OS Ch-1

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

Thus, some operating systems are designed to be convenient, others to be efficient, and others to be some combination of the two.

A computer system can be divided roughly into four components: the hardware, the operating system, the application programs, and the users (Figure 1.1).

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

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

control program manages the execution of user programs to prevent errors and improper use of the computer

.A more common definition, and the one that we usually follow, is that the operating system is the one program running at all times on the computer—usually called the kernel. (Along with the kernel, there are two other types of programs: system programs, which are associated with the operating system but are not necessarily part of the kernel, and application programs, which include all programs not associated with the operation of the system.)

A modern general-purpose computer system consists of one or more CPUs and a number of device controllers connected through a common bus that provides access to shared memory (Figure 1.2). Each device controller is in charge of a specific type of device (for example, disk drives, audio devices, or video displays). The CPU and the device controllers can execute in parallel, competing for memory cycles. To ensure orderly access to the shared memory, a memory controller synchronizes access to the memory. For a computer to start running—for instance, when it is powered up or rebooted—it needs to have an initial program to run. This initial program, or bootstrap program, tends to be simple. Typically, it is stored within the computer hardware in read-only memory (ROM) or electrically erasable programmable read-only memory (EEPROM), known by the general term firmware. It

initializes all aspects of the system, from CPU registers to device controllers to memory contents. The bootstrap program must know how to load the operating system and how to start executing that system.

the bootstrap program must locate the operating-system kernel and load it into memory. Once the kernel is loaded and executing, it can start providing services to the system and its users. Some services are provided outside of the kernel, by system programs that are loaded into memory at boot time to become system processes, or system daemons that run the entire time the kernel is running. On UNIX, the first system process is “init,” and it starts many other daemons. Once this phase is complete, the system is fully booted, and the system waits for some event to occur. The occurrence of an event is usually signaled by an interrupt from either the hardware or the software. Hardware may trigger an interrupt at any time by sending a signal to the CPU, usually by way of the system bus. Software may trigger an interrupt by executing a special operation called a system call (also called a monitor call). When the CPU is interrupted, it stops what it is doing and immediately transfers execution to a fixed location. The fixed location usually contains the starting address where the service routine for the interrupt is located

The interrupt must transfer control to the appropriate interrupt service routine.

The routine, in turn, would call the interrupt-specific handler. However, interrupts must be handled quickly. Since only a predefined number of interrupts is possible, a table of pointers to interrupt routines can be used instead to provide the necessary speed. The interrupt routine is called indirectly through the table, with no intermediate routine needed. Generally, the table of pointers is stored in low memory(the first hundred or so locations)

The CPU can load instructions only from memory, so any programs to run must be stored there. General-purpose computers run most of their programs from rewritable memory, called main memory (also called

EEPROM can be changed but cannot be changed frequently and so contains mostly static programs. For example, smartphones have EEPROM to store their factory-installed programs. All forms of memory provide an array of bytes. Each byte has its own address. Interaction is achieved through a sequence of load or store instructions to specific memory addresses. The load instruction moves a byte or word from main memory to an internal register within the CPU, whereas the”**store instruction moves the content of a register to main memory.

A typical instruction–execution cycle, as executed on a system with a von Neumann architecture, first fetches an instruction from memory and stores that instruction in the instruction register. The instruction is then decoded and may cause operands to be fetched from memory and stored in some internal register. After the instruction on the operands has been executed, the result may be stored back in memory. Notice that the memory unit sees only a stream of memory addresses

sequence of memory addresses generated by the running program. Ideally, we want the programs and data to reside in main memory permanently. This arrangement usually is not possible for the following two reasons: 1. Main memory is usually too small to store all needed programs and data permanently. 2. Main memory is a volatile storage device that loses its contents when power is turned off or otherwise lost. Thus, most computer systems provide secondary storage as an extension of main memory. The main requirement for secondary storage is that it be able to hold large quantities of data permanently. The most common secondary-storage device is a magnetic disk, which provides storage for both programs and data. Most programs (system and application) are stored on a disk until they are loaded into memory.

registers, main memory, and magnetic disks — is only one of many possible storage systems. Others include cache memory, CD-ROM, magnetic tapes, and so on. Each storage system provides the basic functions of storing a datum and holding that datum until it is retrieved at a later time. The main differences among the various storage systems lie in speed, cost, size, and volatility.

In the hierarchy shown in Figure 1.4, the storage systems above the solid-state disk are volatile, whereas those including the solid-state disk and below are nonvolatile. Solid-state disks have several variants but in general are faster than magnetic disks and are nonvolatile. One type of solid-state disk stores data in a large DRAM array during normal operation but also contains a hidden magnetic hard disk and a battery for backup power. If external power is interrupted, this solid-state disk's controller copies the data from RAM to the magnetic disk. When external power is restored, the controller copies the data back into RAM. Another form of solid-state disk is flash memory, which is popular in cameras and personal digital assistants (PDAs), in robots, and increasingly for storage on general-purpose computers

Another form of non-volatile storage is NVRAM, which is DRAM with battery backup power

Storage is only one of many types of I/O devices within a computer. A large portion of operating system code is dedicated to managing I/O, both because of its importance to the reliability and performance of a system and because of the varying nature of the devices. Next, we provide an overview of I/O. A general-purpose computer system consists of CPUs and multiple device controllers that are connected through a common bus. Each device controller is in charge of a specific type of device. Depending on the controller, more than one device may be attached. For instance, seven or more devices can be attached to the small computer-systems interface (SCSI) controller. A device controller maintains some local buffer storage and a set of special-purpose registers. The device controller is responsible for moving the data between the peripheral devices that it controls and its local buffer storage. Typically, operating systems have a device driver for each device controller. This device driver understands the device controller and provides the rest of the operating system with a uniform interface to the device. To start an I/O operation, the device driver loads the appropriate registers within the device controller. The device controller, in turn, examines the contents

of these registers to determine what action to take (such as “read a character from the keyboard”). The controller starts the transfer of data from the device to its local buffer. Once the transfer of data is complete, the device controller informs the device driver via an interrupt that it has finished its operation. The device driver then returns control to the operating system, possibly returning the data or a pointer to the data if the operation was a read. For other operations, the device driver returns status information. This form of interrupt-driven I/O is fine for moving small amounts of data but can produce high overhead when used for bulk data movement such as disk I/O. To solve this problem, direct memory access (DMA) is used. After setting up buffers, pointers, and counters for the I/O device, the device controller transfers an entire block of data directly to or from its own buffer storage to memory, with no intervention by the CPU

Such systems have two or more processors in close communication, sharing the computer bus and sometimes the clock, memory, and peripheral devices

Multiprocessor systems have three main advantages

Increased throughput.

Economy of scale.

Increased reliability

The ability to continue providing service proportional to the level of surviving hardware is called graceful degradation. Some systems go beyond graceful degradation and are called fault tolerant, because they can suffer a failure of any single component and still continue operation. Fault tolerance requires a mechanism to allow the failure to be detected, diagnosed, and, if possible, corrected.

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

The most common systems use symmetric multiprocessing (SMP), in which each processor performs all tasks within the operating system. SMP means that all processors are peers; no boss–worker relationship exists between processors.

It is important to note that while multicore systems are multiprocessor systems,not all multi processor systems are multicore

n Figure 1.7, we show a dual-core design with two cores on the same chip. In this design, each core has its own register set as well as its own local cache. Other designs might use a shared cache or a combination of local and shared caches.

Another type of multiprocessor system is a clustered system, which gathers together multiple CPUs. Clustered systems differ from the multiprocessor systems described in Section 1.3.2 in that they are composed of two or more individual systems—or nodes—joined together. Such systems are considered loosely coupled. Each node may be a single processor system or a multicore system. We should note that the definition of clustered is not concrete; many commercial packages wrestle to define a clustered system and why one form is better than another. The generally accepted definition is that clustered computers share storage and are closely linked via a local-area network LAN

Clustering can be structured asymmetrically or symmetrically. In asymmetric clustering, one machine is in hot-standby mode while the other is running the applications. The hot-standby host machine does nothing but monitor the active server. If that server fails, the hot-standby host becomes the active server. In symmetric clustering, two or more hosts are running applications and are monitoring each other. This structure is obviously more efficient, as it uses all of the available hardware.

The application must have been written specifically to take advantage of the cluster, however. This involves a technique known as parallelization, which divides a program into separate components that run in parallel on individual computers in the cluster. Typically, these applications are designed so that once each computing node in the cluster has solved its portion of the problem, the results from all the nodes are combined into a final solution.

One of the most important aspects of operating systems is the ability to multiprogram

The idea is as follows: The operating system keeps several jobs in memory simultaneously (Figure 1.9). Since, in general, main memory is too small to accommodate all jobs, the jobs are kept initially on the disk in the job pool. This pool consists of all processes residing on disk awaiting allocation of main memory. The set of jobs in memory can be a subset of the jobs kept in the job pool. The operating system picks and begins to execute one of the jobs in memory. Eventually, the job may have to wait for some task, such as an I/O operation, to complete. In a non-multi programmed system, the CPU would sit idle. In a multi programmed system, the operating system simply switches to, and executes, another job

lawyer

Time sharing (or multitasking) is a logical extension of multiprogramming. In time-sharing systems, 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. Time sharing requires an interactive computer system, which provides direct communication between the user and the system. The user gives instructions to the operating system or to a program directly, using a input device such as a keyboard, mouse, touch pad, or touch screen, and waits for immediateresults on anoutput device.Accordingly, the response time should be short—typically less than one second. Atime-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 his use, even though it is being shared among manyusers.

A program loaded into memory and executing is called a process

Time sharing and multiprogramming require that several jobs be kept simultaneously in 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 involves job scheduling, which we discuss in Chapter6.When the operating system selects a job from the job pool, it loads that job into memory for execution.

In addition, if several jobs are ready to run at the same time, the system must choose which job will run first. Making this decision is CPU scheduling

response time. This goal is sometimes accomplished through swapping, whereby processes are swapped in and out of main memory to the disk. A more common method for ensuring reasonable response time is virtual memory, a technique that allows the execution of a process that is not completely in memory(Chapter9).The main advantage of the virtual-memory scheme is that it enables users to run programs that are larger than actual physical memory. Further, it abstracts main memory into a large, uniform array of storage, separating logical memory as viewed by the user from physical memory. This arrangement frees programmers from concern over memory-storage limitations.

modern operating systems are interrupt driven

. Events are almost always signaled by the occurrence of an interrupt or a trap. A trap (or an exception) is a software-generated interrupt caused either by an error (for example, division by zero or invalid memory access

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

be adversely affected by a bug in one program. For example, if a process gets stuck in an infinite loop, this loop could prevent the correct operation of many other processes

Without protection against these sorts of errors, either the computer must execute only one process at a time or all output must be suspect. A properly designed operating system must ensure that an incorrect (or malicious) program cannot cause other programs to execute incorrectly

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

At the very least, we need two separate modes of operation: user mode and kernel mode (also called supervisor mode, system mode, or privileged mode). A bit, called the mode bit, is added to the hardware of the computer to indicate the current mode: kernel (0) or user (1). With the mode bit, we can distinguish between a task that is executed on behalf of the operating system and one that is executed on behalf of the user. When the computer system is executing on behalf of a user application, the system is in user mode. However, when a user application requests a service from the operating system (via a system call), the system must transition from user to kernel mode to fulfill the request. This is shown in Figure 1.10.

At system boot time, the hardware starts in kernel mode. The operating system is then loaded and starts user applications in user mode. Whenever a trap or interrupt occurs, the hardware switches from user mode to kernel mode (that is, changes the state of the mode bit to 0). Thus, whenever the operating system gains control of the computer, it is in kernel mode. The system always switches to user mode (by setting the mode bit to 1) before passing control to a user program.

We accomplish this protection by designating some of the machine instructions that may cause harm as privileged instructions. The hardware allows privileged instructions to be executed only in kernel mode. If an attempt is made to execute a privileged instruction in user mode, the hardware does not execute the instruction but rather treats it as illegal and traps it to the operating system.

The concept of modes can be extended beyond two modes (in which case the CPU uses more than one bit to set and test the mode). CPUs that support virtualization (Section 16.1) frequently have a separate mode to indicate when the virtual machine manager (VMM)—and the virtualization management software—is in control of the system. In this mode, the VMM has more privileges than user processes but fewer than the kernel. It needs that level of privilege so it can create and manage virtual machines, changing the CPU state to do so.

The lack of a hardware-supported dual mode can cause serious shortcom ings in an operating system. For instance, MS-DOS was written for the Intel 8088 architecture, which has no mode bit and therefore no dual mode. A user program running awry can wipe out the operating system by writing over it with data; and multiple programs are able to write to a device at the same time, with potentially disastrous results. Modern versions of the Intel CPU do provide dual-mode operation

Timer We must ensure that the operating system maintains control over the CPU. We cannot allow a user program to get stuck in an infinite loop or to fail to call system services and never return control to the operating system. To accomplish this goal, we can use a timer. A timer can be set to interrupt the computer after a specified period. The period may be fixed (for example, 1/60 second) or variable (for example, from 1 millisecond to 1 second). A variable timer is generally implemented by a fixed-rate clock and a counter. The operating system sets the counter. Every time the clock ticks, the counter is decremented. When the counter reaches 0, an interrupt occurs

ns when it is created, various initialization data (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 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.

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. (Threads are covered in Chapter 4.) The execution of such a process must be sequential. The CPU executes one instruction of the process after another, until the process completes

. A multithreaded process has multiple program counters, each pointing to the next instruction to execute for a given thread. 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**

. Main memory isa large array of bytes, ranging in size from hundreds of thousands to billions. Each 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 both reads and writes data from main memory during the data-fetch cycle (on a von Neumann architecture)

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**

The operating system maps files onto physical media and accesses these files via the storage devices.

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 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 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.

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

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

the computer system must provide secondary storage to back up main memory.

Most programs—including compilers, assemblers, word processors, editors, and formatters—are stored on a disk until loaded into memory.

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

- **Free-space management**
- **Storage allocation**
- **Disk scheduling**

Caching is an important principle of computer systems. Here's how it works. Information is normally kept in some storage system (such as main memory). As it is used, it is copied into a faster storage system—the cache—on a 28 Chapter 1 Introduction 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. Other caches are implemented totally in hardware. For instance, most systems have an instruction cache to hold the instructions expected to be executed next. Without this cache, the CPU would have to wait several cycles while an instruction was fetched from main memory

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

In a computing environment where only one process executes at a time, this arrangement poses no difficulties, since an access to integer A will always be to the copy at the highest level of the hierarchy. However, in a multitasking environment, where the CPU is switched back and forth among various processes, extreme care must be taken to ensure that, if several processes wish to access A, then each of these processes will obtain the most recently updated value of A. The situation becomes more complicated in a multiprocessor environment where, in addition to maintaining internal registers, each of the CPUs also contains a local cache (Figure 1.6). In such an environment, a copy of A may exist simultaneously in several caches. Since the various CPUs can all execute in parallel, 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 (handled below the operating-system level). In a distributed environment, the situation becomes even more complex. In this environment, several copies (or replicas) of the same file can be kept on different computers. Since the various replicas may be accessed and updated concurrently, some distributed systems ensure that, when a replica is updated in one place, all other replicas are brought up to date as soon as possible. There are various ways to achieve this guarantee, as we discuss in Chapter 17

Protection, then, is any mechanism for controlling the access of processes or users to the resources defined by a computer system.

A system can have adequate protection but still be prone to failure and allow inappropriate access. Consider a user whose authentication information (her means of identifying herself to the system) is stolen. Her data could be copied or deleted, even though file and memory protection are working. It is the job of security to defend a system from external and internal attacks.

directly. For example, main memory is constructed as an array. If the data item being stored is larger than one byte, then multiple bytes can be allocated to the item, and the item is addressed as item number × item size. But what about storing an item whose size may vary ?And what about removing an item if the relative positions of the remaining items must be preserved ?In such situations, arrays give way to other data structures. After arrays, lists are perhaps the most fundamental data structures in computer science. Whereas each item in an array can be accessed directly, the items in a list must be accessed in a particular order. That is, a list represents a collection of data values as a sequence. The most common method for

implementing this structure is a linked list, in which items are linked to one another. Linked lists are of several types: • In a singly linked list, each item points to its successor, as illustrated in Figure 1.13. • In a doubly linked list, a given item can refer either to its predecessor or to its successor, as illustrated in Figure 1.14. • In a circularly linked list, the last element in the list refers to the first element, rather than to null, as illustrated in Figure 1.15. Linked lists accommodate items of varying sizes and allow easy insertion and deletion of items. One potential disadvantage of using a list is that performance for retrieving a specified item in a list of size n is linear — O(n), as it requires potentially traversing all n elements in the worst case.

A stack is a sequentially ordered data structure that uses the last in, first out (LIFO) principle for adding and removing items, meaning that the last item placed onto a stack is the first item removed. The operations for inserting and removing items from a stack are known as push and pop, respectively. An operating system often uses a stack when invoking function calls

A queue, in contrast, is a sequentially ordered data structure that uses the first in, first out (FIFO) principle: items are removed from a queue in the order in which they were inserted. There are many everyday examples of queues, including shoppers waiting

in a checkout line at a store and cars waiting inline at a traffic signal. Queues are also quite common in operating systems—jobs that are sent to a printer are typically printed in the order in which they were submitted, for example.

A tree is a data structure that can be used to represent data hierarchically. Data values in a tree structure are linked through parent–child relationships. In a general tree, a parent may have an unlimited number of children. In a binary tree, a parent may have at most two children, which we term the left child and the right child. A binary search tree additionally requires an ordering between the parent's two children in which left child <= right child.

A hash function takes data as its input, performs a numeric operation on this data, and returns a numeric value. This numeric value can then be used as an index into a table (typically an array) to quickly retrieve the data

One potential difficulty with hash functions is that two inputs can result in the same output value—that is, they can link to the same table location. We can accommodate this hash collision by having a linked list at that table location that contains all of the items with the same hash value

One use of a hash function is to implement a hash map, which associates (or maps) [key:value] pairs using a hash function. For example, we can map the key operating to the value system. Once the mapping is established, we can apply the hash function to the key to obtain the value from the hash map (Figure 1.17). For example, suppose that a username is mapped to a password. Password authentication then proceeds as follows: a user enters his user name and password. The hash function is applied to the user name, which is then used to retrieve the password. The retrieved password is then compared with the password entered by the user for authentication.

A bitmap is a string of n binary digits that can be used to represent the status of n items. For example, suppose we have several resources, and the availability of each resource is indicated by the value of a binary digit: 0 means that the resource is available, while 1 indicates that it is unavailable (or vice-versa). The value of the it h position in the bitmap is associated with the it h resource. As an example, consider the bitmap shown below:

A medium-sized disk drive might be divided into sever al thousand individual units, called disk blocks. A bitmap can be used to indicate the availability of each disk block

Mobile computing refers to computing on handheld smartphones and tablet computers. These devices share the distinguishing physical features of being portable and lightweight.

A distributed system is a collection of physically separate, possibly heterogeneous, computer systems that are networked to provide users with access to the various resources that the system maintains

A network operating system is an operating system that provides features such as file sharing across the network, along with a communication scheme that allows different processes on different computers to exchange messages.

In this model, clients and servers are not distinguished from one another. Instead, all nodes within the system are considered peers, and each may act as either a client or a server, depending on whether it is requesting or providing a service. Peer-to-peer systems offer an advantage over traditional client-server systems. In a client-server system, the server is a bottleneck; but in a peer-to-peer system, services can be provided by sever al nodes distributed throughout the network