MODULE 1. INTRODUCTION: COMPUTER & OPERATING SYSTEMS

1.1 BASIC ELEMENTS

A computer consists of processor, memory and I/O components with one or more modules of each type. These components are interconnected to ease job of computer. Thus, there are four structural elements:

- **Processor:** It controls the operation of the computer and performs its data processing functions. When there is only one processor, it is called as Central Processing Unit (CPU).
- **Main Memory:** It stores data and programs. But, the contents of memory are lost when the computer shuts down. Whereas, the contents of disk memory are retained.
- **I/O Modules:** Move data between the computer and its external environment. The external environment consists of a variety of devices including secondary memory devices, communications equipment and terminals.
- **System Bus:** This provides communication among processors, main memory and I/O modules.

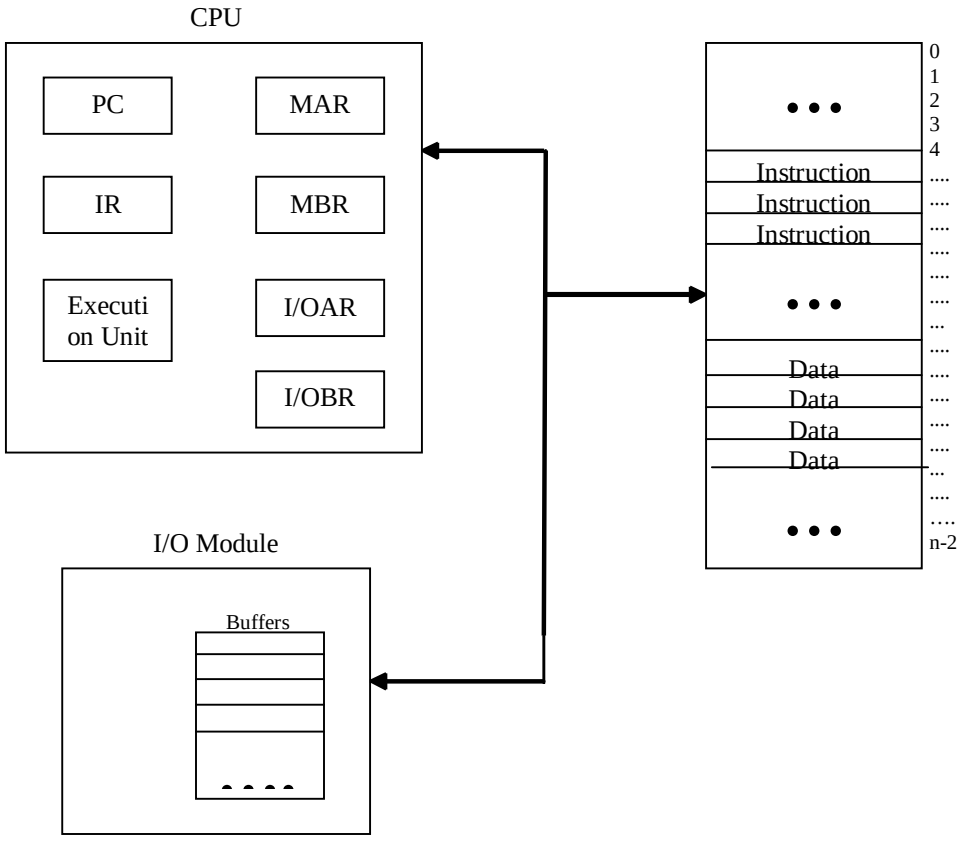


Figure 1.1 Computer Components: Top Level View

The top-level components of computer are shown in Figure 1.1. One of the functionality of a processor is to exchange data with memory. For this purpose, the following registers are used:

- **Memory Address Register (MAR):** specifies the address in memory for the next read or write.
- **Memory Buffer Register (MBR):** contains data to be written into memory or it receives the data read from memory.
- **I/O Address Register (I/OAR):** indicates particular I/O device
- **I/O Buffer Register:** used to exchange data between an I/O module and the processor.

A memory module contains a set of locations – which are sequentially numbered addresses. Each location contains a bit pattern that can be interpreted as an instruction or data. An I/O module transfers data from external devices to processor and memory and vice versa. It contains temporary buffers for holding data till they are sent.

1.2 PROCESSOR REGISTERS

A processor includes a set of registers that provide memory. But, this memory is faster and smaller than the main memory. These registers can be segregated into two types based on their functionalities as discussed in the following sections.

1.2.1 User – visible Registers

These registers enable the assembly language programmer to minimize the main memory references by optimizing register use. Higher level languages have an optimizing compiler which will make a choice between registers and main memory to store variables. Some languages like C allow the programmers to decide which variable has to be stored in register.

A user visible register is generally available to all programs. Types of registers that are available are: data, address and condition code registers.

- **Data Registers:** They can be assigned to different types of functions by the programmer. Sometimes, these are general purpose and can be used with any machine instruction that performs operations on data. Still, there are some restrictions like – few registers are used for floating-point operations and few are only for integers.

- **Address Registers:** These registers contain main memory addresses of data and instructions. They may be of general purpose or may be used for a particular way of addressing memory. Few examples are as given below:
  - **Index Registers:** Indexed addressing is a common mode of addressing which involves adding and index to a base value to get the effective address.
  - **Segment Pointer:** In segmented addressing, a memory is divided into segments (a variable-length block of words). In this mode of addressing, a register is used to hold the base address of the segment.
  - **Stack Pointer:** If there is a user-visible stack addressing, then there is a register pointing to the top of the stack. This allows push and pop operations on instructions stored in the stack.

1.2.2 Control and Status Registers

The registers used by the processor to control its operation are called as Control and Status registers. This registers are also used for controlling the execution of programs. Most of such registers are not visible to the user. Along with MAR, MBR, I/OAR and I/OBR discussed earlier, following registers are also needed for an instruction to execute:

- **Program Counter:** that contains the address of next instruction to be fetched.
- **Instruction Register(IR):** contains the instruction most recently fetched.

All processor designs also include a register or set of registers, known as ***program status word (PSW)***. It contains condition codes and status information like interrupt enable/disable bit and kernel/user mode bit.

Condition codes (also known as ***flags***) are bits set by the processor hardware as the result of operations. For example, an arithmetic operation may produce a positive, negative, zero or overflow result. Condition code bits are collected into one or more registers. And, they are the part of a control register. These bits only can be read to know the feedback of the instruction execution, but they can't be altered.

1.3 INSTRUCTION EXECUTION

A program to be executed contains a set of instructions stored in the memory. The processor takes two steps for processing an instruction:

- Read(or fetch) instructions from the memory one at a time
- Execute each instruction

These two steps are referred as ***fetch stage*** and ***execute stage*** respectively. One instruction requires both of these steps for its execution and such a processing is called as ***instruction cycle*** as shown in Figure 1.2. The program halts its execution only if the processor is turned off, some error occurs or there is an instruction in the program to terminate.

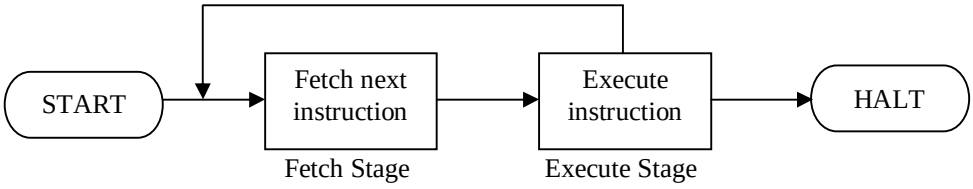


Figure 1.2 Basic Instruction Cycle

1.3.1 Instruction Fetch and Execute

At the beginning of every instruction cycle, the processor fetches an instruction from the memory. The program counter (PC) holds the address of next instruction to be fetched. And, the PC will be incremented whenever the processor fetches the instruction. For example, assume that the current value of PC is 300. When the processor fetches next instruction, PC will be incremented to 301. However, this logic may change in case of possible conditional statements of the program.

Later, the fetched instruction is loaded into the instruction register (IR). The instruction contains bits, and these bits inform the processor about the action to be taken. The

processor understands these bits and performs the required action. Generally, this entire process of instruction execution is segregated into four categories as given below:

- **Processor-memory:** Data may be transferred from processor to memory and vice-versa.
- **Processor-I/O:** Data may be transferred to/from a peripheral device by transferring between the processor and I/O module.
- **Data Processing:** Processor may perform arithmetic/logical operations on data.
- **Control:** An instruction may specify that the sequence of execution be altered. For example, the processor may fetch the instruction from the location 149, which indicates the next address to be fetched should be 182. So, now the program counter is set to 182, instead of 150.

With the help of example, we will discuss these points in details now. Consider the processor with characteristics as shown in Figure 1.3.

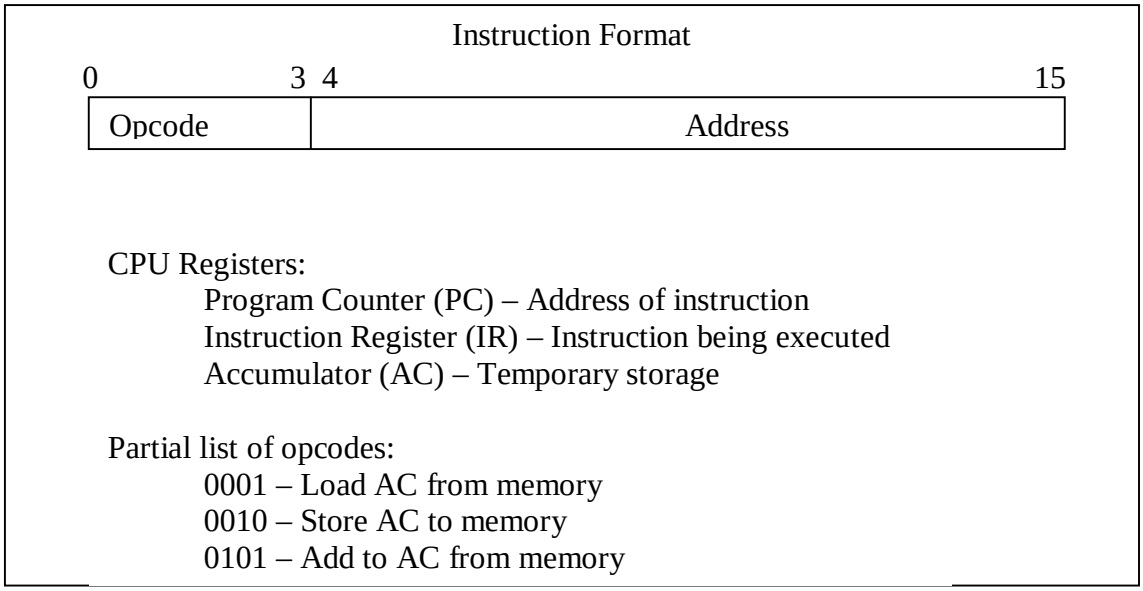


Figure 1.3 Characteristics of a processor

The processor contains single data register called accumulator (AC). Both data and instructions are 16 bit long. The instruction format allows 4 bits for the opcode. So, 16 different opcodes are possible (2^4=16). It will be a single hexadecimal digit. Remaining 12 bits are used for the address in the form of 3 hexadecimal digits. Consider the illustration of program execution (just a portion) shown in Figure 1.4. The program segment is to add the contents at the address 940 and the contents at the address 941 and then storing the result in the address 941. This is something similar to a programming statement x = y+x.

For doing this job, three instruction cycles (3 fetch, 3 execute) are required as given below:

- **Fetch:** The PC contains the address of first instruction, that is 300. The instruction is 1940. Here, 1 indicates opcode and 940 is the memory address. The instruction 1940 is loaded into IR and PC is incremented to 301.

- **Execute:** Since the opcode 1 (or 0001) is for loading the AC from memory, the content of the address 940 is loaded into AC. Hence, now AC contains 0003.
- **Fetch:** Now, the next instruction (5941) is fetched from the location 301 and PC is incremented to 302.
- **Execute:** The opcode 5 (0101) indicates adding AC from memory. So, the content of the location 941 is added to the contents of AC. (0003 + 0002 =0005)
- **Fetch:** The next instruction (2941) from the address 302 is fetched and the PC is incremented to 303.

Execute: The opcode 2 (0010) indicates storing AC to memory. Hence the value 0005 is loaded into the memory address 941.

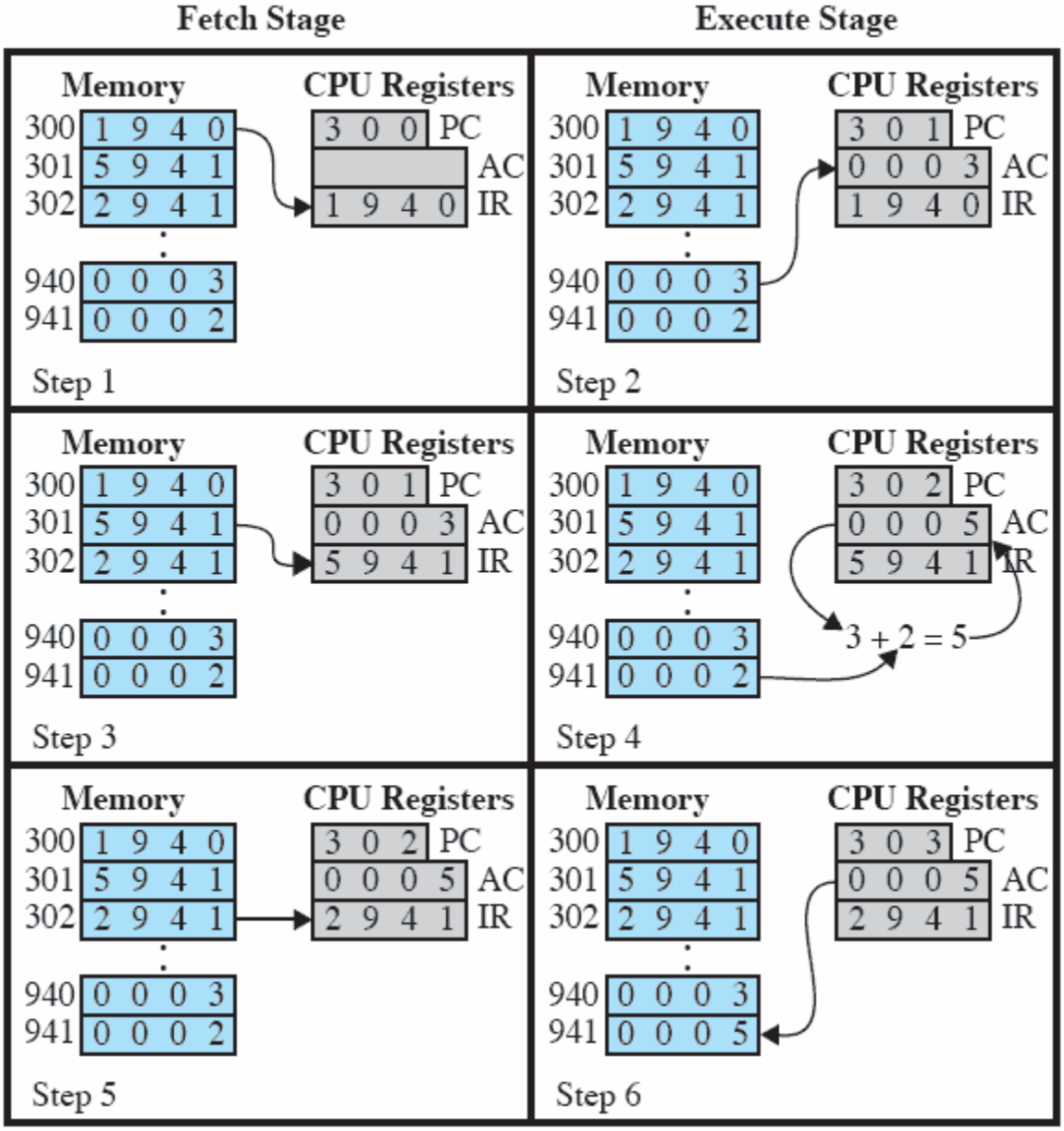


Figure 1.4 Example of program execution

1.3.2 I/O Function

Data can be exchanged directly between an I/O module and the processor. That is, the processor can directly read/write data from/to I/O module, not necessarily from the memory. Here, the processor identifies a specific device that is controlled by a particular I/O module.

In some situations, it is better to allow I/O exchanges to occur directly with main memory to relieve the processor from I/O task. In such cases, the processor should grant the authority to I/O module to read/write to memory. Hence, processor is not tied up with I/O operations. Now, I/O module will issue read/write commands directly to the memory. This operation is known as ***direct memory access (DMA)***.

1.4 THE MEMORY HIERARCHY

The constraints on the design of computer's memory depend on three key points viz.

- Capacity (how much is the size?)
- Access time (how fast it can be accessed?)
- Cost (how expensive it is?)

In general, the relationship between these three points will be as below –

- Faster access time, greater the cost per bit
- Greater capacity, smaller cost per bit
- Greater capacity, slower access speed

Thus, the designers of computer memory face the dilemma on these points. He/she has to optimize and balance the constraints.

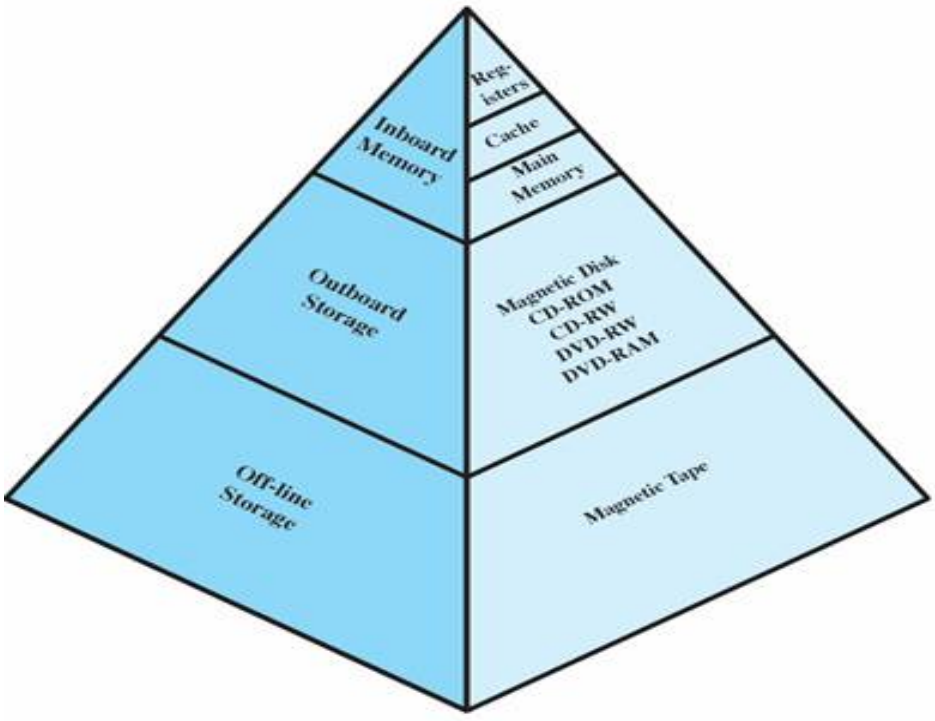


Figure 1.5 Memory Hierarchy

To solve this problem, **memory hierarchy** has to be used instead of relying on a single memory component. A typical memory hierarchy would be as given in Figure 1.5. In this hierarchy, from top to bottom, the following occur:

(i) Decreasing cost per bit
(ii) Increasing capacity
(iii) Increasing access time
(iv) Decreasing frequency of access to the memory by the processor

Hence, the smaller, expensive, faster memories are supplemented by larger, cheaper, slower memories. The levels of memory hierarchy are explained hereunder.

- **Registers:** Generally every processor contains a few dozen or hundreds of registers which are faster, smaller but expensive.
- **Cache Memory:** It is not usually visible to the programmer, but visible only to the processor. It is used for the movement of data between main memory and processor registers.
- **Main Memory:** It is the principal internal memory system of the computer. Each location in the main memory has a unique address. Most of the machine instructions refer to one or more main memory addresses. Main memory is usually extended with a higher speed, smaller cache.
- **Secondary/Auxiliary Memory:** The next two levels of the hierarchy falls into this category. The data are stored permanently on external storage devices like hard disk, removable devices like CD, DVD, tape etc. Programmer can see such data in the form of folders and files.

1.5 CACHE MEMORY

Since cache memory plays very important role in the performance of the processor and managing the memory hardware, it is being discussed in detail here.

1.5.1 Motivation

On every instruction cycle, the processor fetches the memory at least once. If the instruction contains operands or it needs to store some data, then every fetch may need to access memory more than once. Thus, the processor speed is always restricted by the memory cycle time. And, in almost all computers, the processor speed is much higher than that of memory cycle speed. Hence, an intermediate repository of instructions is thought of to keep a portion of memory that needs to be executed by the processor. Such a small and fast memory between the processor and main memory is called as cache.

1.5.2 Cache Principles

The working of cache memory is depicted in Figure 1.6. The cache size will be considerably smaller than then main memory. It contains a copy of some portion of main memory. When the processor tries to read a byte/word from the memory, the cache is checked first. If that byte is available in the cache, it is delivered to the processor. If not, a block of main memory containing that byte will be stored into the cache and then it is delivered to the processor. Figure 1.7 shows the process of read operation using cache.

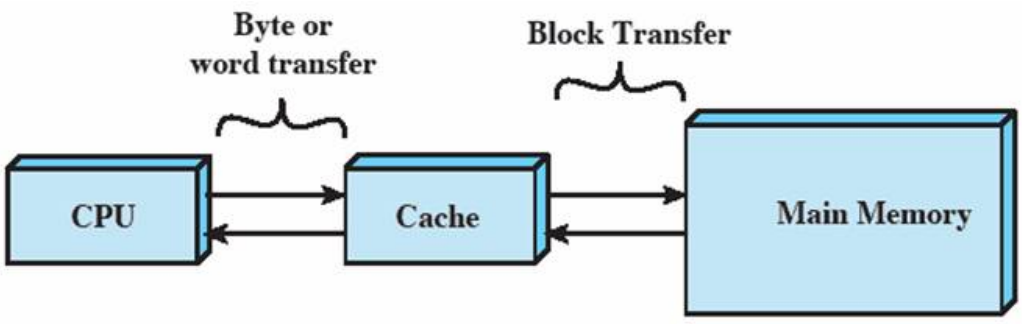


Figure 1.6 Cache and Main Memory

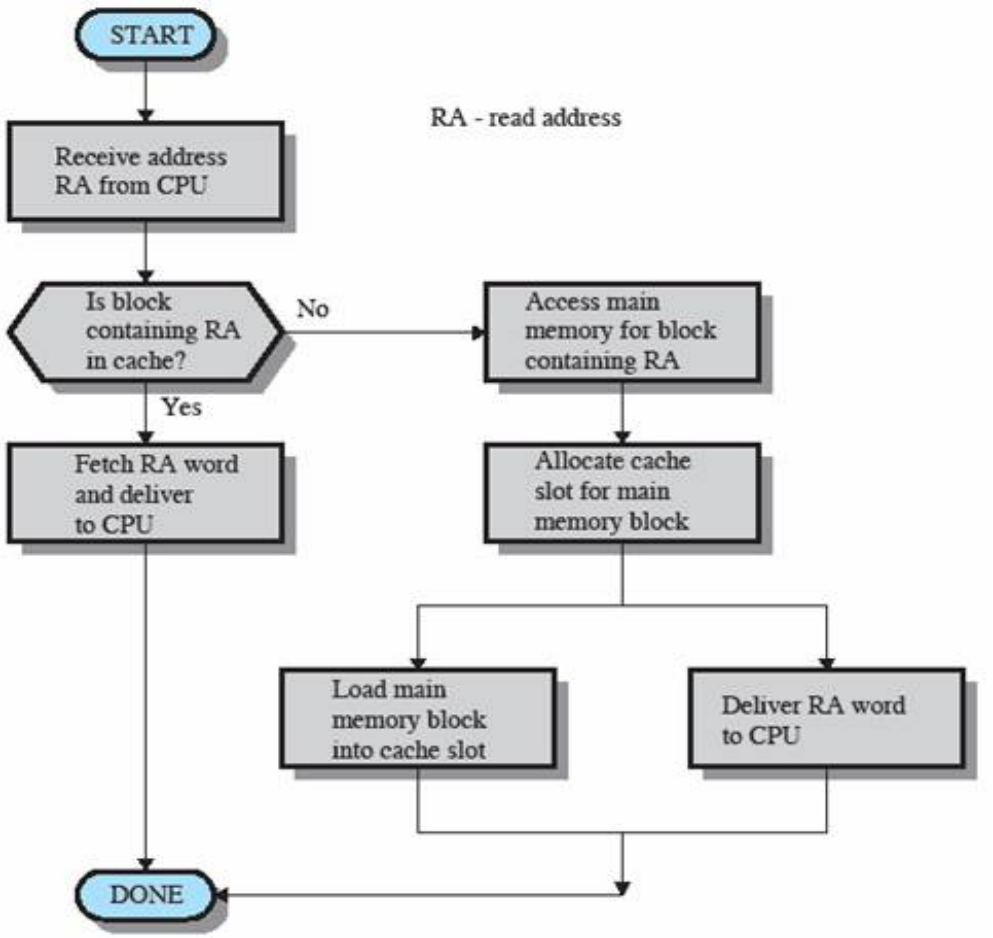


Figure 1.7 Read operation using cache

1.5.3 Cache Design

The design of a cache memory has to consider following aspects:

- **Cache size:** Small size caches have significant impact on the performance of the processor.
- **Block size:** It indicates the amount of data exchanged between cache and main memory. Optimum selection of this size is essential. Because, if the block size is too small, then it cannot hold many instructions and hence the main memory has to be

hit more number of times. Whereas, if the block size is too large, then it becomes almost like a main memory and the very usage of cache will be void.

- **Mapping function:** This function determines the location in the cache to be occupied by the block. It has two constraints: (i) while one block is read, another may need to be replaced. (ii) As mapping function becomes more flexible, the design circuitry becomes complex.
- **Replacement algorithm:** This algorithm decides which block has to be replaced when a new block is loaded into the cache. And the design of this algorithm should work within the constraints of mapping function. In most of the cases, least-recently-used (LRU) method is applied.
- **Write policy:** If the contents of the block in the cache are modified, the same has to be written inside the main memory. The write policy indicates when the memory write operation has to take place.

1.6 I/O COMMUNICATION TECHNIQUES

I/O operations are possible using following three techniques:

- Programmed I/O
- Interrupt-driven I/O
- Direct Memory Access (DMA)

Each of these techniques is explained here and the diagrammatic representation is given in Figure 1.8.

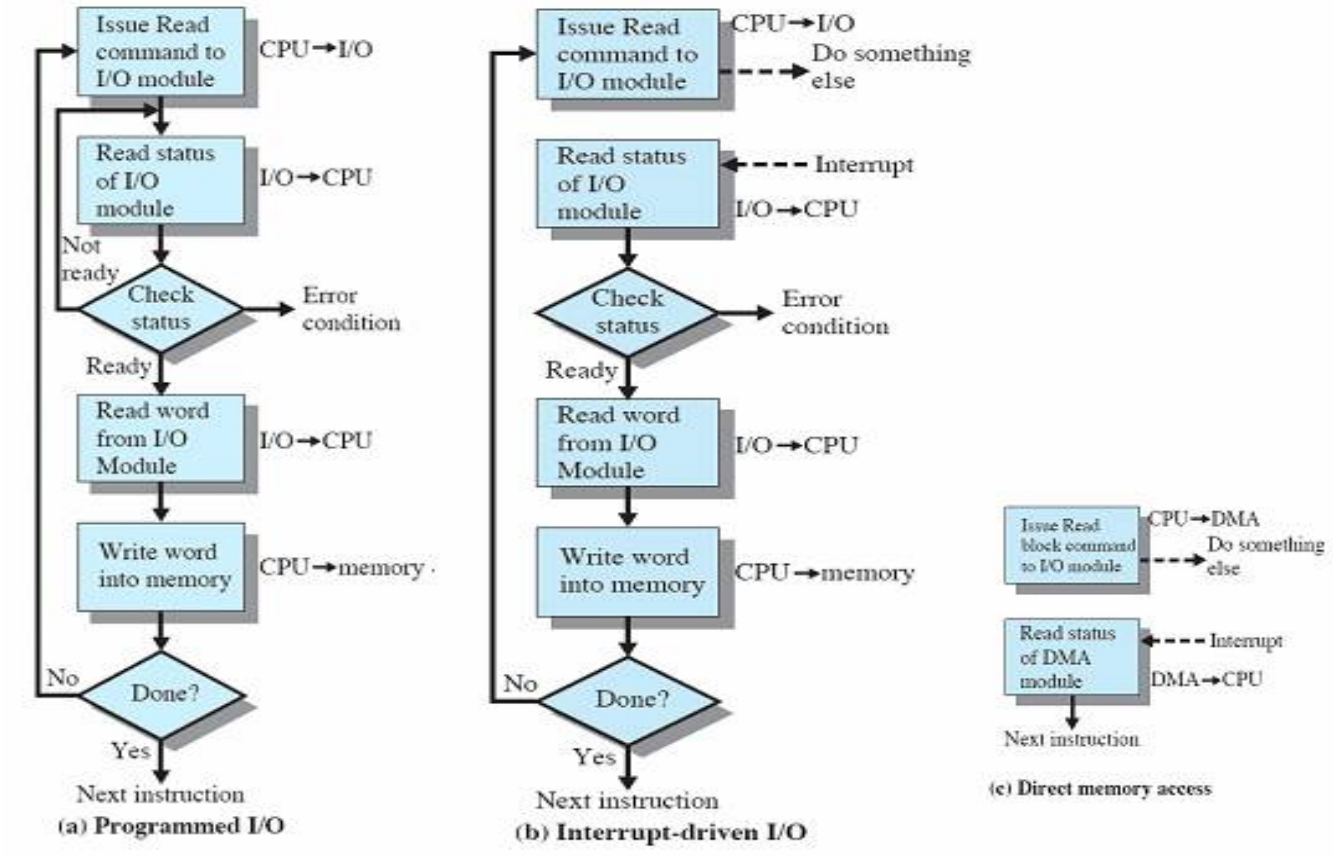


Figure 1.8 Techniques of I/O communication

1.6.1 Programmed I/O

When the processor is executing a program and encounters an I/O instruction, then it will inform I/O module and executes that instruction. In case of programmed I/O, the I/O module performs the task but do not interrupt the processor about the completion of the task. Hence, the processor must periodically keep checking the I/O module for the completion of the task.

Thus, the processor is responsible for extracting/storing data from/to the main memory. So, the instruction set includes the I/O instructions in the following categories:

- **Control:** activates an external device and informs the action to be taken.
- **Status:** used to test various status conditions associated with I/O module.
- **Transfer:** to read/write data between processor registers and external devices.

Note that, the technique of programmed I/O is time-consuming and keeps the processor busy unnecessarily.

1.6.2 Interrupt-Driven I/O

(NOTE: Concepts of Interrupts is out-of-syllabus. But for the knowledge purpose, the notes for interrupts is given at the end of this module.)

As it is observed, the programmed I/O technique will degrade the performance of the processor. An alternative way is to provide interrupt-driven I/O. In this technique, the processor will issue an I/O command to the I/O module and then continue its regular instruction execution. When the I/O module is ready, it will interrupt the processor. Then the processor will execute the requested task and then resume its former processing.

1.6.3 Direct Memory Access

Though interrupt-driven I/O is efficient than the programmed I/O, it requires active participation of the processor for transferring data between memory and I/O module. Hence, both of these techniques have following drawbacks:

- I/O transfer rate is limited
- Processor is tied up in managing I/O transfer

To avoid these problems, direct memory access (DMA) is proposed. It can be put as a separate module on the system bus or as a part of I/O module. In this technique, when the processor has read/write data, it issues a command to DMA by sending following information:

- Whether a read or write is requested
- Address of the I/O device involved
- Starting location in memory to read/write data
- Number of words to be read/written

Then, the processor continues its work. Now the job has been delegated to DMA module. The DMA module will transfer the entire data from the memory and then interrupts the processor. Thus, the processor is involved only at the beginning and ending of the data transfer.

1.7 INTRODUCTION TO OPERATING SYSTEM

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

The design of OS is depending on the purpose for which it is being used. For example,

- Mainframe OS are designed to optimize the utilization of hardware
- Personal computer OS are designed to support complex games, business applications etc.
- Handheld computer (like mobiles, tablets etc) OS provides user-friendly interface and environment to execute programs/applications.

Hence, few OS may be convenient, few may be efficient and some may be combination of both.

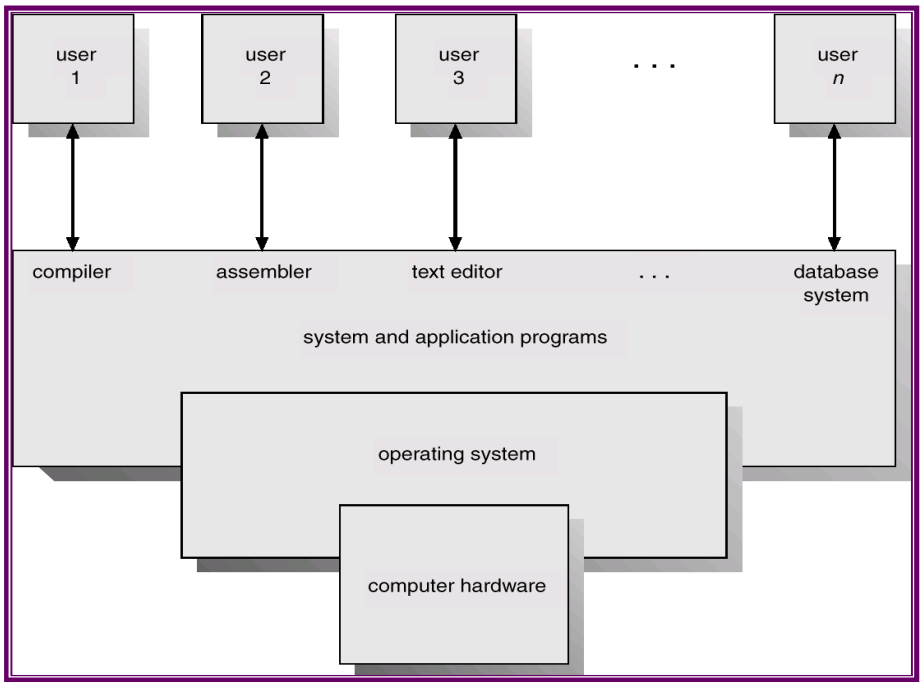


Figure 1.9 Abstract view of components of a computer system

A computer system can be divided into four components as shown in Figure 1.9. They are:

- **Hardware** : Consists of central processing unit (CPU), memory and I/O devices and provides the basic computing resources
- **Operating system:** Controls and coordinates the use of hardware among various application software for different users.
- **Application programs:** defines the ways in which the computing resources are used to solve the problems of users. For example, word processors, spreadsheets, compilers, browsers etc.
- **Users:** users of the computer.

Thus, we can say that OS is like a government. It does not perform any useful function by itself, but provides environment within which other programs can do useful works.

OS can be explored from two viewpoints and are explained below:

- User View
- System View

1.7.1 User View

The user's view of the computer varies according to the interface being used. The design on OS varies depending on type of the user or work to be carried out as explained below:

- **User of PC:** OS for personal computers is aimed at maximizing the work that the user is performing. Hence, the OS design is mostly for the ease of use rather than the performance. And, resource utilization is not taken into consideration.
- **User of Mainframes:** Some users make use of terminals connected to a mainframe or minicomputer. There will be other people accessing the same computer through other terminals. These users share the resources and information. Hence, the OS on such systems is designed to maximize the resource utilization. Thus, efficient sharing of CPU time, memory and I/O are assured for every user.
- **User of Workstations:** Sometimes, computers are connected to networks of workstations and servers. The user working on such workstations may have their own resources and they may also share the resources with other servers. Hence, OS for such situation is designed to optimize between individual usability and resource utilization.
- **User of Handheld Computers:** Handheld devices like mobiles, tablets are common nowadays. Because of power, speed and interface limitations, they may perform relatively less operations. So, their OS are designed for individual usability by keeping battery life in mind.

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

1.7.2 System View

From the computer's point of view, the OS is the program closely involved with the hardware. In this context, we can view an OS 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 OS acts as the manager of these resources by allocating the resources to the programs and users efficiently.

Another view of OS focuses on controlling various I/O devices and user programs. As a control program, OS manages the execution of user programs to prevent errors and improper use of computer.

The aim of any computer system is to execute user programs to solve user’s problems. As computer hardware alone is not easy to use, application programs/softwares have been created for solving the problems. Such programs require certain common operations like

controlling I/O devices etc. The common functions of controlling and allocating resources are brought together into one piece of software known as Operating System.

The widely accepted definition of OS goes like this: ***OS is the program running all times on the computer (also called as kernel), with all other programs being treated as application programs.***

1.7.3 System Goals

It is easier to define an operating system by what it *does* than by what it *is*. The primary goal of some operating system is *convenience for the user.* Operating systems exist because they are supposed to make it easier to compute with them than without them. This view is particularly clear when you look at operating systems for small PCs.

The primary goal of other operating systems is *efficient* operation of the computer system. This is the case for large, shared, multi-user systems. These systems are expensive, so it is desirable to make them as efficient as possible.

These two goals - convenience and efficiency-are sometimes contradictory. In the past, efficiency was often more important than convenience. Thus, much of operating-system theory concentrates on optimal use of computing resources. Operating systems have also evolved over time. For example, UNIX started with a keyboard and printer as its interface, limiting how convenient it could be for the user. Over time, hardware changed, and UNIX was ported to new hardware with more user-friendly interfaces. Many **graphic** user interfaces (GUIs) were added, allowing UNIX to be more convenient for users while still concentrating on efficiency.

The designers of OS face many tradeoffs related to efficiency and convenience. Lot of continuous revision and updation is necessary. Still, the success of OS depends on its users. In past 50 years, OS evolved into different phases. And, OS and computer architecture have influenced each other. To facilitate the use of the hardware, researchers developed operating systems. Users of the operating systems then proposed changes in hardware design to simplify them.

1.8 MAINFRAME SYSTEMS

Mainframe computer systems were the first computers used to tackle many commercial and scientific applications. In this section, we trace the growth of mainframe systems.

1.8.1 Batch Systems

Early computers were huge machines and were run from a console. The common input devices were card readers and tape drives. The common output devices were line printers, tape drives, and card punches. The user did not interact directly with the computer systems. Rather, the user prepared a job and submitted it to the computer operator. Normally, a job would consist of the program, the data, and some control information about the nature of the job (control cards). The job was usually in the form of punch cards (a sample punch card is shown in Figure 1.10). After some time (may be minutes, hours, or

days), the output appeared. The output consisted of the result of the program, as well as a dump of the final memory and register contents for debugging.

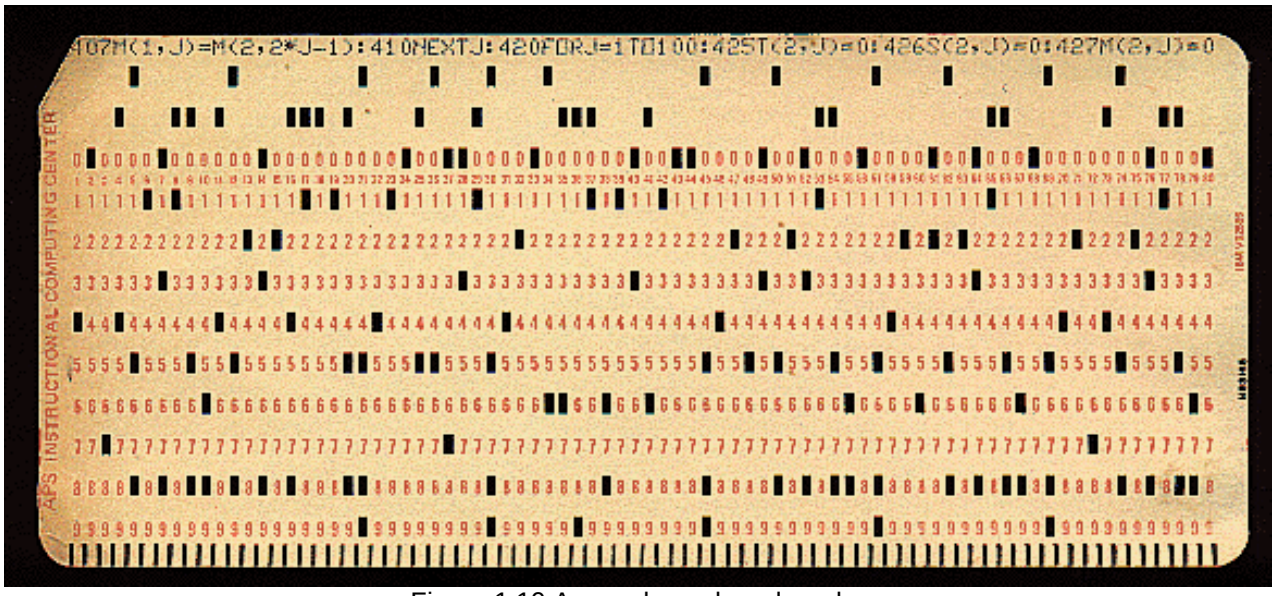


Figure 1.10 A sample puchcard reader

The operating system in these early computers was fairly simple. Its major task was to transfer control automatically from one job to the next. The operating system was always resident in memory as shown in Figure 1.11.

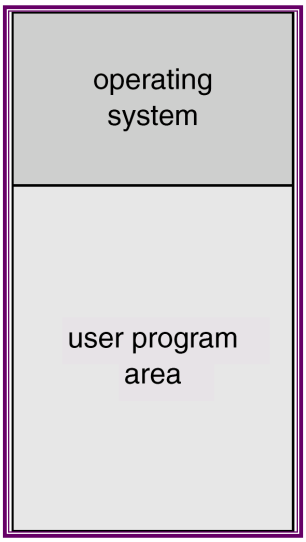


Figure 1.11 Memory layout for simple batch system

The operators batched the jobs with similar needs together and ran them through the computer as a group. Later, the output from each job would be sent back to respective programmer. This would speed up the processing.

In this execution environment, the CPU is often idle, because the speed of I/O devices is much slower than process. Over time, improvements in technology and the introduction of disks resulted in faster I/O devices. However, CPU speeds increased to an even greater extent, so the problem was not only unresolved, but became worse. The introduction of disk technology allowed the operating system to keep all jobs on a disk, rather than in a serial card reader. With direct access to several jobs, the operating system could perform job scheduling, to use resources and perform tasks efficiently. Job scheduling is discussed later in detail.

1.8.2 Multiprogrammed Systems

The most important aspect of job scheduling is the ability to multiprogram. A single user cannot keep either the CPU or the I/O devices busy at all times. Multiprogramming increases CPU utilization by organizing jobs so that the CPU always has one job to execute.

In multiprogrammed systems, the OS keeps several jobs in memory simultaneously as shown in Figure 1.12. The OS picks and begins to execute one of the jobs in the memory. Eventually, the job may have to wait for some task, such as an I/O operation, to complete. In a non-multiprogrammed system, the CPU would sit idle. In a multiprogramming system, the OS simply switches to, and executes, another job. When that job needs to wait, the CPU is switched to another job, and so on. Eventually, the first job finishes waiting and gets the CPU back. As long as at least one job needs to execute, the CPU is never idle.

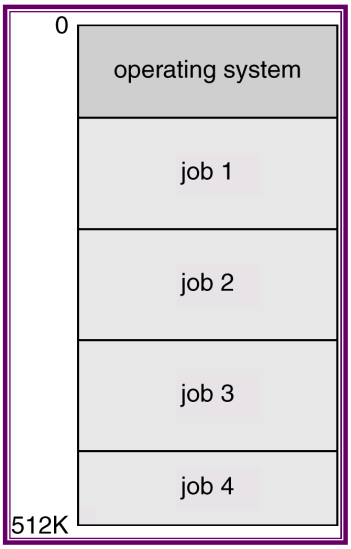


Figure 1.12 Memory layout for multiprogramming system

Multiprogramming creates the situation for OS to make certain decisions:

- There will be many jobs residing on the disk waiting for allocation of main memory for processing. The OS has to take decision of choosing one job among them (This is job scheduling).

- When the OS selects a job from the job pool, it loads that job into memory for execution. Having several programs in memory at the same time requires some form of memory management.
- In addition, if several jobs are ready to run at the same time, the system must choose among them. Making this decision is CPU scheduling.
- Finally, multiple jobs running concurrently require that their ability to affect one another be limited in all phases of the operating system, including process scheduling, disk storage, and memory management. OS has to consider all these aspects.

1.8.3 Time-sharing Systems

Multiprogrammed, batched systems provided effective use of resources, but did not provide for user interaction with the computer system. Time sharing (or multitasking) is a logical extension of multiprogramming. Here, 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. The user gives instructions to the operating system or to a program directly, using a keyboard or a mouse, and waits for immediate results. The response time should be short, typically within 1 second.

A time-shared OS allows many users to share the computer simultaneously. As the system switches rapidly from one user to the other, every user feels that the entire computer is dedicated to him/her.

A time-shared OS makes use of CPU scheduling and multiprogramming to provide a small portion of a time-shared computer to every user. Each user has at least one separate program in memory. ***A program loaded into memory and executing is commonly referred to as a process.*** A process normally executes only for a short time and waits to perform I/O. As I/O is depending on the speed of the user, which is very low compared to that of a system, the user gets enough time. Meanwhile, the OS takes up another process to execute.

In both time-sharing and multiprogrammed OS, several jobs must be kept simultaneously in memory. Hence, the OS should have memory management and protection. To obtain a reasonable response time, jobs may have to be swapped in and out of main memory to the disk. This is achieved by virtual memory, which is a technique that allows the execution of a job that may not be completely in memory. The main advantage of the virtual-memory scheme is that programs can be larger than 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.

Time-sharing systems must also provide a file system. As the file system resides on a collection of disks, the disk management must be provided by OS. Also, time-sharing systems provide a mechanism for concurrent execution, which requires sophisticated CPU-scheduling schemes. To ensure orderly execution, the system must provide mechanisms

for job synchronization and communication and it may ensure that jobs do not get stuck in a deadlock, forever waiting for one another.

1.9 DESKTOP SYSTEMS

Personal computers appeared in the 1970s. During their first decade, the CPUs in PCs lacked the features needed to protect an OS from user programs. Therefore, OS in PC were neither multiuser nor multitasking. However, the goals of these OS have changed with time. Instead of maximizing CPU and peripheral utilization, the systems opt for maximizing user convenience and responsiveness; for example, Microsoft Windows and Apple Macintosh. The MS-DOS from Microsoft has been superseded by multiple flavors of Microsoft Windows, and IBM has upgraded MS-DOS to the OS/2 multitasking system. The Apple Macintosh operating system has been ported to more advanced hardware, and now includes new features, such as virtual memory and multitasking. With the release of MacOS X, the core of OS is now based on Mach and FreeBSD UNIX for scalability, performance, and features, but it retains the same rich GUI. Linux, a UNIX-like operating system available for PCs, has also become popular recently.

OS for these computers have benefited in several ways from the development of OS for mainframes. Microcomputers were immediately able to adopt some of the technology developed for larger OS. On the other hand, the hardware costs for microcomputers are becoming less that an individual user can bear, and CPU utilization is no longer a prime concern. Thus, some of the design decisions made in OS for mainframes may not be appropriate for smaller systems.

Other design decisions still apply. For example, initially file protection was not necessary on a personal machine. However, these computers are now often tied into other computers over LAN or other Internet connections. When other computers and other users can access the files on a PC, file protection again becomes a necessary feature of the OS. The lack of such protection has made it easy for malicious programs to destroy data on systems such as MS-DOS and Macintosh. These programs may be self-replicating, and may spread rapidly via worm or virus mechanisms and disrupt entire companies or even worldwide networks. Advanced timesharing features such as protected memory and file permissions are not enough, on their own, to safeguard a system from attack. But, recent advancements in technology are safeguarding a system up to some extent.

1.10 MULTIPROCESSOR SYSTEMS

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

Multiprocessor systems have three main advantages:

- **Increased throughput:** By increasing the number of processors, we hope to get more work done in less time. The speed-up ratio with N processors is not N; rather, it is less than N. When multiple processors cooperate on a task, a certain amount of

overhead is incurred in keeping all the parts working correctly. This overhead, plus conflict for shared resources, lowers the expected gain from additional processors. Similarly, a group of N programmers working closely together does not result in N times the amount of work being accomplished.

- **Economy of scale**: Multiprocessor systems can save more money than multiple single-processor systems, because they can share peripherals, mass storage, and power supplies. If several programs operate on the same set of data, it is cheaper to store those data on one disk and to have all the processors share them, than to have many computers with local disks and many copies of the data.
- **Increased reliability**: If functions can be distributed properly among several processors, then the failure of one processor will not halt the system, only slows down. If we have ten processors and one fails, then each of the remaining nine processors must take a share of the work of the failed processor. Thus, the entire system runs only 10 percent slower, rather than failing altogether. This ability to continue providing service proportional to the level of surviving hardware is called ***graceful degradation***. Systems designed for graceful degradation are also called ***fault tolerant***.

Continued operation in the presence of failures requires a mechanism to allow the failure to be detected, diagnosed, and, if possible, corrected. The Tandem system uses both hardware and software duplication to ensure continued operation despite faults. The system consists of two identical processors, each with its own local memory. The processors are connected by a bus. One processor is the primary and the other is the backup. Two copies are kept of each process: one on the primary processor and the other on the backup. At fixed checkpoints in the execution of the system, the state information of each job including a copy of the memory image-is copied from the primary machine to the backup. If a failure is detected, the backup copy is activated and is restarted from the most recent checkpoint. This solution is expensive, since it involves considerable hardware duplication.

The most common multiple-processor systems now use symmetric multiprocessing (SMP), in which each processor runs an identical copy of the OS, and these copies communicate with one another as needed. Some systems use asymmetric multiprocessing, in which each processor is assigned a specific task. A master processor controls the system; the other processors either look to the master for instruction or have predefined tasks. This scheme defines a master-slave relationship. The master processor schedules and allocates work to the slave processors. SMP means that all processors are peers; no master-slave relationship exists between processors. Each processor concurrently runs a copy of the operating system.

A typical SMP architecture is shown in Figure 1.13. An example of the SMP system is Encore's version of UNIX for the Multimax computer. This computer can be configured such that it employs dozens of processors, all running copies of UNIX. The benefit of this model is that many processes can run simultaneously-N processes can run if there are N CPUs-without causing a significant deterioration of performance. However, 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 to be shared dynamically among the various processors. Almost all modern OS including Windows NT, Solaris, Digital UNIX, OS/2, and Linux will support SMP.

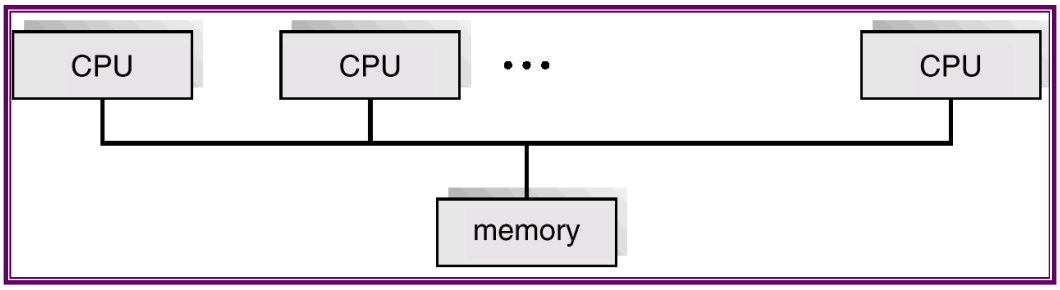


Figure 1.13 Symmetric multiprocessing architecture

The difference between symmetric and asymmetric multiprocessing may be the result of either hardware or software. Special hardware can differentiate the multiple processors, or the software can be written to allow only one master and multiple slaves. For instance, Sun's operating system SunOS Version 4 provides asymmetric multiprocessing, whereas Version 5 (Solaris 2) is symmetric on the same hardware.

As microprocessors become less expensive and more powerful, additional OS functions are off-loaded to slave processors (or back-ends). For example, it is fairly easy to add a microprocessor with its own memory to manage a disk system. The microprocessor could receive a sequence of requests from the main CPU and implement its own disk queue and scheduling algorithm. Thus, the main CPU is relieved from overhead of disk scheduling.

1.11 DISTRIBUTED SYSTEMS

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

Networks vary by the protocols used, the distances between nodes, and the transport media. TCP/IP is the most common network protocol. Similarly, OS support of protocols varies. Most OS support TCP/IP, including Windows and UNIX. Some systems support proprietary protocols to suit their needs. To an OS, a network protocol needs an interface device (for example, a network adapter) with a device driver to manage it, and software for sending/receiving packet data.

Networks are typecast based on the distances between their nodes; for example, LAN, WAN, MAN etc. The media to carry networks are equally varied. They include copper wires, fiber strands, and wireless transmissions between satellites, microwave dishes and radios. When computing devices are connected to cellular phones, they create a network.

Even very short-range infrared communication can be used for networking. These networks also vary by their performance and reliability.

1.11.1 Client-Server Systems

As PCs have become faster, more powerful, and cheaper, designers have shifted away from the centralized system architecture. Terminals connected to centralized systems are now being replaced by PCs. Similarly, user-interface functionality that was handled directly by the centralized systems is now being handled by the PCs. As a result, centralized systems today act as server systems to satisfy requests generated by client systems. The general structure of a client-server system is shown in Figure 1.14. Server systems can be broadly categorized as follows:

- **Compute-server systems** provide an interface to which clients can send requests to perform an action, in response to which they execute the action and send back results to the client.
- **File-server systems** provide a file-system interface where clients can create, update, read, and delete files.

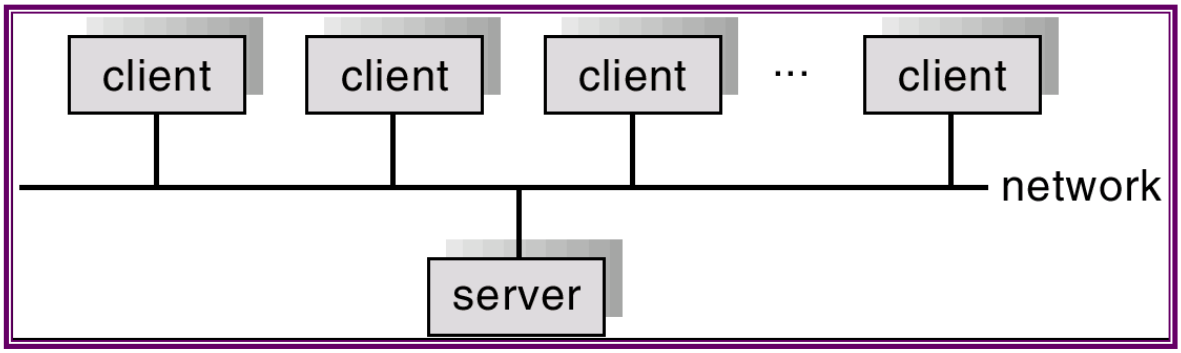


Figure 1.14 General structure of client-server system

1.11.2 Peer-to-peer Systems

The growth of computer networks like the Internet and World Wide Web (WWW) has major influence on the recent development of OS. When PCs were introduced in the 1970s, they were designed for personal use. With the beginning of internet for electronic mail, ftp, and gopher, many PCs became connected to computer networks in 1980s. With the introduction of the Web in the mid 1990s, network connectivity became an essential component of a computer system.

Virtually all modern PCs and workstations are capable of running a web browser. OS like Windows, OS/2, MacOS, UNIX etc also include the system software (such as TCP/IP and PPP) that enables a computer to access the Internet via a LAN or telephone connection.

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

telephone lines. These systems are usually referred to as ***loosely coupled systems*** (or distributed systems).

A network OS is an OS that provides features such as file sharing across the network, and that includes a communication scheme that allows different processes on different computers to exchange messages. A computer running a network OS acts autonomously from all other computers on the network, although it is aware of the network and is able to communicate with other networked computers. A distributed OS is a less autonomous environment. Here, the different OS communicate closely enough to provide the illusion that only a single operating system controls the network.

1.12 CLUSTERED SYSTEMS

In clustered systems, multiple CPUs are gathered together to perform computational work. Clustered systems are composed of two or more individual systems coupled together. The clustered computers share storage and are closely linked via LAN networking.

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

In asymmetric clustering, one machine is in hot standby mode while the other is running the applications. The hot standby host (machine) just monitors the active server. If that server fails, the hot standby host becomes the active server. In symmetric mode, two or more hosts are running applications, and they are monitoring each other. This mode is obviously more efficient, as it uses all of the available hardware.

Other forms of clusters include parallel clusters and clustering over a WAN. Parallel clusters allow multiple hosts to access the same data on the shared storage. Since most OS lack support for this simultaneous data access by multiple hosts, parallel clusters are usually accomplished by special versions of software and special releases of applications. For example, Oracle Parallel Server is a version of Oracle's database that has been designed to run on parallel clusters. Each machine runs Oracle, and a layer of software tracks access to the shared disk. Each machine has full access to all data in the database.

Most systems do not offer general-purpose distributed file systems. Therefore, many clusters do not allow shared access to data on the disk. For this, distributed file systems must provide access control and locking to the files to ensure no conflicting operations occur. This type of service is commonly known as a ***distributed lock manager (DLM).***

Cluster technology is rapidly changing. Cluster directions include global clusters, in which the machines could be anywhere in the. Such projects are still the subject of research and development. Clustered system use and features should expand greatly as storage-area networks (SANS) become prevalent. SANs allow easy attachment of multiple hosts to

multiple storage units. Current clusters are usually limited to two or four hosts due to the complexity of connecting the hosts to shared storage.

1.13 REAL-TIME SYSTEMS

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

A real-time system has well-defined, fixed time constraints. Processing must be done within the defined constraints, or the system will fail. For instance, a robot arm must be instructed to halt before it reaches a wall. A real-time system functions correctly only if it returns the correct result within its time constraints. So in real-time system, a quick response is expected; whereas in batch system time constraints are not at all there.

Real-time systems may be hard or soft. A hard real-time system guarantees that critical tasks be completed on time. That is, various tasks like retrieval of stored data, process by OS etc are bounded by time. Secondary storage is usually limited with data being stored in read-only memory (ROM). ROM is located on nonvolatile storage devices that retain their contents even in the case of electric outage; most other types of memory are volatile. Most advanced OS features are absent too, since they tend to separate the user from the hardware. And this result in uncertainty about the amount of time an operation will take. For example, virtual memory is almost never found on real-time systems. Therefore, hard real-time systems conflict with the operation of time-sharing systems, and the two cannot be mixed. None of the existing general-purpose OS support hard real-time functionality.

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

1.14 HANDHELD SYSTEMS

Handheld systems include personal digital assistants (PDAs), cell phones, tablets with/without internet connection. Developers of handheld systems and applications face many challenges, most of which are due to the limited size of such devices. Due to limited

size, most handheld devices have a small amount of memory, slow processors, and small display screens. The handheld devices have following limitations:

- **Limited Memory:** Many handheld devices have between 512 KB and 2 GB of memory. As a result, the operating system and applications must manage memory efficiently. This includes returning all allocated memory back to the memory manager once the memory is no longer being used. Virtual memory allows developers to write programs that behave as if the system has more memory than actual availability. Currently, many handheld devices do not use virtual memory techniques, thus forcing program developers to work within the limited physical memory.
- **Less Speed:** Processors for most handheld devices run at a fraction of the speed of a processor in a PC. Faster processors require more power. To include a faster processor in a handheld device would require a larger battery that would have to be replaced (or recharged) more frequently. To minimize the size of most handheld devices, smaller, slower processors which consume less power are typically used. Therefore, the OS and applications must be designed accordingly.
- **Small Display:** Handheld devices have a display of about 3-5 inches, whereas, PCs will be having display of about 14 – 40 inches. Familiar tasks, such as reading e-mail or browsing web pages, must be condensed onto smaller displays. One approach for displaying the content in web pages is web clipping, where only a small subset of a web page is delivered and displayed on the handheld device.

1.15 FEATURE MIGRATION

Overall examination of OS for mainframes and microcomputers shows that features once available only on mainframes have been adopted by microcomputers. The same concepts are appropriate for the various classes of computers: mainframes, minicomputers, microcomputers, and handhelds. Figure 1.15 shows the migration of OS features. However, to start understanding modern OS, you need to realize the theme of feature migration and to recognize the long history of many OS features.

A good example of this movement occurred with the MULTIplexed Information and Computing Services (MULTICS) operating system. MULTICS was developed from 1965 to 1970 at the Massachusetts Institute of Technology (MIT) as a computing utility. It ran on a large, complex mainframe computer (the GE 645). Many of the ideas that were developed for MULTICS were subsequently used at Bell Laboratories (one of the original partners in the development of MULTICS) in the design of UNIX. The UNIX operating system was designed circa 1970 for a PDP-11 minicomputer. Around 1980, the features of UNIX became the basis for UNIX-like operating systems on microcomputer systems, and they are being included in more recent operating systems such as Microsoft Windows NT, IBM 0S/2, and the Macintosh operating system. Thus, the features developed for a large mainframe system have moved to microcomputers over time.

At the same time as features of large operating systems were being scaled down to fit PCs, more powerful, faster, and more sophisticated hardware systems were being developed. The personal workstation is a large PC-for example, the Sun SPARCstation, the HP/Apollo,

the IBM RS/6000, and the Intel Pentium class system running Windows NT or a UNIX derivative. Many universities and businesses have large numbers of workstations tied together with local-area networks. As PCs gain more sophisticated hardware and software, the line dividing the two categories-mainframes and microcomputers-is blurring.

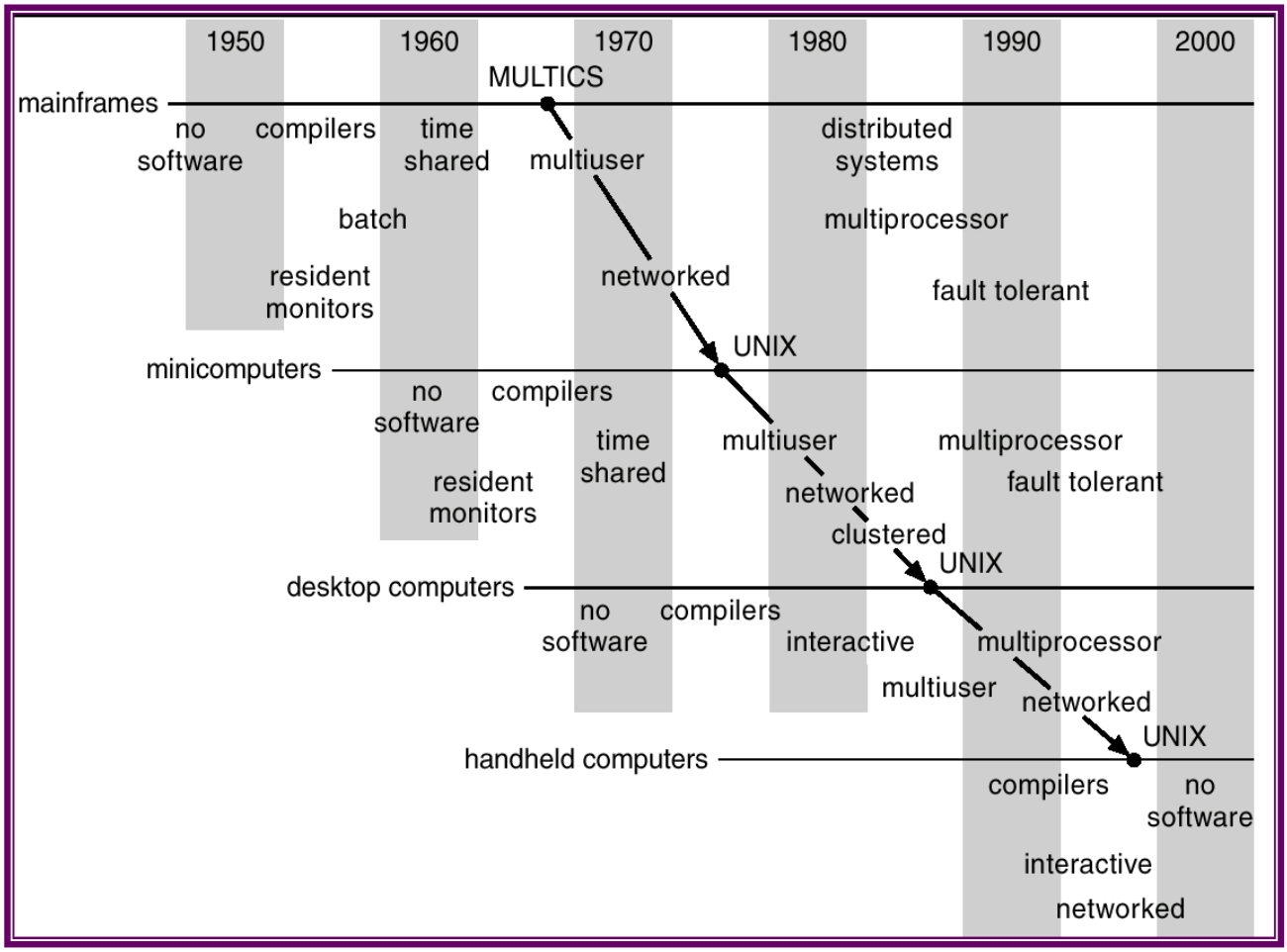


Figure 1.15 Migration of OS concepts and features

1.16 COMPUTING ENVIRONMENTS

Here, we will discuss how multiprogrammed, time-shared, handheld computers are used in variety of computing environment settings.

1.16.1 Traditional Computing

As the time passes by, the computing environment is getting matured. Just a few years ago, a typical office environment consisted of PCs connected to a network, with servers providing file and print service. Remote access was not so easy, and portability was achieved by carrying laptops. Terminals attached to mainframes were common at many companies as well, with even fewer remote access and portability options.

The current trend is toward more ways to access these environments. Web technologies are stretching the boundaries of traditional computing. Companies implement portals which

provide web accessibility to their internal servers. Network computers are essentially terminals that understand web-based computing. Handheld computers can synchronize with PCs to allow very portable use of company information. They can also connect to wireless networks to use the company's web portal.

At home, most users had a single computer with a slow modem connection to the office and Internet. Network connections with good speed are now available for lower cost. Those fast data connections are allowing home computers to serve up web pages and to contain their own networks with printers, client PCs, and servers. Some homes even have firewalls to protect these home environments from security breaches.

1.16.2 Web-based Computing

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

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

1.16.3 Embedded Computing

Embedded computers are the most common form of computers in existence. They run embedded real-time OS. These devices are found everywhere, from car engines and manufacturing robots to VCRs and microwave ovens. They tend to have very specific tasks. The systems they run on are usually primitive, lacking advanced features, such as virtual memory, and even disks. Thus, the OS provide limited features. They usually have little or no user interface, preferring to spend their time monitoring and managing hardware devices, such as automobile engines and robotic arms.

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

The use of embedded systems will increase over the time. Their need as a standalone device or as a member of network/web is increasing. Entire houses can be computerized, so that a central computer-either a general-purpose computer or an embedded system-can control heating and lighting, alarm systems, and even coffee makers. Web access can let a home-owner tell the house to heat up before he arrives home. Someday, the refrigerator may call the grocery store when it notices the milk is gone.

1.17 INTRODUCTION TO SYSTEM STRUCTURES

An OS provides the environment within which the programs are executed. OS varies in their design. Before designing OS, the goals of the OS must be decided. The type of system desired is the basis for choices among various algorithms and strategies.

An OS can be viewed from different angles:

- By examining the services provided by it.
- By looking at the interface that it makes available to users and programmers.
- By disassembling the system into its components and their interconnections.

Here, we will discuss all these aspects of OS, showing the viewpoints of users, programmers, and OS designers. We consider what services an OS provides, how they are provided, and what the various methodologies are for designing such systems.

1.18 SYSTEM COMPONENTS

A large and complex OS can be created by designing small pieces. Each piece should be a well defined portion of the system, with carefully defined inputs, outputs, and functions. Obviously, not all systems have the same structure. However, many modern systems share the goal of supporting the system components as discussed in the following sections.

1.18.1 Process Management

In simple terms, a process can be thought of as a job or time-shared program in execution. A notepad being run, a system task sending output to the printer, a compiler running a program etc. are examples of a process.

A process requires some resources like CPU time, memory, files, I/O devices etc to accomplish its task. These resources are either given to the process when it is created, or allocated to it while it is running. Along with these physical and logical resources, various inputs may be passed when a process is running. For example, consider a process whose function is to display the status of a file on the screen of a terminal. At the beginning, name of the file can be given as an input to the process. It will then execute the appropriate instructions and system calls to obtain and display the information on the terminal. When the process terminates, the OS will claim back the resources.

Program is a passive entity and a process is an active entity. So, a simple program is not a process; whereas a program with a program counter specifying the next instruction to execute is a process. The execution of a process must be sequential and only one instruction is executed at a time. Thus two processes associated with the same program are not treated as two separate execution sequences. A program may generate many processes. A system consists of a collection of processes:

- operating-system processes – those that execute system code
- user processes – those that execute user code

All these processes can potentially execute concurrently, by multiplexing the CPU among them.

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

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

Process management techniques are discussed later.

1.18.2 Main-memory Management

The main memory is central to the operation of a modern computer system. Main memory is a large array of words or bytes. Each word or byte has its own address. Main memory is a repository of quickly accessible data shared by the CPU and I/O devices. The central processor reads instructions from main memory during the instruction-fetch cycle, and it reads/writes data from main memory during the data-fetch cycle. The I/O operations implemented via DMA also read/write data in main memory. The main memory is the only large storage device that the CPU is able to address and access directly. For example, the data stored in the disk must be transferred to main memory for processing that data. Similarly, 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 the utilization of CPU and the speed of computer's response, several programs must be kept in the memory. Different memory management schemes are available and the effectiveness of them depends on situation. Selection of a memory management scheme for a specific system depends on many factors; especially on the hardware design of the system.

The OS is responsible for the following activities in connection with memory management:

- Keeping track of which parts of memory are currently being used and by whom
- Deciding which processes are to be loaded into memory when memory space becomes available
- Allocating and deallocating memory space as needed

1.18.3 File Management

File management is one of the most visible components of an OS. Computers can store information on different types of physical media like magnetic tape, magnetic disk and optical disk etc. Each of such media has its own characteristics and physical organization. And every medium is controlled by a device, such as a disk drive or tape drive. These devices also have unique characteristics like access speed, capacity, data-transfer rate, and access method (sequential or random).

For convenient use of the computer system, the OS provides a uniform logical view of information storage. The OS uses the concepts of the physical properties of its storage devices to define a logical storage unit, the file. OS then maps files onto physical media, and accesses these files via the storage devices.

A file is a collection of related information. Commonly, files represent programs (may be source code or object code) and data. Data files may be numeric, alphabetic, or alphanumeric. Files may be of free-form like text files or they may be formatted rigidly like the files containing fixed fields. A file consists of a sequence of bits, bytes, lines, or records whose meanings are defined by their creators.

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

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

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

1.18.4 I/O System Management

One of the purposes of an OS is to hide the peculiarities of specific hardware devices from the user. For example, in UNIX, the peculiarities of I/O devices are hidden from the bulk of the operating system itself by the I/O subsystem. The I/O subsystem consists of

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

Only the device driver knows the peculiarities of the specific device to which it is assigned. How the I/O subsystem interfaces to the other system components, manages devices, transfers data, and detects I/O completion etc are discussed later in detail.

1.18.5 Secondary Storage Management

The main purpose of a computer system is to execute programs. These programs along with the data they access must be in main memory, or primary storage, during execution. As main memory is not sufficient to store all data and programs; and data may get lost when the power goes off, the computer system must provide secondary storage to back up main memory.

Most modern computer systems use disks as the principal on-line storage medium, for both programs and data. Most programs including compilers, assemblers, sort routines, editors, and formatters-are stored on a disk until loaded into memory, and then use the disk as both

the source and destination of their processing. Hence, the proper management of disk storage is of central importance to a computer system.

The operating system is responsible for the following activities of disk management:

- Free-space management
- Storage allocation
- Disk scheduling
  Because secondary storage is used frequently, it must be used efficiently. The entire speed of operation of a computer may attract the speeds of the disk subsystem and of the algorithms that manipulate that subsystem.

1.18.6 Networking

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

The communication network may be fully or partially connected. Its design must focus on message routing, connection strategies, problems of contention and security. A distributed system collects physically separate and heterogeneous systems into a single coherent system, providing the user with access to the various resources that the system maintains.

Access to a shared resource allows computation speedup, increased functionality, increased data availability, and enhanced reliability. OS generalizes network access as a form of file access and the details of networking are contained in the device driver of network interface.

The protocols of a distributed system influence the utility and popularity of that system. The aim of World Wide Web was to create a new access method for information sharing. It improved file-transfer protocol (FTP) and network file-system (NFS) protocol by removing the need for a user to log in for accessing remote resource. It defined a new protocol, hypertext transfer protocol (HTTP), for use in communication between a web server and a web browser. A web browser then needs to send a request for information to a remote machine's web server, and the information is returned. Hence, the HTTP and web usage got increased.

1.18.7 Protection System

If a computer system has multiple users and concurrent execution of multiple processes is allowed, then these processes must be protected from each other's activities. For that purpose, OS must ensure that the files, memory segments, CPU, and other resources are be operated by only authorized processes. For example, memory-addressing hardware ensures that a process can execute only within its own address space. The timer ensures that no process can gain control of the CPU without eventually relinquishing control.

Protection is any mechanism for controlling the access of programs, processes, or users to the resources defined by a computer system. This mechanism must provide the rules for how and what type of controls must be enforced. Protection can improve reliability by detecting hidden errors at the interfaces between component subsystems. Early detection of interface errors can prevent contamination of a healthy subsystem by a malfunctioning subsystem. An unprotected resource cannot defend against use (or misuse) by an unauthorized or incompetent user. A protection-oriented system provides a means to distinguish between authorized and unauthorized usage.

1.18.8 Command Interpreter System

Command interpreter is the interface between the user and the OS. It is one of the most important system programs for an OS. Some OS include the command interpreter in the kernel. Few OS like MS-DOS and UNIX treat the command interpreter as a special program that is running when a job is initiated, or when a user first logs on.

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

Normally, shell is differentiated from user-friendly command interpreter. For this purpose, mouse-based window and menu systems are used in OS like Microsoft Windows and Macintosh. Here, a mouse pointer and clicks will do the job of navigation and selection.

In a complex shell like MS-DOS and UNIX, commands are typed using a keyboard and displayed on a screen. Then enter key has to be pressed to indicate end of the command and then the task is executed.

The command statements themselves deal with process creation and management, I/O handling, secondary-storage management, main-memory management, file-system access, protection, and networking.

1.19 OPERATING SYSTEM SERVICES

An OS provides an environment for the execution of programs. Also, it provides certain services to the programs and its users. Though these services differ from one OS to the other, following are some general services provided by any OS.

- **Program execution:** The system must be able to load a program into memory and to run that program. The program must be able to end its execution, either normally or abnormally (indicating error).
- **I/O operations:** A running program may require I/O. This I/O may involve a file or an I/O device. For efficiency and protection, users usually cannot control I/O devices directly. Therefore, the OS must provide a means to do I/O.

- **File-system manipulation:** The OS must facilitate the programs to read and write the files. And also, programs must be allowed to create and delete files by name.
- **Communications:** Processes may need to exchange information with each other. These processes may be running on same computer or on different computers. Communications may be implemented via shared memory, or by the technique of message passing, in which packets of information are moved between processes by the OS.
- **Error detection:** The OS constantly needs to be aware of possible errors. Errors may occur in any of CPU, memory hardware, I/O devices, user program etc. For each type of error, the OS should take the appropriate action to ensure correct and consistent computing.

OS also has another set of functionalities to help the proper functioning of itself.

- **Resource allocation:** When multiple users are logged on the system or multiple jobs are running at the same time, resources must be allocated to each of them. OS has to manage many resources like CPU cycles, main memory, and file storage etc. OS uses CPU scheduling routines for effective usage of CPU. These routines manage speed of the CPU, the jobs that must be executed, the number of registers available, and such other factors.
- **Accounting:** We want to keep track of which users use how many and which kinds of computer resources. This record keeping may be used for accounting (so that users can be billed) or simply for accumulating usage statistics. Usage statistics may be a valuable tool for researchers who wish to reconfigure the system to improve computing services.
- **Protection:** Information on a multi-user computer system must be secured. When multiple processes are executing at a time, one process should not interfere with the others. Protection involves ensuring that all access to system resources is controlled. System must be protected from outsiders as well. This may be achieved by authenticating the users by means of a password. It also involves defending external I/O devices, modems, network adapters etc. from invalid access.

1.20 SYSTEM CALLS

System calls provide the interface between a process and OS. These calls are normally assembly-language instructions and used by the assembly-language programmers. Some systems allow system calls to be made directly from a higher level language program. In such cases, the calls resemble predefined function or subroutine calls. They may generate a call to a special run-time routine that makes the system call or the system call may be generated directly in-line.

Several languages-such as C, C++, and Perl have replaced assembly language for systems programming. These languages allow system calls to be made directly. For example, UNIX system calls may be invoked directly from a C or C++ program. System calls for modern Microsoft Windows platforms are part of the Win32 application programmer interface (API), which is available for use by all the compilers written for Microsoft Windows.

To understand how system calls are used, consider an example of writing a program to read data from one file and to copy them to another file. The program needs names of two files as an input. These names may be asked from the user. Now, this will initiate a sequence of system calls: to write a prompting message on the screen, and to read the characters from keyboard which defines names of files. Then, the program opens an input file and creates the output file. This operation requires another system call that may possibly face error conditions. Each of the error conditions (like input file doesn't exist, no permission to read, output file can't be created, no write permission etc) require different system calls. When everything is proper, we enter a loop to read from input file and write into the output file. Both of these operations are system calls. Every read/write must return status information for any possible error or end-of-file. Again, this requires system call. Once the task is done, both the files must be closed using system calls. Finally, the program will be terminated using final system call. Thus, we can make out that a very simple program also uses OS heavily. However, the users never see such details. Because, the set of built-in functions provided by the compiler of programming languages provides very simpler interface.

Occurrence of system calls differ from computer to computer. Apart from the identity of the system call, some more information may also be required. The type and nature of information vary according to OS and call. For example, to get input, we may need to specify the file or device to use as the source, and the address and length of the memory buffer into which the input should be read.

Three general methods are used to pass parameters to the operating system:

- The simplest approach is to pass the parameters in registers.
- In some cases, there may be more parameters than registers. Then, the parameters are stored in a block or table in memory. And the address of the block is passed as a parameter in a register as shown in Figure 1.16. This is the approach taken by Linux. Parameters can also be placed, or pushed, onto the stack by the program, and popped of the stack by the operating system.
- Some OS prefer the block or stack methods, because those approaches do not limit the number or length of parameters being passed.

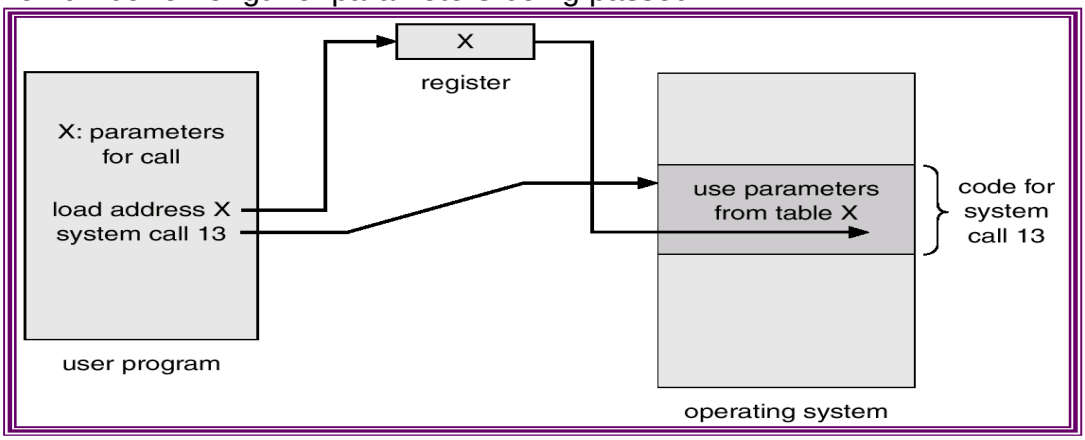


Figure 1.16 Passing of parameters as a table

System calls can be grouped roughly into five major categories: process control, file management, device management, information maintenance, and communications. These are discussed in the following sections. Table 1.1 summarizes the types of system calls provided by OS.

Table 1.1 Types of System Calls

Category	System Calls
Process Control	• end, abort • load, execute • create process, terminate process • get process attributes, set process attributes • wait for time • wait event, signal event • allocate and free memory
File Management	• create file, delete file • open, close • read, write, reposition • get file attributes, set file attributes
Device Management	• request device, release device • read, write, reposition • get device attributes, set device attributes • logically attach or detach devices
Information Maintenance	• get time or date, set time or date • get system data, set system data • get process, file, or device attributes • set process, file, or device attributes
Communications	• create, delete communication connection • send, receive messages • transfer status information • attach or detach remote devices

1.20.1 Process Control

Various system calls for process controls may be listed as below:

- A running program should halt its execution either normally (*end*) or abnormally (*abort*).
- A process or job executing one program may want to *load* and *execute* another program.
- To create a new job or process, a system call *create process* is used.
- We may also want to terminate a job or process that we created (*terminate process*) if we find that it is incorrect or is no longer needed.
- We should be able to control the execution of a process. This control requires the ability to determine and reset the attributes of a job or process, including the job's priority, its maximum allowable execution time, and so on (*get process attributes* and *set process attributes*).
- We may need to wait for a specific amount time (*wait time)* so that the process is completed or, we may need to wait for a particular event to happen (*wait event*).

- The process may need to signal when the event has occurred (*signal event*).
- Memory has to be allocated to the process (*allocate*) when it needs to be executed and the memory has to be deallocated (*free*) when the process is completed.

1.20.2 File Management

Whenever we are dealing with file handling, we should be able to *create* and *delete* files using respective system calls. The files have to be *opened* for using it. One must be able to *read* and *write* the data into/from files. After finishing the job, the file has to be *closed*.

The same set of system calls is required for directories. On both files and directories, few attributes like file name, file type, protection codes (access rights) etc have to be set or retrieved. For this purpose, we may use the system calls *get file attribute* and *set file attribute.*

1.20.3 Device Management

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

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

1.20.4 Information Maintenance

Many system calls exist simply for the purpose of transferring information between the user program and the OS. For example, return the *current time* and *date,* the number of current users, the version number of the OS, the amount of free memory or disk space etc.

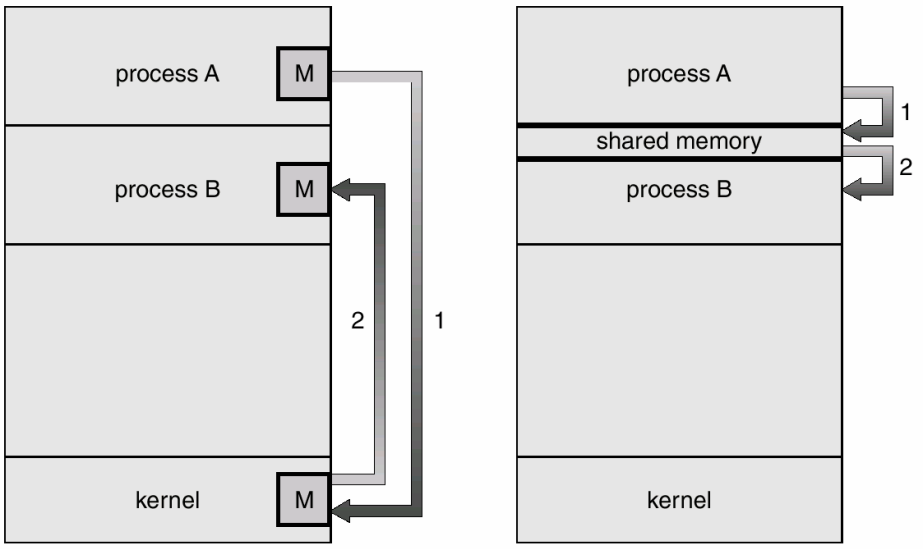
In addition, there are also system calls to reset the process information (get process attributes and set process attributes).

1.20.5 Communication

There are two common models of communication viz.

- Message passing model
- Shared memory model

In the message-passing model, information is exchanged through an inter-process communication facility provided by the OS. Before communication can take place, a connection must be *opened*. The clients and the servers will communicate using *send* and *receive* system calls. In the shared-memory model, processes use *map memory* system calls to gain access to regions of memory owned by other processes. The two communications models are depicted in Figure 1.17.



(a) Message Passing (b) Shared memory

Figure 1.17 Two communication models

1.21 SYSTEM PROGRAMS

System programs provide a convenient environment for program development and execution. Some of them are just user interfaces to system calls; others are considerably more complex. They can be divided into following categories:

- **File Management:** These programs create, delete, copy, rename, print, dump, list, and generally manipulate files and directories.
- **Status Information:** Information like date, time, amount of available memory or disk space, number of users etc. is formatted, and is printed to the terminal or other output device or file.
- **File Modification:** Several text editors may be available to create and modify the content of files stored on disk or tape.
- **Programming-language Support:** Compilers, assemblers, and interpreters for common programming languages (such as C, C++, Java etc) are often provided to the user with the OS.
- **Program loading and execution:** Once a program is assembled or compiled, it must be loaded into memory to be executed. The system may provide absolute loaders, relocatable loaders, linkage editors, and overlay loaders. Debugging systems for either higher-level languages or machine language are required.
- **Communications**: These programs provide the mechanism for creating virtual connections among processes, users, and different computer systems. They allow users to send messages to one another's screens, to browse web pages, to send electronic-mail messages, to log in remotely, or to transfer files from one machine to another.

Most operating systems will supply the programs for solving common problems, or to perform common operations. For example, web browsers, word processors and text

formatters, spreadsheets, database systems, compilers, plotting and statistical-analysis packages, and games. These programs are known as **system utilities** or **application programs.**

1.22 SYSTEM STRUCTURE

The modern OS are very large and complex. To design such an OS, proper care has to be taken. So, it is advised to partition the task into smaller components. Each of such components should be a well-defined unit with properly designed input, output and other functions. We have discussed important components in Section 1.18. Here, we will discuss how these components are interconnected and integrated into kernel.

1.22.1 Simple Structure

Many commercial systems do not have a well-defined structure. Frequently, such operating systems started as small, simple, and limited systems, and then grew into wider range. MS-DOS is an example for one such OS. It was written to provide the most functionality in the least space, so it was not divided into modules carefully. Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated as shown in Figure 1.18.

UNIX is another system that was initially limited by hardware functionality. The UNIX OS consists of two separable parts:

- **Systems programs**
- **The kernel**
  - Consists of everything below the system-call interface and above the physical hardware
  - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level.

The traditional UNIX OS can be layered as shown in Figure 1.19.

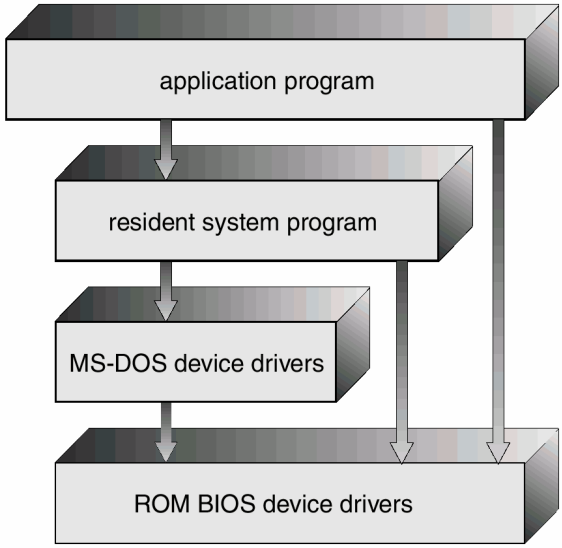


Figure 1.18 MS DOS Layer Structure

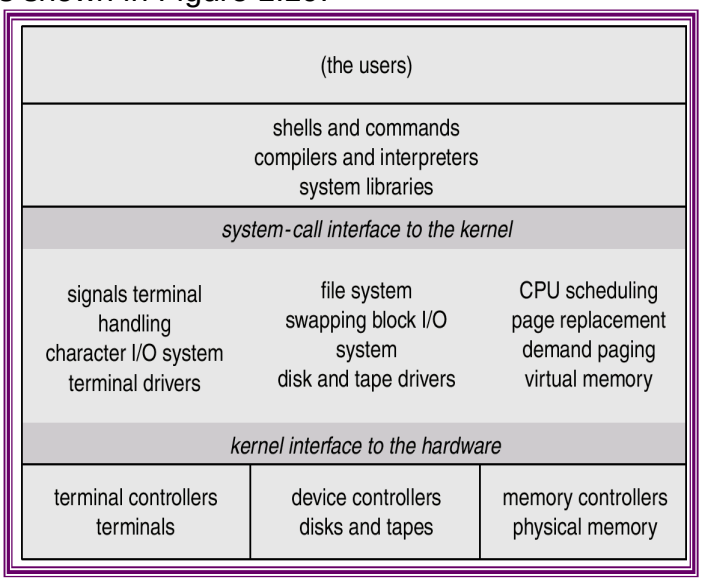


Figure 1.19 UNIX System Structure

1.22.2 Layered Approach

In the layered approach, the OS is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface. A typical OS with layered approach is shown in Figure 1.20.

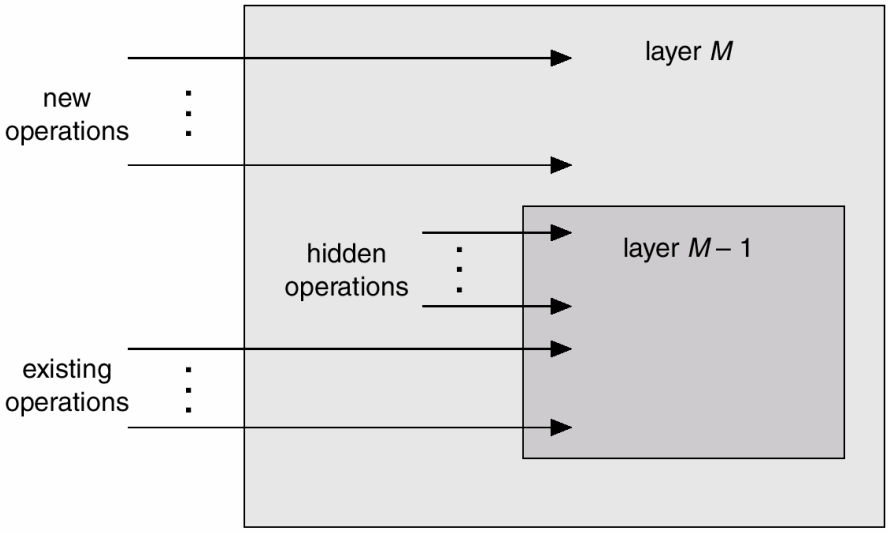


Figure 1.20 A layered approach of OS

The main advantage of layered approach is modularity. The layers are selected such that each uses functions (operations) and services of only lower-level layers. This approach simplifies debugging and system verification.

The major difficulty with the layered approach involves the careful definition of the layers, because a layer can use only those layers below it.

1.22.3 Microkernels

In this approach, all non-essential components of earlier UNIX kernel have been removed and only system and user level programs have been implemented. Hence, it became a smaller kernel. Generally, microkernels provide minimal process and memory management, in addition to a communication facility. Here, the communication takes place between user modules using message passing. The major benefits of using microkernels are:

- easier to extend a microkernel
- easier to port the operating system to new architectures
- more reliable (less code is running in kernel mode)
- more secure

1.23 VIRTUAL MACHINES

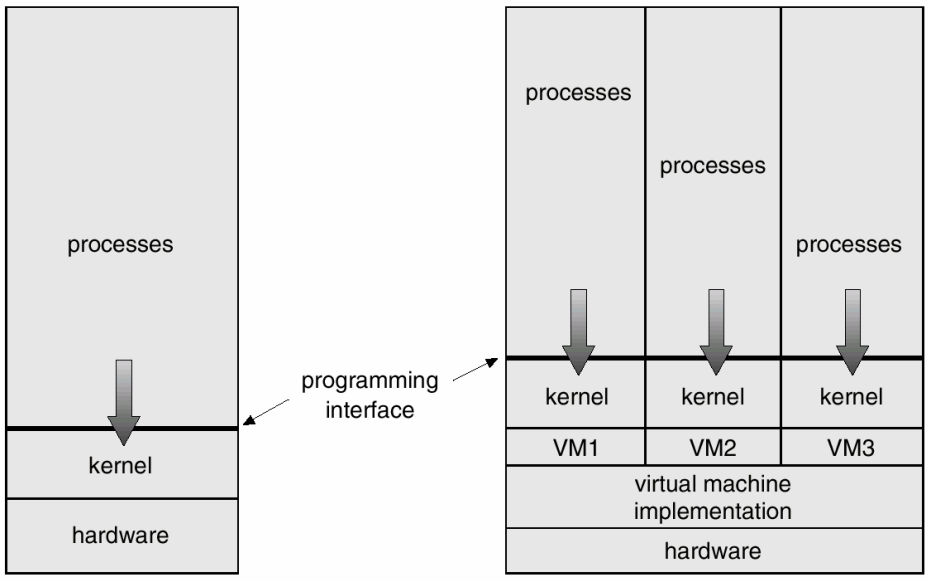
A *virtual machine* takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware. A virtual machine provides an interface *identical* to the underlying bare hardware. The operating system

creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory.

The resources of the physical computer are shared to create the virtual machines. That is,

- CPU scheduling can create the appearance that users have their own processor.
- Spooling and a file system can provide virtual card readers and virtual line printers.
- A normal user time-sharing terminal serves as the virtual machine operator's console.

The difference between non-virtual and virtual machines can be understood by referring Figure 1.21.



(a) (b)

Figure 1.21 System Models (a) Non-virtual machine (b) Virtual Machine

1.23.1 Implementation

Though the virtual-machine (VM) concept is useful, it is difficult to implement due to the effort required to provide an exact duplicate of the underlying machine. Compared to actual I/O, the virtual I/O may take considerably more time, as it is interpreted. Also, since CPU is multiprogrammed among many virtual machines, the VM will slow down unpredictably.

1.23.2 Benefits

The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines. This isolation, however, permits no direct sharing of resources. A virtual-machine system is a perfect means for operating-systems research and development. System development is done on the virtual machine, instead of on a physical machine and so does not disrupt normal system operation.

1.23.3 Java

Java is a popular object oriented programming language which provides specifications for Java Virtual machine (JVM). The java compiler produces a platform independent bytecode (.class) file for every class in the program. This bytecode can run on any JVM. The JVM consists of a class loader, a class verifier and a java interpreter. The just-in-time (JIT) compiler converts the bytecode into native code for a particular computer. Thus, the JVM helps to develop platform independent and portable programs. The JVM can be seen in Figure 1.22.

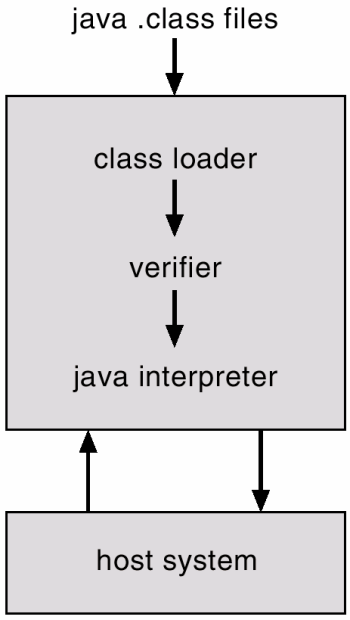


Figure 1.22 Java Virtual Machine

1.24 SYSTEM DESIGN AND IMPLEMENTATION

Here, we will discuss the various problems in designing and implementing a system. No concrete solution exists for the problem, but some approaches are useful.

1.24.1 Design Goals

The very first challenge in a system design is specifying the goals of the system. At the highest level, the design of the system will be affected by the choice of hardware and type of system: batch, time shared, single user, multiuser, distributed, real time, or general purpose. Beyond this highest level, the requirements can be divided into two types:

- **User goals:** operating system should be convenient to use, easy to learn, reliable, safe, and fast.
- **System goals:** operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient.

1.24.2 Mechanisms and Policies

The specification and design of an OS is a highly creative task. There are some general software engineering principles which are applicable to OS. One important principle is the separation of policy from mechanism. Mechanisms determine how to do something; policies determine what will be done.

The separation of policy and mechanism is important for flexibility. Policies are likely to change across places or over time. Each change in policy may require a change in the underlying mechanism. Policy decisions must be made for all resource-allocation and scheduling problems.

1.24.3 Implementation

After designing an OS, the next task would be implementation. Earlier, OS would have been written in assembly language. Now, they can be written in higher languages like C, C++ etc. Code written in higher languages has certain advantages:

- Can be written faster
- Is more compact
- Is easier to understand and debug
- Easier to port (move to some other hardware)

Some may say that implementing OS in higher languages may reduce the speed and performance. But, the modern processors have deep pipelining and multiple functional units. So, they can handle many complexities and perform better. Moreover, the performance of OS depends on the selection of data structures and algorithms rather than the assembly-language code.

1.25 SYSTEM GENERATION

We can design, code, and implement an operating system specifically for one machine at one site. The system must then be configured or generated for each specific computer site, a process sometimes known as system generation (SYSGEN).

The SYSGEN program reads from a given file, or asks the operator of the system for information concerning the specific configuration of the hardware system, or probes the hardware directly to determine what components are there. The following kinds of information must be determined.

- What CPU will be used? For multiple-CPU systems, each CPU must be described.
- How much memory is available?
- What devices are available?
- What operating-system options are desired, or what parameter values are to be used?

Using this information the OS can be compiled for a specific requirement. After an OS is generated, it must be made available for use by the hardware. But how does the hardware know where the kernel is, or how to load that kernel? The procedure of starting a computer by loading the kernel is known as ***booting the system***. Most computer systems have a small piece of code, stored in ROM, known as the ***bootstrap program*** or bootstrap loader. This code is able to locate the kernel, load it into main memory, and start its execution.

ADD – ON TOPIC

INTERRUPTS

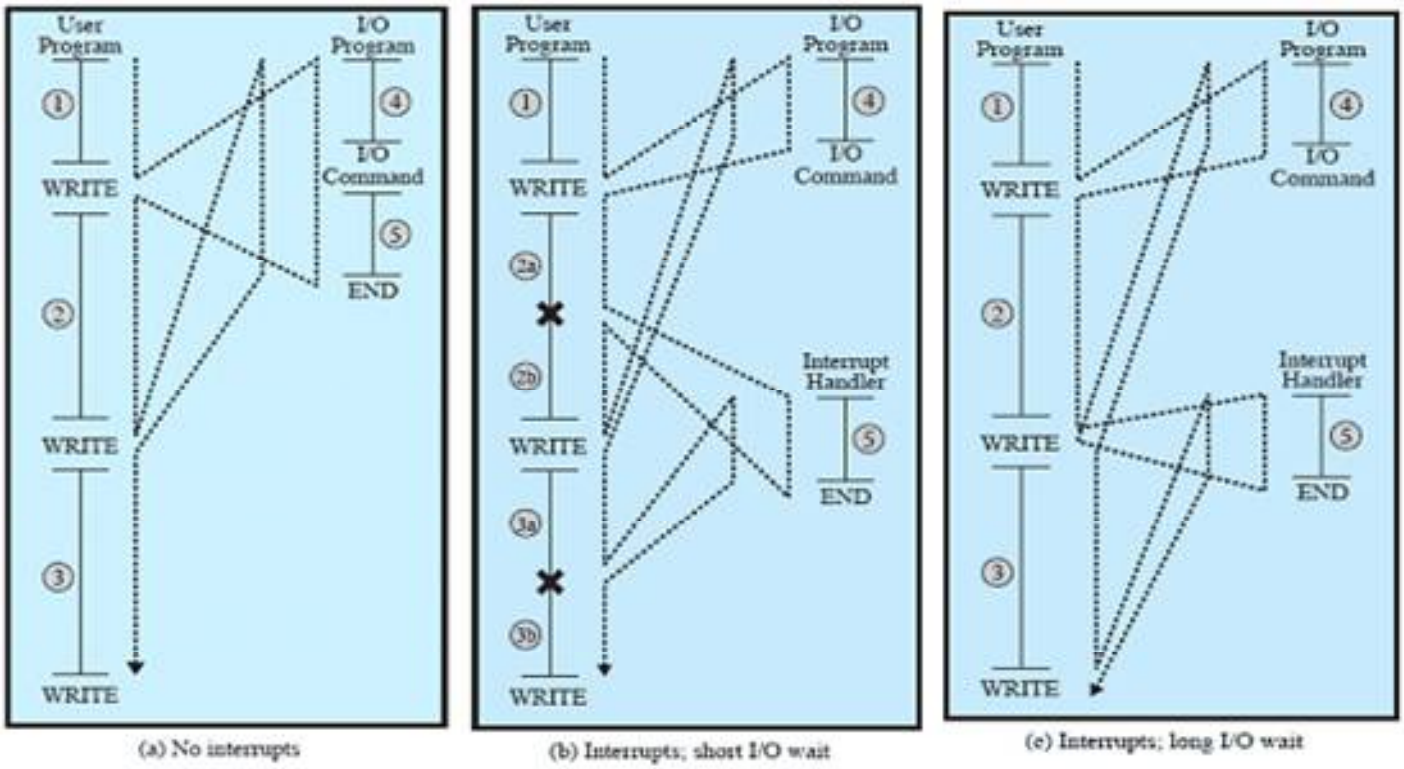
The normal sequence of the processor may be interrupted by other modules like I/O, memory etc. Following table gives the list of common interrupts.

Classes of Interrupts

Class	Description
Program	Generated as a result of an instruction execution. For example, arithmetic overflow, division by zero etc.
Timer	Generated by timer within the processor. It allows OS to perform certain functions on regular basis.
I/O	Generated by I/O controller to indicate any error or normal completion of an operation.
Hardware Failure	Generated by failure like power failure or memory parity error.

The aim of interrupts is to improve the processor utilization. For example, most I/O devices are slower than the processor. If the processor gives the instruction for WRITE something on I/O devices, the I/O unit takes two steps for the job –

- I/O program may copy the data to be written into the buffer etc. and prepare for the actual I/O operation.
- The actual I/O command has to be executed.

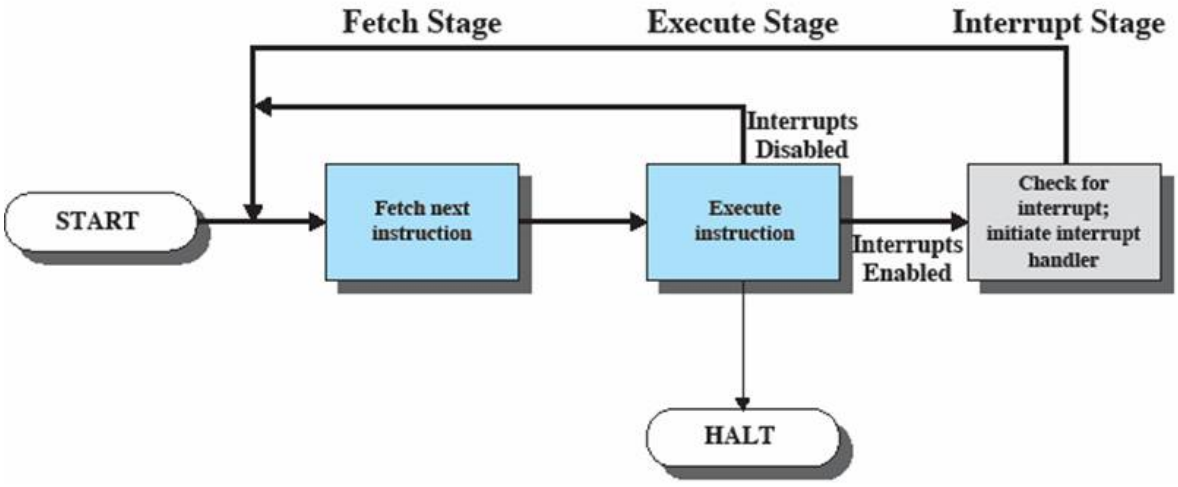


Program flow of control with and without interrupts

Without interrupts, the processor would sit idle while the I/O unit is preparing (the first step) itself for the job. But, in case of interrupts, the processor just gives intimation to the I/O unit first. While I/O unit prepares itself, the processor would continue to execute the next instructions in the program. When I/O unit is ready, in between, it will do the actual I/O command and come back to normal flow of execution. Refer the above given figure for understanding this concept.

Interrupts and the Instruction Cycle

Whenever interrupts are introduced in the system, the processor gives information to the I/O unit and without waiting for I/O operation to complete; it will continue to execute next instruction. When the external device is ready to accept more data from the processor, the I/O module sends an ***interrupt request*** signal to the processor. Now, the processor suspends the current operation of the program and responds to a routine (or a function) of I/O device, which is called as ***interrupt handler***. When the interrupt processing is completed, the processor resumes the execution. To allow interrupts, an ***interrupt stage*** is added along with fetch stage and execute stage as shown in below figure.



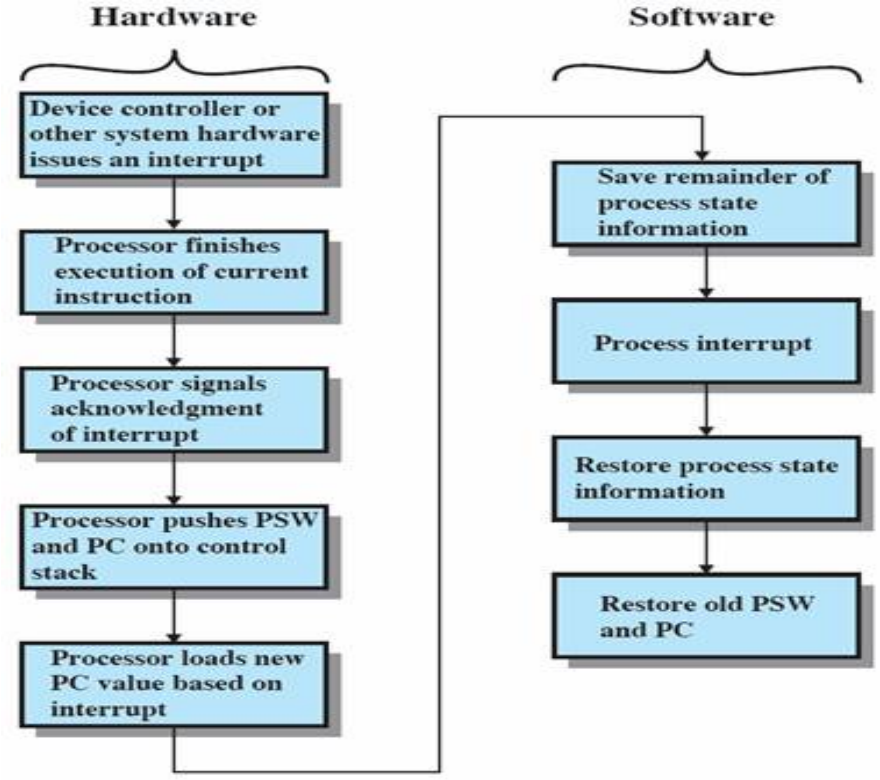
Instruction Cycle with interrupts

In the interrupt stage, the processor checks for any possible interrupt signal. If no interrupt is pending, it will go the fetch stage. If an interrupt signal is there, the processor suspends current execution and executes interrupt handler routine. The interrupt handler routine is a part of OS, which identifies nature of interrupt and performs necessary action. After completing interrupt handler routine, the processor resumes the program execution from the point where it was suspended.

It is understood that some overhead is involved in this process. Extra instructions have to be executed in the interrupt handler to determine type of interrupt, to decide the appropriate action etc. But, instead of processor sitting idle for I/O operation and wasting huge amount of time, the concepts of interrupts are found to be efficient.

Interrupt Processing

An interrupt triggers many events in the processor hardware and software. The following figure shows the sequence of these events and is explained below:



Simple interrupt processing

i. The I/O device (or any other interrupt) issues an interrupt signal to the processor.
ii. The processor finishes the execution of current instruction.
iii. The processor checks for the interrupt signal and send the acknowledgement to the I/O device. This acknowledgement helps the device to clear the interrupt signal.
iv. Now, the processor has to transfer the control to interrupt routine. Before that, it saves the current status of PC and PSW (Program Status Word) in the control stack. This will help the processor to resume its execution after finishing the interrupt routine.
v. Then, the processor loads the entry location of interrupt handling routine into the PC.

Now, the processor has to go for next instruction cycle by fetching the address at PC. But, PC now contains address of interrupt routine (as per step (v)) and hence, the following operations will be carried out.

vi. The PC and PSW relating to the interrupted program have been saved on the control stack. Now, the contents of all registers are also pushed into the control stack. Thus, the top of the stack contains the address of interrupt routine.
vii. The interrupt handler will now process the interrupt.
viii. When interrupt processing is complete, the saved register values are retrieved from the stack and stored back into the registers.

ix. Finally, PC and PSW are loaded with their old values from the stack. Hence, the next instruction of the program will be executed.

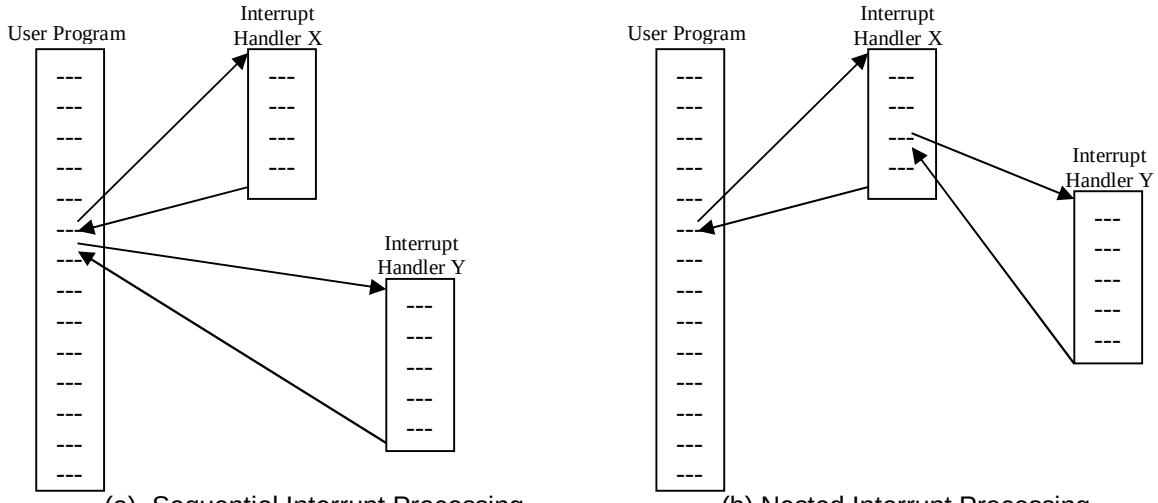
Multiple Interrupts

It is possible to have multiple interrupts in a single program. One or more interrupt can occur during another interrupt is being processed. For example, an input may be taken while printing the data onto the printer. Each time, the printer finishes its job, an interrupt will occur. There are two approaches to deal with multiple interrupts as explained below (Refer Figure given below).

- **Disable the interrupts while one interrupt is being processed:** Here, the processor ignores any new interrupt signal when another interrupt is in progress. The processor keeps such signals as pending, and considers when the previous interrupt routine is completed. Hence, all the interrupts will be processed sequentially.

  But, in this approach, the priorities or inter-dependencies between the interrupt routines are not considered. Hence, time-critical needs cannot be satisfied.

- **Define priorities for interrupts:** Here, a higher priority interrupt will pause the lower priority interrupt which is in execution. Hence, a nested interrupt processing will be achieved based on the priority.



(a) Sequential Interrupt Processing (b) Nested Interrupt Processing

Transfer of control during multiple interrupts

Multiprogramming

Even though interrupts are used, in most of the situations, the processor is not being used efficiently. For example, if there is a long I/O wait, then time taken for I/O is more than the actual user code. So, processor would sit idle. To solve this problem, multiple programs may be made to execute. Thus, when one program is busy with I/O, the other program's instructions may be getting executed; and vice-versa. When the processor is dealing with many programs, the sequence in which the programs are executed will depend on their

relative priority. In this case, when one interrupt routine is completed, the processor may not come back to the user program instructions, but it may consider interrupt of another program based on the priority.