(Latest Revision:Aug 24, 2020)
[Aug 24, 2020: corrected two typographical errors]
[Jan 27, 2020: clarified language in first bullet under section 1.4.2]

Chapter One -- Introduction -- Lecture Notes

An OS performs as an intermediary between the user and the hardware, functioning as a virtual computer that makes the use of the hardware more convenient and/or efficient.

- **1.0 Chapter Goals**
    - Describe the general organization of a computer system and the role of interrupts.
    - Describe the components in a modern multiprocessor computer system.
    - Illustrate the transition from user mode to kernel mode.
    - Discuss how operating systems are used in various computing environments.
    - Provide examples of free and open-source operating systems.
- **1.1 What Operating Systems Do**
    - Roughly speaking, a computing system consists of hardware, operating system, application programs, and a user.

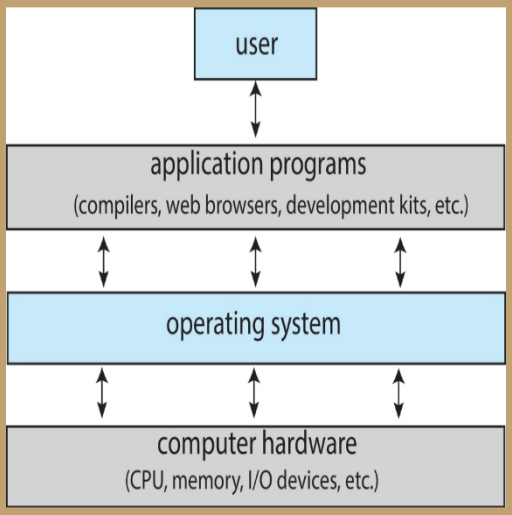

Figure 1.1: Abstract view of the components of a computer system

    - **1.1.1 User View** - For many personal devices, the main goal of the OS is to make the computer easy to use.

- **1.1.2 System View** - the computer relies on the OS to allocate, manage, and control the resources, such as CPU time, forms of memory, and I/O devices.
- **1.1.3 Defining Operating Systems**
  - Operating systems first emerged as a form of automation, taking over tasks that had previously been performed by the user, the *operator*, of the computer.
  - We may think of an OS as a collection of automatic functions that make using the hardware easier, safer, and more efficient.
  - Usually when computer science professionals refer to the operating system, they mean something called the *kernel*.
  - The study of operating systems is mostly about the study of kernels.

- **1.2 Computer-System Organization**

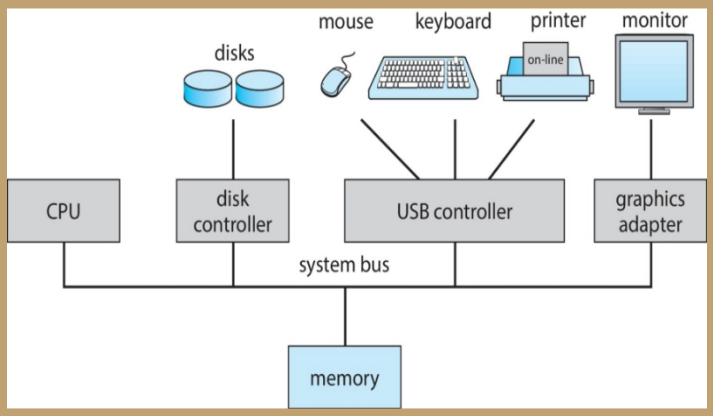

Figure 1.2: A typical PC computer system

- Figure 1.2 in the text shows the typical components of a personal computer: a bus, CPUs, primary memory, device controllers, and peripherals. An operating system is responsible for operating the peripherals by sending instructions to device controllers. Different device controllers have to be operated with their own special commands, so usually an OS has a separate section of code called a device driver, for interacting with each different kind of device controller.

- **1.2.1 Interrupts**
- The OS tells devices what to do by sending commands through the bus to registers in the device controllers, but how do the devices communicate with the OS??
- INTERRUPTS is by far the most common method.

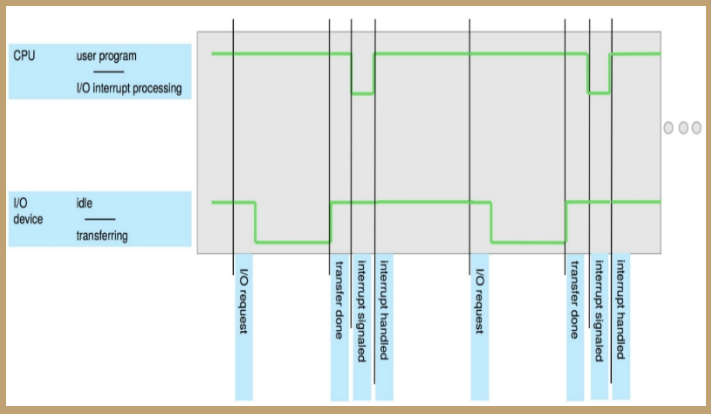

Figure 1.3: Interrupt timeline for a single program doing output

- **1.2.1.1 Overview**
  - Hardware can trigger an interrupt at any time by putting a signal on the bus that is received by the CPU.
  - For example device controllers interrupt the CPU when they need attention from the operating system.
  - When the CPU receives an interrupt, it saves some information about whatever is currently executing in the CPU, and then jumps into a part of the OS, called an interrupt handler.
  - The interrupt handler performs whatever actions are required to respond to the device.
  - The operating system is able to use saved information to resume the execution of whatever task was interrupted.
- **1.2.1.2 Implementation**
  - After executing each instruction, the CPU senses the part of the bus that carries interrupt signals, the interrupt-request line.
  - The details of the interrupt architecture may be quite complex, employing such features as an interrupt vector, interrupt chaining, interrupt priority levels, interrupt preemption, and interrupt masking.

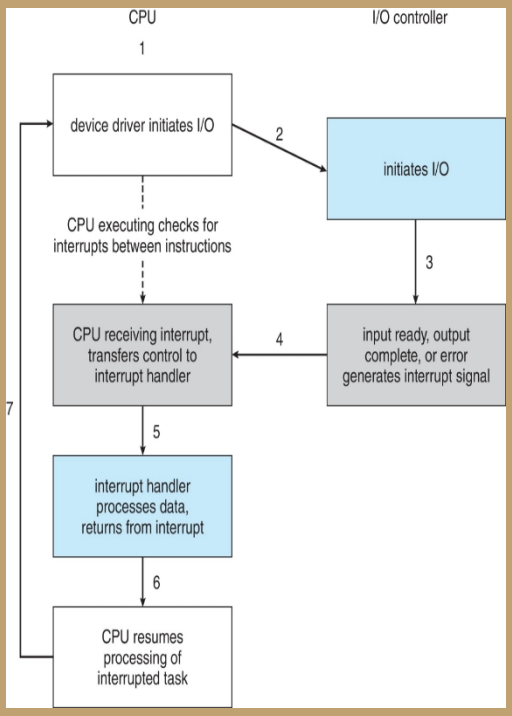

Figure 1.4: Interrupt-driven I/O cycle

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

Figure 1.5: Intel processor event-vector table

- **1.2.2 Storage Structure**
  - It's important to know, and to bear in mind, that computers can only fetch items from a few locations, like registers, main memory & its caches, and certain types of read-only memory (ROM). Therefore a computing system must copy data and instructions from secondary memory to main memory before the CPU can directly access those items.
  - Modern computers utilize storage hierarchies, in which each level of memory functions as a back-up for the levels above it, and as a cache for the levels below it.
  - The forms of memory in the upper levels are typically fast, but expensive and volatile.
  - Non-volatile memory is required for cost-effective permanent storage of large amounts of programs and data.
  - The management of the various forms of memory is a major part of what operating systems do. Chapter 9-15 of the text book all deal, in whole or in part, with memory management issues.

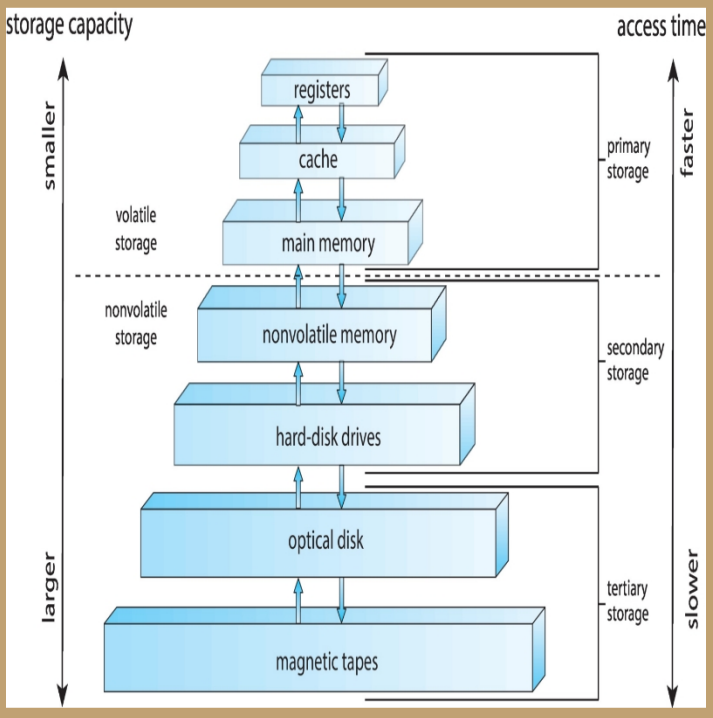

Figure 1.6: Storage-device hierarchy

- **1.2.3 I/O Structure**
  - Some forms of I/O require that data be moved one byte at a time between the device controller and the main memory. When doing I/O this way, the controller has to interrupt the CPU after transferring every single byte. That's OK for devices that are inherently slow, like keyboards.
  - For devices like hard drives with the ability to transfer data at high speed, computing systems utilize direct memory access (DMA). Only one

interrupt per block transfer is required with DMA. Software (usually the OS) gives the I/O controller the base location and size of a large array of bytes in main memory. The controller then transfers an entire block (multiple bytes) between the memory array and the controller's internal buffer. The controller interrupts the CPU only once, after the entire block has been transferred.

- With DMA, the CPU is not interrupted as often, and so there's more opportunity to use the CPU for other things during the time that the I/O transfer is being performed. That helps the system get more work done sooner.

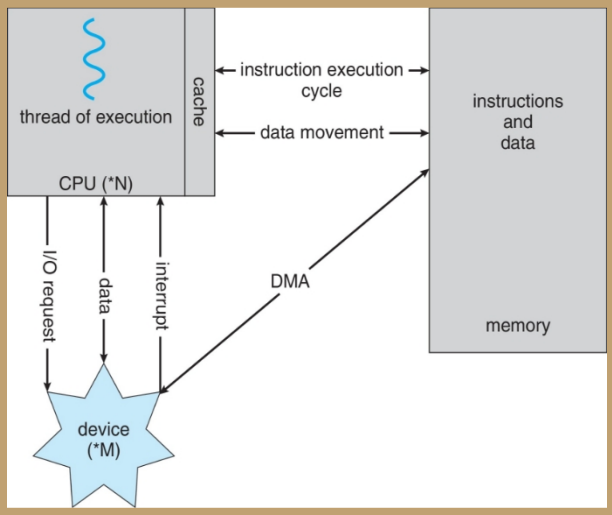

Figure 1.7: How a modern computer system works

- **1.3 Computer-System Architecture**
  - **1.3.1 Single-Processor Systems**
    - A single-processor computing system (aka a uni-processor) is a computer with only one CPU core, only one component capable of executing a general-purpose instruction set.
    - Modern computing systems may contain processors that are NOT general-purpose. For example, a disk-controller might have a special-purpose microprocessor that queues and schedules requests for reading from and writing to the disk.
    - Even if it has such special-purpose processors, a system with only one core is still a single-processor system.
  - **1.3.2 Multiprocessor Systems**
    - What is a multiprocessor? The term refers to systems that have more than one CPU core. Even if a computer has only one CPU chip, it will usually be considered a multiprocessor if that chip contains more than one (general-purpose) core.

- Most multiprocessors nowadays are symmetric multiprocessors (SMPs). All the cores are equal peers. Each is made available to perform all the same kinds of work as the others, including operating system functions and the execution of user applications.

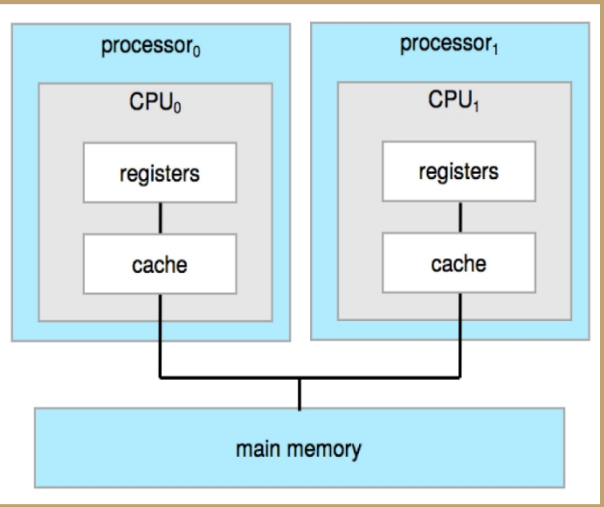

Figure 1.8: Symmetric multiprocessing architecture

- Multiprocessors can run more than one sequence of instructions simultaneously. More work can be done faster than on a uni-processor.
- It's a challenge for operating systems to derive the maximum benefits from a multiprocessor, to keep all the cores doing useful work as often as possible.
- Multiprocessors can be more or less *tightly coupled* depending on how much of the hardware resources (clock, cache, memory, component connections) they share in common.

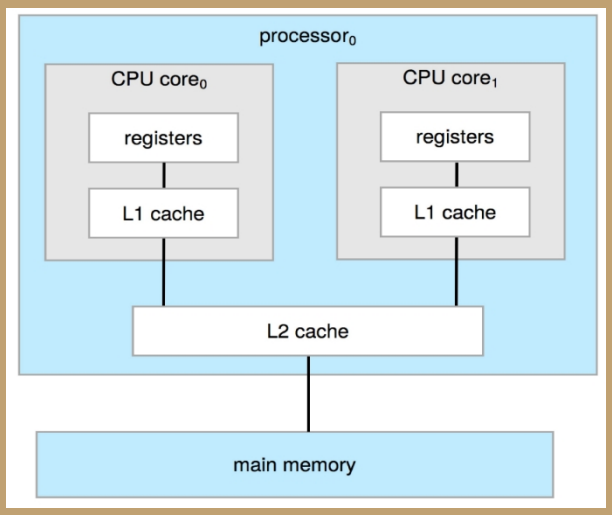

Figure 1.9: A dual-core design with two cores on the same chip

- Non-uniform memory access (NUMA) architectures tend to be easier to scale up with very large numbers of cores, and so they are of increasing importance.

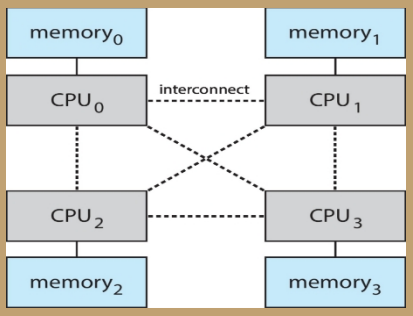

Figure 1.10: NUMA multiprocessing architecture

o **1.3.3 Clustered Systems**

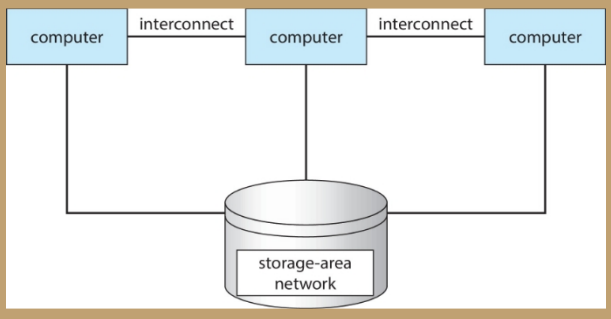

Figure 1.11: General structure of a clustered system

- A clustered system is a loosely-coupled form of multiprocessing.
- It is a group of individual computers that communicate closely using a LAN or other high-speed interconnect.
- A cluster usually has the job of providing a *high-availability service*. Typically each of the machines is monitored by one of the others, and if one machine has a failure, another machine can take over for it.

- **1.4 Operating-System Operations**
  - Usually computer hardware is designed to automatically run a program in the ROM when the machine is turned on. This is the bootstrap program. Some typical things it does are:
    - Initialize the hardware,
    - Find the operating system on disk or on a network,
    - Copy the OS to the primary memory, and
    - Execute the OS
  - Many operating systems, at boot time, will start up a group of continuously running system programs, often called daemons, to provide services.
  - An operating system that has nothing to do will sleep or wait until a request for service arrives (an interrupt).
  - Interrupts can be generated by hardware or software. A software interrupt is often called a trap or exception.
  - **1.4.1 Multiprogramming and Multitasking**

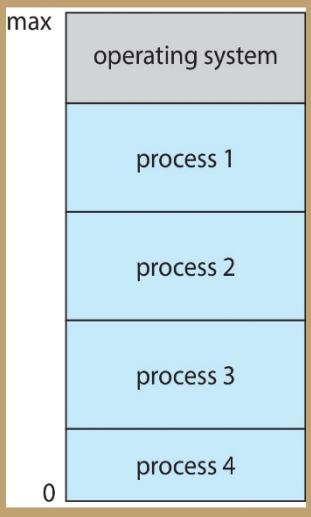

Figure 1.12: Memory layout for a multiprogramming system

- A process is an executing program.

- Most processes need to pause very often to wait for something to happen. Usually they're waiting for I/O to complete, which often takes a huge amount of time, relative to the speed of the CPU.
- How can the OS keep the CPUs busy when processes keep pausing all the time?? Multiprogramming is the answer. Keep a large number of processes in main memory. When a process in a CPU pauses, choose one of the other process to run in the CPU.
- There's a form of multiprogramming called multitasking. The OS switches processes in and out of CPUs so rapidly that a user interacting with a process does not normally notice the delay when the process is not executing.
- Multiprogramming and multitasking are very desirable operating system features, but the OS has to be very complex and sophisticated to make it work.
- Challenges include memory management, memory protection, CPU scheduling, process synchronization, and process communication.
- Also interactive multitasking systems require an efficient online file system, which requires many sophisticated data structures and secondary memory management techniques.

o **1.4.2 Dual-Mode and Multimode Operation**

- In order to assure that the hardware is operated properly, usually only the operating system should be allowed to execute certain commands, for example the commands that perform I/O, or commands that allow one process to change the contents of the main memory of a different process.
- How is it possible to enforce such rules? The answer is to have more than one mode of execution.
- It is only possible to make this work by building it into the hardware of the computer.
- A simple way to do it is to have two modes of execution: user mode and kernel mode (kernel mode may also called system mode, privileged mode, or supervisor mode). The hardware is built with a special mode bit. The mode bit functions as if it was an additional bit added to the code for each machine language instruction. There are privileged instructions that behave differently depending on whether the mode bit is 0 (kernel mode) or 1 (user mode).
- As an example, suppose a process tries to execute an instruction that performs disk I/O. That is a privileged instruction. If the value of the mode bit is 0, then the disk I/O happens. However, if the mode bit is 1, then something completely different happens: the hardware sends a trap to the CPU. Next, the interrupt mechanism causes the operating system to start executing, which gives the OS the opportunity to deal as it wishes with the user process that attempted to execute the privileged instruction.

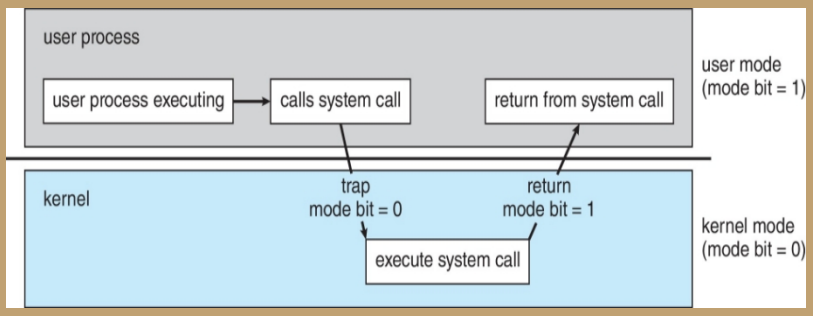

Figure 1.13: Transition from user to kernel mode

- The mode bit is supposed to be 0 whenever the OS is executing, and it's supposed to be 1 whenever a user process is executing. How is this assured? Well, the hardware automatically sets the mode bit to 0 when someone turns on the computer. Therefore the booting system always starts in kernel mode. The operating system always changes the mode bit to 1 just before executing a user process. Therefore user processes always start up (or resume) in user mode. Whenever there is an interrupt, the hardware automatically sets the value of the mode bit to 0. Therefore the OS always resumes executing in kernel mode. Finally changing the value of the mode bit is a privileged instruction, so a user process is not able to change it from 1 to 0.
- Processes need to make requests for the services of the OS. For example, a processes may request the OS to perform I/O, or to allocate some memory for a data structure. These requests are *system calls*. Processes typically make system calls by executing an instruction that causes a trap. The routine that responds to the trap then performs the service requested by the process.

1.4.3 Timer

- A timer is a special CPU register. Each time it executes a user process in a CPU, the OS first places a number N in the timer. After each clock tick, the hardware automatically decrements the timer. If the value reaches zero, the hardware immediately generates an interrupt. This assures that the user process will allow the OS to resume execution in the CPU after no more than N clock ticks.
- The user process cannot disable or change the value in the timer. Such operations are privileged instructions.

1.5 Resource Management

1.5.1 Process Management

The OS has to

- create and delete processes,
- schedule processes (and threads) on CPUs,
- suspend and resume processes,
- provide mechanisms for process synchronization, and

- provide mechanisms for process communication.
- Chapters 3-7 discuss these things.

- **1.5.2 Memory Management** The OS has to
  - keep track of which parts of memory are currently being used, and which processes are using them;
  - allocate and deallocate memory space as needed; and
  - decide which processes (or parts of processes) and data to copy into or out of memory.
  - Chapters 9-10 discuss these things.
- **1.5.3 File-System Management** The OS has to
  - create and delete files,
  - create and delete directories (folders),
  - support primitives for manipulating files and directories,
  - map files onto mass storage, and
  - back up files on stable (nonvolatile) storage media.
  - Chapters 13-15 discuss these things.
- **1.5.4 Mass Storage Management** Regarding mass storage, the OS has to
  - mount and unmount devices,
  - manage free space,
  - allocate storage,
  - schedule disk accesses,
  - perform partitioning, and
  - provide protection.
  - Chapter 11 discusses these things.
- **1.5.5 Cache Management**

Level	1	2	3	4	5
Name	registers	cache	main memory	solid-state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25-0.5	0.5-25	80-250	25,000-50,000	5,000,000
Bandwidth (MB/sec)	20,000-100,000	5,000-10,000	1,000-5,000	500	20-150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape

Figure 1.14: Characteristics of various types of storage

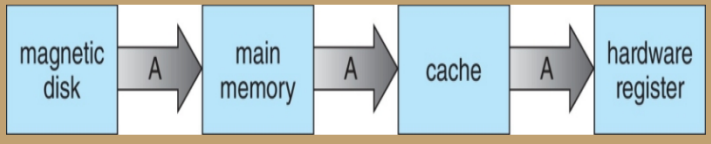

Figure 1.15: Migration of integer A from disk to register

        - Some of the caching that goes on in computers is controlled by the operating system, and some is not.
        - Chapter 10 covers some cache-replacement algorithms.
    - **1.5.6 I/O System Management**
        - Chapter 12 discusses the I/O subsystem, including such things as memory management, a general device-driver interface, and drivers for specific devices.

- **1.6 Security and Protection**

    - Modern computing systems require security and protection.
    - The system grants only an authorized process access to a resource.
    - The OS must surveil the system and ensure that resources are utilized safely and correctly.
    - Chapter 16 discusses security.
    - Chapter 17 discusses protection.
    - To maintain protection and security, the operating system must manage certain kinds of data, such as user ID numbers, group ID numbers, and passwords.

- **1.7 Virtualization**

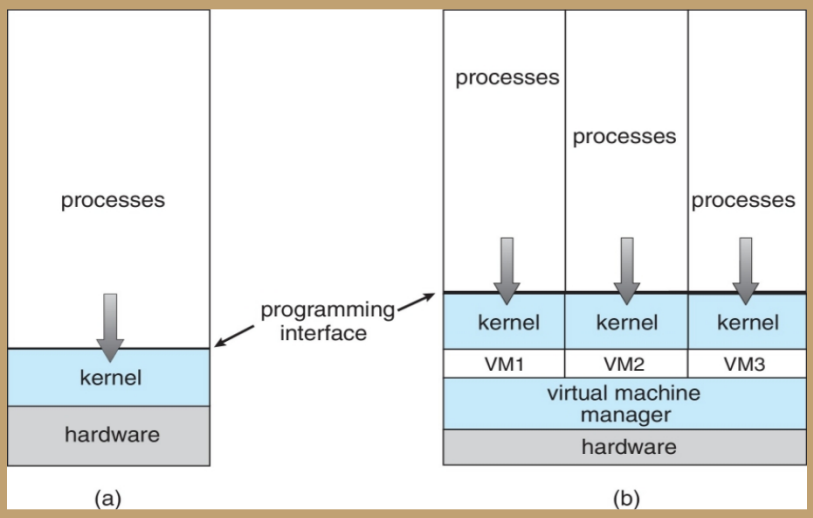

Figure 1.16: A computer running (a) a single operating system and (b) three virtual machines

    - Virtualization is a technology that enables one computer to function simultaneously as several different computers, each with its own operating system.
    - Under virtualization several guest operating systems can run 'on top of' a host operating system, and all of these systems use the computer hardware natively.

- A person might use a virtualized operating systems to run an applications that is not available on the host operating system.
- If someone develops versions of a new application to execute on several different operating systems, virtualization could allow the developer to test and debug all the version on one computer.
- There is a Linux virtual machine provided with the text.
- Chapter 18 discusses virtualization.

1.8 Distributed Systems

- A distributed system is a group of computers that are connected by a network in a way that allows users to get services from all the individual computers.
- Some distributed systems are very *transparent*, which means that to use the system you only have to be aware of the one machine in front of you. You don't have to be aware of the other machines, or the network.
- Chapter 19 has information about distributed systems.

1.9 Kernel Data Structures

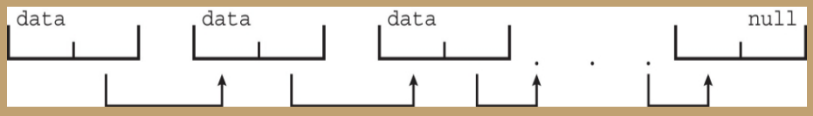

Figure 1.17: Singly linked list

1.9.1 Lists, Stacks, and Queues

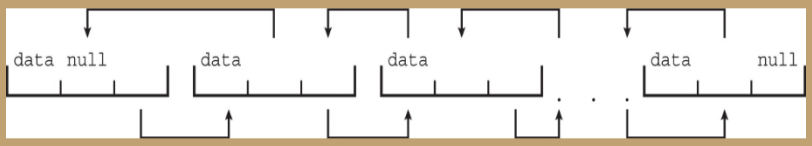

Figure 1.18: Doubly linked list

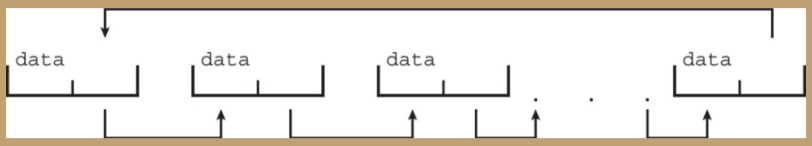

Figure 1.19: Circularly linked list

1.9.2 Trees

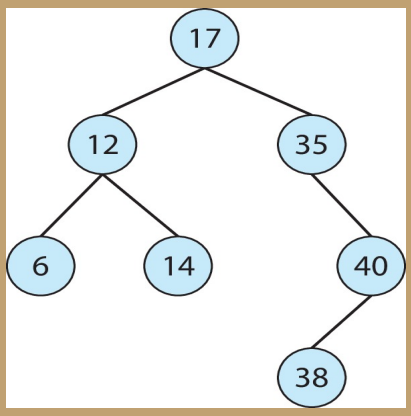

Figure 1.20: Binary search tree

- **1.9.3 Hash Functions and Maps**
- **1.9.4 Bitmaps**

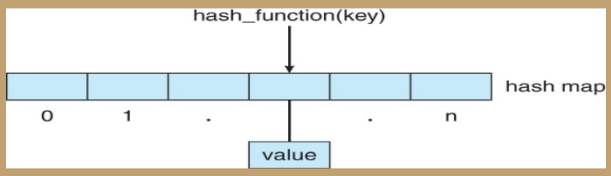

Figure 1.21: Hash map

- A bitmap is just a string of bits used to keep track of resource availability. To check on the i-th resource, examine the i-th bit. If it's 0, the resource is available. If it's 1, the resource is not available.

- **1.10 Computing Environments**
  - **1.10.1 Traditional Computing**
    - Earlier in the history of computing, a common goal of the technology was to help large numbers of people share a very small number of computers, which was necessary for affordability.
    - In the current era, it's common for each person to have exclusive access to multiple advanced devices. Convenience is now often a higher priority than efficiency. Networks facilitate communication, as well as the sharing of resources.
  - **1.10.2 Mobile Computing**
    - A mobile computer is one that is small and light enough so you can walk around with it all the time - like a smart phone or a tablet computer.
    - Nowadays mobile devices function as networked computers, cameras, phones, navigational aids, they play and record music and video, and more.

- However, mobile computers are under-powered compared with a typical laptop computer, and their memory capacities, and processing speeds are significantly lower.
- Chapter 2 has some discussion of Apple iOS and Google Android, which are two leading operating systems used on mobile devices.

1.10.3 Client-Server Computing

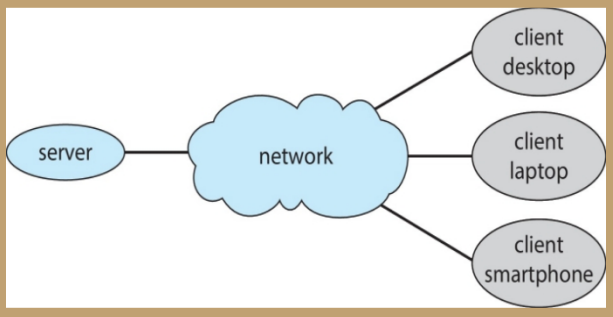

Figure 1.22: General structure of a client-server system

- In computer networks, programs called *clients* connect to programs called *servers* and the clients ask the servers to do something for them. This is how a lot of things get done on computer networks.
- For example, a client on a PC can ask a server on a super computer to do a calculation that would take too long if it was done on the PC. That is an example where the service is computation.
- Networks also use the client-server paradigm for file service. A client running on a machine with a small amount of space for files can keep most of its files on a remote machine and get help from a server on the remote machine whenever it needs to read from, or write to, one of the files.

1.10.4 Peer-to-Peer Computing

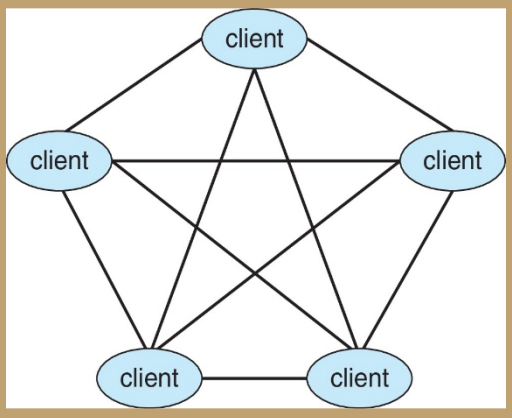

Figure 1.23: Peer-to-peer system with no centralized service

- If you start with client-server computing, but let clients be servers sometimes, and vice-versa, then you get the idea of peer-to-peer computing.
- Successful applications include the file-sharing technologies of Napster and Gnutella, as well as the voice-over-IP technology of Skype.
- **1.10.5 Cloud Computing**

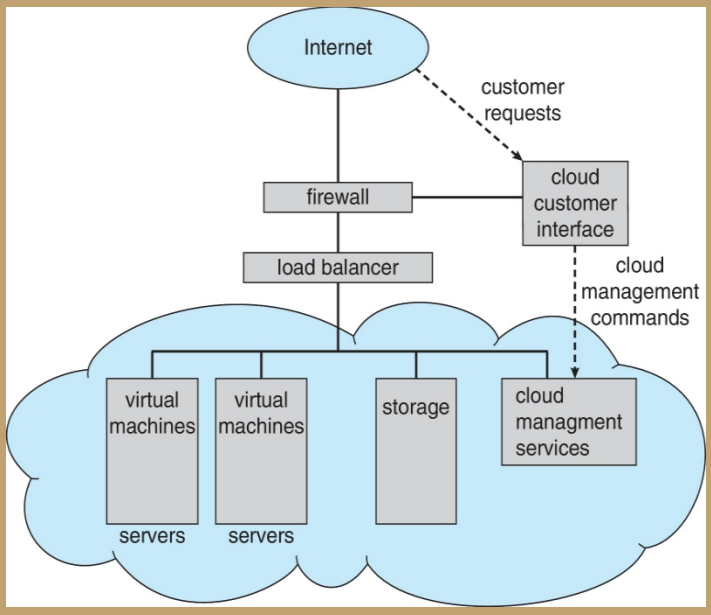

Figure 1.24: Cloud computing

  - Cloud computing might be considered a form of client-server computing.
  - Often the facilities where the servers execute have very large numbers of processors, and amounts of secondary storage.
  - Clients pay by the month just for what they use.
  - Servers can run software for clients, which is referred to as *software as a service* (SaaS). The software can be something sophisticated, or quite common, like a word processor.
  - There is also *platform as a service* (PaaS). A database server would be an example.
  - There is *infrastructure as a service* (IaaS). Paying for the ability to keep your files backed up on the server machines in the cloud would be an example of IaaS.
- **1.10.6 Real-Time Embedded Systems**
  - Most computers are actually embedded in devices like car engines, factory robots, microwave ovens, and computer peripherals.
  - Many embedded computers don't have user interfaces.
  - Many don't have operating systems.
  - When they do have an OS it's usually a real-time OS.

        - A real-time system has to deal with time constraints imposed by the (real) external environment, which means starting and stopping activities at optimal times, as time is measured in the outside world.
        - Issues regarding real time systems are discussed in chapter 5 and chapter 20.

- **1.11 Free and Open Source Operating Systems**

    - When software is open-source, it means that anyone is allowed to obtain a copy of the source code. Free software is open-source software to which additional rights are added, such as low or no cost, permission to redistribute, possibly after modification.
    - Often communities of users and developers associated with open source and free software spring up. They help debug and make useful modifications to the software.
    - Reading and experimenting with free and open source operating systems can be a good way to learn about operating systems.
    - **1.11.1 History**
        - In about the period 1950-1975, free and open source software was very common.
        - By 1980, it was much less common. Companies had decided to try to protect their profits by drastically limiting access to source code, and giving most customers just compiled code.
    - **1.11.2 Free Operating Systems**
        - In the mid 1980s, notably led by Richard Stallman, the idea was put forward that software should be "free," not necessarily in the sense of "the price is 0," but in the sense of the word "freedom."
        - The four principles of this kind of free software are that users are entitled to:
            1. freely run the program,
            2. study and change the source code, and give or sell copies either
            3. with, or
            4. without changes.
        - Stallman published a GNU Manifesto that argued all software should be "free." He also formed a foundation called the Free Software Foundation (FSF).
        - Nowadays "free" software can be routinely released through the GNU General Public License (GPL).
    - **1.11.3 GNU/Linux**
        - GNU/Linux is a free operating system released under the (GPL).
        - The development of the software was greatly influenced by Richard Stallman and Linus Torvalds. However there were many other contributors.
        - There are references to Linux throughout the text, and chapter 20 is about Linux.
    - **1.11.4 BSD UNIX**

- BSD Unix - The flavor of unix developed at UC Berkeley - was tied to AT&T unix for some time & required a license, but now is free of AT&T code, and is open source.
- There are many descendants of BSD Unix, including the Darwin kernel component code used in the Mac OS X.

o **1.11.5 Solaris**

- Solaris - originally the OS of Sun Microsystems was based on Berkeley unix. Sun migrated to the AT&T based Solaris in the 1990s, and eventually open-sourced most of the code it developed.

o **1.11. 6 Open-Source Systems as Learning Tools**

- Open-sourcing is encouraging lots of creative projects.
- Computer enthusiasts of all levels can participate in the study and development of shared software.
