OPERATING SYSTEM INTRODUCTION

OS is a program that acts as an intermediary between a user of a computer and the computer hardware

E.g: Windows, Linux

Operating system goals:

1. Execute user programs and make solving user problems easier
2. Make the computer system convenient to use
3. Use the computer hardware in an efficient manner

1.1 COMPUTER SYSTEM OVERVIEW- BASIC ELEMENTS

At a top level, a computer consists of processor, memory, and I/O components, with one or more modules of each type. These components are interconnected to execute programs.

Four main structural elements:

Processor: Controls the operation of the computer and performs its data processing functions. When there is only one processor, it is often referred to as the central processing unit (CPU).

Main memory: Stores data and programs. It is typically volatile; that is, when the computer is shut down, the contents of the memory are lost. In contrast, the contents of disk memory are retained even when the computer system is shut down. Main memory is also referred to as real memory or primary memory.

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 (e.g., disks), communications equipment, and terminals.

System bus: Provides for communication among processors, main memory, and I/O modules.

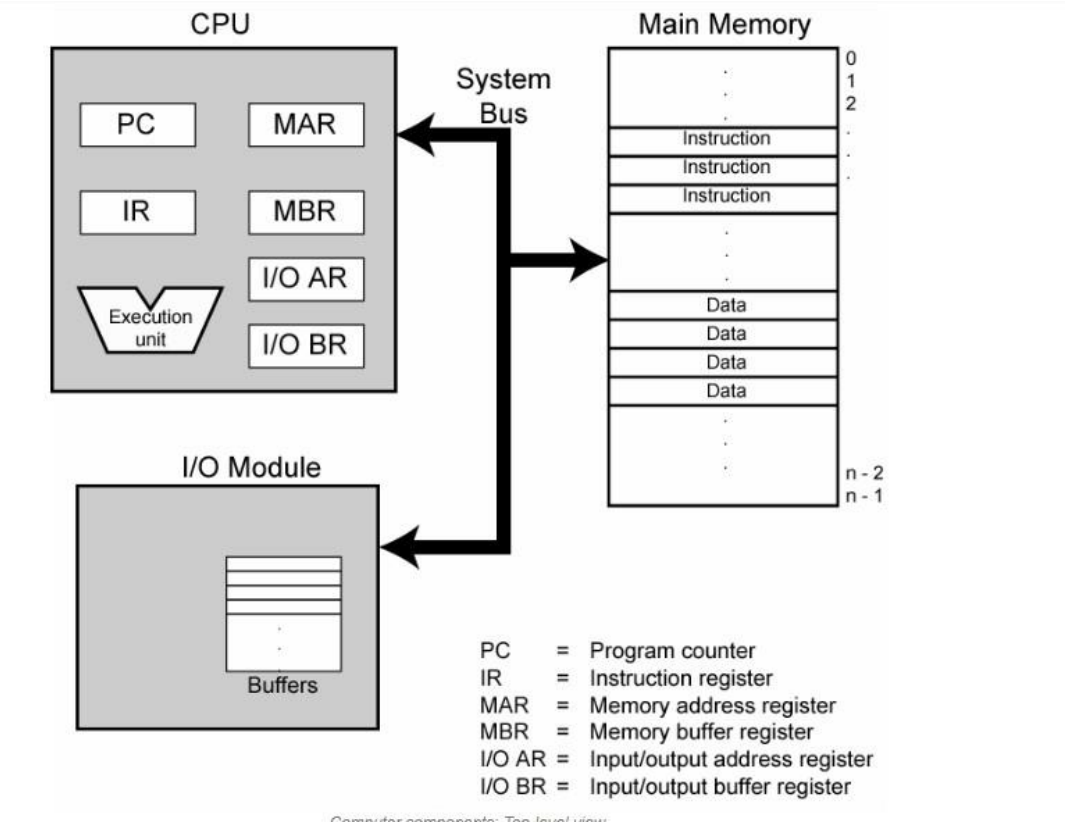


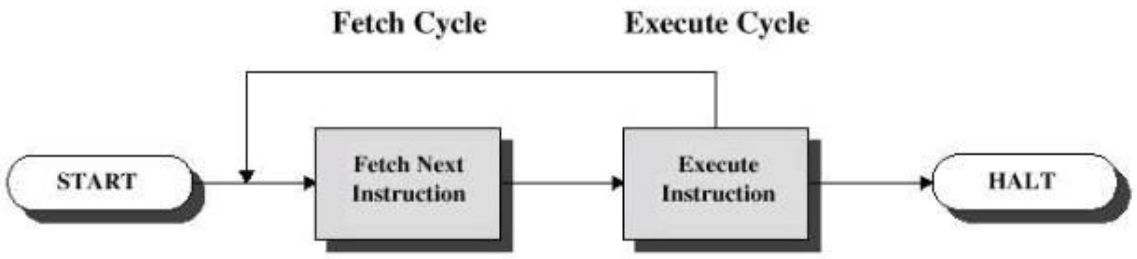
Fig: Computer Components Top Level View

- One of the processor's functions is to exchange data with memory. For this purpose, it typically makes use of two internal (to the processor) registers:
- A memory address register (MAR), which specifies the address in memory for the next read or write;
- A memory buffer register (MBR), which contains the data to be written into memory or which receives the data read from memory.
- Similarly, an I/O address register (I/OAR) specifies a particular I/O device.
- An I/O buffer register (I/OBR) is used for the exchange of data between an I/O module and the processor.
- A memory module consists of a set of locations, defined by sequentially numbered addresses. Each location contains a bit pattern that can be interpreted as either an instruction or data.

- An I/O module transfers data from external devices to processor and memory, and vice versa. It contains internal buffers for temporarily holding data until they can be sent on.

1.2 Instruction Execution

- A program to be executed by a processor consists of a set of instructions stored in memory.
- Instruction processing consists of two steps: The processor reads (*fetches*) instructions from memory one at a time and executes each instruction.
- Program execution consists of repeating the process of instruction fetch and instruction execution.
- The processing required for a single instruction is called an *instruction cycle.*



Instruction cycle:

The two steps are referred to as the *fetch stage* and the *execute stage.*

1. At the beginning of each instruction cycle, the processor fetches an instruction from memory.
2. Typically, the program counter (PC) holds the address of the next instruction to be fetched. PC value is incremented after each instruction fetch.
3. The fetched instruction is loaded into the instruction register (IR). The instruction contains bits that specify the action. The processor interprets the instruction and performs the required action.

In general, these actions fall into four categories:

- **Processor-memory:** Data may be transferred from processor to memory or from memory to processor.

- **Processor-I/O:** Data may be transferred to or from a peripheral device by transferring between the processor and an I/O module.
- **Data processing:** The processor may perform some arithmetic or logic operation on data.
- **Control:** An instruction may specify that the sequence of execution be altered.

Figure illustrates a partial program execution, showing the relevant portions of memory and processor registers. The program fragment shown adds the contents of the memory word at address 940 to the contents of the memory word at address 941 and stores the result in the latter location.

Steps:

1. The PC contains 300, the address of the first instruction. This instruction (the value 1940 in hexadecimal) is loaded into the IR and the PC is incremented.

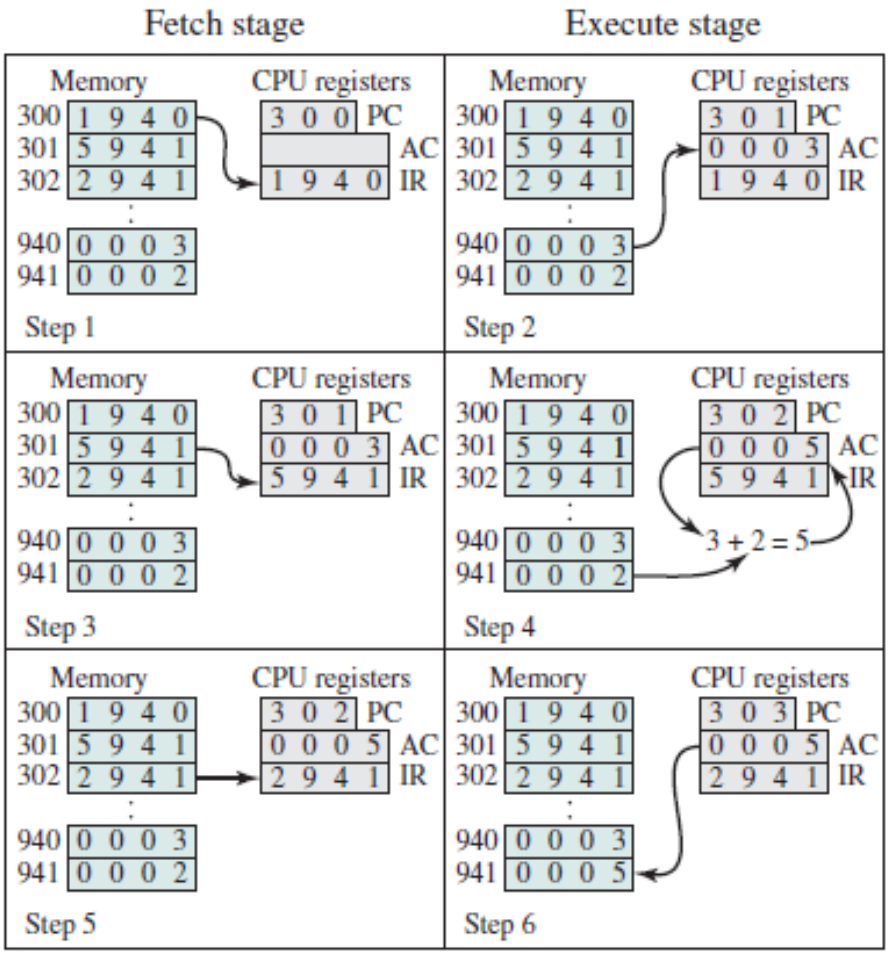


Figure 1.4 **Example of Program Execution** (contents of memory and registers in hexadecimal)

2. The first 4 bits (first hexadecimal digit) in the IR indicate that the AC is to be loaded from memory. The remaining 12 bits (three hexadecimal digits) specify the address, which is 940.

3. The next instruction (5941) is fetched from location 301 and the PC is incremented.

4. The old contents of the AC and the contents of location 941 are added and the result is stored in the AC.

5. The next instruction (2941) is fetched from location 302 and the PC is incremented.

6. The contents of the AC are stored in location 941.

1.6 DIRECT MEMORY ACCESS

Three techniques are possible for I/O operations:

- Programmed I/O
- Interrupt-driven I/O
- Direct memory access (DMA).

Programmed I/O:

- When the processor is executing a program and encounters an instruction relating to I/O, it executes that instruction by issuing a command to the appropriate I/O module.
- The I/O module performs the requested action and then sets the appropriate bits in the I/O status register but takes no further action to alert the processor. In particular, it does not interrupt the processor.
- The processor periodically checks the status of the I/O module until it finds that the operation is complete.
- With programmed I/O, the processor has to wait a long time for the I/O module of concern to be ready for either reception or transmission of more data. The processor, while waiting, must repeatedly cross-examine the status of the I/O module. As a result, the performance level of the entire system is severely degraded.

Interrupt-driven I/O ,

- An alternative to Programmed I/O is for the processor to issue an I/O command to a module and then go on to do some other useful work.
- The I/O module will then interrupt the processor to request service when it is ready to exchange data with the processor. The processor then executes the data transfer, as before, and then resumes its former processing.
- Interrupt-driven I/O, though more efficient than simple programmed I/O, still requires the active intervention of the processor to transfer data between memory and an I/O module, and any data transfer must traverse a path through the processor. Thus, both of these forms of I/O suffer from two inherent drawbacks:

1. The I/O transfer rate is limited by the speed with which the processor can test and service a device.

2. The processor is tied up in managing an I/O transfer; a number of instructions must be executed for each I/O transfer.

Definition

When large volumes of data are to be moved, a more efficient technique is required: **direct memory access (DMA).**

Direct Memory Access (DMA) transfers the block of data between the *memory* and *peripheral devices* of the system, **without the participation** of the **processor**.

The DMA function can be performed by a separate module on the system bus or it can be incorporated into an I/O module.

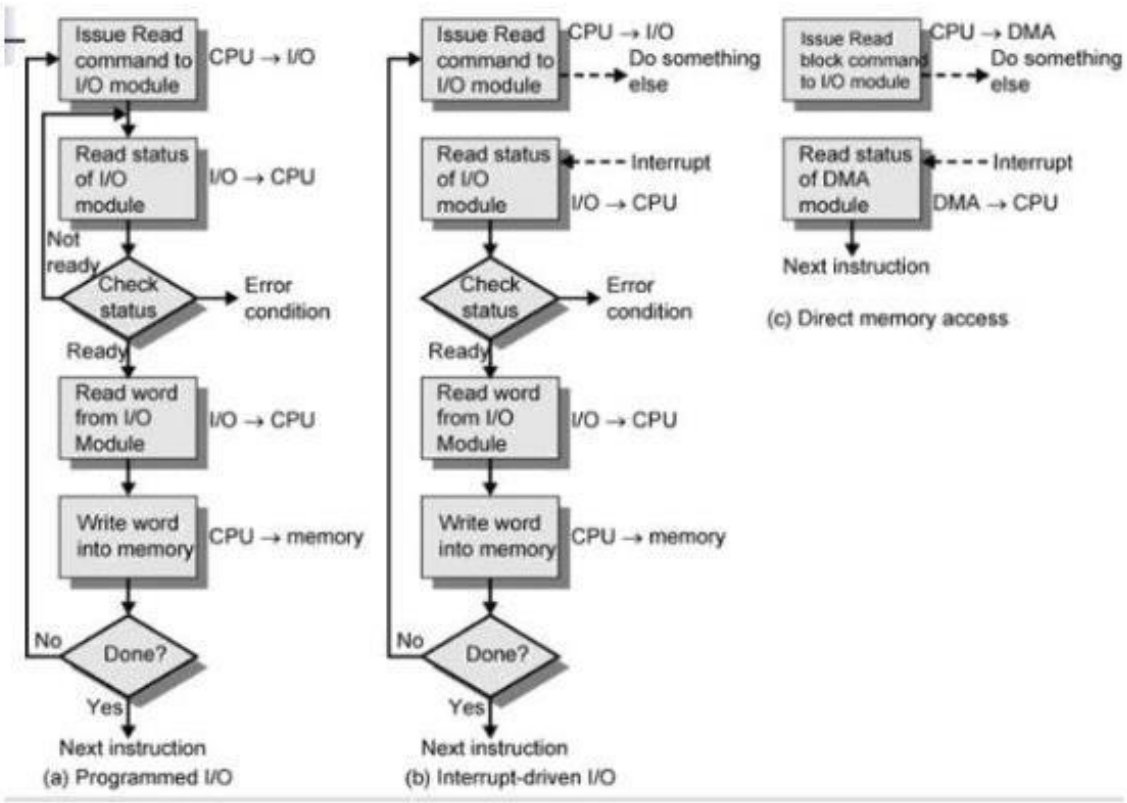
Working

When the processor wishes to read or write a block of data, it issues a command to the DMA module, by sending to the

DMA module the following information:

- Whether a read or write is requested
- The address of the I/O device involved
- The starting location in memory to read data from or write data to
- The number of words to be read or written
- The processor then continues with other work. It has delegated this I/O operation to the DMA module, and that module will take care of it.
- The DMA module transfers the entire block of data, one word at a time, directly to or from memory without going through the processor.
- When the transfer is complete, the DMA module sends an interrupt signal to the processor. Thus, the processor is involved only at the beginning and end of the transfer.
- The DMA module needs to take control of the bus to transfer data to and from memory. Because of this competition for bus usage, there may be times when the processor needs the bus and must wait for the DMA module.
- Note that this is not an interrupt; the processor does not save a context and do something else.

- Rather, the processor pauses for one bus cycle (the time it takes to transfer one word
- across the bus).
- The overall effect is to cause the processor to execute more slowly during a DMA transfer when processor access to the bus is required.
- Nevertheless, for a multiple-word I/O transfer, DMA is far more efficient than interrupt-driven or programmed I/O.



1.7 MULTI PROCESSOR & MULTICORE ORGANIZATION

- A processor executes programs by executing machine instructions in sequence and one at a time. Each instruction is executed in a sequence of operations (fetch instruction, fetch operands, perform operation, store results).
- As computer technology has evolved and as the cost of computer hardware has dropped, computer designers have sought more and more opportunities for parallelism, usually to improve performance and, in some cases, to improve reliability.

The three most popular approaches to providing parallelism by replicating processors: symmetric multiprocessors (SMPs), multicore computers, and clusters.

Multiprocessor Systems

Types

Asymmetric multiprocessing, in which each processor is assigned a specific task. A boss processor, controls the system; the other processors either look to the boss for instruction or have predefined tasks.

- This scheme defines a boss–worker relationship. The boss processor schedules and allocates work to the worker processors.

Symmetric multiprocessing (SMP), in which each processor performs all tasks within the operating system.

SMP means that all processors are peers; no boss–worker relationship exists between processors.

Symmetric Multiprocessors

DEFINITION An SMP can be defined as a stand-alone computer system with the following characteristics:

1. There are two or more similar processors of comparable capability.
2. These processors share the same main memory and I/O facilities and are interconnected by a bus or other internal connection scheme, such that memory access time is approximately the same for each processor.
3. All processors share access to I/O devices, either through the same channels or through different channels that provide paths to the same device.
4. All processors can perform the same functions (hence the term *symmetric*).
5. The system is controlled by an integrated operating system that provides interaction between processors and their programs at the job, task, file, and data element levels.

In an SMP, individual data elements can constitute the level of interaction, and there can be a high degree of cooperation between processes.

- **Advantages**
  - **Performance:** If the work to be done by a computer can be organized so that some portions of the work can be done in parallel, then a system with multiple processors will yield greater performance than one with a single processor of the same type.
  - **Availability:** In a symmetric multiprocessor, because all processors can perform the same functions, the failure of a single processor does not halt the machine. Instead, the system can continue to function at reduced performance.
  - **Incremental growth:** A user can enhance the performance of a system by adding an additional processor.
  - **Scaling:** Vendors can offer a range of products with different price and performance characteristics based on the number of processors configured in the system.

ORGANIZATION Figure 1.19 illustrates the general organization of an SMP. There are multiple processors, each of which contains its own control unit, arithmetic logic unit, and registers.

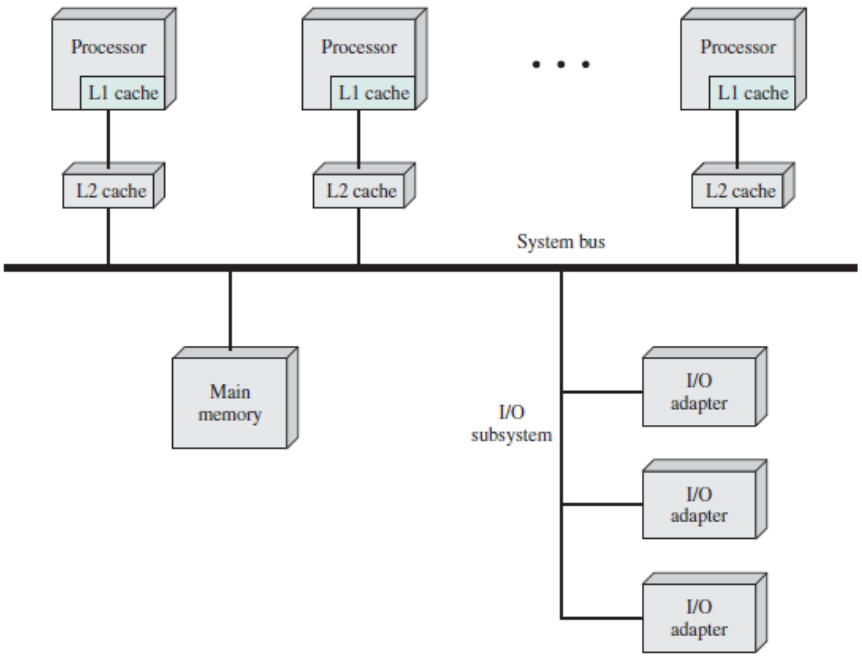


Figure 1.19 Symmetric Multiprocessor Organization

- Each processor has access to a shared main memory and the I/O devices through some form of interconnection mechanism; a shared bus is a common facility.

- The processors can communicate with each other through memory (messages and
- status information left in shared address spaces).
- It may also be possible for processors to exchange signals directly.
- The memory is often organized so that multiple simultaneous accesses to separate blocks of memory are possible.
    - In modern computers, processors generally have at least one level of cache memory that is private to the processor.
    - This use of cache introduces some new design considerations. Because each local cache contains an image of a portion of main memory, if a word is altered in one cache, it could conceivably invalidate a word in another cache.
    - To prevent this, the other processors must be alerted that an update has taken place. This problem is known as the cache coherence problem and is typically addressed in hardware rather than by the OS.

Multicore Computers

- A multicore computer, also known as a chip multiprocessor, combines two or more processors (called cores) on a single piece of silicon (called a die).
- Typically, each core consists of all of the components of an independent processor, such as registers, ALU, pipeline hardware, and control unit, plus L1 instruction and data caches.
- In addition to the multiple cores, contemporary multicore chips also include L2 cache and, in some cases, L3 cache.
- Designers have found that the best way to improve performance to take advantage of advances in hardware is to put multiple processors and a substantial amount of cache memory on a single chip.
- An example of a multicore system is the Intel Core i7, which includes four x86 processors, each with a dedicated L2 cache, and with a shared L3 cache
- One mechanism Intel uses to make its caches more effective is prefetching, in which
- the hardware examines memory access patterns and attempts to fill the caches with data that's likely to be requested soon.

- The Core i7 chip supports two forms of external communications to other chips. DDR3 memory controller, QPI
- The DDR3 memory controller brings the memory controller for the DDR (double data rate) main memory onto the chip. The interface supports three channels that are 8 bytes wide for a total bus width of 192 bits, for an aggregate data rate of up to 32 GB/s.
- The QuickPath Interconnect (QPI) is a point-to-point link electrical interconnect specification. It enables high-speed communications among connected processor chips. The QPI link operates at 6.4 GT/s (transfers per second).

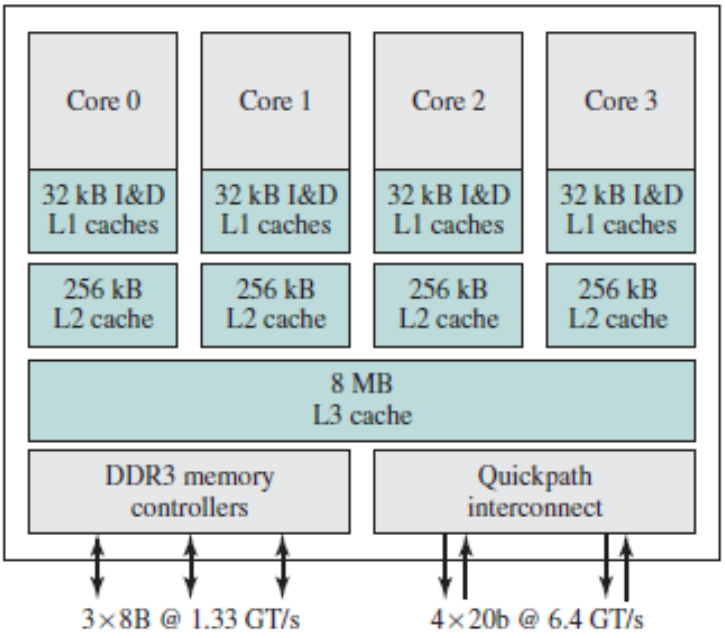


Figure 1.20 Intel Core i7 Block Diagram

1.9 EVOLUTION OF OPERATING SYSTEM

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

The evolution of operating system is explained at various stages.

- Serial Processing
- Simple Batch Systems
- Multi programmed batch systems.
- Time sharing systems

1.9.1 Serial Processing

- With the earliest computers, from the late 1940s to the mid-1950s, the programmer interacted directly with the computer hardware; there was no OS.
- These computers were run from a console consisting of display lights, toggle switches, some form of input device, and a printer.
- Programs in machine code were loaded via the input device (e.g., a card reader).
- If an error halted the program, the error condition was indicated by the lights.
- If the program proceeded to a normal completion, the output appeared on the printer.

These early systems presented two main problems:

- **Scheduling**: Most installations used a hardcopy sign-up sheet to reserve computer time. A user might sign up for an hour and finish in 45 minutes; this would result in wasted computer processing time. On the other hand, the user might run into problems, not finish in the allotted time, and be forced to stop before resolving the problem.
- **Setup time**: A single program, called a job, could involve loading the compiler plus the high-level language program (source program) into memory, saving the compiled program (object program) and then loading and linking together the object program and common functions. Thus, a considerable amount of time was spent just in setting up the program to run.
- This mode of operation could be termed serial processing, reflecting the fact that users have access to the computer in series.

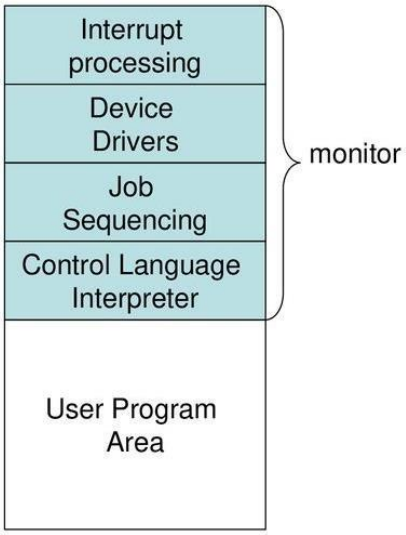
1.9.2 Simple Batch Systems

To improve utilization, the concept of a batch OS was developed.

- The central idea behind the simple batch-processing scheme is the use of a piece of software known as the monitor.
- With this type of OS, the user no longer has direct access to the processor.
- Instead, the user submits the job on cards or tape to a computer operator, who batches the jobs together sequentially and places the entire batch on an input device, for use by the monitor.
- Each program is constructed to branch back to the monitor when it completes processing, at which point the monitor automatically begins loading the next program.

Monitor point of view: The monitor controls the sequence of events. For this to be so, much of the monitor must always be in main memory and available for execution. That portion is referred to as the resident monitor. The rest of the monitor consists of utilities and common functions that are loaded as subroutines to the user program at the beginning of any job that requires them. The monitor reads in jobs one at a time from the input device (typically a card reader or magnetic tape drive). As it is read in, the current job is placed in the user program area, and control is passed to this job. When the job is completed, it returns control to the monitor, which immediately reads in the next job. The results of each job are sent to an output device, such as a printer, for delivery to the user.

Processor point of view: At a certain point, the processor is executing instructions from the portion of main memory containing the monitor. These instructions cause the next job to be read into another portion of main memory. Once a job has been read in, the processor will encounter a branch instruction in the monitor that instructs the processor to continue execution at the start of the user program. The processor will then execute the instructions in the user program until it encounters an ending or error.



- With each job, instructions are included in a primitive form of job control language (JCL).
- This is a special type of programming language used to provide instructions to the monitor.

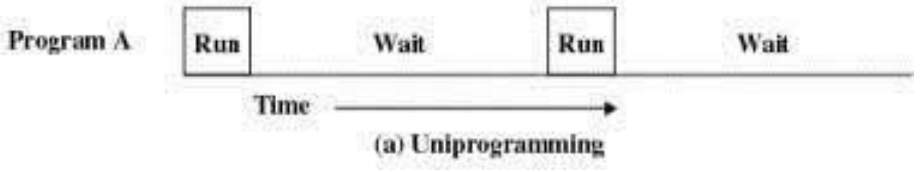
Certain other hardware features are also desirable

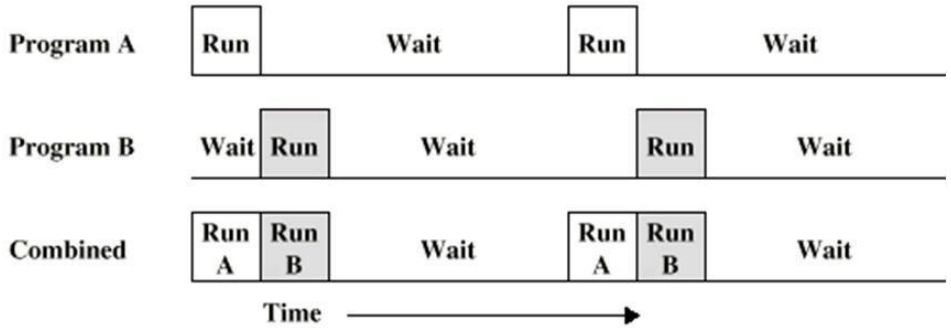
- Memory protection
- Timer
- Privileged instructions
- Interrupts

- A user program executes in a user mode, in which certain areas of memory are protected from the user's use and in which certain instructions may not be executed.
- The monitor executes in a system mode, or what has come to be called kernel mode, in which privileged instructions may be executed and in which protected areas of memory may be accessed.

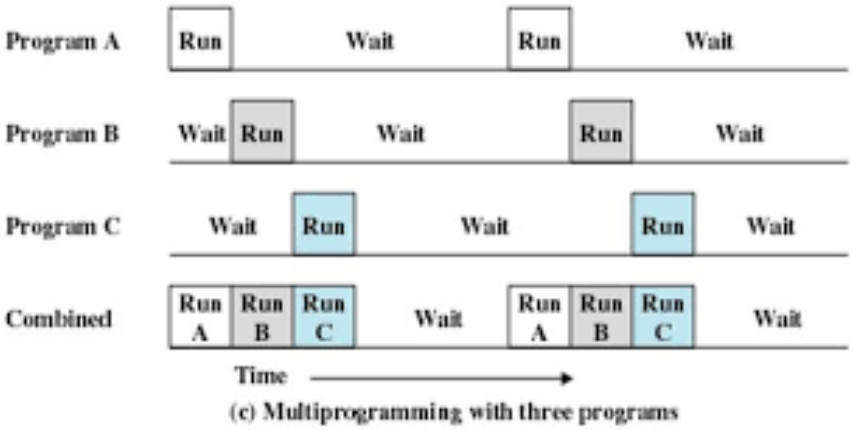
1.9.3 Multi programmed Batch Systems

- Even with the automatic job sequencing provided by a simple batch OS, the processor is often idle.
- The approach used to improve processor utilization is known as multi programming, or multitasking.
- It is the central theme of modern operating systems.





Multiprogrammed with two programs



1.9.4 Time-Sharing Systems:

- In time sharing systems the processor time is shared among multiple users.
- In a time-sharing system, multiple users simultaneously access the system through terminals, with the OS interleaving the execution of each user program in a short burst or quantum of computation.
- If there are n users actively requesting service at one time, each user will only see on the average 1/n of the effective computer capacity.

Batch Multiprogramming Vs Time Sharing systems

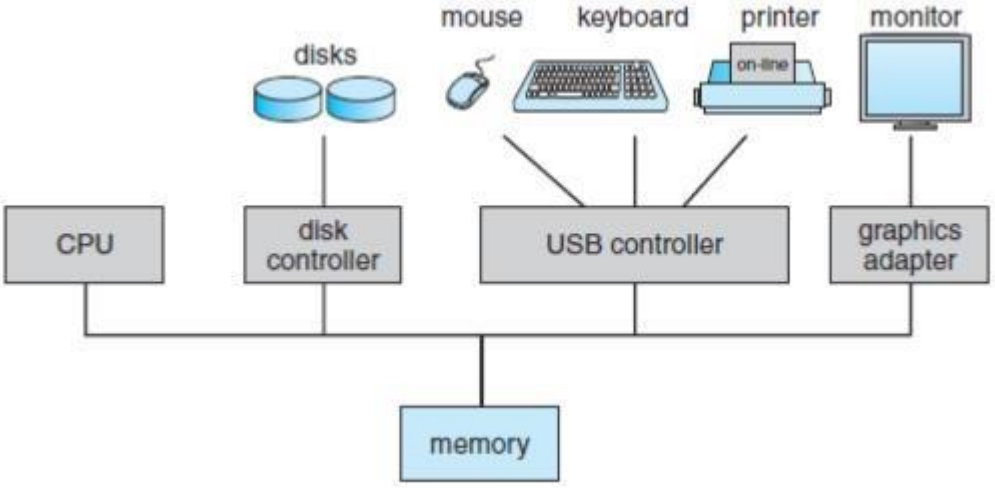
	Batch Multiprogramming	Time Sharing
Principal objective	Maximize processor use	Minimize response time
Source of directives to operating system	Job control language commands provided with the job	Commands entered at the terminal

1.10 COMPUTER SYSTEM ORGANIZATION:

Computer system organization deals with the structure of the computer system.

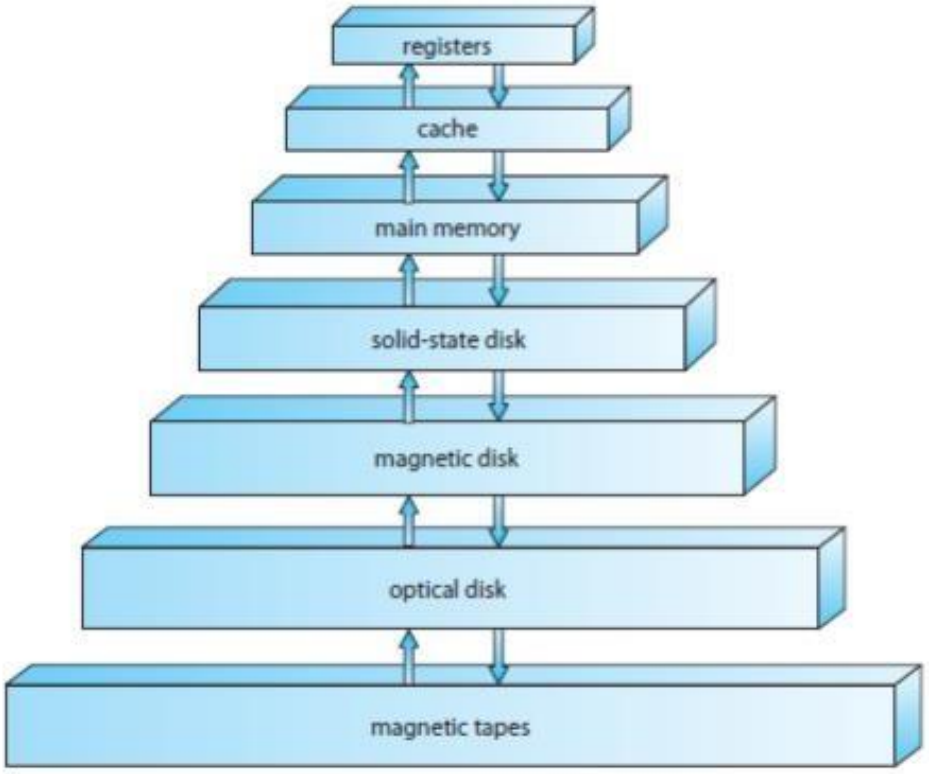
Computer system operation:

- A modern general-purpose computer system consists of one or more CPUs and a number of device controllers connected through a common bus that provides access to shared memory.
- For a computer to start running when it is powered up or rebooted—it needs to have an initial program to run. This initial program is called as the Bootstrap program.
- It is stored within the computer hardware in read-only memory (ROM) or electrically erasable programmable read-only memory (EEPROM), known by the general term firmware.
- The bootstrap loader: It initializes all aspects of the system, from CPU registers to device controllers to memory contents.
- The bootstrap program loads the operating system and start executing that system.
- Once the kernel is loaded and executing, it can start providing services to the system and its users.



Storage structure:

- The CPU can load instructions only from memory, so any programs to run must be stored in main memory.
- Main memory commonly is implemented in a semiconductor technology called dynamic random-access memory
- ROM is a read only memory that is used to store the static programs such as bootstrap loader.
- Ideally, we want the programs and data to reside in main memory permanently.
- Main memory is usually too small to store all needed programs and data permanently
- Main memory is a volatile storage device that loses its contents when power is turned off or otherwise lost.
- Most computer systems provide secondary storage as an extension of main memory.
- Volatile storage loses its contents when the power to the device is removed so that the data must be written to nonvolatile storage for safekeeping.
- Caches can be installed to improve performance.



I/O Structure:

A large portion of operating system code is dedicated to managing I/O, both because of its importance to the reliability and performance of a system.

- A general-purpose computer system consists of CPUs and multiple device controllers that are connected through a common bus. Each device controller is in charge of a specific type of device.
- The device controller is responsible for moving the data between the peripheral devices that it controls and its local buffer storage
- Operating systems have a device driver for each device controller.
- To start an I/O operation, the device driver loads the appropriate registers within the device controller.
- The controller starts the transfer of data from the device to its local buffer. Once the transfer of data is complete, the device controller informs the device driver via an interrupt that it has finished its operation. This is called as interrupt driven I/O.
- The direct memory access I/O technique transfers a block of data directly to or from its own buffer storage to memory, with no intervention by the CPU. Only one interrupt is generated per block, to tell the device driver that the operation has completed,

1.3 INTERRUPTS

Interrupt is a signal that prompts the OS to stop one process and start work on another process.

- Virtually all computers provide a mechanism by which other modules (I/O, memory) may interrupt the normal sequencing of the processor.
- Interrupts are provided primarily as a way to improve processor utilization.

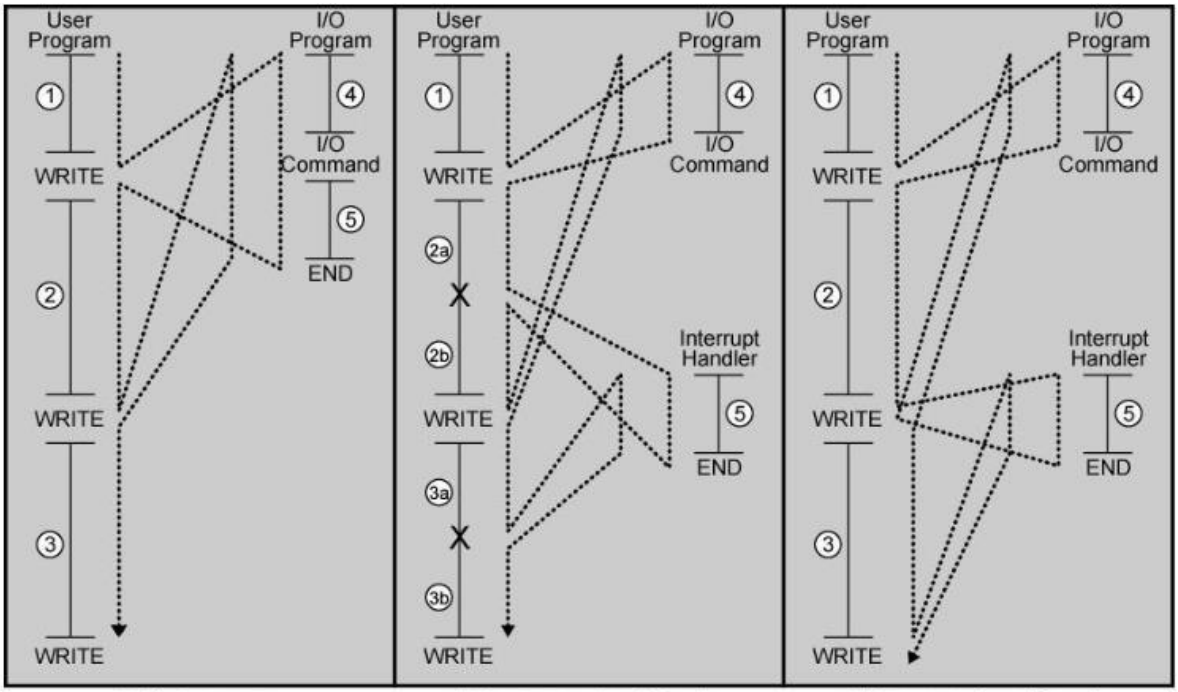
Table lists the most common classes of interrupts.

Program	Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, and reference outside a user's allowed memory space.
Timer	Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.
I/O	Generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions.
Hardware failure	Generated by a failure, such as power failure or memory parity error.

Figure (a) illustrates this state of affairs. The user program performs a series of WRITE calls interleaved with processing. The solid vertical lines represent segments of code in a program. Code segments 1, 2, and 3 refer to sequences of instructions that do not involve I/O. The WRITE calls are to an I/O routine that is a system utility and that will perform the actual I/O operation.

The I/O program consists of three sections:

- A sequence of instructions, labeled 4 in the figure, to prepare for the actual I/O operation. This may include copying the data to be output into a special buffer and preparing the parameters for a device command.
- The actual I/O command. Without the use of interrupts, once this command is issued, the program must wait for the I/O device to perform the requested function (or periodically check the status, or poll, the I/O device).
- A sequence of instructions, labeled 5 in the figure, to complete the operation. This may include setting a flag indicating the success or failure of the operation.

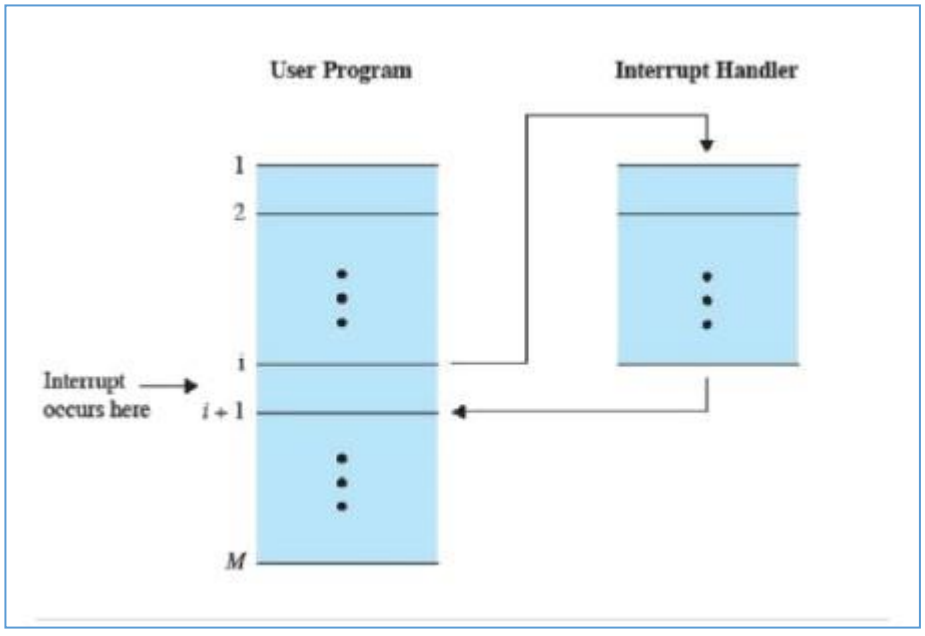


- The dashed line represents the path of execution followed by the processor; Thus, after the first WRITE instruction is encountered, the user program is interrupted and execution continues with the I/O program.
- After the I/O program execution is complete, execution resumes in the user program immediately following the WRITE instruction.
- Because the I/O operation may take a relatively long time to complete, the I/O program is hung up waiting for the operation to complete; hence, the user program is stopped at the point of the WRITE call for some considerable period of time.

Interrupts and the Instruction Cycle

- With interrupts, the processor can be engaged in executing other instructions while an I/O operation is in progress.
- Consider the flow of control in Figure b.
- As before, the user program reaches a point at which it makes a system call in the form of a WRITE call.
- The I/O program that is invoked in this case consists only of the preparation code and the actual I/O command.

- After these few instructions have been executed, control returns to the user program. Meanwhile, the external device is busy accepting data from computer memory and printing it.
- This I/O operation is conducted concurrently with the execution of instructions in the user program.
- When the external device becomes ready to be serviced, that is, when it is ready to accept more data from the processor, the I/O module for that external device sends an *interrupt request* signal to the processor. The processor responds by suspending operation of the current program; branching off to a routine to service



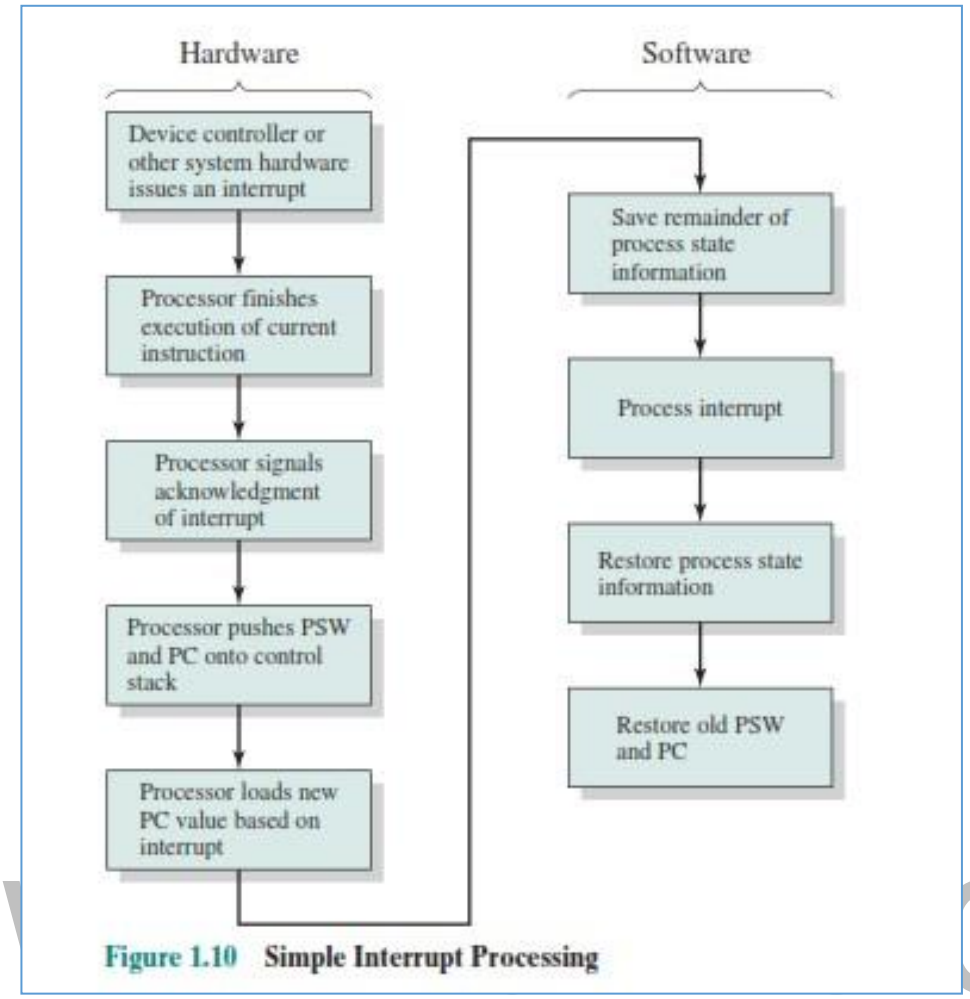


Figure 1.10 Simple Interrupt Processing

Transfer of control via Interrupts

- To accommodate interrupts, an interrupt stage is added to the instruction cycle. In the interrupt stage, the processor checks to see if any interrupts have occurred, indicated by the presence of an interrupt signal.
- If no interrupts are pending, the processor proceeds to the fetch stage and fetches the next instruction of the current program.
- If an interrupt is pending, the processor suspends execution of the current program and executes an interrupt-handler routine.
- This routine determines the nature of the interrupt and performs whatever actions are needed.

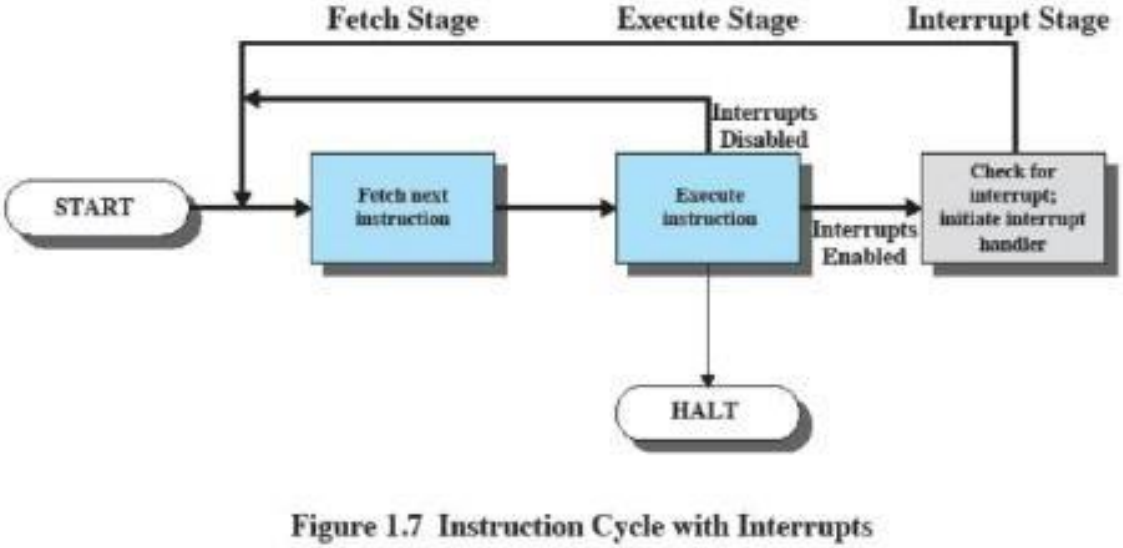


Figure 1.7 Instruction Cycle with Interrupts

1.4 MEMORY HIERARCHY

- The three key characteristics of memory: namely, capacity, access time, and cost.
- There is a trade-off among the three key characteristics of memory namely-
    - Cost
    - Capacity
    - Access time
- Memory hierarchy is employed to balance this trade-off. Memory hierarchy is the hierarchy of memory and storage devices found in a computer system. It ranges from the slowest but high capacity auxiliary memory to the fastest but low capacity cache memory.
- A typical hierarchy is illustrated in Figure. As one goes down the hierarchy, the following occur:
    a. Decreasing cost per bit
    b. Increasing capacity
    c. Increasing access time
    d. Decreasing frequency of access to the memory by the processor
- Thus, smaller, more expensive, faster memories are supplemented by larger, cheaper, slower memories.
- The key to the success of this organization is the decreasing frequency of access at lower levels.
- Suppose that the processor has access to two levels of memory. Level 1 contains 1,000 bytes and has an access time of 0.1 μs; level 2 contains 100,000 bytes and has an access time of 1 μs.
- Assume that if a byte to be accessed is in level 1, then the processor accesses it directly. If it is in level 2, then the byte is first transferred to level 1 and then accessed by the processor.

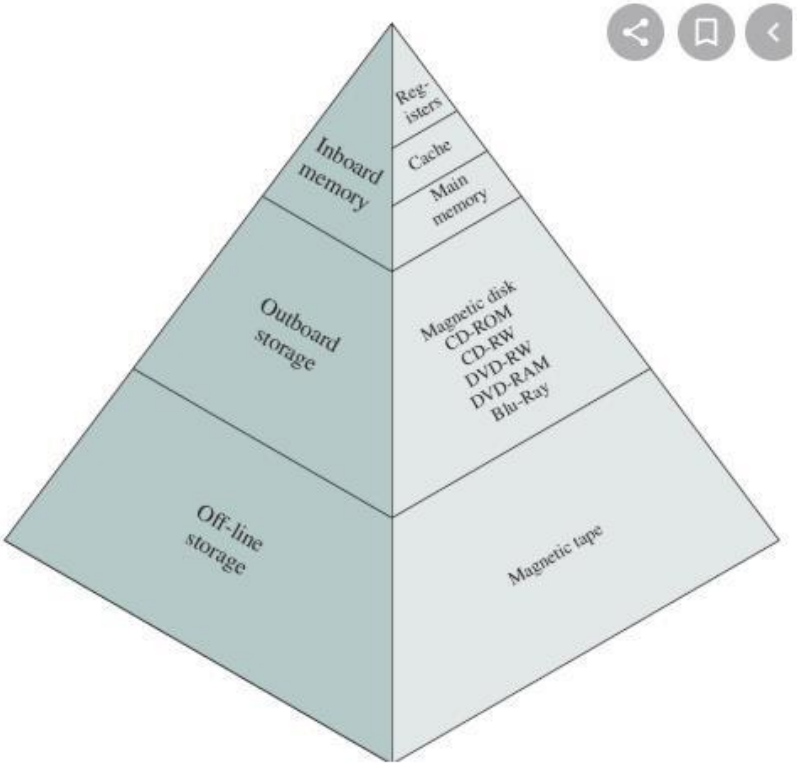


Fig: Memory Hierarchy

- Following Figure shows the general shape of the curve that models this situation. The figure shows the average access time to a two-level memory as a function of the hit ratio H , where H is defined as the fraction of all memory accesses that are found in the faster memory (e.g., the cache), T1 is the access time to level 1, and T2 is the access time to level 2.

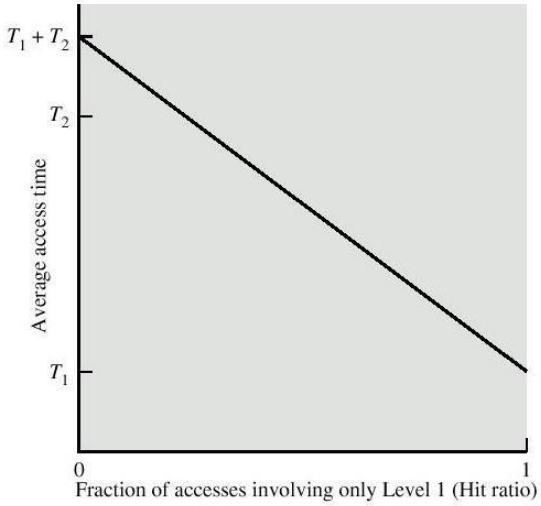


Fig: Performance of Two Level Memory

- As can be seen, for high percentages of level 1 access, the average total access time is much closer to that of level 1 than that of level 2. In our example, suppose 95% of the memory accesses are found in the cache (H = 0.95) . Then the average time to access a byte can be expressed as

$$(0.95)\,(0.1\ \mu s) + (0.05)\,(0.1\ \mu s + 1\ \mu s) = 0.095 + 0.055 = 0.15\ \mu s$$

- The result is close to the access time of the faster memory. So the strategy of using two memory levels works in principle.
- The basis for the validity of condition (Decreasing frequency of access to the memory by the processor) is a principle known as locality of reference.
- Locality of reference, also known as the principle of locality, is the tendency of a processor to access the same set of memory locations repetitively over a short period of time.
- Accordingly, it is possible to organize data across the hierarchy such that the percentage of accesses to each successively lower level is substantially less than that of the level above.
- The fastest, smallest, and most expensive type of memory consists of the registers internal to the processor.

- Skipping down two levels, main memory is the principal internal memory system of the computer. Each location in main memory has a unique address.
- Main memory is usually extended with a higher-speed, smaller cache. The cache is not usually visible to the programmer or, indeed, to the processor. It is a device for staging the movement of data between main memory and processor registers to improve performance.
- The three forms of memory just described are, typically, volatile and employ semiconductor technology.
- Data are stored more permanently on external mass storage devices, of which the most common are hard disk and removable media, such as removable disk, tape, and optical storage.
- **External, nonvolatile memory** is also referred to as **secondary memory or auxiliary memory**. These are used to store program and data files.
- A hard disk is also used to provide an extension to main memory known as **virtual memory.**
- Additional levels can be effectively added to the hierarchy in software. For example, a portion of main memory can be used as a buffer to temporarily hold data that are to be read out to disk. Such a technique, sometimes referred to as a **disk cache**, improves performance in two ways:
- Disk writes are **clustered**. Instead of many small transfers of data, we have a few large transfers of data. This improves disk performance and minimizes processor involvement.
- Some data destined for write-out may be referenced by a program before the next dump to disk. In that case, the data are retrieved rapidly from the software cache rather than slowly from the disk.

1.5 CACHE MEMORY

Cache memory is the fast and small memory placed between processor & main memory. The basic idea of using a cache is simple. When CPU need a word, it first looks in the cache. Only if the word is not there does it go to main memory.

Motivation

- The rate at which the processor can execute instructions is clearly limited by the memory cycle time (the time it takes to read one word from or write one word to memory).

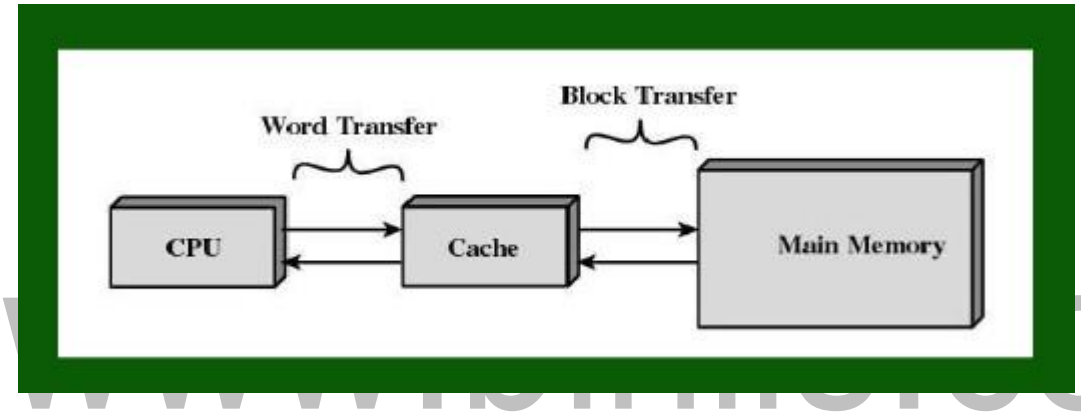
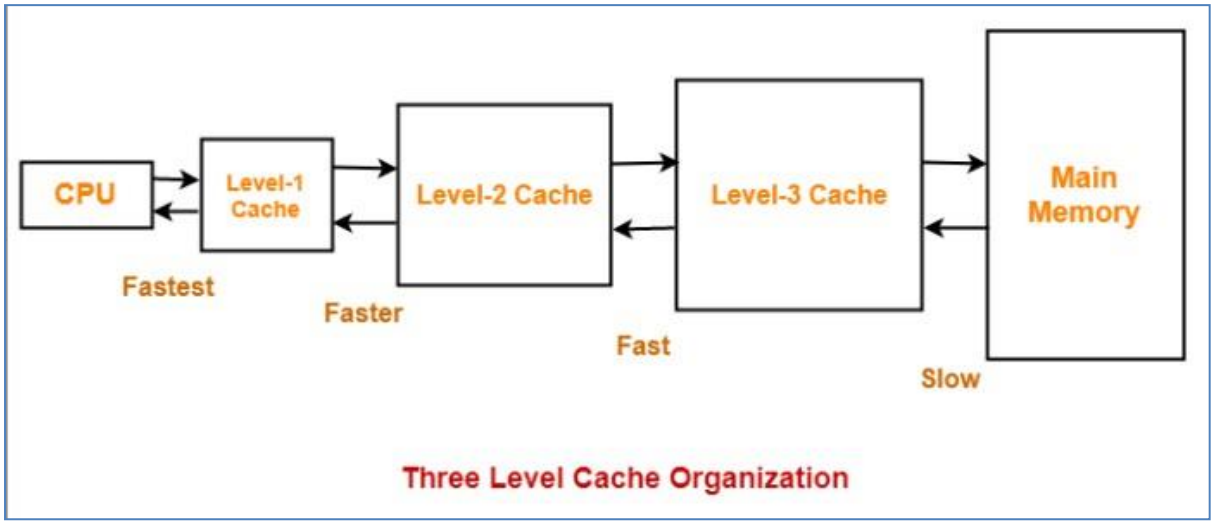


Fig: Single Cache



- This limitation has been a significant problem because of the persistent mismatch between processor and main memory speeds:
- Over the years, processor speed has consistently increased more rapidly than memory access speed.
- We are faced with a trade-off among speed, cost, and size.
- Ideally, main memory should be built with the same technology as that of the processor registers, giving memory cycle times comparable to processor cycle times.
- This has always been too expensive a strategy. The solution is to exploit the principle of locality by providing a small, fast memory between the processor and main memory, namely the cache.

Cache Principles

- The cache contains a copy of a portion of main memory.
- When the processor attempts to read a byte or word of memory, a check is made to determine if the byte or word is in the cache.
- If so, the byte or word is delivered to the processor. **(CACHE HIT)** If not, a block of main memory, consisting of some fixed number of bytes, is read into the cache and then the byte or word is delivered to the processor. **(CACHE MISS)**
- The use of multiple levels of cache:

  The L2 cache is slower and typically larger than the L1 cache, and the L3 cache is slower and typically larger than the L2 cache.

  Following Figure depicts the structure of a cache/main memory system.

- Main memory consists of up to 2^n addressable words, with each word having a unique n –bit address.
- This memory is considered to consist of a number of fixed-length blocks of K words each. That is, there are $M= 2^n/K$ blocks.
- Cache consists of C slots (also referred to as lines) of K words each, and the number of slots is considerably less than the number of main memory blocks (C<<M) (<< much less than)

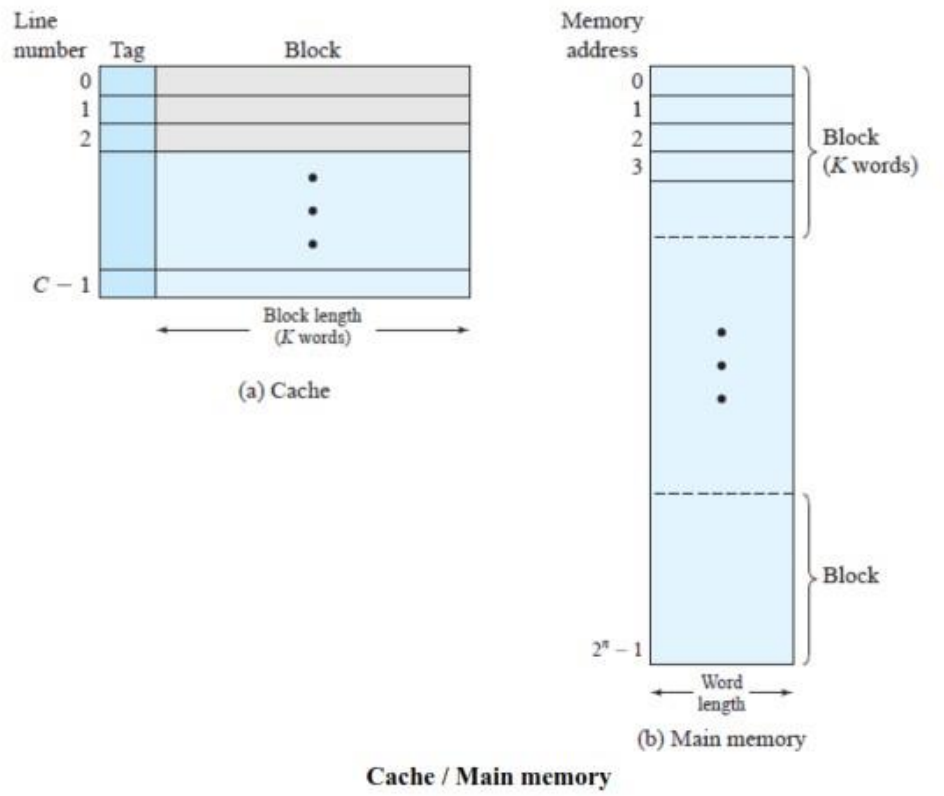
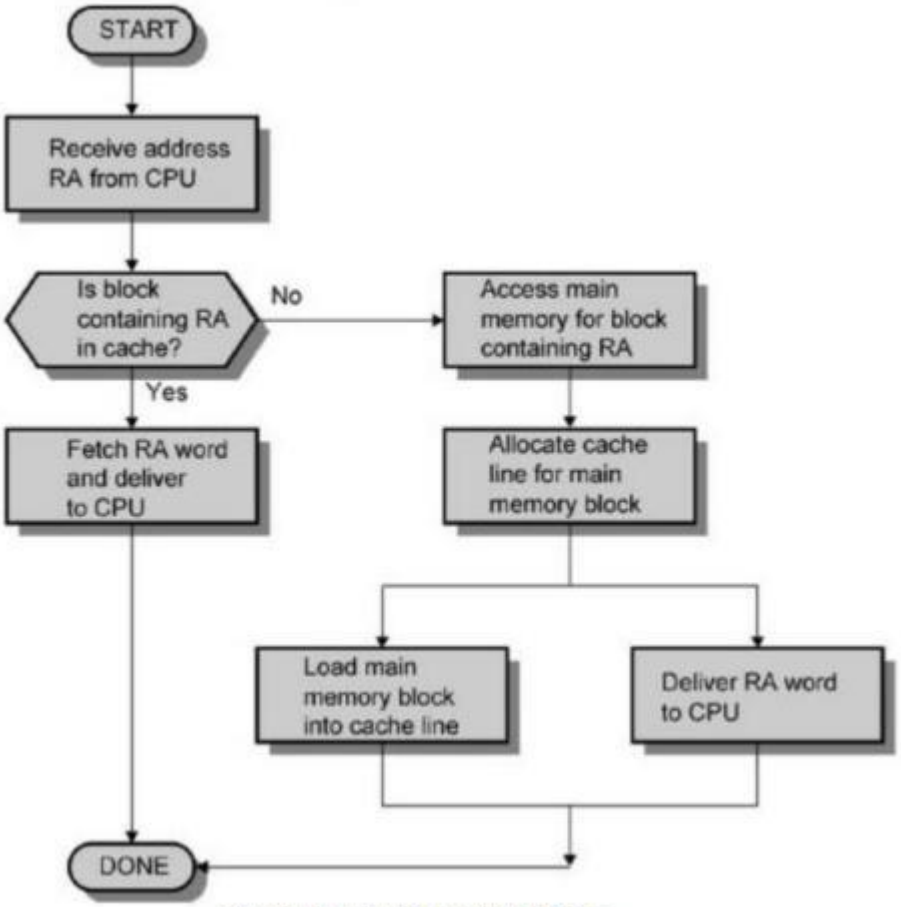


Fig : Cache - Main Memory Structure

- If a word in a block of memory that is not in the cache is read, that block is transferred to one of the slots of the cache.
- Each slot includes a tag that identifies which particular block is currently being stored.
- The tag is usually some number of higher-order bits of the address and refers to all addresses that begin with that sequence of bits.
- As a simple example, suppose that we have a 6-bit address and a 2-bit tag. The tag 01 refers to the block of locations with the following addresses: 010000, 010001, 010010, 010011, 010100, 010101, 010110, 010111, 011000, 011001, 011010, 011011, 011100, 011101, 011110, 011111.

Cache Read Operation

The processor generates the address, RA, of a word to be read. If the word is contained in the cache, it is delivered to the processor. Otherwise, the block containing that word is loaded into the cache and the word is delivered to the processor.



CACHE READ OPERATION

CACHE DESIGN

Key elements in cache design.

- Cache size
- Block size
- Mapping function
- Replacement algorithm
- Write policy
- Number of cache levels

Cache size: It turns out that reasonably small caches can have a significant impact on performance.

Block size: The unit of data exchanged between cache and main memory. As the block size increases from very small to larger sizes, the hit ratio will at first increase because of the principle of locality:

Mapping Function: When a new block of data is read into the cache, the mapping function determines which cache location the block will occupy. Two constraints affect the design of the mapping function.

First, when one block is read in, another may have to be replaced.

The more flexible the mapping function, the more scope we have to design a replacement algorithm to maximize the hit ratio.

Second, the more flexible the mapping function, the more complex is the circuitry required to search the cache to determine if a given block is in the cache.

The replacement algorithm: Effective strategy is to replace the block that has been in the cache longest with no reference to it. This policy is referred to as the least-recently-used (LRU) algorithm.

Write policy: If the contents of a block in the cache are altered, then it is necessary to write it back to main memory before replacing it. The write policy dictates when the memory write operation takes place.

At one extreme, the writing can occur every time that the block is updated. At the other extreme, the writing occurs only when the block is replaced. (Write Through, Write Back)

1.8 OPERATING SYSTEM OVERVIEW: OBJECTIVES AND FUNCTIONS

Operating system acts an interface between applications and the computer hardware.

It can be thought of as having three objectives:

- Convenience
- Efficiency
- Ability to evolve

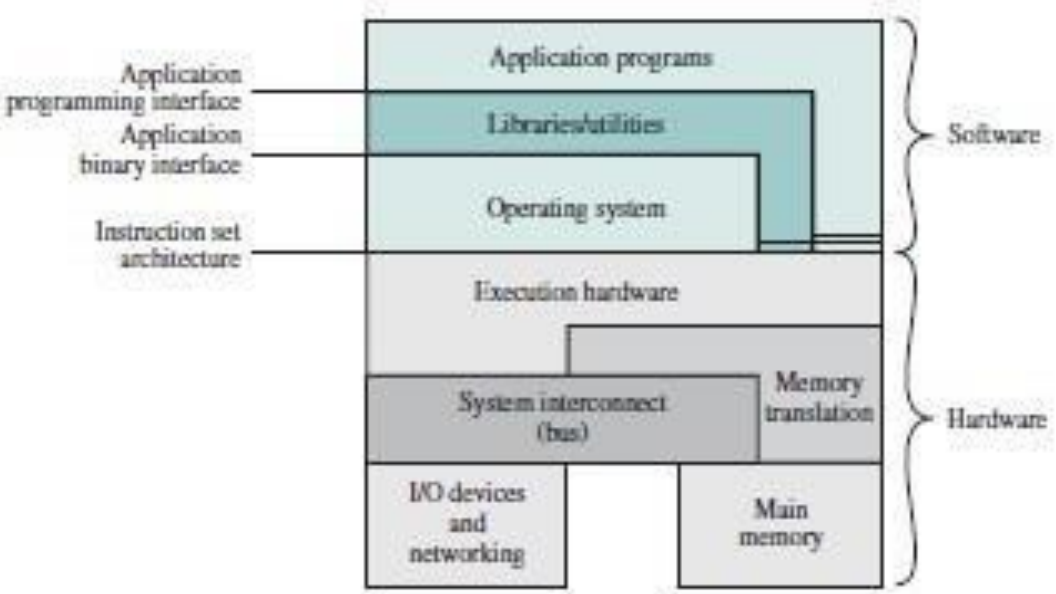
1.8.1 The Operating System as a User/Computer Interface

- The hardware and software used in providing applications to a user can be viewed in a layered or hierarchical fashion, as in Figure.
- The user of those applications, the end user, generally is not concerned with the details of computer hardware.
- Thus, the end user views a computer system in terms of a set of applications.
- An application can be expressed in a programming language and is developed by an application programmer.
- A set of system programs referred to as utilities implement frequently used functions that assist in program creation, the management of files, and the control of I/O devices.

The OS typically provides services in the following areas:

- **Program development**: The OS provides a variety of facilities and services, such as editors and debuggers, to assist the programmer in creating programs.
- **Program execution**: A number of steps need to be performed to execute a program.
- Instructions and data must be loaded into main memory, I/O devices and files must be initialized, and other resources must be prepared. The OS handles these scheduling duties for the user.
- **Access to I/O devices**: The OS provides a uniform interface that hides inner details so that programmers can access I/O devices using simple reads and writes.
- **Controlled access to files**: In the case of a system with multiple users, the OS provides a protection mechanism to control access to the files.
- **System access**: For shared or public systems, the OS controls access to the system as a whole and to specific system resources.

- **Error detection and response**: A variety of errors can occur while a computer system is running. In each case, the OS must provide a response that clears the error condition with the least impact on running applications.
- **Accounting:** A good OS will collect usage statistics for various resources and monitor performance parameters such as response time.



Three key interfaces in a typical computer system:

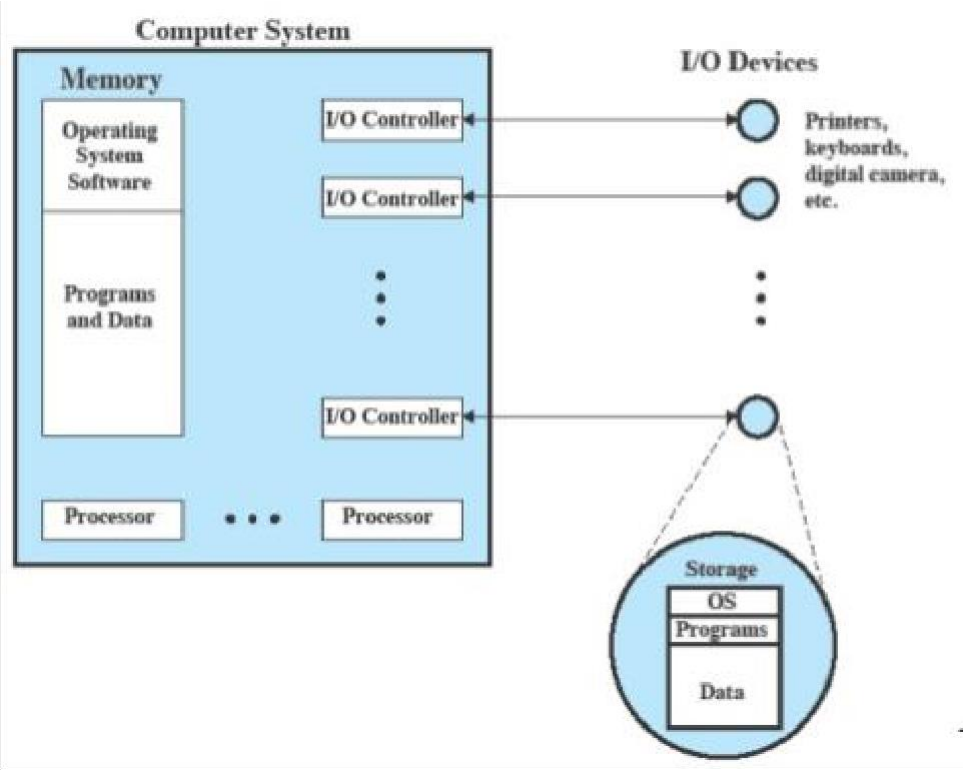
Instruction set architecture (ISA) : The ISA defines the collection of machine language instructions that a computer can follow. This interface is the boundary between hardware and software.

Application binary interface (ABI) : The ABI defines a standard for binary portability across programs. The ABI defines the system call interface to the operating system and the hardware resources and services available in a system through the user ISA.

Application programming interface (API) : The API gives a program access to the hardware resources and services available in a system through the user ISA supplemented with high-level language (HLL) library calls.

1.8.2 The Operating System as Resource Manager

- A computer is a set of resources for the movement, storage, and processing of data and for the control of these functions.
- The OS is responsible for managing these resources.



Control mechanism is unusual in two respects:

- The OS functions in the same way as ordinary computer software; that is, it is a program or suite of programs executed by the processor.
- The OS frequently relinquishes control and must depend on the processor to allow it to regain control.

1.8.3 Ease of Evolution of an Operating System

A major OS will evolve over time for a number of reasons:

- Hardware upgrades plus new types of hardware
- New services
- Fixes

1.11 OPERATING SYSTEM STRUCTURE:

The operating systems are large and complex. A common approach is to partition the task into small components, or modules, rather than have one monolithic system.

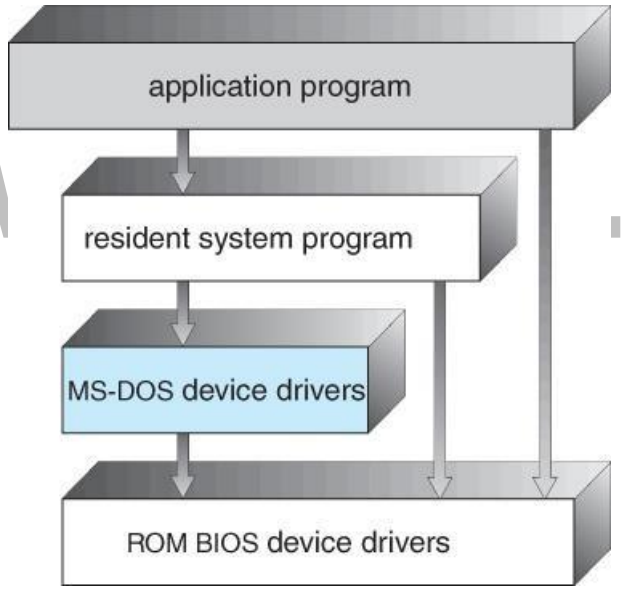
The structure of an operating system can be defined the following structures.

- Simple structure
- Layered approach
- Microkernels
- Modules
- Hybrid systems

Simple structure:

The Simple structured operating systems do not have a well-defined structure. These systems will be simple, small and limited systems.

Example: MS-DOS.

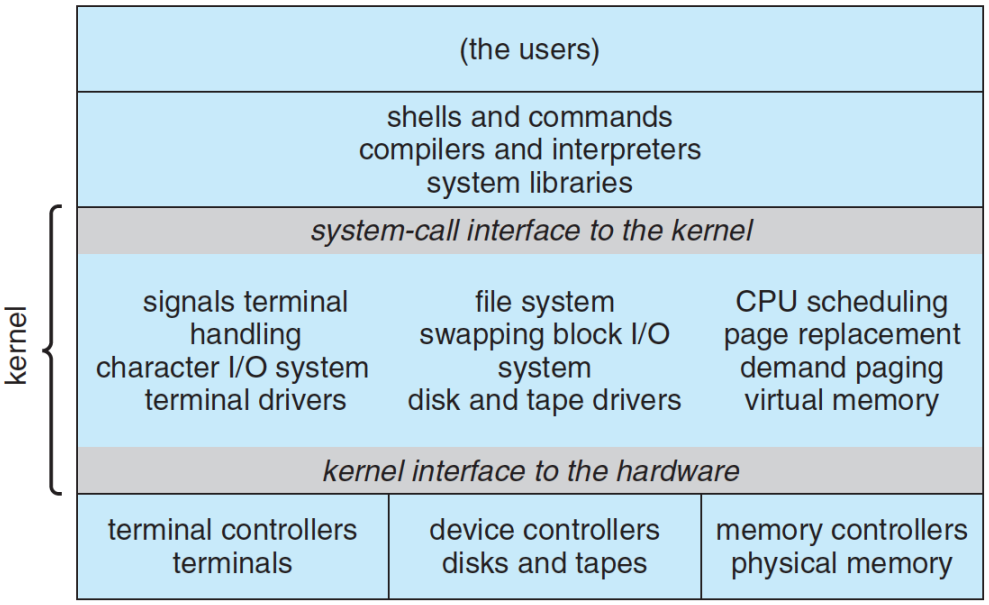


- In MS-DOS, the interfaces and levels of functionality are not well separated.
- In MS-DOS application programs are able to access the basic I/O routines. This causes the entire systems to be crashed when user programs fail.

Example: Traditional UNIX OS

It consists of two separable parts: the kernel and the system programs.

- The kernel is further separated into a series of interfaces and device drivers
- The kernel provides the file system, CPU scheduling, memory management, and other operating-system functions through system calls.



Layered approach:

- A system can be made modular in many ways. One method is the layered approach, in which the operating system is broken into a number of layers (levels). The bottom layer (layer 0) is the hardware; the highest layer is the user interface.
- An operating-system layer is an implementation of an abstract object made up of data and the operations that can manipulate those data.
- The main advantage of the layered approach is simplicity of construction and debugging. The layers are selected so that each uses functions (operations) and services of only lower-level layers.
- Each layer is implemented only with operations provided by lower-level layers. A layer does not need to know how these operations are implemented; it needs to know only what these operations do.
- The major difficulty with the layered approach involves appropriately defining the various layers because a layer can use only lower-level layers.
- A problem with layered implementations is that they tend to be less efficient than other types.

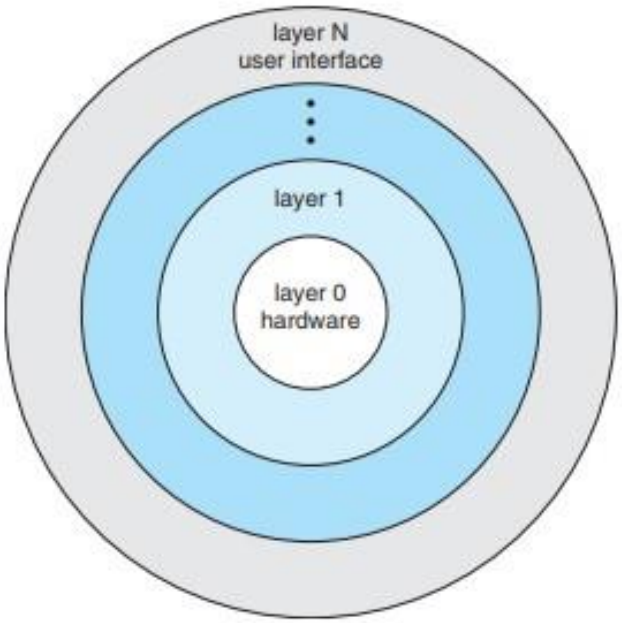


Fig : Layered Approach

Microkernels:

- In the mid-1980s, researchers at Carnegie Mellon University developed an operating system called Mach that modularized the kernel using the microkernel approach.
- This method structures the operating system by removing all nonessential components from the kernel and implementing them as system and user-level programs.
- Microkernel provide minimal process and memory management, in addition to a communication facility.
- The main function of the microkernel is to provide communication between the client program and the various services that are also running in user space.
- The client program and service never interact directly. Rather, they communicate indirectly by exchanging messages with the microkernel.
- One benefit of the microkernel approach is that it makes extending the operating system easier. All new services are added to user space and consequently do not require modification of the kernel.
- The performance of microkernel can suffer due to increased system-function overhead.

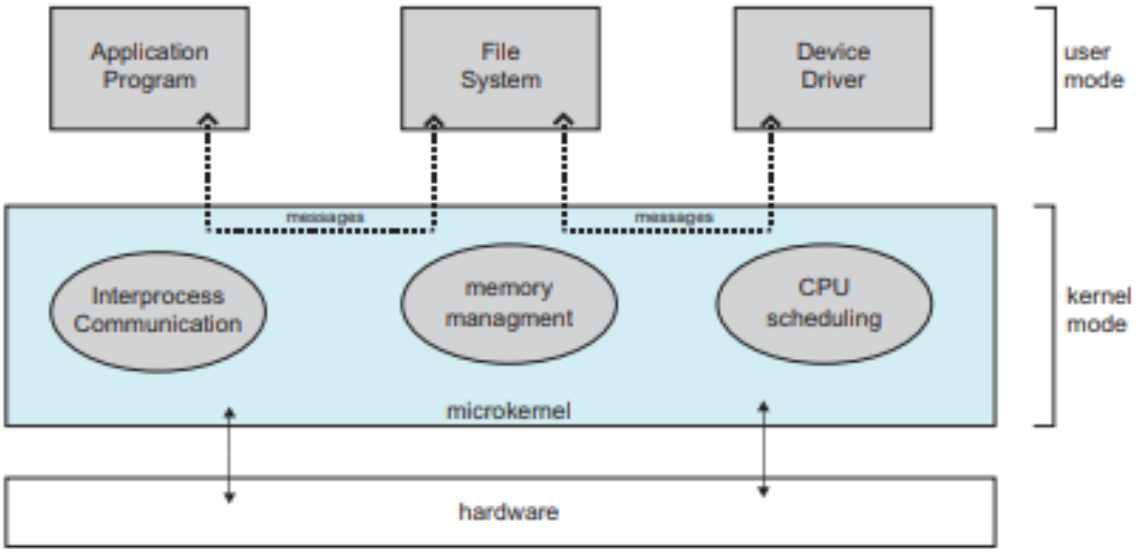


Fig : Architecture of typical microkernel

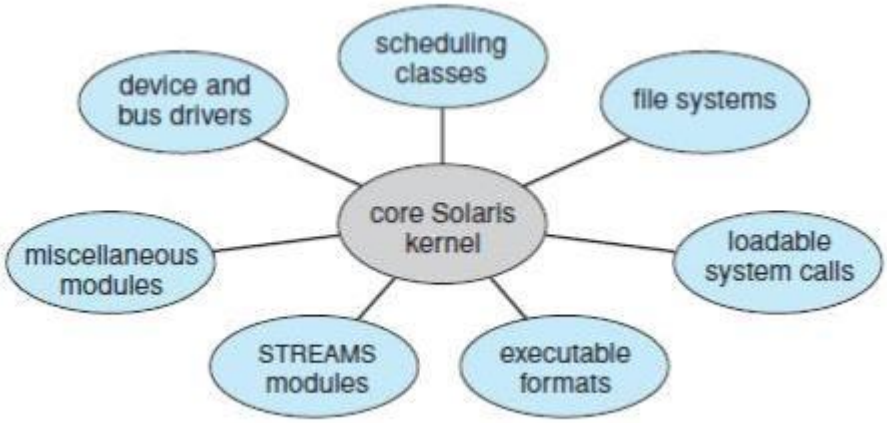
Modules:

- The best current methodology for operating-system design involves using loadable kernel modules
- The kernel has a set of core components and links in additional services via modules, either at boot time or during run time.
- The kernel provides core services while other services are implemented dynamically, as the kernel is running.
- Linking services dynamically is more comfortable than adding new features directly to the kernel, which would require recompiling the kernel every time a change was made.

Example: Solaris OS

The Solaris operating system structure is organized around a core kernel with seven types of loadable kernel modules:

- Scheduling classes
- File systems
- Loadable system calls
- Executable formats
- STREAMS modules
- Miscellaneous
- Device and bus drivers



Hybrid Systems:

- The Operating System combines different structures, resulting in hybrid systems that address performance, security, and usability issues.
- They are monolithic, because having the operating system in a single address space provides very efficient performance.
- However, they are also modular, so that new functionality can be dynamically added to the kernel.

Example: Linux and Solaris are monolithic (simple) and also modular, IOS.

1.12 OPERATING-SYSTEM OPERATIONS

Interrupt-driven nature of modern OS requires that erroneous processes not be able to disturb anything else.

Dual-Mode and Multimode Operation

- User mode when executing harmless code in user applications
- Kernel mode (a.k.a. system mode, supervisor mode, privileged mode) when executing potentially dangerous code in the system kernel.
- Certain machine instructions (privileged instructions) can only be executed in kernel mode.
- Kernel mode can only be entered by making system calls. User code cannot flip the mode switch.
- Modern computers support dual-mode operation in hardware, and therefore most modern OSes support dual-mode operation.

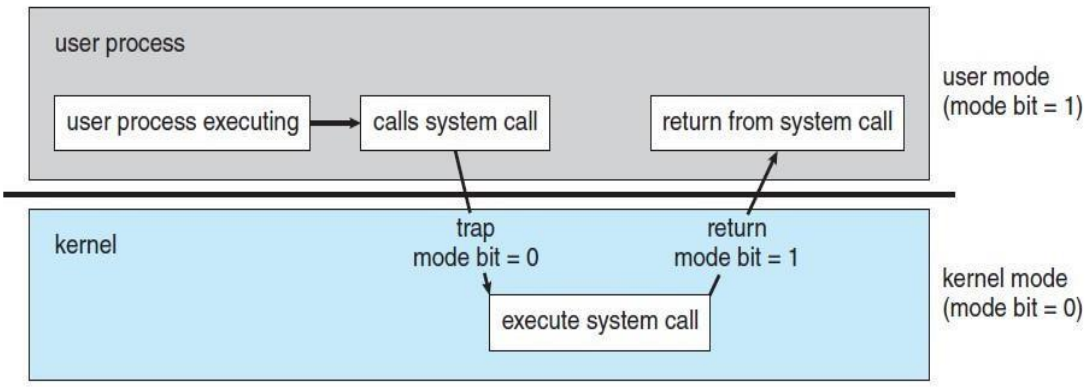


Fig : Transition from user to kernel mode

- The concept of modes can be extended beyond two, requiring more than a single mode bit
- CPUs that support virtualization use one of these extra bits to indicate when the virtual machine manager, VMM, is in control of the system. The VMM has more privileges than ordinary user programs, but not so many as the full kernel.
- System calls are typically implemented in the form of software interrupts, which causes the hardware's interrupt handler to transfer control over to an appropriate interrupt handler, which is part of the operating system, switching the mode bit to kernel mode in the process.
- The interrupt handler checks exactly which interrupt was generated, checks additional parameters (generally passed through registers) if appropriate, and then calls the appropriate kernel service routine to handle the service requested by the system call.
- User programs' attempts to execute illegal instructions (privileged or non- existent instructions), or to access forbidden memory areas, also generate software interrupts, which are trapped by the interrupt handler and control is transferred to the OS, which issues an appropriate error message, possibly dumps data to a log (core) file for later analysis, and then terminates the offending program.

Timer

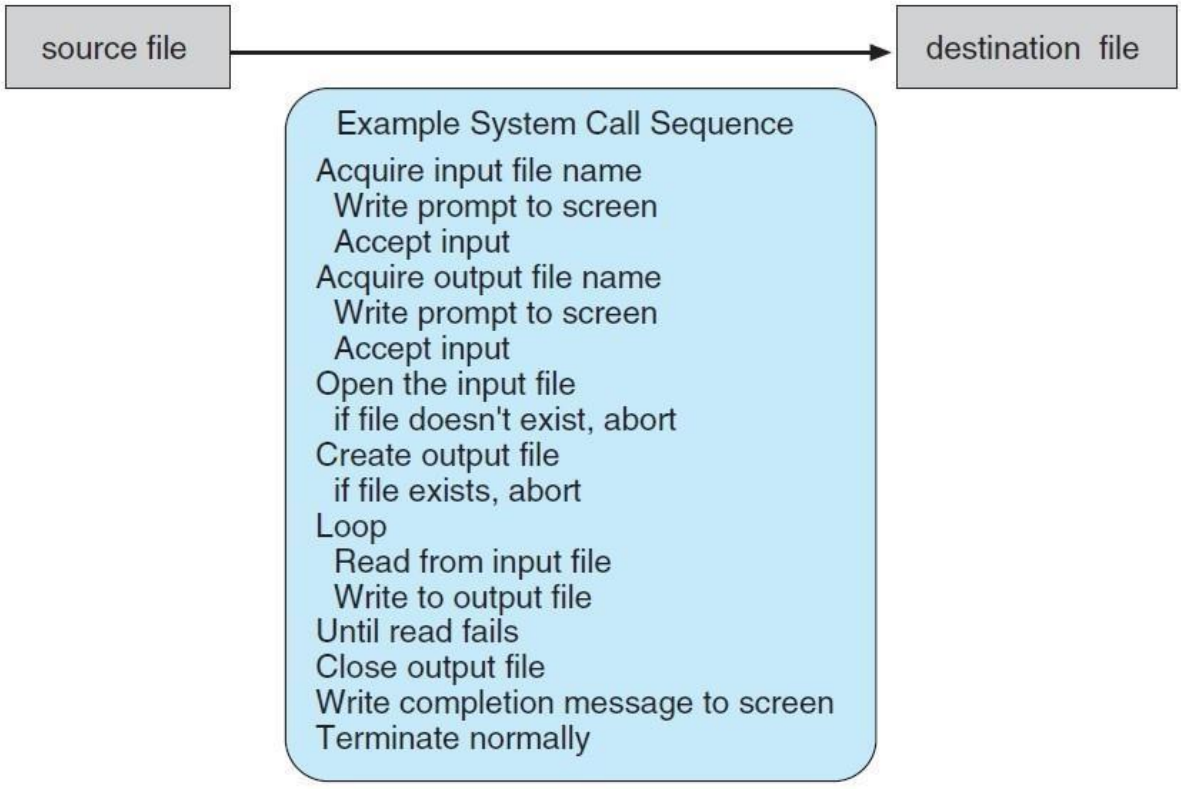
- Before the kernel begins executing user code, a timer is set to generate an interrupt.
- The timer interrupt handler reverts control back to the kernel.
- This assures that no user process can take over the system.
- Timer control is a privileged instruction, (requiring kernel mode.)

1.13 SYSTEM CALLS

- System calls provide a means for user or application programs to call upon the services of the operating system.
- Generally written in C or C++, although some are written in assembly for optimal performance.

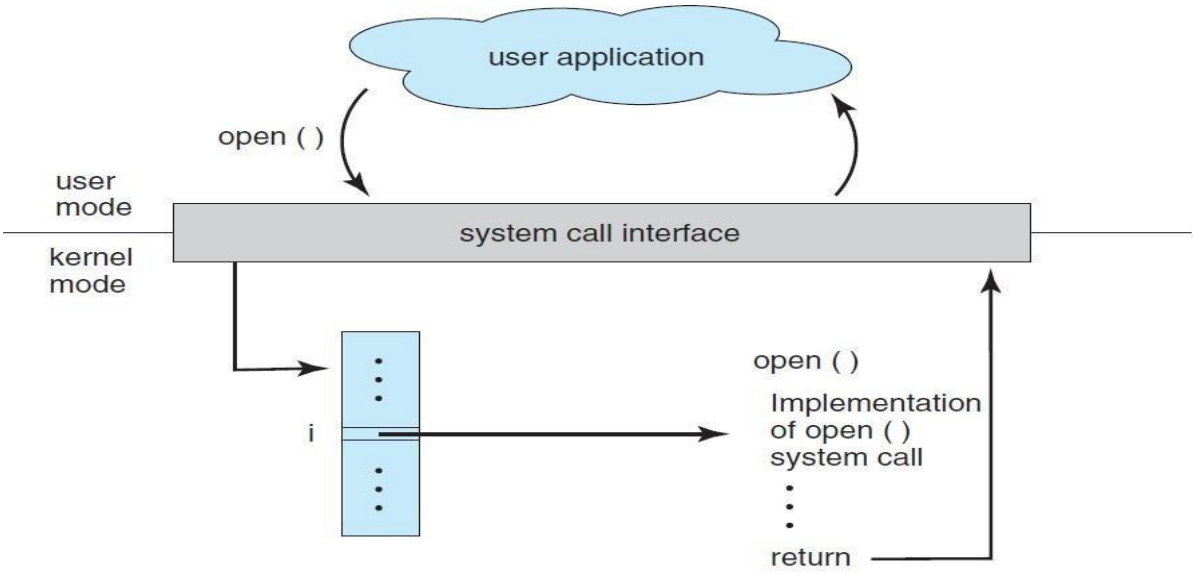
Definition

The interface between a process and an operating system is provided by system calls. In general, system calls are available as assembly language instructions. To call upon the services of the operating system we use system calls.



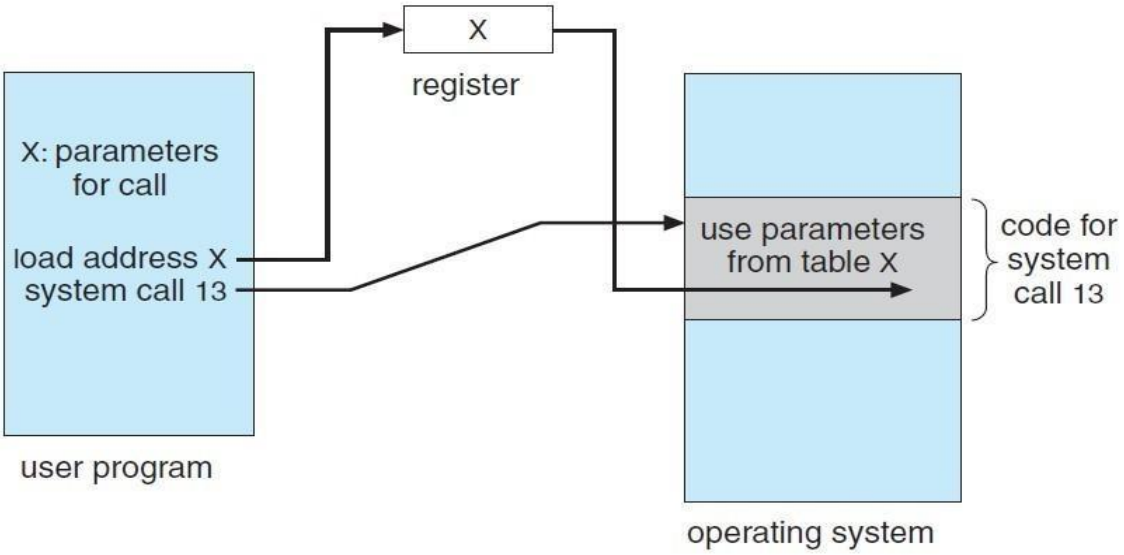
Example of how system calls are used.

- You can use "strace" to see more examples of the large number of system calls invoked by a single simple command.
- The API then makes the appropriate system calls through the system call interface, using a table lookup to access specific numbered system calls, as shown in figure



The handling of a user application invoking the open( ) system call

- Parameters are generally passed to system calls via registers, or less commonly, by values pushed onto the stack. Large blocks of data are generally accessed indirectly, through a memory address passed in a register or on the stack, as shown in Figure



Passing of parameters as a table

Types of System Calls

There are six major categories

1. Process Control:

- Process control system calls include end, abort, load, execute, create process, terminate process, get/set process attributes, wait for time or event, signal event, and allocate and free memory.
- Processes must be created, launched, monitored, paused, resumed,and eventually stopped.
- When one process pauses or stops, then another must be launched or resumed
- When processes stop abnormally it may be necessary to provide core dumps and/or other diagnostic or recoverytools.

2. File Management

- File management system calls include create file, delete file, open, close, read, write, reposition, get file attributes, and set file attributes.
- These operations may also be supported for directories as well as ordinary files.

3. Device Management

- Device management system calls include request device, release device, read, write, reposition, get/set device attributes, and logically attach or detach devices.
- Devices may be physical (e.g. disk drives), or virtual / abstract (e.g. files, partitions, and RAM disks).
- Some systems represent devices as special files in the file system, so that accessing the "file" calls upon the appropriate device drivers in the OS. See for example the /dev directory on any UNIX system.

4. Information Maintenance

- Information maintenance system calls include calls to get/set the time, date, system data, and process, file, or device attributes.
- Systems may also provide the ability to dump memory at any time, single step programs pausing execution after each instruction, and tracing the operation of programs, all of which can help to debug programs.

5. Communication

- Communication system calls create/delete communication connection, send/receive messages, transfer status information, and attach/detach remote devices.
  - The message passing model must support calls to:
  - Identify a remote process and/or host with which to communicate. Establish a connection between the two processes.
  - Open and close the connection as needed.
- Transmit messages along the connection.
- Wait for incoming messages, in either a blocking or non-blocking state.
- Delete the connection when no longer needed.
- The shared memory model must support calls to:
- Create and access memory that is shared amongst processes (and threads.)
- Provide locking mechanisms restricting simultaneous access.
- Free up shared memory and/or dynamically allocate it as needed.
- Message passing is simpler and easier, (particularly for inter-computer communications), and is generally appropriate for small amounts of data.
- Shared memory is faster, and is generally the better approach where large amounts of data are to be shared, (particularly when most processes are reading the data rather than writing it, or at least when only one or a small number of processes need to change any given data item.)

6. Protection

- Protection provides mechanisms for controlling which users / processes have access to which system resources.
- System calls allow the access mechanisms to be adjusted as needed, and for non-privileged users to be granted elevated access permissions under carefully controlled temporary circumstances.
- Once only of concern on multi-user systems, protection is now important on all systems, in the age of ubiquitous network connectivity.

1.14 SYSTEM PROGRAMS

- System programs provide OS functionality through separate applications, which are not part of the kernel or command interpreters.
- They are also known as system utilities or system applications.
- Most systems also ship with useful applications such as calculators and simple editors, (e.g. Notepad). Some debate arises as to the border between system and non-system applications.

System programs may be divided into these categories:

1. **File management** - programs to create, delete, copy, rename, print, list, and generally manipulate files and directories.
2. **Status information** - Utilities to check on the date, time, number of users, processes running, data logging, etc. System registries are used to store and recall configuration information for particular applications.
3. **File modification** - e.g. text editors and other tools which can change file contents.
4. **Programming-language support** - E.g. Compilers, linkers, debuggers, profilers, assemblers, library archive management, interpreters for common languages, and support for make.
5. **Program loading and execution** - loaders, dynamic loaders, overlay loaders, etc., as well as interactive debuggers.
6. **Communications** - Programs for providing connectivity between processes and users, including mail, web browsers, remote logins, file transfers, and remote command execution.
7. **Background services** - System daemons are commonly started when the system is booted and run for as long as the system is running, handling necessary services. Examples include network daemons, print servers, process schedulers, and system error monitoring services.

1.15 OPERATING-SYSTEM GENERATION

OS may be designed and built for a specific HW configuration at a specific site, but more commonly they are designed with a number of variable parameters and components, which are then configured for a particular operating environment.

- Systems sometimes need to be re-configured after the initial installation, to add additional resources, capabilities, or to tune performance, logging, or security.

Information that is needed to configure an OS include:

- What CPU(s) are installed on the system, and what optional characteristics does each have?
- How much RAM is installed? (This may be determined automatically, either at install or boot time.)
- What devices are present? The OS needs to determine which device drivers to include, as well as some device-specific characteristics and parameters.
- What OS options are desired, and what values to set for particular OS parameters.

The latter may include the size of the open file table, the number of buffers to use, process scheduling (priority) parameters, disk scheduling algorithms, number of slots in the process table, etc.

2 Options

- At one extreme the OS source code can be edited, re-compiled, and linked into a new kernel.
- At the other extreme a system configuration may be entirely defined by table data, in which case the "rebuilding" of the system merely requires editing data tables.
- Once a system has been regenerated, it is usually required to reboot the system to activate the new kernel. Because there are possibilities for errors, most systems provide some mechanism for booting to older or alternate kernels.

1.16 System Boot

The procedure of starting a computer by loading the kernel is known as booting the system.

- A small piece of code known as the bootstrap program or bootstrap loader locates the kernel, loads it into main memory, and starts its execution.
- First a simple bootstrap loader fetches a more complex boot program from disk
- A complex boot program loads the OS
- The bootstrap program can perform a variety of tasks. Usually, one task is to run diagnostics to determine the state of the machine.
- It can also initialize all aspects of the system, from CPU registers to device controllers and the contents of main memory and then it starts the Operating system.

- All forms of ROM are also known as firmware, since their characteristics fall somewhere between those of hardware and those of software.
- A problem with firmware in general is that executing code there is slower than executing code in RAM.
  Some systems store the operating system in firmware and copy it to RAM for fast execution.
- A final issue with firmware is that it is relatively expensive, so usually only small amounts are available.
- For large operating systems the bootstrap loader is stored in firmware, and the operating system is on disk.
- The Bootstrap program has a piece of code that can read a single block at a fixed location from disk into memory and execute the code from that Boot block.
- The program stored in the boot block may be sophisticated enough to load the entire operating system into memory and begin its execution.
- A disk that has a Boot partition is called as a Boot Disk.
- GRUB is an example of an open-source bootstrap program for Linux systems.