as well as a dump of the final memory and register contents for debugging.

The operating system in these early computers was fairly simple. Its major task was to transfer control automatically from one job to the next. The operating system was always resident in memory (Figure 1.2).

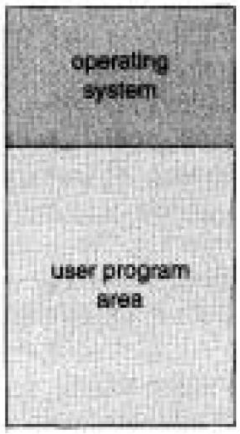

Figure **1.2** Memory layout for a simple batch system.

To speed up processing, operators batched together jobs with similar needs and ran them through the computer as a group. Thus, the programmers would leave their programs with the operator. The operator would sort programs into batches with similar requirements and, as the computer became available, would run each batch. The output from each job would be sent back to the appropriate programmer.

Multiprogrammed Systems

The most important aspect of job scheduling is the ability to multiprogram. A single user cannot, in general, keep either the CPU or the I/O devices busy at all times. Multiprogramming increases CPU utilization by organizing jobs so that the CPU always has one to execute. The idea is as follows: The operating system keeps several jobs in memory simultaneously (Figure below).

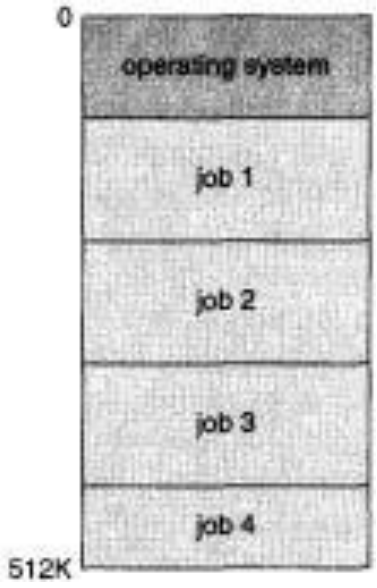

Fig: Memory Layout for a Multiprogramming System

This set of jobs is a subset of the jobs kept in the job pool-since the number of jobs that can be kept simultaneously in memory is usually much smaller than the number of jobs that can be in the job pool. The operating system picks and begins to execute one of the jobs in the memory. Eventually, the job may have to wait for some task, such as an I/O operation, to complete. In a non-multiprogrammed system, the CPU would sit idle. In a multiprogramming system, the

operating system simply switches to, and executes, another job. When that job needs to wait, the CPU is switched to another job, and so on. Eventually, the first job finishes waiting and gets the CPU back. As long as at least one job needs to execute, the CPU is never idle.

Multiprogramming is the first instance where the operating system must make decisions for the users. Multiprogrammed operating systems are therefore fairly sophisticated. All the jobs that enter the system are kept in the job pool. This pool consists of all processes residing on disk awaiting allocation of main memory. If several jobs are ready to be brought into memory, and if there is not enough room for all of them, then the system must choose among them. Making this decision is job scheduling. When the operating system selects a job from the job pool, it loads that job into memory for execution. Having several programs in memory at the same time requires some form of memory management. In addition, if several jobs are ready to run at the same time, the system must choose among them. Making this decision is CPU scheduling. Finally, multiple jobs running concurrently require that their ability to affect one another be limited in all phases of the operating system, including process scheduling, disk storage, and memory management.

Time-Sharing Systems

Time sharing (or multitasking) is a logical extension of multiprogramming. The CPU executes multiple jobs by switching among them, but the switches occur so frequently that the users can interact with each program while

it is running.

An interactive (or hands-on) computer system provides direct communication between the user and the system. The user gives instructions to the operating system or to a program directly, using a keyboard or a mouse, and waits for immediate results. Accordingly, the response time should be short typically within 1 second or so.

A time-shared operating system allows many users to share the computer simultaneously. Since each action or command in a time-shared system tends to be short, only a little CPU time is needed for each user. As the system switches rapidly from one user to the next, each user is

given the impression that the entire computer system is dedicated to her use, even though it is being shared among many users.

Time-sharing operating systems are even more complex than multiprogrammed operating systems. In both, several jobs must be kept simultaneously in memory, so the system must have memory management and protection. To obtain a reasonable response time, jobs may have to be swapped in and out of main memory to the disk that now serves as a backing store for main memory.

Time-sharing systems must also provide a file system. The file system resides on a collection of disks; hence, disk management must be provided. Also, time-sharing systems provide a mechanism for concurrent execution, which requires sophisticated CPU-scheduling schemes. To ensure orderly execution, the system must provide mechanisms for job synchronization and communication, and it may ensure that jobs do not get stuck in a deadlock, forever waiting for one another.

Desktop Systems

Personal computers PCs appeared in the 1970s. During their first decade, the CPUs in PCs lacked the features needed to protect an operating system from user programs. PC operating systems therefore were neither multiuser nor multitasking. However, the goals of these operating systems have changed with time; instead of maximizing CPU and peripheral utilization, the systems opt for maximizing user convenience and responsiveness.

Multiprocessor Systems

Most systems to date are single-processor systems; that is, they have only one main CPU. However, multiprocessor systems (also known as parallel systems or tightly coupled systems) are growing in importance. Such systems have more than one processor in close communication, sharing the computer bus, the clock, and sometimes memory and peripheral devices.

Multiprocessor systems have three main advantages.

Increased throughput. By increasing the number of processors, we hope to get more work done in less time. The speed-up ratio with N processors is not N; rather, it is less than N. When multiple processors cooperate on a task, a certain amount of overhead is incurred in keeping all the parts working correctly. This overhead, plus contention for shared resources, lowers the expected gain from additional processors. Similarly, a group of N programmers working closely together does not result in N times the amount of work being accomplished.

Economy of scale. Multiprocessor systems can save more money than multiple single-processor systems, because they can share peripherals, mass storage, and power supplies. If several programs operate on the same set of data, it is cheaper to store those data on one disk and to have all the processors share them, than to have many computers with local disks and many copies of the data.

Increased reliability. If functions can be distributed properly among several processors, then the failure of one processor will not halt the system, only slow it down. If we have ten processors and one fails, then each of the remaining nine processors must pick up a share of the work of the failed processor. Thus, the entire system runs only 10 percent slower, rather than failing altogether. This ability to continue providing service proportional to the level of surviving hardware is called **graceful degradation**. Systems designed for graceful degradation are also called **fault tolerant**.

The most common multiple-processor systems now use **symmetric multiprocessing (SMP**), in which each processor runs an identical copy of the operating system, and these copies communicate with one another as needed. Some systems use **asymmetric multiprocessing**, in which each processor is assigned a specific task. A master processor controls the system; the other processors either look to the master for instruction or have predefined tasks. This scheme defines a master-slave relationship. The master processor schedules and allocates work to the slave processors.

SMP means that all processors are peers; no master-slave relationship exists between processors. Each processor concurrently runs a copy of the operating system. Figure below illustrates a typical SMP architecture. An example of the SMP system is Encore's version of UNIX for the Multimax computer. This computer can be configured such that it employs dozens of

processors, all running copies of UNIX. The benefit of this model is that many processes can run simultaneously-N processes can run if there are N CPUs-without causing a significant deterioration of performance.

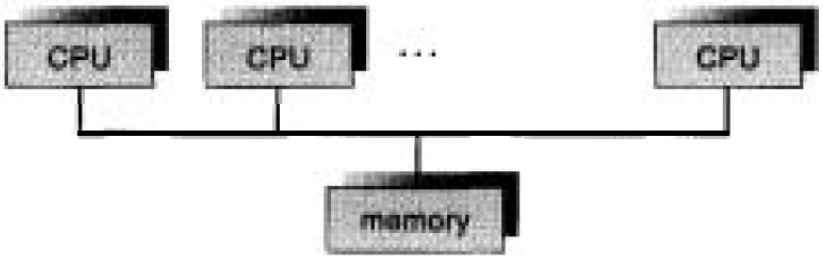

Fig: Symmetric Multiprocessing Architecture

The difference between symmetric and asymmetric multiprocessing may be the result of either hardware or software. Special hardware can differentiate the multiple processors, or the software can be written to allow only one master and multiple slaves.

Distributed Systems

A network, in the simplest terms, is a communication path between two or more systems. Distributed systems depend on networking for their functionality. By being able to communicate, distributed systems are able to share computational tasks, and provide a rich set of features to users.

Networks are typecast based on the distances between their nodes. A **local-area network (LAN),** exists within a room, a floor, or a building. A **wide-area network (WAN),** usually exists between buildings, cities, or countries. A global company may have a WAN to connect its offices, worldwide. These networks could run one protocol or several protocols. The continuing advent of new technologies brings about new forms of networks. For example, a **metropolitan-area network (MAN)**, could link buildings within a city. BlueTooth devices communicate over a short distance of several feet, in essence creating a **small-area network**.

Client-Server Systems

As PCs have become faster, more powerful, and cheaper, designers have shifted away from the centralized system architecture. Terminals connected to centralized systems are now being supplanted by PCs. Correspondingly, user-interface functionality that used to be handled directly by the centralized systems is increasingly being handled by the PCs. As a result, centralized systems today act as server systems to satisfy requests generated by client systems. The general structure of a client-server system is depicted in Figure below.

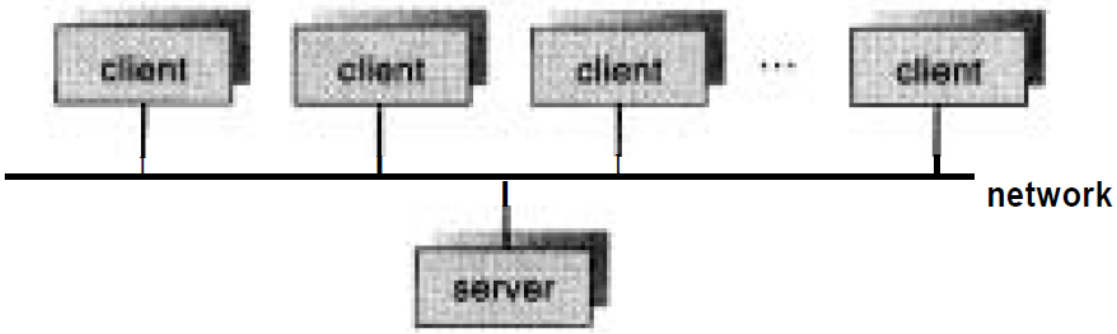

Fig: General Structure of a Client-Server System

Server systems can be broadly categorized as compute servers and file servers.

Compute-server systems provide an interface to which clients can send requests to perform an action, in response to which they execute the action and send back results to the client.

File-server systems provide a file-system interface where clients can create, update, read, and delete files.

Peer-to-Peer Systems

In contrast to the tightly coupled systems, the computer networks used in these applications consist of a collection of processors that do not share memory or a clock. Instead, each processor has its own local memory. The processors communicate with one another through various communication lines, such as high-speed buses or telephone lines. These systems are usually referred to as **loosely coupled systems (or distributed systems).**

Some operating systems have taken the concept of networks and distributed systems further than the notion of providing network connectivity. A network operating system is an operating system that provides features such as file sharing across the network, and that includes a communication scheme that allows different processes on different computers to exchange messages. A computer running a network operating system acts autonomously from all other computers on the network, although it is aware of the network and is able to communicate with other networked computers. A distributed operating system is a less autonomous environment: The different operating systems communicate closely enough to provide the illusion that only a single operating system controls the network.

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. The generally accepted definition is that clustered computers share storage and are closely linked via LAN networking.

Clustering is usually performed to provide high availability. A layer of cluster software runs on the cluster nodes. Each node can monitor one or more of the others (over the LAN). If the monitored machine fails, the monitoring machine can take ownership of its storage, and restart the application(s) that were running on the failed machine. The failed machine can remain down, but the users and clients of the application would only see a brief interruption of service.

In **asymmetric clustering**, one machine is in hot standby mode while the other is running the applications. The **hot standby host** (machine) does nothing but monitor the active server. If that server fails, the hot standby host becomes the active server. In **symmetric mode**, 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. It does require that more than one application be available to run.

Real-Time Systems

Another form of a special-purpose operating system is the **real-time system**. A real-time system is used when rigid time requirements have been placed on the operation of a processor or the flow of data; thus, it is often used as a control device in a dedicated application. Sensors bring data to the computer. The computer must analyze the data and possibly adjust controls to modify the sensor inputs. Systems that control scientific experiments, medical imaging systems, industrial control systems, and certain display systems are real-time systems. Some automobile-engine fuel-injection systems, home-appliance controllers, and weapon systems are also real-time systems.

A real-time system has well-defined, fixed time constraints. Processing must be done within the defined constraints, or the system will fail. For instance, it would not do for a robot arm to be instructed to halt after it had smashed into the car it was building. A real-time system functions correctly only if it returns the correct result within its time constraints. Contrast this requirement to a time-sharing system, where it is desirable (but not mandatory) to respond quickly, or to a batch system, which may have no time constraints at all.

Real-time systems come in two flavors: **hard and soft**. A **hard real-time system** guarantees that critical tasks be completed on time. This goal requires that all delays in the system be bounded, from the retrieval of stored data to the time that it takes the operating system to finish any request made of it. Such time constraints dictate the facilities that are available in hard real-time systems. Secondary storage of any sort is usually limited or missing, with data instead being stored in short-term memory or in read-only memory (ROM). ROM is located on nonvolatile storage devices that retain their contents even in the case of electric outage; most other types of memory are volatile. Most advanced operating-system features are absent too, since they tend to separate the user from the hardware, and that separation results in uncertainty about the amount of time an operation will take. For instance, virtual memory is almost never found on real-time systems. Therefore, hard real-time systems conflict with the operation of time-sharing systems, and the two cannot be mixed. Since none of the existing

general-purpose operating systems support hard real-time functionality, we do not concern ourselves with this type of system in this text.

A less restrictive type of real-time system is a **soft real-time system**, where a critical real-time task gets priority over other tasks, and retains that priority until it completes. As in hard real-time systems, the operating-system kernel delays need to be bounded: A real-time task cannot be kept waiting indefinitely for the kernel to run it. Soft real time is an achievable goal that can be mixed with other types of systems. Soft real-time systems, however, have more limited utility than hard real-time systems. Given their lack of deadline support, they are risky to use for industrial control and robotics. They are useful, however in several areas, including multimedia, virtual reality, and advanced scientific projects-such as undersea exploration and planetary rovers. These systems need advanced operating-system features that cannot be supported by hard real-time systems. Because of the expanded uses for soft real-time functionality, it is finding its way into most current operating systems, including major versions of UNIX.

Handheld Systems

Handheld systems include personal digital assistants (PDAs), such as Palm-Pilots or cellular telephones with connectivity to a network such as the Internet. Developers of handheld systems and applications face many challenges, most of which are due to the limited size of such devices. For example, a PDA is typically about 5 inches in height and 3 inches in width, and it weighs less than one-half pound. Due to this limited size, most handheld devices have a small amount of memory, include slow processors, and feature small display screens. We will take a look now at each of these limitations.

Many handheld devices have between 512 KB and 8 MB of **memory.** (Contrast this with a typical PC or workstation, which may have several hundred megabytes of memory!) As a result, the operating system and applications must manage memory efficiently. This includes returning all allocated memory back to the memory manager once the memory is no longer being used.

Virtual memory allows developers to write programs that behave as if the system has more memory than may be physically available. Currently, many handheld devices do not use virtual memory techniques, thus forcing program developers to work within the confines of limited physical memory.

A second issue of concern to developers of handheld devices is the **speed of the processor** used in the device. Processors for most handheld devices often run at a fraction of the speed of a processor in a PC. Faster processors require more power. To include a faster processor in a handheld device would require a larger battery that would have to be replaced (or recharged) more frequently. To minimize the size of most handheld devices, smaller, slower processors which consume less power are typically used. Therefore, the operating system and applications must be designed not to tax the processor.

The last issue confronting program designers for handheld devices is the **small display screens** typically available. Whereas a monitor for a home computer may measure up to 21 inches, the display for a handheld device is often no more than 3 inches square. Familiar tasks, such as reading e-mail or browsing web pages, must be condensed onto smaller displays. One approach for displaying the content in web pages is web clipping, where only a small subset of a web page is delivered and displayed on the handheld device.

Some handheld devices may use wireless technology, such as BlueTooth, allowing remote access to e-mail and web browsing. Cellular telephones with connectivity to the Internet fall into this category.

Feature Migration

Overall, an examination of operating systems for mainframes and microcomputers shows that features once available only on mainframes have been adopted by microcomputers. The same concepts are appropriate for the various classes of computers: mainframes, minicomputers, microcomputers, and handhelds.

At the same time as features of large operating systems were being scaled down to fit PCs, more powerful, faster, and more sophisticated hardware systems were being developed. The personal workstation is a large PC-for example, the Sun SPARCstation, the HP/Apollo, the IBM RS/6000, and the Intel Pentium class system running Windows NT or a UNIX derivative. Many universities and businesses have large numbers of workstations tied together with local-area networks. As PCs gain more sophisticated hardware and software, the line dividing the two categories-mainframes and microcomputers-is blurring.

Computing Environments

- Traditional Computing
- Web-based Computing
- Embedded Computing

Traditional Computing

As computing matures, the lines among many of the traditional computing environments are blurring. Consider the "typical office environment." Just a few years ago, this environment consisted of PCs connected to a network, with servers providing file and print service. Remote access was awkward, and portability was achieved by laptop computers carrying some of the user's workspace. Terminals attached to mainframes were prevalent at many companies as well, with even fewer remote access and portability options.

The current trend is toward more ways to access these environments. Web technologies are stretching the boundaries of traditional computing. Companies implement portals which provide web accessibility to their internal servers. Network computers are essentially terminals that understand web based computing. Handheld computers can synchronize with PCs to allow very portable use of company information. They can also connect to wireless networks to use the company's web portal (as well as the myriad other web resources).

At home, most users had a single computer with a slow modem connection to the office, the Internet, or both. Network connection speeds once attainable only at great cost are now available at low cost, allowing more access to more data at a company or from the Web. Those fast data connections are allowing home computers to serve up web pages and to contain their own networks with printers, client PCs, and servers. Some homes even have firewalls to protect these home environments from security breaches. Those firewalls cost thousands of dollars a few years ago, and did not even exist a decade ago.

Web-based Computing

The Web has become ubiquitous, leading to more access by a wider variety of devices than was dreamt about a few years ago. PCs are still the most prevalent access devices, with workstations (high-end graphics-oriented PCs), handheld PDAs, and even cell phones also providing access.

Web computing has increased the emphasis on networking. Devices that were not previously networked now have wired or wireless access. Devices that were networked now have faster network connectivity, either by improved networking technology, optimized network implementation code, or both.

The implementation of web-based computing has given rise to new categories of devices, such as load balancers which distribute network connections among a pool of similar servers. Operating systems like Windows 95, which acted as web clients, have evolved into Windows ME and Windows 2000, which can act as web servers as well as clients. Generally, the Web has increased the complexity of devices as their users require them to be web-enabled.

Embedded Computing

Embedded computers are the most prevalent form of computers in existence. They run embedded real-time operating systems. These devices are found everywhere, from car engines and manufacturing robots to VCRs and microwave ovens. They tend to have very specific tasks.

The systems they run on are usually primitive, lacking advanced features, such as virtual memory, and even disks. Thus, the operating systems provide limited features. They usually have little or no user interface, preferring to spend their time monitoring and managing hardware devices, such as automobile engines and robotic arms.

As an example, consider the aforementioned firewalls and load balancers. Some are general-purpose computers, running standard operating systems such as UNIX-with special-purpose applications loaded to implement the functionality. Others are hardware devices with a special-purpose operating system embedded within, providing just the functionality desired.

The use of embedded systems continues to expand. The power of those devices, both as standalone units and as members of networks and the Web, is sure to increase as well. Entire houses can be computerized, so that a central computer-either a general-purpose computer or an embedded system-can control heating and lighting, alarm systems, and even coffee makers. Web access can let a home-owner tell the house to heat up before he arrives home. Someday, the refrigerator may call the grocery store when it notices the milk is gone.

System Components

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

Process Management

A program does nothing unless its instructions are executed by a CPU. A process can be thought of as a program in execution, but its definition will broaden as we explore it further. A time-shared user program such as a compiler is a process. A word-processing program being run by an individual user on a PC is a process. A system task, such as sending output to a printer, is also a process.

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. In addition to the various physical and logical resources that a process obtains when it is created, various initialization data (or input) may be passed along. For example, consider a process whose function is to display the status of a file on the screen of a terminal. The process will be given as an input the name of the file, and will execute the appropriate instructions and system calls to obtain and display on the terminal the desired information. When the process terminates, the operating system will reclaim any reusable resources.

A program by itself is not a process; a program is a passive entity, such as the contents of a file stored on disk, whereas a process is an active entity, with a program counter specifying the next instruction to execute. The execution of a process must be sequential. The CPU executes one instruction of the process after another, until the process completes. Further, at any time, at most one instruction is executed on behalf of the process. Thus, although two processes may be associated with the same program, they are nevertheless considered two separate execution sequences. It is common to have a program that spawns many processes as it runs.

A process is the unit of work in a system. Such 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). All these processes can potentially execute concurrently, by multiplexing the CPU among them.

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

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

Main-Memory Management

Main memory is a large array of words or bytes, ranging in size from hundreds of thousands to billions. Each word or byte has its own address. Main memory is a repository of quickly accessible data shared by the CPU and I/O devices. The central processor reads instructions from main memory during the instruction-fetch cycle, and it both reads and writes data from main memory during the data-fetch cycle. The I/O operations implemented via DMA also read and write data in main memory. The main memory is generally the only large storage device that the CPU is able to address and access directly. For example, for the CPU to process data from disk, those data must first be transferred to main memory by CPU-generated I/O calls. Equivalently, instructions must be in memory for the CPU to execute them.

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, we must keep several programs in memory. Many different memory-management schemes are available, and the effectiveness of the different algorithms depends on the particular situation. Selection of a memory-management scheme for a specific system depends on many factors -especially on the hardware design of the system. Each algorithm requires its own hardware support.

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 by whom
- Deciding which processes are to be loaded into memory when memory space becomes available
- Allocating and deallocating memory space as needed

File 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 tape, magnetic disk, and optical disk are the most common media. Each of these media has its own characteristics and physical organization. Each medium is controlled by a device, such as a disk drive or tape drive, that also has 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, or alphanumeric. Files may be free-form (for example, text files), or may be formatted rigidly (for example, fixed fields). A file consists of a sequence of bits, bytes, lines, or records whose meanings are defined by their creators. The concept of a file is an extremely general one.

The operating system implements the abstract concept of a file by managing mass storage media, such as disks and tapes, and the devices that control them. Also, files are normally organized into directories to ease their use. Finally, when multiple users have access to files, we may want to control by whom and in what ways (for example, read, write, append) files may be accessed.

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

- Creating and deleting files
- Creating and deleting directories
- Supporting primitives for manipulating files and directories
- Mapping files onto secondary storage
- Backing up files on stable (nonvolatile) storage media

I/O System Management

One of the purposes of an operating system is to hide the peculiarities of specific hardware devices from the user. For example, in UNIX, 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

- A memory-management component that includes buffering, caching, and spooling
- A general device-driver interface
- Drivers for specific hardware devices

Secondary-Storage Management

The main purpose of a computer system is to execute programs. These programs, with the data they access, must be in main memory, or primary storage, during execution. 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, sort routines, editors, and formatters-are stored on a disk until loaded into memory, and 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

Networking

A **distributed system** is a collection of processors that do not share memory, peripheral devices, or a clock. Instead, each processor has its own local memory and clock, and the processors communicate with one another through various communication lines, such as high-speed buses or networks. The processors in a distributed system vary in size and function. They may include small microprocessors, workstations, minicomputers, and large, general-purpose computer systems.

The processors in the system are connected through a communication network, which can be configured in a number of different ways. The network may be fully or partially connected. The communication-network design must consider message routing and connection strategies, and the problems of contention and security.

A distributed system collects physically separate, possibly heterogeneous, systems into a single coherent system, providing the user with access to the various resources that the system maintains. Access to a shared resource allows computation speedup, increased functionality, increased data availability, and enhanced reliability. Operating systems usually generalize network access as a form of file access, with the details of networking being contained in the network interface's device driver. The protocols that create a distributed system can have a great effect on that system's utility and popularity. The innovation of the World Wide Web was to create a new access method for information sharing. It improved on the existing file-transfer protocol (**FTP**) and network file-system (**NFS**) protocol by removing the need for a user to log in before she is allowed to use a remote resource. It defined a new protocol, hypertext transfer protocol (http), for use in communication between a web server and a web browser. A web browser then just needs to send a request for information to a remote machine's web server, and the information (text, graphics, links to other information) is returned. This increase in convenience fostered huge growth in the use of http and of the Web in general.

Protection System

If a computer system has multiple users and allows the concurrent execution of multiple processes, then the various processes must be protected from one another's activities. For that purpose, mechanisms ensure that the files, memory segments, CPU, and other resources can be operated on by only those processes that have gained proper authorization from the operating system.

For example, memory-addressing hardware ensures that a process can execute only within its own address space. The timer ensures that no process can gain control of the CPU without eventually relinquishing control. Device control registers are not accessible to users, so that the integrity of the various peripheral devices is protected.

Protection is any mechanism for controlling the access of programs, processes, or users to the resources defined by a computer system. This mechanism must provide means for specification of the controls to be imposed and means for enforcement.

Protection can improve reliability by detecting latent errors at the interfaces between component subsystems. Early detection of interface errors can often prevent contamination of a healthy subsystem by another subsystem that is malfunctioning. An unprotected resource cannot defend against use (or misuse) by an unauthorized or incompetent user. A protection-oriented system provides a means to distinguish between authorized and unauthorized usage.

Command-Interpreter System

One of the most important systems programs for an operating system is the command interpreter, which is the interface between the user and the operating system. Some operating systems include the command interpreter in the kernel. Other operating systems, such as MS-DOS and UNIX, treat the command interpreter as a special program that is running when a job is initiated, or when a user first logs on (on time-sharing systems).

Many commands are given to the operating system by control statements. When a new job is started in a batch system, or when a user logs on to a time-shared system, a program that reads and interprets control statements is executed automatically. This program is sometimes called the control-card interpreter or the command-line interpreter, and is often known as the shell. Its function is simple: To get the next command statement and execute it.

Operating System Services

An operating system provides an environment for the execution of programs. It provides certain services to programs and to the users of those programs. The specific services provided, of course, differ from one operating system to another, but we can identify common classes. These operating-system services are provided for the convenience of the programmer, to make the programming task easier.

Program execution: The system must be able to load a program into memory and to run that program. The program must be able to end its execution, either normally or abnormally (indicating error).

I/O operations: A running program may require I/O. This I/O may involve a file or an I/O device. For specific devices, special functions may be desired (such as to rewind a tape drive, or to blank a CRT screen). For efficiency and protection, users usually cannot control I/O devices directly. Therefore, the operating system must provide a means to do I/O.

File-system manipulation: The file system is of particular interest. Obviously, programs need to read and write files. Programs also need to create and delete files by name.

Communications: In many circumstances, one process needs to exchange information with another process. Such communication can occur in two major ways. The first takes place between processes that are executing on the same computer; the second takes place between processes that are executing on different computer systems that are tied together by a

computer network. Communications may be implemented via shared memory, or by the technique of message passing, in which packets of information are moved between processes by the operating system.

Error detection: The operating system constantly needs to be aware of possible errors. Errors may occur in the CPU and memory hardware (such as a memory error or a power failure), in I/O devices (such as a parity error on tape, a connection failure on a network, or lack of paper in the printer), and in the user program (such as an arithmetic overflow, an attempt to access an illegal memory location, or a too-great use of CPU time). For each type of error, the operating system should take the appropriate action to ensure correct and consistent computing.

In addition, another set of operating-system functions exists not for helping the user, but for ensuring the efficient operation of the system itself.

Resource allocation: When multiple users are logged on the system or multiple jobs are running at the same time, resources must be allocated to each of them. Many different types of resources are managed by the operating system. Some (such as CPU cycles, main memory, and file storage) may have special allocation code, whereas others (such as I/O devices) may have much more general request and release code.

Accounting: We want to keep track of which users use how many and which kinds of computer resources. This record keeping may be used for accounting (so that users can be billed) or simply for accumulating usage statistics. Usage statistics may be a valuable tool for researchers who wish to reconfigure the system to improve computing services.

Protection: The owners of information stored in a multiuser computer system may want to control use of that information. When several disjointed processes execute concurrently, it should not be possible for one process to interfere with the others, or with the operating system itself. Protection involves ensuring that all access to system resources is controlled.

Security of the system from outsiders is also important. Such security starts with each user having to authenticate himself to the system, usually by means of a password, to be allowed access to the resources.

System Calls

System calls provide the interface between a process and the operating system. These calls are generally available as assembly-language instructions. Certain systems allow system calls to be made directly from a higher level language program. Several languages-such as C, C++, and Perl-have been defined to replace assembly language for systems programming. These languages allow system calls to be made directly.

System calls can be grouped roughly into five major categories: process control, file management, device management, information maintenance, and communications.

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

A running program needs to be able to halt its execution either normally (end) or abnormally (abort). Under either normal or abnormal circumstances, the operating system must transfer control to the command interpreter. The command interpreter then reads the next command. A process or job executing one program may want to load and execute another program. An interesting question is where to return control when the loaded program terminates. This question is related to the problem of whether the existing program is lost, saved, or allowed to continue execution concurrently with the new program. If control returns to the existing

program when the new program terminates, we must save the memory image of the existing program; thus, we have effectively created a mechanism for one program to call another program. If both programs continue concurrently, we have created a new job or process to be multiprogrammed. Often, system calls exists specifically for this purpose (create process or submit job).

If we create a new job or process, or perhaps even a set of jobs or processes, we should be able to control its execution. This control requires the ability to determine and reset the attributes of a job or process, including the job's priority, its maximum allowable execution time, and so on (get process attributes and set process attributes). We may also want to terminate a job or process that we created (terminate process) if we find that it is incorrect or is no longer needed.

Having created new jobs or processes, we may need to wait for them to finish their execution. We may want to wait for a certain amount of time (wait time); more likely, we may want to wait for a specific event to occur (w a i t event). The jobs or processes should then signal when that event has occurred (signal event).

File Management

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

We first need to be able to create and delete files. Either system call requires the name of the file and perhaps some of the file's attributes. 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, a file type, protection codes, accounting information, and so

on. At least two system calls, get file attributes and set file attribute, are required for this function.

Device Management

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

A program, as it is running, may need additional resources to proceed. Additional resources may be more memory, tape drives, access to files, and so on. If the resources are available, they can be granted, and control can be returned to the user program; otherwise, the program will have to wait until sufficient resources are available.

Files can be thought of as abstract or virtual devices. Thus, many of the system calls for files are also needed for devices. If the system has multiple users, however, we must first request the device, to ensure exclusive use of it. After we are finished with the device, we must release it. These functions are similar to the open and close system calls for files. Once the device has been requested (and allocated to us), we can read, write, and (possibly) reposition the device, just as we can with ordinary files.

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

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. In addition, the operating system keeps information about all its processes, and there are system calls to access this information. Generally, there are also calls to reset the process information (get process attributes and set process attributes).

Communication

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

There are two common models of communication. In the **message-passing model**, information is exchanged through an interprocess-communication facility provided by the operating system. 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 CPU, or a process on another computer connected by a communications network.

In the **shared-memory model**, processes use map memory system calls to gain access to regions of memory owned by other processes. They may then exchange information by reading and writing data in these shared areas. The form of the data and the location are determined by these processes and are not under the operating system's control. The processes are also responsible for ensuring that they are not writing to the same location simultaneously.

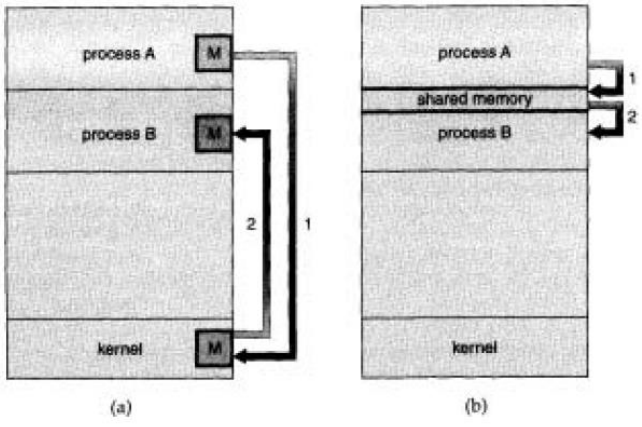

Figure 3.5 Communications models. (a) Msg passing. (b) Shared memory.

System Programs

System programs provide a convenient environment for program development and execution. 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. That information is then formatted, and is printed to the terminal or other output device or file.

File modification: Several text editors may be available to create and modify the content of files stored on disk or tape.

Programming-language support: Compilers, assemblers, and interpreters for common programming languages (such as C, C++, Java, Visual Basic, and PERL) are often provided to the user with the operating system. Some of these programs are now priced and provided separately.

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 also.

Communications: These programs provide the mechanism for creating virtual connections among processes, users, and different computer systems. They allow users to send messages to one another's screens, to browse web pages, to send electronic-mail messages, to log in remotely, or to transfer files from one machine to another.

Most operating systems are supplied with programs that solve common problems, or perform common operations. Such programs include web browsers, word processors and text formatters, spreadsheets, database systems, compiler compilers, plotting and statistical-analysis packages, and games. These programs are known as **system utilities or application programs.**

The most important system program for an operating system is the **command interpreter**, the main function of which is to get and execute the next user-specified command.

System Structure

(Notes given in class)

Question Bank

1. Explain OS with respect to User view and System view
2. Explain the following:
    - Mainframe Systems
    - Desktop Systems
    - Multiprocessor Systems
    - Distributed Systems
    - Clustered Systems
    - Real - Time Systems
    - Handheld Systems
3. What are the different computing environments?
4. List all the system components.
5. What is the role of the OS towards each of the system components?
6. What are the various services provided by the OS.
7. What is a System call? Explain the different categories of System calls?
8. Write a note on System Programs.
9. Explain the various structures of implementing OS.