# Operating systems

# What are the different types?

**Linux** is a Unix-like computer operating system assembled under the model of free and open source software

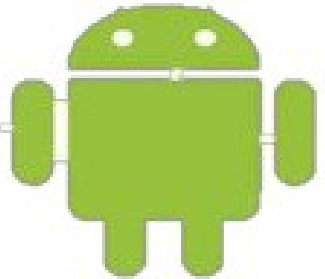
development and distribution.

**Microsoft Windows** is a series of graphical interface operating systems developed, marketed, and sold by Microsoft.

**Mac OS** is a series of graphical user interface-based operating systems developed by Apple Inc. for their Macintosh

**iOS** (previously **iPhone OS**) is a mobile operating system developed and distributed by Apple Inc. Originally unveiled in 2007 for the iPhone, it has been extended to support other Apple devices such as

the iPod Touch

**Android** is a Linux-based operating system designed primarily for touchscreen mobile devices such as smartphones and tablet computers. Initially developed by Android, Inc.

BSD/OS had a reputation for reliability in server roles; the renowned Unix programmer and author W. Richard Stevens used it for his own personal web server for this reason.

- A computer system can be divided roughly into four components: **the hardware, the operating system, the application programs, and the users**.
- The hardware—the central processing unit ( CPU ), the memory, and the input/output ( I/O ) devices—provides the basic computing resources for the system.
- The application programs—such as word processors, spreadsheets, compilers, and Web browsers—define the ways in which these resources are used to solve users' computing problems.
- The operating system controls the hardware and coordinates its use among the various application programs for the various users.
- Like a government, it performs no useful function by itself. It simply provides an environment within which other programs can do useful work

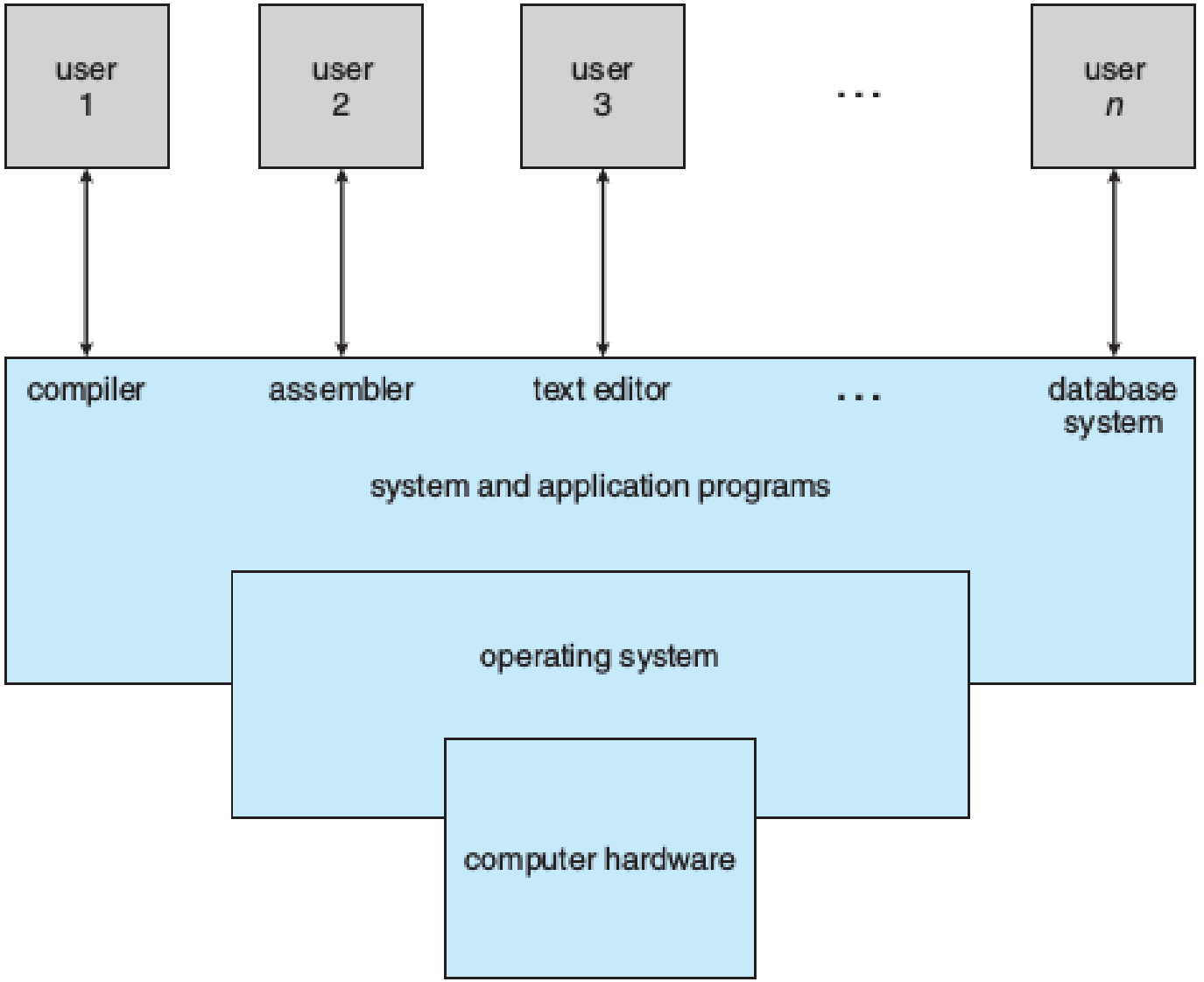



**Figure 1.1** Abstract view of the components of a computer system.

## User View

### 1. PC

- 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.
- The goal is 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 —how various hardware and software resources are shared.
- Performance is, of course, important to the user; but such systems are optimized for the single-user experience rather than the requirements of multiple users.

## 2. Mainframe

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

## System View

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

# Computer System Architecture

1. Single Processor Systems
2. Multiprocessor Systems
3. Clustered Systems

## 1. Single Processor Systems

- On a single- processor system, there is one main CPU capable of executing a general-purpose instruction set, including instructions from user processes.
- Almost all single-processor systems have other special-purpose processors as well. They may come in the form of device-specific processors, such as disk, keyboard, and graphics controllers; or, on mainframes, they may come in the form of more general-purpose processors, such as I/O processors that move data rapidly among the components of the system.
- All of these special-purpose processors run a limited instruction set and do not run user processes.
- For example, a disk-controller microprocessor receives a sequence of requests from the main CPU and implements its own disk queue and scheduling algorithm. This arrangement relieves the main CPU of the overhead of disk scheduling.

## 2. Multiprocessor Systems

- Multiprocessor systems (also known as parallel systems or multicore 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:

  a. **Increased throughput**. By increasing the number of processors, we expect to get more work done in less time. The speed-up ratio with N processors is not N , however; 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.

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

**3. 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 can pick up a share of the work of the failed processor. Thus, the entire system runs only 10 percent slower, rather than failing altogether.

The multiple-processor systems in use today are of two types.

- **Asymmetric multiprocessing** Each processor is assigned a specific task. A boss processor controls the system; the other processors either look to the boss for instruction or have predefined tasks.
- This scheme defines a boss–worker relationship. The boss processor schedules and allocates work to the worker processors.
- **Symmetric multiprocessing ( SMP )** Each processor performs all tasks within the operating system. SMP means that all processors are peers; no boss–worker relationship exists between processors.
- The benefit of this model is that many processes can run simultaneously— N processes can run if there are N CPU s— without causing performance to deteriorate significantly.
- However, we must carefully control I/O to ensure that the data reach the appropriate processor. Also, since the CPU s are separate, one may be sitting idle while another is overloaded, resulting in inefficiencies.

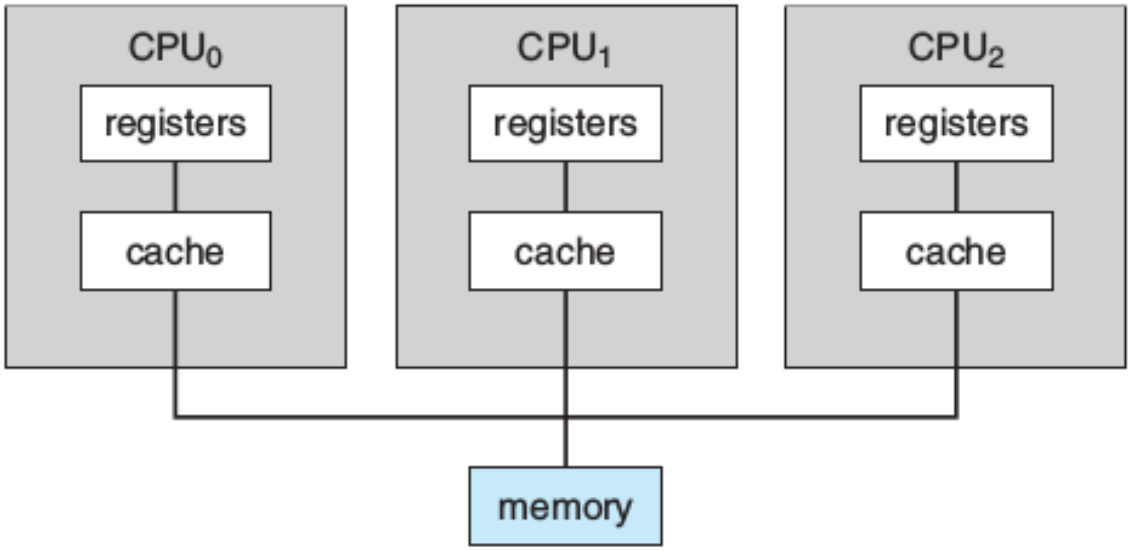



**Figure 1.6** Symmetric multiprocessing architecture.

- **Multicore System** Multiple computing cores on a single chip. Such multiprocessor systems are termed multicore.
- They can be more efficient than multiple chips with single cores because on-chip communication is faster than between-chip communication. In addition, one chip with multiple cores uses significantly less power than multiple single-core chips.
- While multicore systems are multiprocessor systems, not all multiprocessor systems are multicore

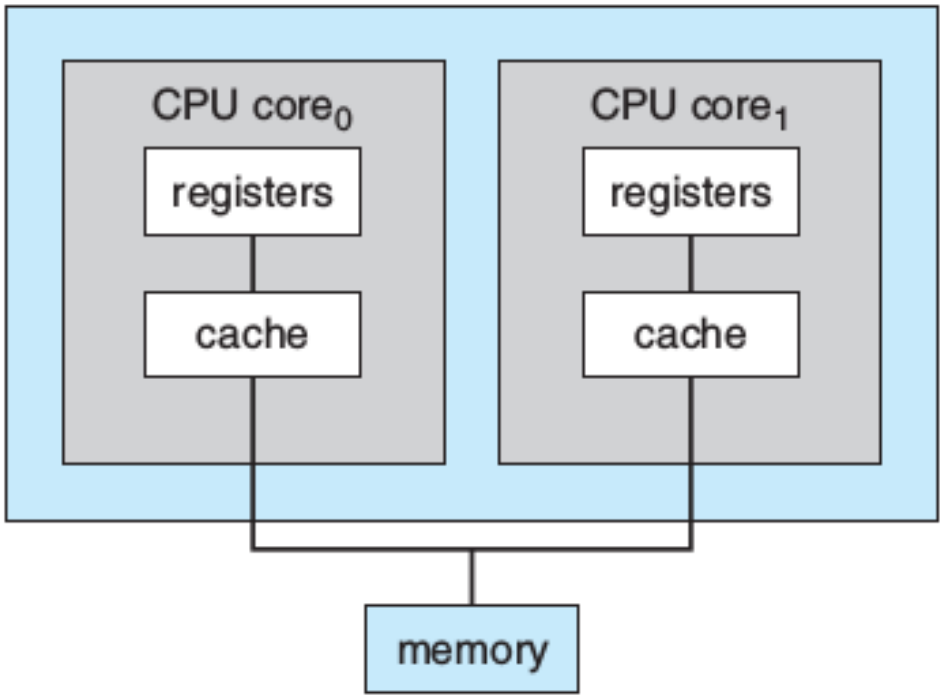



**Figure 1.7** A dual-core design with two cores placed on the same chip.

## 3. Clustered Systems

- Another type of multiprocessor system is a clustered system, which gathers together multiple CPU s.
- 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.
- Clustering is usually used to provide high-availability service —that is, service will continue even if one or more systems in the cluster fail.
- 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. However it does require that more than one application be available to run.

- Since a cluster consists of several computer systems connected via a network, clusters can also be used to provide high-performance computing environments.
- Such systems can supply significantly greater computational power than single-processor or even SMP systems because they can run an application concurrently on all computers in the cluster.
- 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.

# Operating System Structure

## 1. Multiprogramming

- A single program cannot, in general, keep either the CPU or the I/O devices busy at all times. Single users frequently have multiple programs running. Multiprogramming increases CPU utilization by organizing jobs (code and data) so that the CPU always has one to execute.
- The operating system keeps several jobs in memory simultaneously. 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.

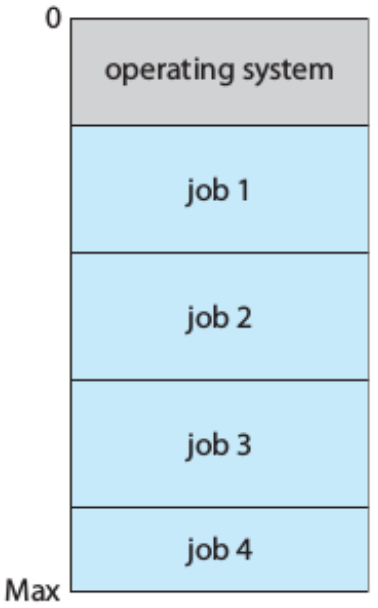



**Figure 1.9** Memory layout for a multiprogramming system.

- 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-multiprogrammed system, the CPU would sit idle. In a multiprogrammed system, the operating system simply switches to, and executes, another job.
- When that job needs to wait, the CPU switches 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.
- Multiprogrammed systems provide an environment in which the various system resources (for example, CPU , memory, and peripheral devices) are utilized effectively, but they do not provide for user interaction with the computer system.

- **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 immediate results on an output device. Accordingly, the response time should be short—typically less than one second.
- 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 his use, even though it is being shared among many users.
- A time-shared operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared computer. Each user has at least one separate program in memory.

- A program loaded into memory and executing is called a **process**. When a process executes, it typically executes for only a short time before it either finishes or needs to perform I/O .
- I/O may be interactive; that is, output goes to a display for the user, and input comes from a user keyboard, mouse, or other device. It may take a long time to complete.
- Rather than let the CPU sit idle as this interactive input takes place, the operating system will rapidly switch the CPU to the program of some other user.
- 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.
- 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 which job will run first. Making this decision is CPU scheduling.

## Operating System Operations

- Modern operating systems are interrupt driven. If there are no processes to execute, no I/O devices to service, and no users to whom to respond, an operating system will sit quietly, waiting for something to happen.
- Events are almost always signaled by the occurrence of an interrupt or a trap. A trap (or an exception) is a software-generated interrupt caused either by an error (for example, division by zero or invalid memory access) or by a specific request from a user program that an operating-system service be performed.
- For each type of interrupt, separate segments of code in the operating system determine what action should be taken. An **interrupt service routine** is provided to deal with the interrupt.
- 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. More subtle errors can occur in a multiprogramming system, where one erroneous program might modify another program, the data of another program, or even the operating system itself.
- 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.

## 1. Dual-Mode Operation

- In order to ensure the proper execution of the operating system, we must be able to distinguish between the execution of operating-system code and user-defined code.
- 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)**.
- When the computer system is executing on behalf of a user application, the system is in user mode. However, when a user application requests a service from the operating system (via a system call), the system must transition from user to kernel mode to fulfill the request.

- At system boot time, the hardware starts in kernel mode. The operating system is then loaded and starts user applications in user mode.
- Whenever a trap or interrupt occurs, the hardware switches from user mode to kernel mode (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.

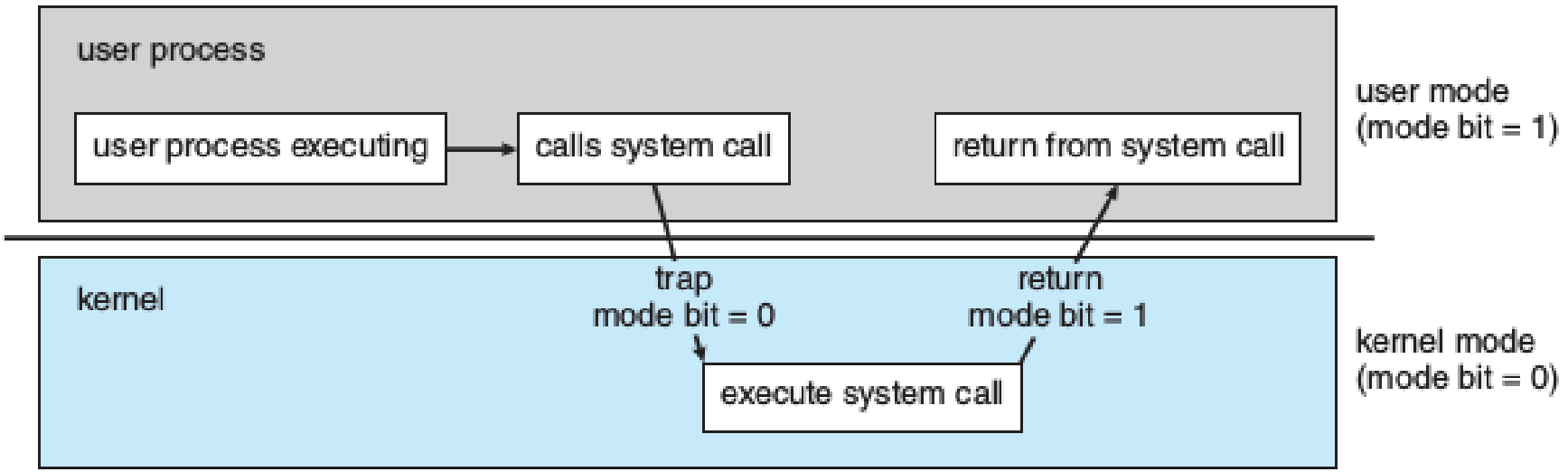



**Figure 1.10** Transition from user to kernel mode.

- The dual mode of operation provides us with the means for protecting the operating system from errant users—and errant users from one another.
- Some of the machine instructions that may cause harm are **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 instruction to switch to kernel mode is an example of a privileged instruction.

## System Call

- Initial control resides in the operating system, where instructions are executed in kernel mode. When control is given to a user application, the mode is set to user mode. Eventually, control is switched back to the operating system via an interrupt, a trap, or a system call.
- **System calls** provide the means for a user program to ask the operating system to perform tasks reserved for the operating system on the user program's behalf.
- A system call usually takes the form of a trap to a specific location in the interrupt vector. When a system call is executed, it is typically treated by the hardware as a software interrupt. Control passes through the interrupt vector to a service routine in the operating system, and the mode bit is set to kernel mode. The system-call service routine is a part of the operating system.
- The kernel examines the interrupting instruction to determine what system call has occurred; a parameter indicates what type of service the user program is requesting. Additional information needed for the request may be passed in registers, on the stack, or in memory (with pointers to the memory locations passed in registers).
- The kernel verifies that the parameters are correct and legal, executes the request, and returns control to the instruction following the system call.

## 2. Timer

- 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.
- A timer can be set to interrupt the computer after a specified period. The period may be **fixed** or **variable**.
- 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. If the timer interrupts, control transfers automatically to the operating system, which may treat the interrupt as a fatal error or may give the program more time.
- Clearly, instructions that modify the content of the timer are privileged.
- We can use the timer to prevent a user program from running too long. A simple technique is to initialize a counter with the amount of time that a program is allowed to run. Every second, the timer interrupts, and the counter is decremented by 1. As long as the counter is positive, control is returned to the user program. When the counter becomes negative, the operating system terminates the program for exceeding the assigned time limit.

## Process

- A program does nothing unless its instructions are executed by a CPU . A program in execution, is a process. 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 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 (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. When the process terminates, the operating system will reclaim any reusable resources.

- A program is a passive entity, like the contents of a file stored on disk, whereas a process is an active entity. The CPU executes one instruction of the process after another, until the process completes.
- The operating system is responsible for the following activities in connection with process management:

  a. Scheduling processes and threads on the CPU s

  b. Creating and deleting both user and system processes

  c. Suspending and resuming processes

  d. Providing mechanisms for process synchronization

  e. Providing mechanisms for process communication

## Memory Management

- Main memory is a 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
- 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. In the same way, 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, 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:

  a. Keeping track of which parts of memory are currently being used and who is using them

  b. Deciding which processes (or parts of processes) and data to move into and out of memory

  c. Allocating and deallocating memory space as needed

## File System Management

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

  a. Creating and deleting files

  b. Creating and deleting directories to organize files

  c. Supporting primitives for manipulating files and directories

  d. Mapping files onto secondary storage

  e. Backing up files on stable (nonvolatile) 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.
- The operating system is responsible for the following activities in connection with disk management:

  a. Free-space management

  b. Storage allocation

  c. Disk scheduling

## Caching

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

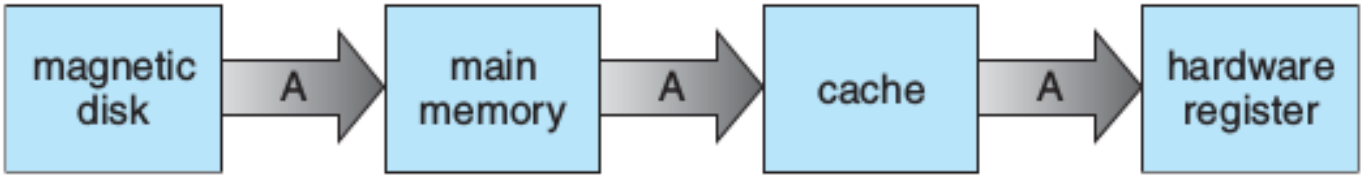



**Figure 1.12** Migration of integer A from disk to register.

- 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.
- 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.

- 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 CPU s also contains a local cache.
- In such an environment, a copy of A may exist simultaneously in several caches. Since the various CPU s 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**.