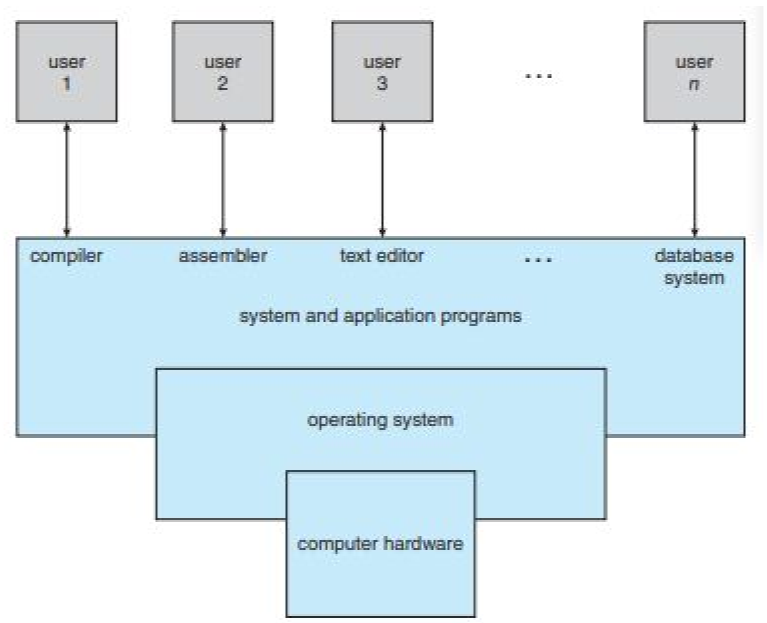
Operating System Concepts

Chapter 1

The purpose of an operating system is to provide an environment in which a user can execute programs in a convenient and efficient manner.
The hardware must provide appropriate mechanisms to ensure the correct operation of the computer system and to prevent user programs from interfering with the proper operation of the system.

Introduction



1.1 What Operating Systems Do

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 - such as word processors, spreadsheets, compilers, and the 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.

1.1.1 User View

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.

1.1.2 System View

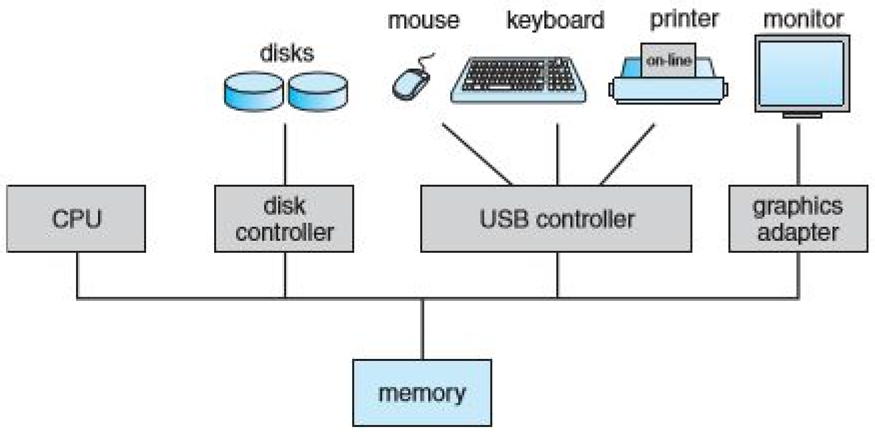
From the computer's point of view, the operating system is the program most intimately involved with the hardware. We can view an operating system as a resource allocator.
The operating system acts as the manager of these resources. Facing numerous and possibly conflicting requests for resources, the 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.

1.1.3 Defining Operating Systems

Operating systems exist because they offer a reasonable way to solve the problem of creating a usable computing system. The fundamental goal of computer systems is to execute user programs and to make solving user problems easier.

1.2 Computer-System Organization

1.2.1 Computer-System Operation



Bootstrap program starts the computer from the ROM, know to be called firmware.

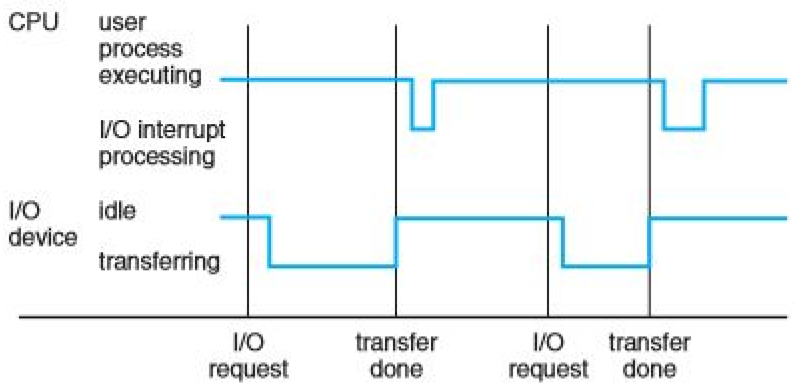


Figure 1.3 Interrupt timeline for a single process doing output.

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. 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.
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 (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 service routine executes, on completion, the CPU resumes the interrupted computation.
Interrupts are an important part of a computer architecture. Each computer design has its own interrupt mechanism, but several functions are common. The interrupt must transfer control to the appropriate interrupt service routine. The straightforward method for handling this transfer would be to invoke a generic routine to examine the interrupt information. 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 is called indirectly through the table, with no intermediate routine needed. Generally, the table of pointers is stored in low memory. The interrupt vector, of addresses is then indexed by a unique device number, given with the interrupt request, to provide the address of the interrupt service routine for the interrupting device.
The interrupt architecture must also save the address for the interrupted instruction. Many old designs simply stored the interrupt address in a fixed location or in a location indexed by the device number. If the interrupt routine needs to modify the processor state - for instance, by

modifying register values - it must explicitly save the current state and then restore that is loaded into the program counter, and the interrupted computation resumes as though the interrupt had not occurred.

1.2.2 Storage Structure

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.

1.2.3 I/O Structure

A general-purpose computer system consists of CPU's and multiple device controllers that are connected through a common bus. Each device controller is in charge of specific type of device.

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. 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.

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. Only one interrupt is generated per block, to tell the device driver that the operation has completed, rather than the one interrupt per byte generated for low-speed devices. While the device controller is performing these operations, the CPU is available to accomplish other work.

1.3 Computer-System Architecture

1.3.1 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.
All of the special-purpose processors run a limited instruction set and do not run processes. Sometimes, they are managed by the operating system, in that the operating system sends them information about their next task and monitors their status.

1.3.2 Multiprocessor Systems

Multiprocessor systems have three main advantages:

1. Increased throughput
2. Economy of scale
3. Increased reliability

The multiple-processor systems in use today are 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 boss processor schedules and allocates work to the worker processors.

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

We must carefully control I/O to ensure that the data reach the appropriate processor. Also, since the CPUs are separate, one may be sitting idle while another is overloaded, resulting in inefficiencies. These inefficiencies can be avoided if the processors share certain data structures.

A multiprocessor system of this form will allow processes and resources—such as memory— to be shared dynamically among the various processors and can lower the variance among the processors.

Either way, multiprocessing can cause a system to change its memory access model from uniform memory access (**UMA**) to non-uniform memory access (**NUMA**). UMA is defined as the situation in which access to any RAM from any CPU takes the same amount of time.

1.3.3 Clustered Systems

Another type of multiprocessor system is a **clustered system**, which gathers together multiple CPUs. Clustered systems differ from multiprocessor systems 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.

Clustering is usually used to provide **high-availability** service—that is, service will continue even if one or more systems in the cluster fail. Generally, we obtain high availability by adding a level of redundancy in the system.

Clustering can be structured asymmetrically or symmetrically. In **asym- metric 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.

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.

1.4 Operating-System Structure

Multiprogramming increases CPU utilization by organizing jobs (code and data) so that the CPU always has one to execute.

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.

1.5 Operating-System Operations

Modern operating systems are interrupt driven.
A trap or an exception is a software-generated interrupt caused either by an error or by a specific request from a user program than an operating-system service be performed.

1.5.1 Dual-Mode and Multimode 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.

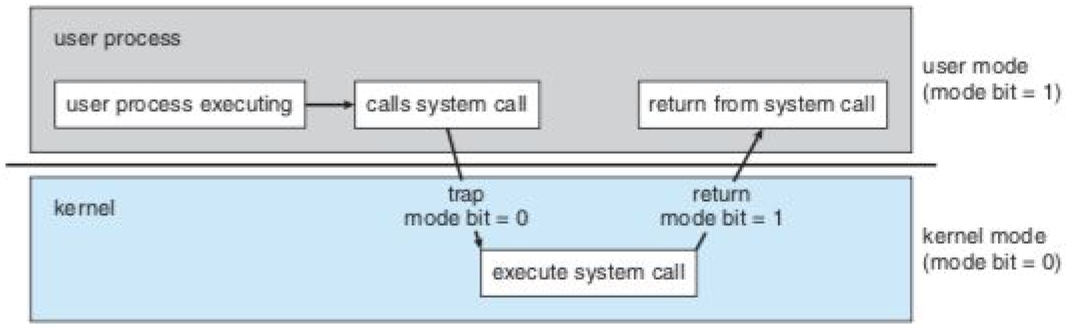


Figure 1.10 Transition from user to kernel mode.

Two modes of operation:

1. User mode
    a.
2. Kernel 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 bit mode, we can distinguish between a task that is executed on behalf of the user. The system always switches to user mode before passing control to a user program.
The dual mode of operation provides us with the means for protecting the operating system from errant users - and errant users from one another. 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.

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.

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.

When a system call is executed, it is typically treated by the hardware as a software interrupt.

1.5.2 Timer

A timer can be set to interrupt the computer after a specified period, a variable timer is generally implemented by a fixed-rate clock and a counter.

1.6 Process Management

A program does nothing unless its instructions are executed by a CPU. A program in execution, as mentioned, 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 system task, such as sending output to a printer, can also be a process.

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.

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). All these processes can potentially execute concurrently—by multiplexing on a single CPU, for example.

The operating system is responsible for the following activities in connec- tion with process management:

- Scheduling processes and threads on the CPUs
- Creating and deleting both user and system processes
- Suspendingandresumingprocesses
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication

1.7 Memory Management

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

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:

- Keep 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

1.8 Storage Management

The operating system provides a uniform, logical view of information storage. The operating system abstracts from the physical properties of its storage devices to define a logical storage unit, the file.

18.1 File-System Management

File management is one of the most visible components of an operating system.
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

18.2 Mass-Storage Management

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

- Free-space management
- Storage allocation
- Disk scheduling

18.3 Caching

Caching is an important principle of computer systems. Here's how it works. Information is normally kept in some storage system. As it is used, it is copied into a faster 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. Main memory can be viewed as a fast cache for secondary storage, since data in secondary storage must be copied into main memory for use and data must be in main memory before being moved to secondary storage for safekeeping. At the highest level, the operating system may maintain a cache of file-system data in main memory.

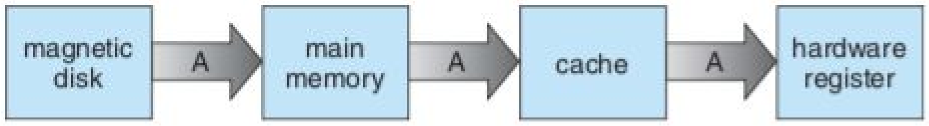


Figure 1.12 Migration of integer A from disk to register.

The movement of information between levels of a storage hierarchy may be either explicit or implicit, depending on the hardware design and the controlling operating-system software. INa hierarchical storage structure, the same data may appear in different levels of the storage system.

1.8.4 I/O Systems

The I/O subsystem consists of several components:

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

1.9 Protection and Security

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

Protection can improve reliability by detecting latent errors at the interfaces between component subsystems. A system can have adequate protection but still be prone to failure and allow inappropriate access. It is the job of security to defend a system from external and internal attacks.

1.10 Kernel data Structures

1.10.1 Lists, Stacks and Queues

An array is a simple data structure in which each element can be accessed directly.
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 a doubly linked list, a given item can refer either to its predecessor or to its successor
- In a circularly linked list, the last element in the list refers to the first element, rather than to null.

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.

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.

1.10.2 Trees

A tree is a data structure that can be used to represent data hierarchically.
In a general tree, a parent may have an unlimited number of children.

1.10.3 Hash Functions and Maps

A hash function takes data as its input, performs a numeric operation on this data, and return 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 accomodate this hash collision by having a linked list at the table location that contains all of the items with the same hash value.
One use of hash function is to implement a hash map, which associates pairs using a hash function.

1.10.4 Bitmaps

A bitmap is a string of n binary digits that can be used to represent the status of n items.
Bitmaps are commonly used when there is a need to represent the availability of a large number of resources. A medium-sized disk drive might be divided into several thousand individual units, called disk blocks. A bitmap can be used to indicate the availability of each disk block.

1.11 Computing Environments

1.11.1 Traditional Computing

Companies establish portals, which provide Web accessibility to their internal servers. Network computers which are essentially terminals that understand web-based computing - are used in place of traditional workstations where more security or easier maintenance is desired.

1.11.3 Distributed Systems

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.
Access to a shared resource increases computation speed, functionality, data availability, and reliability.
A network is a communication path between two or more systems. TCP/IP is the most common network protocol, and it provides the fundamental architecture of the Internet.
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.

1.11.4 Client-Server Computing

Today's systems act as server systems to satisfy requests generated by client systems
The compute-server system provides an interface to which a client can send a request to perform an action. In response, the server executes the action and sends the results to the client. The file-server system provides a file-system interface where clients can create, update, read, and delete files.

1.11.6 Virtualization

Virtualization is a technology that allows operating systems to run as applications within other operating systems.
Emulation is used when the source CPU type is different from the target CPU type.

1.13 Summary

An operating system is software that manages the computer hardware, as well as providing an environment for application programs to run. Perhaps the most visible aspect of an operating system is the interface to the computer system it provides to the human user.

For a computer to do its job of executing programs, the programs must be in main memory. Main memory is the only large storage area that the processor can access directly. It is an array of bytes, ranging in size from millions to billions. Each byte in memory has its own address.

The main memory is usually a volatile storage device that loses its contents when power is turned off or lost. Most computer systems provide secondary storage as an extension of main memory. Secondary storage provides a form of nonvolatile storage that is capable of holding large quantities of data permanently.

The wide variety of storage systems in a computer system can be organized in a hierarchy according to speed and cost. The higher levels are expensive, but they are fast. As we move down the hierarchy, the cost per bit generally decreases, whereas the access time generally increases.

There are several different strategies for designing a computer system. Single-processor systems have only one processor, while multiprocessor systems contain two or more processors that share physical memory and peripheral devices.

The most common multiprocessor design is symmetric multiprocessing (or SMP), where all processors are considered peers and run independently of one another. Clustered systems are a specialized form of multiprocessor systems and consist of multiple computer systems connected by a local-area network.

To best utilize the CPU, modern operating systems employ multiprogram- ming, which allows several jobs to be in memory at the same time, thus ensuring that the CPU always has a job to execute. Time-sharing systems are an exten- sion of multiprogramming wherein CPU scheduling algorithms rapidly switch between jobs, thus providing the illusion that each job is running concurrently.

The operating system must ensure correct operation of the computer system. To prevent user programs from interfering with the proper operation of the system, the hardware has two modes: user mode and kernel mode. Various instructions (such as I/O instructions and halt instructions) are privileged and can be executed only in kernel mode.

The memory in which the operating system resides must also be protected from modification by the user. A timer prevents infinite loops. These facilities (dual mode, privileged instructions, memory protection, and timer interrupt) are basic building blocks used by operating systems to achieve correct operation.

A process (or job) is the fundamental unit of work in an operating system. Process management includes creating and deleting processes and providing mechanisms for processes to communicate and synchronize with each other.

An operating system manages memory by keeping track of what parts of memory are being used and by whom. The operating system is also responsible for dynamically allocating and freeing memory space. Storage space is also managed by the operating system; this includes providing file systems for representing files and directories and managing space on mass-storage devices.

Operating systems must also be concerned with protecting and securing the operating system and users. Protection measures control the access of processes or users to the resources made available by the computer system. Security measures are responsible for defending a computer system from external or internal attacks.

Several data structures that are fundamental to computer science are widely used in operating systems, including lists, stacks, queues, trees, hash functions, maps, and bitmaps.

Concepts

Operating system

An **operating system** acts as an intermediary between the user of a computer and the computer hardware.
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.

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.

Kernel

A program running at all times

System programs

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.

Middleware

A set of software frameworks that provide additional services to application developers.

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

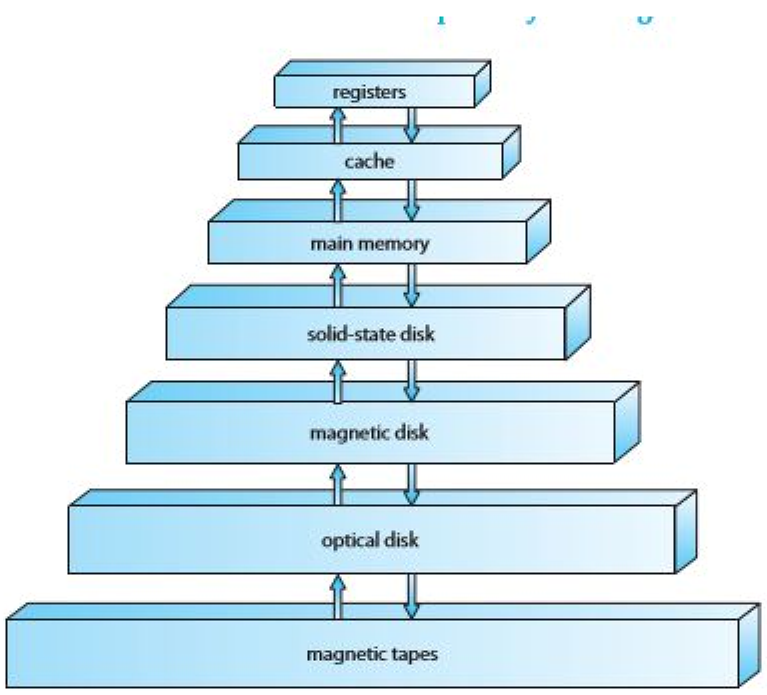


Figure 1.4 Storage-device hierarchy.

Graceful degradation

The ability to continue providing service proportional to the level of surviving hardware.

Fault Tolerant

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.
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 program loaded into memory and executing is called a **process**.

In a time-sharing system, the operating system must ensure reasonable 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 inmemory (Chapter 9). 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.