SUNBEAM

Institute of Information Technology

Operating System concepts

CONTENTS

Why Study Computer Architecture and Organization?

- The Computer lies at the heart of computing. Without it, most of the computing disciplines today would be a branch of theoratical mathematics.
- To be a professional in any field of computing today, one should not regard the computer as just a black box that executes programs by magic.
- All students of computing should acquire some understanding and appreciation of a computer system's functional components, their characteristics, their performance, and their interactions.
- There are practical implications as well. Students need to understand computer architecture in order to structure a program so that it runs more efficiently on a real machine.
- In selecting a system to use, they should be able to understand the tradeoff among various components, such as CPU clock speed vs. memory size.

What is Computer?

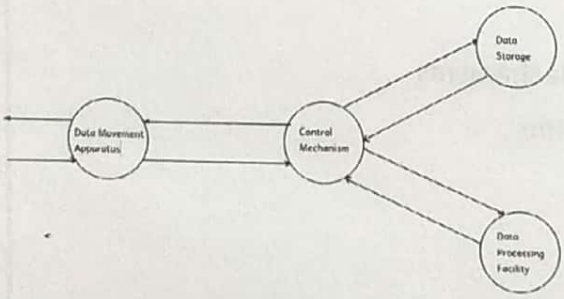
A Computer is a **complex hardware system**; Computer contains millions of elementary electronics components.

A Computer System, like any system, consists of an interrelated set of components. The system is best characterized in terms of **Structure**-the way in which the components are interconnected, and **Function**-the operation of each individual components as part of the structure.

Computer Architecture is refers to **those attributes of a system visible to a programmer or those attributes that have the direct impact on the logical execution of a program.**

e.g. the instruction set, the no. of bits used to represent various data types (e.g. numbers, characters), IO mechanism and techniques for addressing memory.

Computer Organization refers to **the operational units and their interconnections that realize the architecture specifications.** e.g. control signals, interfaces between the computer and peripherals, and memory technologies used



Functional view of Computer:

Basic functions that computer can perform:

1. Data processing
2. Data storage
3. Data movement
4. Control

- The computer must be able to **process data**. The data may take a wide variety of forms, and the range of processing requirements is broad.
- It also essential that a computer **store data**, computer must store data temporarily as well as permanently.
- The computer must be able to **move data** between itself and the outside world. When data are received from or delivered to a device that is directly connected to the computer, the process is known as **input-output(I/O)**, and the device is referred to as a **peripheral**.
- When data are moved along longer distances, to or from a remote device, the process is known as **data communications.**
- There must be **control** of these three functions. Within the computer, a control unit manages the computer's resources and coordinates the performance of its functional parts in response to those instructions.

Top level structure of the Computer:

Computer System: Major components are Processor(CPU), Main Memory(M), I/O equipments(IO).

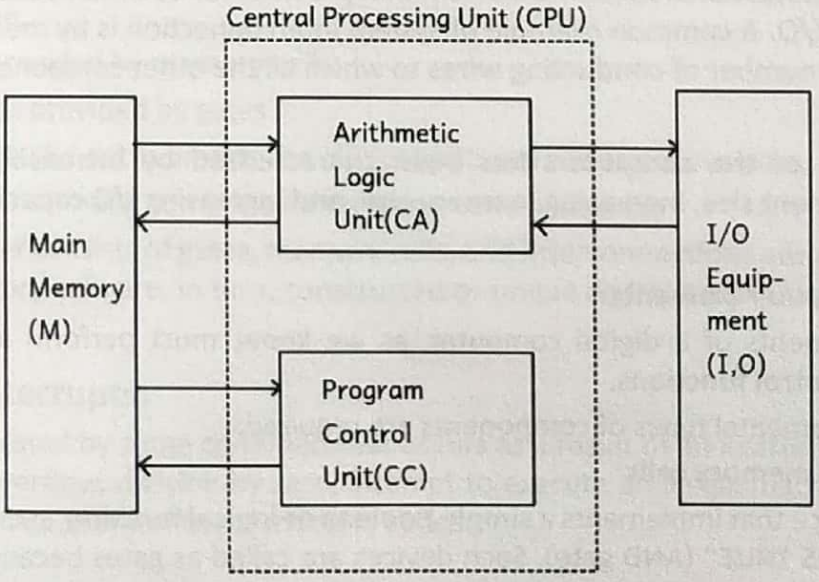


Figure 1.1

I. **Central Processing Unit (CPU):controls the operations** of the computer and **performs its data processing functions**; often simply referred to as a **processor.** Major components of CPU are

Arithmetic Logic Unit(ALU), Registers, Control Unit(CU), and **Instruction Execution Unit.**

An Arithmetic and Logical Unit (ALU) capable of operating on binary data. A Program **Control Unit(CU): provide control signals for the operations and coordination of all processor components**, which interprets the instructions in memory and causes them to execute.

CU (Control Unit) and **ALU(Arithmetic Logic Unit)** contains **the storage locations**, called as **registers.**

1. **Memory Buffer Register(MBR):** Contains a word to be stored in memory or sent to the I/O unit, or is used to receive a word from memory or from the I/O unit.
2. **Memory Address Register(MAR):** Specifies the address in memory of the word to be written from or read into MBR.
3. **Instruction Register(IR):** Contains 8-bit opcode, **instruction being executed.**
4. **Instruction Buffer Register(IBR):** Employed to hold temporarily the right hand instruction-pair to be fetched from memory.
5. **Program Counter(PC):** Contains **the address of the next instruction-pair to be fetched from memory.**
6. **Accumulator(AC) & Multiplier Quotient(MQ):** Employed to hold temporarily operands and results of ALU operations.

II. **Main Memory(M):** stores both data and instructions.

III. **I/O:** moves data between the **computer** and its **external environments.**

IV. **System Interconnections:** some mechanism that provides for communication among **CPU, main memory**, and **I/O**. A common example of system interconnection is by means of a **system bus**, consisting of a number of **conducting wires** to which all the other component attach.

The evolution of the computers has been characterized by increasing processor speed, decreasing component size, increasing memory size, and increasing I/O capacity and speed.

Fundamental Computer Elements:

The basic elements of a digital computer, as we know, must perform **storage, movement, processing, and control functions.**

Only two fundamental types of components are required:

1. gates and 2. memory cells

1. **A gate** is a device that implements a **simple Boolean or logical function**, such as "**IF A AND B ARE TRUE THEN C IS TRUE**" (AND gate). Such devices are called as gates because they control data flow in much the same way that water canal gates do.

   A **Gate** will have one or two data inputs plus a control signal input that activates the gate. When the control signal is ON, the gate performs its function on the data inputs and produces a data output.

2. **The memory cell** is a device that can store one bit of data; that is the device can be one of two stable states at any time. The memory cell will store the bit that is on its input lead when the **WRITE** control signal is **ON** and will place the bit that is in the cell on its output lead when **READ** control signal is **ON**.

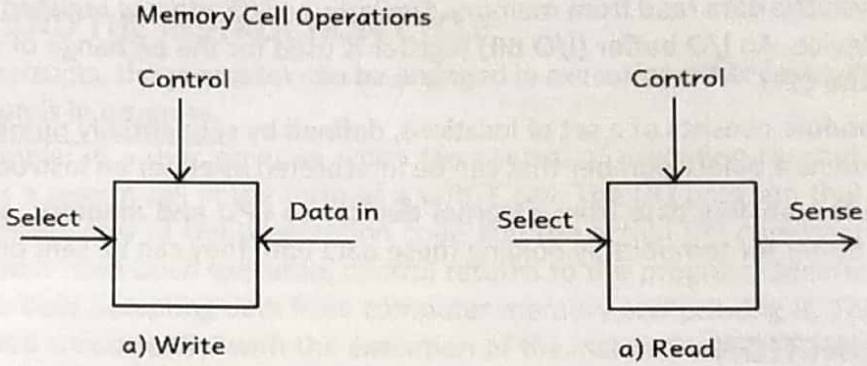


Figure 1.2

By interconnecting large number of these fundamental devices, we can construct a computer.

The computer must be able to **move data between itself and outside world.** The computer's operating environment consists of devices that serves as either sources or destinations of data. When data are received from or delivered to a device that is directly connected to the computer, the process is known as **input-output(I/O)**, and the device is referred as a **peripheral.**

When data moves over longer distances, to or from a remote device, the process is known as **data communication.**

Data storage: provided by memory cells

Data processing: provided by gates

Data movement: the paths among the components are used to move data.

Control: the paths among components can carry **control signals,**

Thus, a computer consists of **gates, memory cells,** and **interconnections among these elements.** The gates and memory cells are, in turn, constructed of simple **digital electronic components.**

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 instructions, or 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.

The CPU exchanges data with memory => For this purpose it typically make use of two internal(to the CPU) registers: a **memory address register (MAR)**, which specifies the address in memory for the next read and write, and a **memory buffer register (MBR)**, which contains the data to be written into memory or receives the data read from memory. Similarly, an **I/O address register(I/O AR)** specifies a particular I/O device. An **I/O buffer (I/O BR)** register is used for the exchange of data between **an I/O module** and **the CPU**.

A **memory module** consists of a set of locations, defined by **sequentially numbered addresses**. Each location contains a binary number that can be interpreted as either **an instruction or data**.

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

COMPUTER FUNCTION:

- The basic function performed the computer **is execution of a program**, which consists of a set of instructions stored in a memory.
- The processor does the actual work by executing instructions specified in the program.
- **Instruction processing consists of two steps: fetch cycle and execute cycle:** 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.**
- At the beginning of each instruction cycle, the processor fetches an instruction from memory.
- In a typical processor, a register called **program counter (PC)** holds **the address of the instruction to be fetched next**, unless told otherwise, the processor always increments the PC after each instruction fetch so that it will fetch the next instruction in sequence (i.e. instruction located at the next higher memory address).
- The fetched instruction is loaded into a register in the processor known as the **instruction register (IR).** The instruction contains bits that specify the action the processor is to take. The processor interprets the instruction and performs the required actions.

In general, these actions fall into four categories:

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

INTERRUPTS:

- **Interrupt is a signal received by the processor due to which processor stops the execution**

of current program and starts execution of another program.

- Virtually all Processors provides a mechanism by which **other modules (I/O, memory) may interrupt the normal processing of the processor.**
- Interrupts are provided primarily as a way **to improve processing efficiency.**

INTERRUPTS AND THE INSTRUCTION CYCLE:

- With interrupts, **the processor can be engaged in executing other instructions while an I/O operation is in progress.**
- For example: In a user program when the control of execution 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 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 the instructions in the user program. When the external device becomes ready to be serviced i.e. when it is ready to accept more data from processor, the I/O module from external device sends an **interrupt request signal** to the processor.
- The processor responds by suspending operation of the current program, **branching off to a program to service that particular I/O device, known as interrupt handler,** and resuming the original execution after the device is serviced.
- From the point of view of the user program, an interrupt is just that: **an interruption of the normal sequence of execution.** When an interrupt processing is completed, execution resumes.
- Thus the user program does not have to contain any special code to accommodates interrupts; **the processor and the operating system are responsible for suspending the user program and then resuming at the same point.**
- To accommodate interrupts, an **interrupt cycle** is added to the instruction cycle. In the interrupt cycle, 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 cycle and fetches the next instruction of the current program. **If the interrupt is pending the processor does the following:**
  - -> It suspends execution of current program being executed and saves its context. This means saving the address of the next instruction to be executed (current contents of the program counter) and any other data relevant to the processor's current activity.
  - -> It sets the program counter to the starting address of an **interrupt handler** routine.
  - -> The processor now proceeds to the fetch cycle and fetches the first instruction in the interrupt handler program, which will service the interrupt.
  - -> The **interrupt handler program** is generally part of the operating system, typically this program **determines the nature of the interrupt** and **performs whatever actions needed.**

INTERCONNECTION STRUCTURE:

- A computer consists of set of components or **modules** of three basic types **(processor, memory, I/O)** that communicates with each other.
- In effect, **a computer is a network of basic modules**. Thus, there must be paths for connecting the modules.
- The collection of paths connecting the various modules is called the **interconnection structure**.
- **Memory:** Typically, a **memory module** consists of N words of equal length. Each word is assigned a unique numerical address **(0,1,.....,N-1)**. **A word of data can be read from or written into the memory.** The nature of the operation is indicated by read and write control signals. The location for the operation is specified by an address.
- **I/O module:** From an internal (to the computer system) point of view, I/O is functionally similar to memory. There are two operations, read and write. Further **an I/O module may control more than one external device. We can refer to each of the interfaces to an external device as a port and give each unique address (e.g. 0,1,...,M-1).** In addition, there are external data paths for the input and output of data with an external device. Finally, an I/O module may be able to send interrupt signals to the processor.
- **Processor:** The processor,
  -> reads an instructions and data
  -> writes out data after processing
  -> uses control signals to control overall operations of the system.
  -> it also receives interrupt signals.

The interconnection structure must support the following types of transfers:

1. **Memory to Processor:** the processor reads an instruction or a unit of data from memory.
2. **Processor to Memory:** the processor writes a unit of data to memory.
3. **I/O to Processor:** the processor reads data from an I/O device via an I/O module.
4. **Processor to I/O:** the processor sends data to the I/O device.
5. **I/O to Memory and vice-versa:** For these two cases, an I/O module is allowed to exchange data directly with memory, without going through the processor, using **direct memory access (DMA).**

BUS INTERCONNECTION:

- **A bus** is a communication pathway connecting two or more devices.
- A key characteristics of a bus is that **it is a shared transmission medium**: multiple devices connect to the bus, and a signal transmitted by any one device is available for reception by all other devices attached to the bus.
- Typically, a bus consists of multiple communication pathways, or **lines**. Each line is capable of transmitting signals representing binary 1 and binary 0. Over time, a sequence of binary digits can be transmitted across single line. Taken together, several lines of a bus can be used to transmit binary digits simultaneously(in parallel). e.g. an 8-bit unit of data can be

transmitted over eight bus lines.

- **A bus that connects major computer components (processor, memory, I/O) is called a system bus.** The most common computer interconnection structures are based on the use of one or more system buses.
- A system bus consist, typically, of from about **50 to hundreds of separate lines**. Each line is assigned a particular meaning or function.
- The data bus may consists of **32, 64, 128 or even more separate lines**. The number of lines being referred to as the **width** of the data bus.
- Because each line can carry only 1 bit at a time, **the number of lines determines how many bits can be transferred at a time.**
- The width of a **data bus** is a key factor in determining **overall system performance.**
- The **address lines** are use to designate the source or destination of the data on the data bus.
- The **control lines** are used to control the access to and the use of the data and address lines.
- **Control signals** transmit both **command** and **timing** information among system modules. **Timing signals** indicate the validity of data and address information. **Command signals** specify operations to be performed.

Typical control lines include:

1. **Memory write:** causes data on the data bus to be written into the addressed location.
2. **Memory read:** causes data from the addressed location to be placed onto the bus.
3. **I/O write:** causes data on the data bus to be output to the addressed I/O port.
4. **I/O read:** causes data from the addressed I/O port to be placed on the bus.
5. **Transfer ACK:** indicates that the data have been accepted from or placed on the bus.
6. **Bus request:** indicates that a module needs to gain control of the bus.
7. **Bus grant:** indicates that a requesting module has been granted control of the bus.
8. **Interrupt request:** indicates that an interrupt is pending
9. **Interrupt ACK:** acknowledges that the pending interrupt has been recognized.
10. **Clock:** it is used to synchronize operations
11. **Reset:** Initializes all modules.

- Physically, **a system bus is actually a number of parallel electrical conductors.**
- In the classic bus arrangement, these conductors are metal lines etched in a card or board(PCB).

COMPUTER MEMORY SYSTEM OVERVIEW:

- Computer memory is organized into a **hierarchy**. At highest level(closest to the processor) are the **processor/CPU registers**.
- Next comes **one or more levels of cache**, when multiple levels are used they are denoted **L1, L2**, and so on.
- Next comes **main memory,** which usually made out of **dynamic random access memory (DRAM)**.

- All of these are considered internal to the computer system.
- The hierarchy continues with **external memory**, with the next level typically being a **fixed hard disk**, and one or more levels below that consisting of **removable media** such as **optical disks and tape.**
- As one goes down the memory hierarchy, one finds **decreasing cost/bit, increasing capacity**, and **slower access time**.

Characteristics of Memory Systems:

1. **Location:** location refers to whether memory is **internal** or **external** to the computer.

   Internal memory is often equated with **main memory**, but there are other forms of internal memory. The processor requires its own local memory in the form of **registers**. **Cache** is another form of internal memory.

   External memory consists of **peripheral storage devices**, such as **disk and tape**, that are accessible to the processor via I/O controller.

2. **Capacity:** For **internal memory**, this is typically expressed in terms of **bytes (1 byte = 8 bits) or words**.

   Common **word lengths** are **8, 16 and 32 bits**. **External memory** capacity is typically expressed in terms of **bytes**.

3. **Unit of transfer:** For **internal memory** unit of transfer is equal to **the number of electrical lines into and out of the memory module.** This may be equal to the **word length**, but it is often larger, **such as 64, 128 or 256 bits.**

   Word: the natural unit of organization of memory. The size/length of word is typically equal to **the number of bits used to represent an integer** and **to the instruction length.**

   Addressable units: In some systems, the addressable unit is the word. **However many systems allow addressing at the byte level.**

   Unit of memory: For **main memory**, this is the number of bits read out of or written into memory at a time. For **external memory**, data are often transferred in much larger units than a word, and these are referred to as **blocks.**

4. **Access methods:** there are four different **method of accessing units** of data.
   1. **Sequential access:** memory is organized into units of data, called records. Access must be made **in a specific linear sequence**. e.g. tape units
   2. **Direct access:** e.g. disk units
   3. **Random access:** this is fastest access method. data can be accesses randomly. e.g. main memory and some cache systems.
   4. **Associative:** data can be access by using **"locality by reference"** principal e.g. cache memory.

   From users point of view, the two most important characteristics of memory are **Capacity and Performance**

5. **Performance:** Three performance parameters are used:

   Access time: for **random access memory:** this is the time it takes to perform a read or write operation, that is, the time from the instant that an address is presented to the memory to the instant that data have been stored or made available for use, and for **non-random access** memory, **disk access time = seek time + rotational latency.**

Memory cycle time: this concept is primarily applied to **random-access memory** and **consists of the access time plus any additional time required before a second access can commence.** Memory cycle time is concerned with the system bus, not the processor.

Transfer rate: this is the rate at which data can be transferred into or out of a memory unit. For random-access memory, it is equal to 1/cycle time.

6. **Physical Type:** A variety of physical types of memory have been employed. The most common today are **semi-conductor memory**, **magnetic surface memory**, used for disk and tape, and **optical** and **magneto optical**.
7. **Physical characteristics:**

volatile: in a volatile memory, information decays naturally or is lost when electrical power is switched off.

non-volatile: in a non-volatile memory, information once recorded remains without deterioration until deliberately changed; no electrical power is needed to retain the information. **Magnetic surface memories** are **non-volatile. Semi conductor memories** are may be either **volatile** or **non-volatile**.

Non-erasable memory cannot be altered, except by destroying the storage unit. Semi-conductor memory of this type is known as **read-only memory(ROM)**

8. **Organization:** for random access memory organization is the key design issue. By organization is meant the physical arrangement of bits to form words.

Cache Memory:

- There is relatively large slow main memory together with a small, faster cache memory.
- The cache contains **copy of portions of main memory**.
- When the processor attempts to read a word of memory, a check is made to determine if the word is in the cache. If so, the word is delivered to the processor, referred as **cache hit.**
- If not, a block of main memory consisting of some fixed number of words is read into the cache and then word is delivered to the processor, referred as **cache miss.**
- Main memory consists of up to **2^n addressable words**, with each word having a unique n-bit address. For mapping purposes, this memory is considered to consists of number of fixed length blocks of **K words** each. **That is, there are M = 2^n/K blocks in main memory.**
- The cache consists of m blocks, called as **lines**. **Each line contains K words, plus tag of few bits.**
- Each line also includes **control bits**, such as **a bit** to indicate whether the line has been modified since being loaded into the cache.
- **The length of the line not including tag and control bits, is the line size.**
- The line size may be as small as 32 bits, with each word being a single byte.

Disk Cache:

A **portion of main memory** can be used as a **buffer** to hold data temporarily that is to be read out to disk. Such a technique sometimes referred to as a disk cache. **Disk cache is generally a purely software technique.**

SEMI CONDUCTOR MAIN MEMORY:

- In earlier computers, the most common form of random-access storage for computer main memory employed **an array of doughnut-shaped ferromagnetic loops** referred to as **cores**. Hence, main memory was often referred to a core, a term that persists this days.
- The two basic forms of **semiconductor random access memory** are **dynamic RAM (DRAM)** and **static RAM (SRAM).**
- **SRAM** is **faster**, **more expensive**, and **less dense** than **DRAM.**
- **SRAM** is used for **cache memory,** whereas **DRAM** is used for **main memory.**
- The basic element of semiconductor memory is the **memory cell**.
- Although a variety of electronic technologies are used, all semi conductor memory cells shares certain properties.
- They exhibit two stable(semi stable) states, which can be used to represent binary 1 and 0.
- They are capable of being written into (at least once), to set the state.
- They are capable being read to sense the state.
- Most commonly the cell has **three functional terminals** capable of carrying an electrical signal.

1. **The select terminal:** as the name suggest selects a memory cell for a read or write operation.
2. **The control terminal** indicates read or write.
3. **For writing,** the other terminal provides an electrical signal that sets the state of the cell to 1 or 0. **For reading**, that terminal is used for output of the cell's state.

Characteristics of RAM are:

1. It is possible both to read data from the memory and to write new data into the memory easily and rapidly. Both the reading and writing are accomplished through the use of electrical signals.
2. It is volatile.

RAM is divided into technologies: **dynamic and static**

DRAM: DYNAMIC RAM: used in main memory(RAM), is made with cells that store data as charge on **capacitors**. **The presence or absence of charge in a capacitor is interpreted as a binary 1 or 0.**

SRAM: STATIC RAM: used in cache memory, a static RAM is **digital device** that uses the same logic elements used in the processor. In SRAM, binary values are stored using **traditional flip-flops logic gate** configurations.

A DRAM cell is **simpler** and **smaller** than a SRAM memory cell. Thus, a DRAM is more dense(smaller cells = more cells per unit area) and less expensive than corresponding SRAM. On the other hand **DRAM requires the supporting refresh circuitry.**

Types of ROM:

- ROM contains permanent pattern of data that cannot be changed.
- A ROM is **non-volatile**; that is no power source is required to maintain the bit values in memory.
- While it is possible to read a ROM, it is not possible to write new data into it.
- An important application of ROM is **microprogramming**. Other potential applications include: **Library subroutines for frequently wanted functions, System programs, Function tables.**

Input and Output:

- The computer **system's I/O architecture** is its interface to the outside world.
- This architecture provides **systematic means of controlling interaction with the outside world and provides the operating system with the information needs to manage I/O activity effectively.**

There are three principal I/O techniques:

1. **Programmed I/O:** in which I/O occurs under the direct and continuous control of the program requesting the I/O operation.
2. **Interrupt driven I/O:** in which program issues an I/O command and then continues to execute, until it is interrupted by the I/O hardware to signal the end of the I/O operation.
3. **Direct Memory Access (DMA):** in which specialized I/O processor takes over control of an I/O operations to move a large block of data.

Two important examples of external I/O interfaces are FireWire and InfiniBand.

1. **FireWire:** IEEE1394 is an interface standard for a serial bus for high-speed communications and isochronous(if events occurs regularly, or at equal time intervals) real-time data transfer. It was developed in 1980's by Apple, which called it FireWire
2. **InfiniBand:** (abbreviated as IB) is a computer-networking communications standards used in high-performance computing that features very high throughput and very low latency. It is used for data interconnect both among and within computers. It is also utilized as either a direct, or switched interconnect between servers and storage systems, as well as an interconnection between storage systems.

I/O modules:

- In addition to **processor** and **a set of memory modules**, the third key element of a computer system is **a set of I/O modules**.
- Each module interfaces to the system bus or central switch and controls one or more peripheral devices.
- **I/O modules contains logic for performing a communication function between the peripheral and the bus.**

The reasons why one does not connect peripherals directly to the system bus:

- There are a wide variety of peripherals with various methods of operations. It would be impractical to incorporate the necessary logic within the processor to control range of devices.
- The data transfer rate of peripherals is often much slower than that of the memory or processor. Thus, it is impractical to use high-speed system bus to communicate directly with

a peripheral.

- On the other hand the data transfer rate of some peripherals is faster than that of the memory or processor. Again the mismatch would lead to inefficiencies if not managed properly.
- Peripherals often use different data formats and word lengths than the computer to which they are attached.

Thus an I/O module is required.

This module has two major functions:

1. Interface to the processor and memory via the system bus or central switch and
2. Interface to one or more peripheral devices by tailored data links.

EXTERNAL DEVICES:

- I/O operations are accomplished through a wide assortment of external devices that provide a means of exchanging data between the external environment and the computer.
- An external device attaches to the computer by a link to an I/O module.
- The link is used **to exchange control, status,** and **data** between the I/O module and external device.
- The external device connected to an I/O module is often referred to as a **peripheral device** or simple, **peripheral.**

We can broadly classify external devices into three categories:

1. **Human readable:** suitable for communicating with the computer user
   e.g. Video Display Terminals (VDTs) and Printers.
2. **Machine readable:** suitable for communicating with the equipment
   e.g. magnetic disks and tape systems, sensors and actuators, such are used in a robotics application.
3. **Communication:** suitable for communicating with remote devices. Communication devices allow a computer to exchange data with a remote device, which may be a human readable device, such as terminal, a machine readable device, or even another computer.

Structure of External Device:

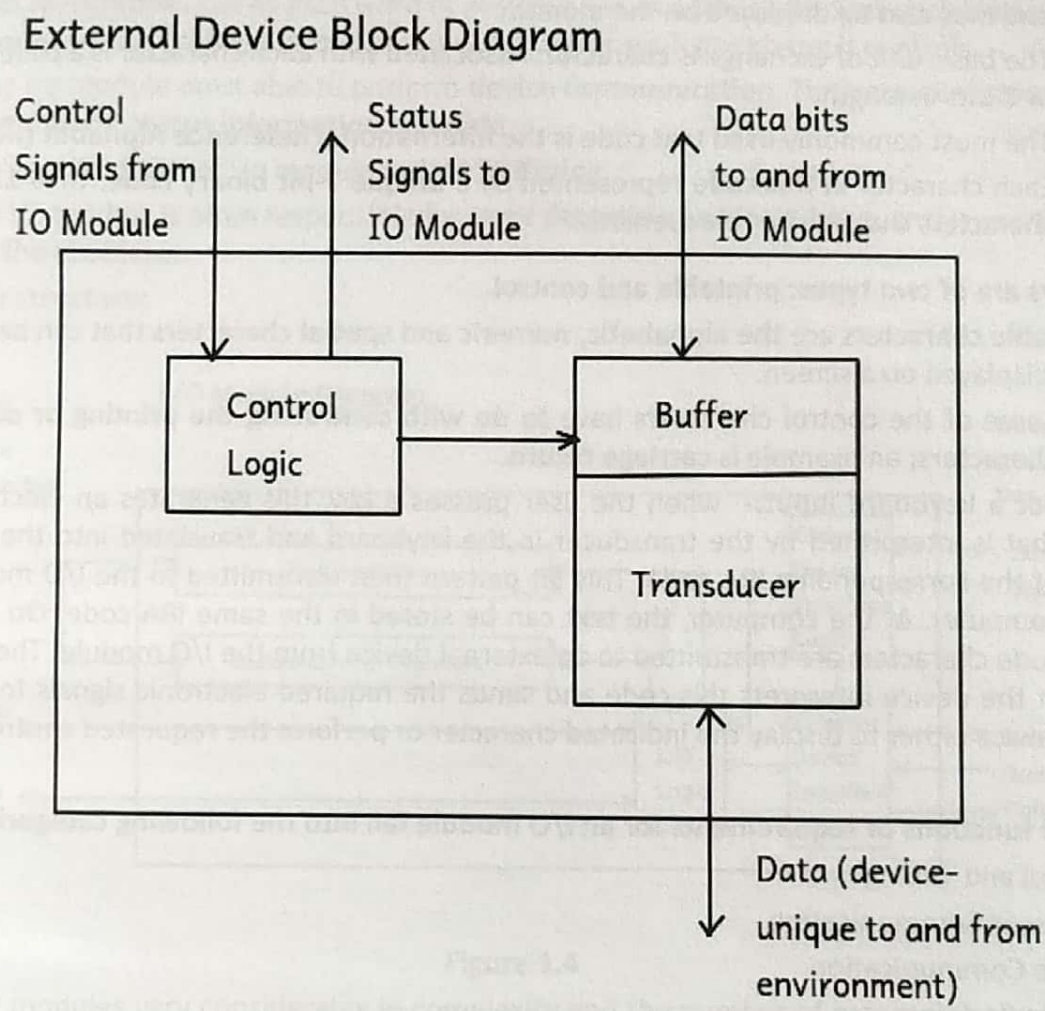


Figure 1.3

- The interface to the I/O module is in the form of **control, data,** and **status signals.** Control signals determines the function that the device will perform, such as **send data to the I/O module(INPUT/READ), accept data from the I/O module (OUTPUT or WRITE), report status,** or **perform some control function particular to the device.(e.g. position a disk head).**
- **Data** are in the form of set of bits to be sent or received from the I/O module.
- **Status signals** indicate the state of the device. e.g. **READY/NOT-READY** to show whether the device is ready for data transfer.
- **Control logic** associated with the device controls the device's operation in response to direction from the I/O module.
- The **transducer** converts data from electrical to other forms of energy during output and from other forms to electrical during input.
- Typically, a **buffer** is associated with the transducer to temporarily hold data being transferred between the I/O module and the external environment; a buffer size of 8 to 16 bit is common.

For example: Keyboard/Monitor:

- The most common means of computer/user interaction is a keyboard/monitor arrangement.
- The user provides input through keyboard. This input is then transmitted to the computer and may also be displayed on the monitor.
- The basic unit of exchange is **character**. Associated with each character is a code, typically 7 or 8 bits in length.
- The most commonly used text code is the **International Reference Alphabet (IRA).**
- Each character in this code represented by a **unique 7-bit binary code**; thus 128 different characters that can be represented.

Characters are of two types: printable and control.

Printable characters are **the alphabetic, numeric** and **special characters** that can be printed on paper or displayed on a screen.

- Some of the control characters have **to do with controlling the printing or displaying of characters;** an example is **carriage return**.
- **For a keyboard input:-** when the user presses a key, this generates an electronic signal that is interpreted by the transducer in the keyboard and translated into the bit pattern of the corresponding IRA code. This bit pattern then transmitted to the I/O module in the computer. At the computer, the text can be stored in the same IRA code. On output, IRA code characters are transmitted to an external device from the I/O module. The transducer at the device interprets this code and sends the required electronic signals to the output device either to display the indicated character or perform the requested control function.

The major functions or requirements for an I/O module fall into the following categories:

1. Control and Timing
2. Processor Communication
3. Device Communication
4. Data buffering
5. Error detection

The control of the transfer of data from an external device to the processor might involve the following sequence of steps:

1. The processor interrogates the i/o module to check the status of the attached device.
2. the i/o module returns the device status
3. if the device is operational and ready to transmit, the processor requests the transfer of data, by means of a command to the i/o module
4. the i/o module obtains a unit of data (e.g. 8 or 16 bits) from the external device.
5. the data are transferred from the i/o module to the processor.

Processor communication involves:

1. **Command decoding:** the i/o module accepts commands from the processor, typically sent as a signals on the control bus
2. **Data:** Data are exchanged between the processor and the i/o module over the data bus.

3. **Status reporting:** because peripherals are so slow, it is important to know the status of the i/o module. Common status signals are **BUSY** and **READY**.
4. **Address recognition:** just as each word of memory has an address, so does each i/o device. Thus an i/o module must recognize one unique address for each peripheral it controls.
   - The i/o module must able to perform **device communication**. This communication involves **commands, status information** and **data**.
   - An essential task of i/o module is **data buffering**.
   - An i/o module is often responsible for **error detection** and for subsequently reporting errors to the processor.

I/O module structure:

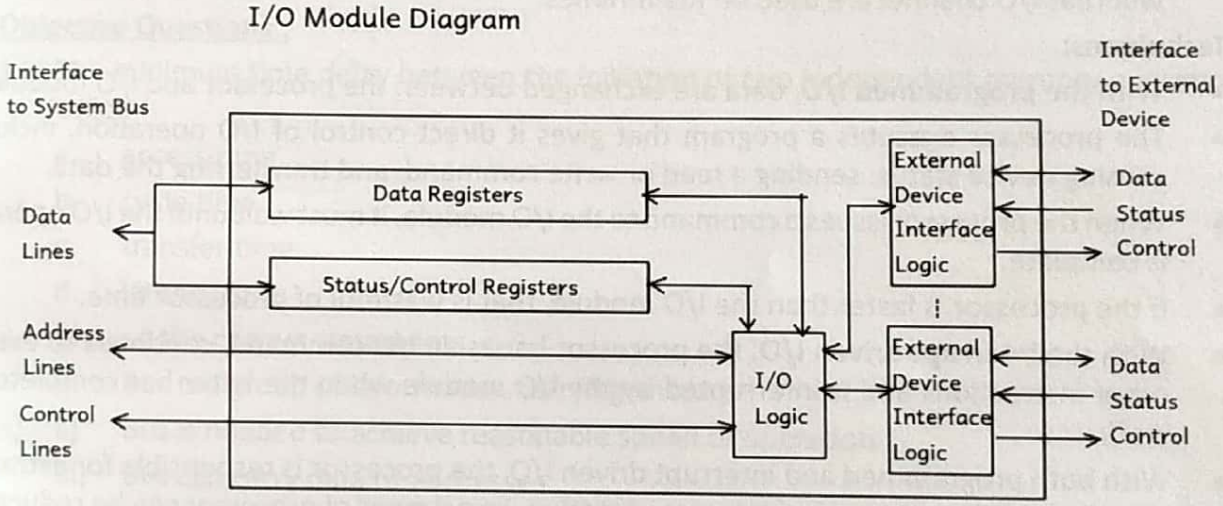


Figure 1.4

- I/O modules vary considerably in complexity and the number of external devices that they control.
- The module connects to the rest of the computer through a set of signal lines(e.g. system bus lines). Data transferred to and from the module are buffered in one or more data registers.
- There may also be one or more status registers that provide current status information. A status register may also function as a control register, to accept detailed control information from the processor. The logic within the module interacts with the processor via set of control lines.
- The processor uses the control lines to issue commands to the i/o module.
- Some of the control lines may be used by the i/o module (e.g. for arbitration and status signals).
- The module must also be able to recognize and generate addresses associated with devices it controls.
- Each i/o module has a unique address or, if it controls more than one external device, a unique set of addresses.
- The i/o module contains logic specific to the interface with each device that it controls.

- An i/o module functions to allow the processor to view a wide range of devices in a simple-minded way.
- I/O module may hide the details of timing, formats and the electro mechanics of an external device so that the processor can function in terms of simple read and write commands, and possibly open and close file commands.
- An I/O module may still leave much of the work of controlling a device(e.g. rewind a tape) visible to the processor.
- An I/O module that takes on most of the detailed processing burden, presenting a high level interface to the processor, is usually referred to as an I/O channel or I/O processor.
- An I/O module that is quite primitive and requires detailed control is usually referred to as an I/O controller or device controller. I/O controllers are commonly seen on microcomputers, whereas I/O channel are used on mainframes.

I/O Techniques:

- With the **programmed I/O**, data are exchanged between the processor and I/O module.
- The processor executes a program that gives it direct control of I/O operation, including sensing device status, sending a read or write command, and transferring the data.
- When the processor issues a command to the I/O module, it must wait until the I/O operation is complete.
- If the processor is faster than the I/O module, that is wasteful of processor time.
- With the **interrupt driven I/O**, the processor issues an I/O command, continues to execute other instructions and is interrupted by the I/O module when the latter has completed its work.
- With both programmed and interrupt driven I/O, the processor is responsible for extracting data from main memory for output and storing, investment of processor can be reduced by using **DMA technique** and results in to increasing CPU utilization

Hard Disk Drive(HDD):

- HDD, hard disk drive, hard disk or fixed disk is a data storage device used for storing and retrieving digital information using one or more rigid rapidly rotating disks platters coated with magnetic material.
- The platters are paired with magnetic heads arranged on a moving actuator arm, which read and write data to the platter surfaces.
- Introduced by IBM in 1956.
- HDD become the dominant secondary storage device for general purpose computers.
- Most current units manufactured by **Seagate**, **Toshiba** and **Western Digital**.
- HDD's are connected to the systems by standard interface cables such as **PATA(Parallel ATA)**, **SATA (Serial ATA)**, **USB or SAS(Serial Attached SCSI)** cables.
- Originally **AT Attachment**, is an interface standard for the connection of storage devices such as hard disk drives, floppy disk drives and optical disk drives in computers.
- A modern HDD records data by magnetizing a thin film of ferromagnetic material on a disk. Sequential changes in the direction of magnetization represents the binary data bits.

Motherboard:

- A motherboard (sometimes alternatively known as the **main board, planer board** or **logic board**) is the main **printed circuit board (PCB)** found in general purpose microcomputers and other expandable systems.
- It holds and allows communication between many of the crucial electronics components of a system, such as **central processing unit(CPU)** and **memory**, and provides **connectors** for other peripherals.
- Motherboard specifically refers to a PCB with expansion capability and as the name suggest, this board is often referred to as the mother of all components attached to it, which often include **peripherals, interface cards, network cards, hard drives**, or other forms of **persistent storage**; TV tuner cards, cards providing extra USB or FireWire slots and variety of other custom components.

Objective Questions :

1. The minimum time delay between the initiation of two independent memory operations is called
   a. access time
   b. cycle time
   c. transfer time
   d. latency time
2. Choose the correct statements
   i) Bus is a group of information carrying wires
   ii) Bus is needed to achieve reasonable speed of operation
   iii) Bus can carry data or addresses
   iv) A bus can be shared by more by more than one device.
      a. i,ii,iii,iiii
      b. i,ii,iii
      c. iii,iv
      d. none of the above
3. On receiving interrupt from an I/O device, the CPU
   a. halts for predetermined time
   b. hands over control of address bus and data bus to the interrupt device
   c. branches off to the interrupt service routine immediately
   d. branches off to the interrupt service routine after completion of the current instruction
4. A Computer handles several interrupt sources of which of the following are relevant for this question
   -> Interrupt from CPU temperature sensor(raises interrupt if CPU temp is too high)
   -> Interrupt from mouse (raises interrupt if the mouse is moved or a button is pressed)
   -> Interrupt from Keyboard(raises interrupt when a key is pressed)
   -> Interrupt from Hard Disk(raises interrupt when disk read in completed)
   Which one of these will be handled at the highest priority

a. Interrupt from hard disk
b. Interrupt from Mouse
c. Interrupt from Keyboard
d. Interrupt from CPU temperature sensor

5. The number of bits in the tag field of an address is
   a. 11
   b. 14
   c. 16
   d. 27

6. RAID level 1 refers to
   a. disk arrays with striping
   b. disk mirroring
   c. both (a) and (b)
   d. none of the mentioned

7. To differentiate the many network services a system supports _____ are used.
   a. Variables
   b. Sockets
   c. Ports
   d. Service names

8. The heads of the magnetic disk are attached to a _____ that moves all the heads as a unit.
   a. spindle
   b. disk arm
   c. track
   d. None of these

9. The set of tracks that are at one arm position make up a _____
   a. magnetic disk
   b. electrical disk
   c. assemblies
   d. cylinders

10. _____ controller sends the command placed into it, via message to the _____ controller
    a. host, host
    b. disk, disk
    c. host, disk
    d. disk, host

Introduction to Operating System:

What is Operating System?

An operating system (OS) is system software(collection of programs) that manages computer hardware **and software resources and provides common services for computer programs.**

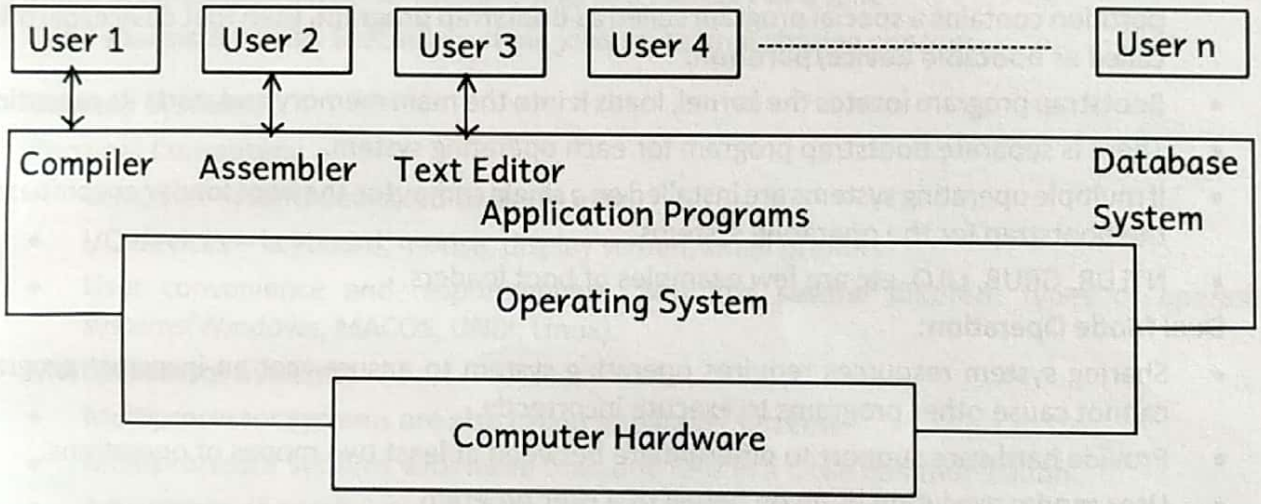


Figure 2.1

There are four components of Computer System:

1. **Hardware:** provides basic computing resources: CPU, memory and I/O device.
2. **Operating System:** Controls and coordinates use of hardware among various applications and users.
3. **Application Programs:** define the way in which the system resources are used to solve the computing problems of the users. Examples are word processors, web browsers spreadsheets and compiler.
4. **What is OS ? :** People, machines, other computers.
   - Operating System acts as an interface between user and hardware.
   - Operating System acts as an interface programs(user, application and system) and hardware.
   - Operating system controls the hardware devices in a computer system, so it is also called as **hardware controller program**.
   - It allocates resources like main memory, cpu time and io device access to all running programs, so it is also called as **resource allocator.**
   - OS manages available limited number of resources among all running programs, so it is also

called as **resource manager.**

- **Kernel** is a core part/program of an operating system, which runs continuously into memory, does basic minimal functionalities of it.

Installation of OS on PC: it is the process to store OS software onto the hard disk of the computer machine.

System Booting :

- The procedure of starting a computer system by loading kernel from secondary storage device into main memory is called booting the system.
- The first sector of any storage device is called as **boot sector.** If boot sector of any device/ partition contains a special program called as **bootstrap program,** then that device/partition called as **bootable device/partition.**
- Bootstrap program locates the kernel, loads it into the main memory and starts its execution.
- There is separate bootstrap program for each operating system.
- If multiple operating systems are installed on a single computer, the **boot loader** encompasses the bootstrap for the operating systems.
- NTLDR, GRUB, LILO, etc are few examples of boot loaders.

Dual Mode Operation:

- Sharing system resources requires operating system to ensure that an incorrect program cannot cause other programs to execute incorrectly.
- Provide hardware support to differentiate between at least two modes of operations.
- **User mode:** execution done on behalf of a user program.
- **Monitor mode:** (also called as kernel mode or system mode)- execution mode on behalf of operating system.
- Mode bit added to the computer hardware to indicate the current mode:

for system mode -> mode bit = 0 and

for user mode -> mode bit = 1.

When an interrupt occurs hardware switches to monitor mode.

Types of Operating System:

- Few operating systems are designed to be **efficient** while other are designed to be **convenient.**
- Depending in design goals and hardware constraints different types of operating system exists.

1. Mainframe Systems
2. Multiprocessor Systems
3. Distributed Systems
4. Real-Time Systems
5. Handheld Systems
6. Desktop Systems

Mainframe Systems:

Resident monitor:

- initial control in monitor
- control transfers to job
- when job completes control transfers back to monitor

Batch systems:

- Reduce setup time by batching similar jobs
- Transfers control from one job to another
-

Multi programmed Systems: Multiple jobs in a memory at a time

Time sharing Systems: Execute multiple jobs using time sharing concept

Desktop Systems:

Personal Computers:

- computer system dedicated to single user.
- I/O devices – keyboard, mouse, display screen, small printers.
- User convenience and responsiveness. May run several different types of operating systems(Windows, MACOS, UNIX, Linux).

Multiprocessor Systems:

- Multiprocessor systems are also called as parallel systems.
- Multiprocessor systems with more than one CPU in a close communication.
- Advantages of parallel systems: increase throughput, Economical and increased reliability.

Distributed Systems:

- Distribute computation among several physical Computers.
- It is also called as Loosely Coupled Systems, because each processor has its own local memory; processors communicates with one another through various communication lines, such as high-speed network or telephone lines.
- Advantage of distributed system: Resource sharing, Load balancing, Reliability.
- Requires networking infrastructure.
- Local Area Networks(LAN) or Wide Area networks(WAN)
- May be either client-server or peer-to-peer systems.

Real Time Operating Systems:

- Often used as a control device in a dedicated application such as controlling scientific experiments, medical imaging systems and some display systems.
- Well defined fixed time constraints.
- These systems may be either hard or soft real time.
- **Hard real time:** secondary storage limited or absent, data stored in short term memory, or read-only memory (ROM). They are extremely time critical missing deadlines may cause Cata Strophic effects.
- **Soft real time:** more flexible and hence widely used Less time Critical.

Handheld Systems:

- These systems are used for mobile hardware, Personal Digital Assistants (PDA's), Cellular phones, Portable multimedia systems
- **Limitations** of this type of system are: limited memory, slow processors and small display screens.

Functions of Operating System:

1. Process Management
2. CPU Scheduling
3. Memory Management
4. Hardware Abstraction
5. File to Management
6. User Interfacing
7. Networking
8. Protection and Security

First five are the basic minimal/core functionalities of any operating system and last three are the additional optional functionalities which supported by most of modern OS.

Operating System Design:

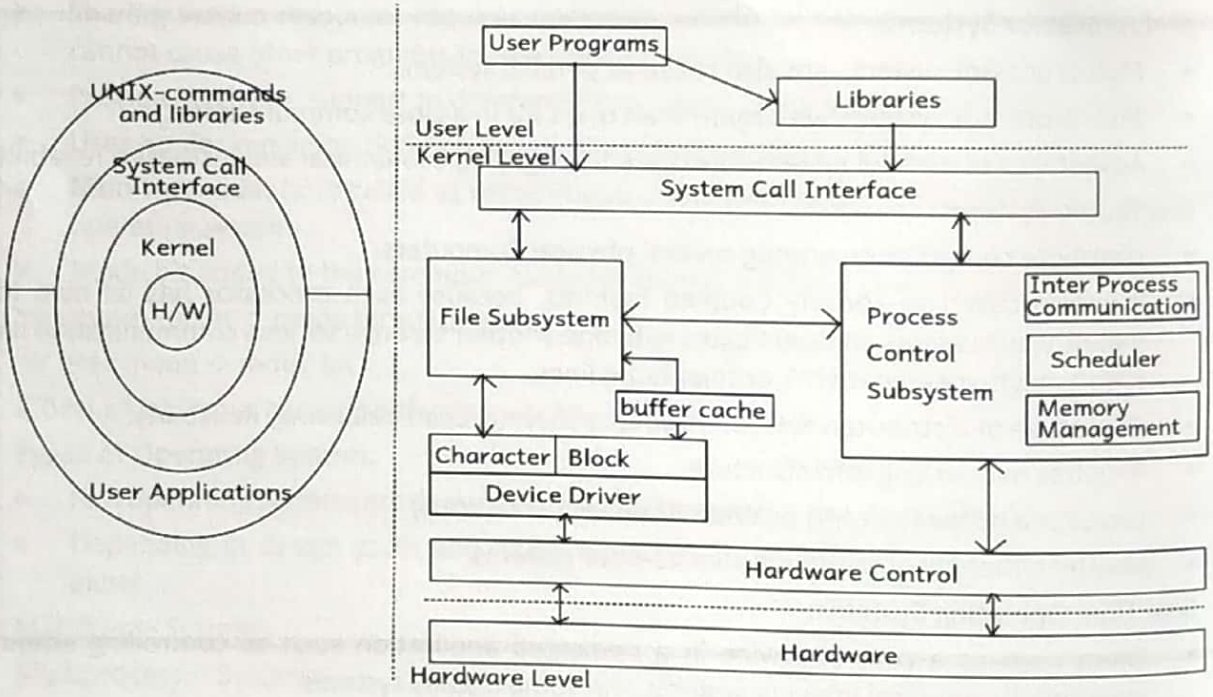


Figure 2.2

- The operating system is divided into number of layers (levels), each built on top of lower layers.
- The innermost/lower layer is the hardware, and the outermost/top layer is the user interface layer.
- With modularity, layers are selected such that each uses functions(operations) and services

of only lower -level layers.

- UNIX consists of two parts:

1. **System Programs(system call interface):**
2. **Kernel:** Contains File Subsystem block, Process Control Subsystem block which provides support for Inter Process Communication, CPU Scheduling and Memory Management.

From UNIX system point of view there are two categories of devices:

1. **Character devices:** data transfers character by character, e.g. keyboard, printer etc...

2. **Block devices:** data transfers block by block, e.g. all storage devices. for block devices one buffer cache is there, which stores the frequently accessed data temporarily into it, to get maximum throughput in minimum disk head movements.

- **Device drivers** are the modules/programs of the operating system that operates or controls a particular type of device that attached to the computer.
- **Device Controller** is of part of each hw device that controls hw functianality of each device. Device drivers in turn send instructions to device controllers.
- **System Calls:**

  System calls provides the interface of the services which are made available by kernel for users/user programs. **System calls are the functions defined either in C, C++ or assembly language.**

There are 6 categories of system calls:

1. **Process Control:** which are associated with the process control e.g. fork(), exit(), wait(), etc...

2. **File Manipulation:** to manipulate files and directories, e.g. open(), read(), write() etc...

3. **Device Management:** read(), write(), ioctl()

4. **Information Maintainance:**
   - Ask the system for information- date, time, amount of available memory, disk space, number of users.
   - Provide detailed performance, logging and debugging information.
   - Some systems implement a registry – used to store and retrieve configuration information.

5. **Communication:** provide the mechanism for creating virtual connections among processes, users, and computer systems.

6. **Protection and Security:** provide the protection and security of the files, directories, hardware devices etc. e.g. chmod(), chown() etc.

User Interfacing:

- **CUI:** A **Command-Line-Interface** or **Command-Language-Interpreter(CLI)**, also known as **command-line user interface** and **character user interface (CUI)**, is a means of interacting with a computer program where the user issues commands to the program in the form of successive lines of text (commands).
- **A program which handles the interface is called as command language interpreter or command processor or shell.**
- The CLI was the primary means of interactions with most computer systems on computer terminals in the mid-1960's.
- Shell is a program that accepts the commands as a text input and converts commands into appropriate operating system functions (system calls).
- **Command Prompt,** (cmd.exe or cmd) is the command-line- interpreter on Windows NT, Windows CE, OS/2 and eComStation operating systems.
- It is the same as **Command.com** in MS-DOS and Windows 9x systems (where it is also called as **MS-DOS Prompt**), and analogous to UNIX shells used on UNIX like systems.
- **Graphical User Interface(GUI):** is a type of user interface that allows users to interact with electronic device through graphical icons and visual indicators such as secondary notations, instead of text-based user interfaces, typed command labels or text navigations.
- **explorer.exe** is the Windows Program Manager or Windows Explorer. It manages the Windows Graphical Shell including the Start menu, taskbar, desktop, and File Manager.

Objective Questions :

1. Some computer systems supports dual mode operation -the user mode and the supervisor or monitor mode. These refers to the modes
   a. by which user programs handle their data
   b. by which the operating system executes user programs
   c. in which the processor and the associated hardware operate
   d. of memory access

2. Which of the following is a service not provided by the operating system?
   a. Protection
   b. Accounting
   c. Compilation
   d. I/O operation

3. For which of the following device there is requirement of device driver?
   a. Register
   b. Cache

c. Disk
d. none of the above

4. Which one of the following is not a real time operating system?
   a. VxWorks
   b. Windows CE
   c. RTLinux
   d. Palm OS

5. The main function of the command interpreter is
   a. to get and execute the next user-specified command
   b. to provide the interface between the API and application program
   c. to handle the files in operating system
   d. none of the mentioned

6. By operating system, the resource management can be done via
   a. time division multiplexing
   b. space division multiplexing
   c. both (a) and (b)
   d. none of the mentioned

7. Which one of the following error will be handle by the operating system?
   a. power failure
   b. lack of paper in printer
   c. connection failure in the network
   d. all of the mentioned

8. What is operating system?
   a. collection of programs that manages hardware resources
   b. system service provider to the application programs
   c. link to interface the hardware and application programs
   d. all of the mentioned

9. Operating system is responsible for
   a. disk initialization
   b. booting from disk

c. bad-block recovery
d. all of the mentioned

10. _____ program loads kernel from hard disk into the main memory
a. loader
b. BIOS
c. boot loader
d. bootstrap

Process Management:

What is Program?

- Program is a set of instructions given to the computer.
- Process is program in execution. A process will need certain resources such as **CPU time, memory, files, and I/O devices** to accomplish its task.
- Operating System is responsible for the following activities in connection with process and thread management.
  1. The creation and termination of both user and system processes.
  2. The scheduling of processes.
  3. Provision of mechanism for synchronization
  4. Inter Process Communication
  5. Deadlock handling for the processes.
- **Program** is a file containing a list of instructions stored on disk**(often called an executable file). Executable file contains following sections:**
  1. **Primary Header:** contains **magic number, address of entry point function** and **information about remaining sections(metadata: data about data).**
  2. **RO-Data Section(Read-Only Data Section):** contains constant string literals.
  3. **Data Section:** contains global and static variables.
  4. **Code Section/Text Section:** contains executable instructions.
  5. **Symbol Table:** Symbol Table contains information about variables and functions in tabular format.
- **Program is passive entity, whereas Process** is an **active entity.**

Structure of a Process:

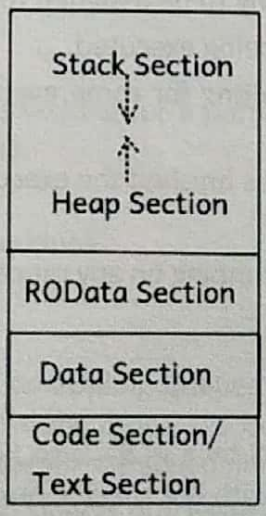


Figure 3.1

- **A Process contains following sections:**
  1. **Code Section/Text Section or Segment:** contains executable instructions.
  2. **Data Section or Segment:** contains constant string literals.
  3. **Heap Section or Segment:** contains memory which is allocated dynamically during process runtime.
  4. **Stack Section or Segment:** which contains temporary data(such as function parameters, return address and local variables).

States of the Process:

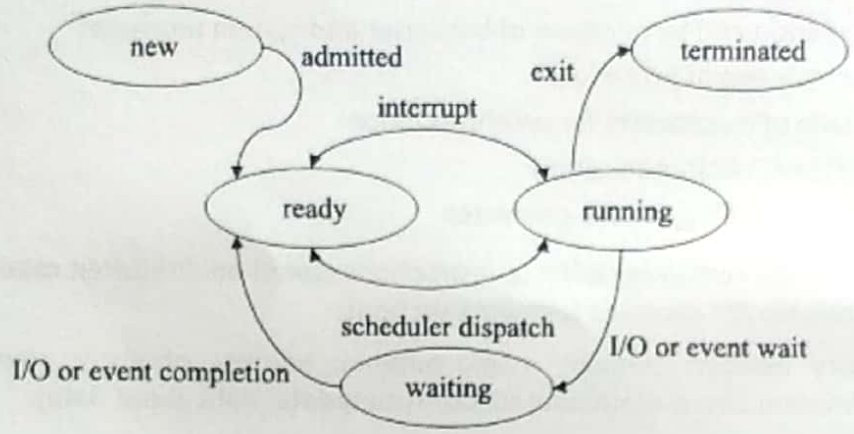


Figure 3.2

- As process executes, it changes state. **The state of the process is defined in part by the current activity of that process.** Each process may be in one of the following states.
  1. **New:** the process is being created.
  2. **Ready:** the process is waiting to be assigned to a processor.
  3. **Running:** instructions are being executed.
  4. **Waiting:** the process is waiting for some event to occur(such as an IO completion or reception of signal)
  5. **Terminated:** the process has finished the execution.

Note that, only one process can be running on any processor at any instant. Many processes may be ready and waiting.

- **PCB: Process Control Block:**

Each process represented in the OS by a PCB - also known as **Task Control Block.** It contains many pieces of information associated with the specific process, including these:

1. Process State

2. Program Counter
3. Open files Information
4. CPU Scheduling Information
5. Memory Management Information
6. Execution Context
7. Accounting Information
8. I/O Status Information etc...

- **Multi-Programming:** If in a system multiple programs can be loaded in memary at a time then such a system is called as multi-programming system.
- The no. of programs that can be loaded in memary at a time is called as **degree of multi-programming.**
- **Multi-Tasking/Time Sharing:** CPU can execute multiple programs concurrently or simultaneously. OR CPU time gets shared among all running programs.
- **Multi-User:** Multiple users can execute one or more programs in a single system concurrently.
- **Multi-Processor:** The system in which multiple processors are connected in a closed circuit, is called as multi-processor system.
- The objective of multi-programming **to maximize CPU utilization.**

Scheduling Queues:

1. **Job Queue:** when new proces is created, it is kept into Job Queue, it contains of all processes in the system.
2. **Ready Queue:** the processes that are residing in main memory and are ready to execute are kept on a list called as Ready Queue.
3. **Device Queue:** The list of processes waiting for a particular i/o device is called device queue. Each device has its own device queue.

Processes migrates among various queues.

Scheduler:

A process migrates among various scheduling queues throughout its lifetime. The operating system must select , for scheduling purposes, processes from these queues in some fashion. The selection process is carried out by the appropriate scheduler.

1. **Long Term Scheduler/Job Scheduler:** selects processes from job queue and load them into the

main memory in ready queue.

2. **The Short Term Scheduler/CPU Scheduler:** selects the processes that are waiting in ready queue and allocates the CPU to one of them.
   - The difference between these two schedulers is in frequency of execution. The long term scheduler executes less frequently than short term scheduler.
   - The long term scheduler controls the **degree of multiprogramming.**

Context Switch:

- **Interrupts** cause the operating system to change a CPU from its current task and to run a kernel routine. Such operations happen frequently on general purpose systems.
- When interrupt occurs, the system needs to save the current context of the of the process running on the CPU so that it can restore that context when its processing is done, essentially suspending the process and then resuming it. The context is represented in the PCB of the process; it includes the value of CPU registers, the process state, and memory management information.
- Generally, we perform a **state save** of the current state of the CPU, be it in kernel or user mode, and then a **state restore** to resume operations.
- **Switching the CPU to another process requires performing a state save of the current (old) process and state restore of a another(new) process. This task is known as context switch.** When context switch occurs, the kernel saves the context of old process in its PCB and loads the saved context of new process scheduled to run.
- **Dispatcher:** It is the module of an operating system, that gives control of CPU to the process selected by CPU scheduler.

This function involves the following:

1. Switching context
2. Switching to user mode
3. Jumping to the proper location in the user program to restart the program.

The time it takes to perform the same is called as dispatcher latency.

CPU Scheduling:

- CPU scheduling is the task of selecting a waiting process from the ready queue and allocating the CPU to it.
- CPU scheduling is the basis of multiprogrammed operating systems, By switching CPU among processes, the operating system can make the computer more productive.
- In a **single processor system/uniprocessor system**, only one process can run at a time; any others must wait until the CPU is free and can be rescheduled.
- The objective of multi-programming is to have some process running all the times, to maximize the CPU utilization.
- CPU Scheduling decisions may take place under the following four circumstances:

case 1: Running -> Waiting (when Current process makes io request.)

case 2: Running -> Ready (when interrupt occurs)

case 3: Waiting -> Ready (when a process complates io)

case 4: Running -> Terminated (when current process is terminated)

- There are two types of CPU scheduling: **Non-Pre-emptive** and **Pre-emptive.**
- In **Non-Pre-emptive** scheduling process releases the control of CPU in only **case 1** and **case 4**, whereas under **pre-emptive** scheduling process releases the control of CPU in **case 2** and **case 3** as well.
- That is, in **Non-Pre-emptive control of the CPU release by the process only after termination or switched to waiting state, i.e. process is releasing the CPU by its own**, whereas in **Pre-emptive scheduling process is forced to leave the CPU i.e. to preempt the CPU.**
- Different CPU-Scheduling algorithms have different properties, and the choice of a particular algorithm may favour one class of processes over another.
- In choosing which algorithm to use in a particular situation, we must consider the properties of the various algorithms.
- Many **criteria** have been suggested for comparing CPU-Scheduling algorithms

CPU Scheduling Criteria:

1. **CPU Utilization(Max):** We want to keep CPU as busy as possible. Conceptually, CPU utilization can range from 0% to 100%.

2. **Throughput (Max):** If the CPU is busy executing the processes, then work is being done. **One measure of work is the number of processes that are completed per unit time, called throughput.**

3. **Turn Around Time-TAT (Min):** the time interval from the time of submission of a process to the time of completion is the turnaround time. Turnaround time is the sum of periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing IO.

Turn Around Time = Execution Time + Waiting Time.

4. **Waiting Time (Min):** Waiting time is the sum of periods spent waiting by the process in the ready queue. CPU Scheduling algorithms does not affect the amount of the time during which a process executes or does I/O; it affects only the amount time that a process spends waiting in the ready queue.

5. **Response Time (Min):** response time of process is the time, it takes to get first response from CPU, from time of submission.

Gantt Chart: It is a **bar chart** that illustrates a particular schedule, including the start and finish

times of each participating processes with respect to **CPU cycles.**

CPU Scheduling Algorithms:

1. **First-Come, First-Served Scheduling (FCFS):**
   - It is the **simplest** CPU-Scheduling algorithm.
   - With this scheme, the process that requests the CPU first is allocated the CPU first.
   - The implementation of FCFS policy easily managed with a FIFO queue.
   - When a process enters the ready queue, its PCB is linked onto the tail of the queue. When the CPU is free, it is allocated to the process at the head of the queue. The running process then remove from the queue.
   - The **average waiting time often quite long** under FCFS policy.
   - FCFS algorithm is **non-pre emptive**.
   - **scheduling overhead is minimum:** as context switch only occur upon termination.
   - **throughput can be low,** because long processes can be holding CPU, waiting the shorter processes for long time(**known as convoy effect).**
   - **no starvation,** because each process gets chance to be executed after a definite time.
   - turnaround time, waiting time and response time depends on their arrival and can be high for the same reason above.

2. **Shortest Job First(SJF):**
   - In this policy, when the CPU is available, it is assigned to the process that has the **smallest next CPU burst time,** if the next CPU bursts of two processes are the same, FCFS scheduling is used to break the tie. It is also called as **shortest-next-CPU burst algorithm.**
   - **SJF algorithm is provably optimal,** in that it give minimum average waiting time for a given set of processes.
   - The real difficulty with the SJF algorithm is knowing the length of next CPU request.
   - **SJF is used in specialized environments where accurate estimates of running time are available.**
   - **Shortest Job First (Pre-emptive):**
   - The SJF algorithm can be either non-pre-emptive or pre-emptive. The choice arises when a new process arrives at the ready queue while a previous process is still executing. The next CPU burst of the newly arrived process may be shorter than what is left of the currently executing process, whereas a non-pre-emptive SJF allow the currently running process to finish its CPU burst. Pre-emptive SJF is also called as **shortest-remaining-time-first-scheduling.**

3. **Priority Scheduling:**
   - The SJF algorithm is a special case of general priority scheduling algorithm.
   - The OS assigns a fixed priority rank to every process, and then scheduler arranges the processes in the ready queue in order of their priority.

- Lower priority processes gets interrupted by incoming high priority processes.
- Equal priority processes are scheduled in FCFS order.
- **Priorities can be defined internally or externally.**
- Internally defined priorities use some measurable quantity or quantities to compute the priority of the process. For example, time limits, memory requirement etc.
- Priority of the process is there in its PCB
- **Starvation** of lower priority processes is possible with large number of high priority processes ready for execution.
- **Ageing:** it is the technique by which system gradually increments the priority of the process which is starred in ready queue due to very low priority. This will ensure that process priority will be raised enough, to get scheduled.

4. **Round Robin Scheduling:**
   - The scheduler assigns a fixed **time slice/time quantum** per process, and cycles through them. If process is complete within that time slice, it gets terminated, otherwise it is reschedules after giving a chance to all other processes.
   - This scheduling involves **extensive overhead,** especially with a small time unit.
   - Balanced throughput between FCFS and SJF, shorter jobs are completed faster than in FCFS and longer processes are completed faster than SJF.
   - **Good average response time,** waiting time is dependent on number of processes, and not on average process length.
   - **Starvation can never occurs,** since no priority is given.
   - There is no universal "best" scheduling algorithm, and many operating systems use extended or combinations of the scheduling algorithms above.

5. **Multi-level Queue Scheduling:**
   - This algorithm partitions the ready queue into several separate queues. The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type.
   - Each queue has its own scheduling algorithm. For example separate queues might be used for foreground and background processes. The foreground queue might be scheduled by RR algorithm, while background queue might be scheduled by FCFS algorithm.
   - Example of multilevel queue scheduling algorithm with five queues: 1. System Processes 2. Interactive Processes 3. Interactive editing processes 4. Batch Processes and 5. Student Processes.

6. **Multi-level Feedback Queue:**
   - Normally, when the multilevel queue scheduling algorithm is used, processes are permanently assigned to a queue when they enter into the system.
   - The multilevel feedback queue scheduling algorithm, in contrast, **allows a process move/ switch between queues.**
   - The idea is to separate processes according to the characteristics of their CPU bursts. If any process uses too much CPU time, it will be moved to a lower priority queue. In addition,

a process that waits too long in a lower-priority queue may be moved to a higher-priority queue. This form of aging prevents starvation.

- Whatever we discussed so far has focused on the problems of scheduling the CPU in a system with a single processor. If multiple CPU's are available, load sharing becomes possible; however, the scheduling problem becomes more complex.

Approaches to Multi-Processor Scheduling:

1. **Asymmetric Multiprocessing System(ASMPS):** One approach to CPU scheduling in a multiprocessor system has all scheduling decisions, I/O processing, and other system activities handled by a single processor-the master. The other processors execute only user code. This is simple because only one processor accesses the system/kernel data structures, reducing the need for data sharing.

2. **Symmetric Multiprocessing System(SMPS):** in this approach each processor is **self-scheduling**. All processes may be in one common queue, or each processor may have its own private queue of ready processes.

Inter Process Communication:

- Processes executing concurrently in the operating system may be either **independent processes** or **cooperating processes.**
- A process is **independent** if it cannot affect or be affected by the other processes executing in the system or any process that does not share data with any other process is independent.
- A process is **cooperating** if it can affect or be affected by other processes executing in the system or any process that shares data with other process is a cooperating process.

- There are several reasons for providing an environment that allows process cooperation: **Information sharing**, **Computation speedup**, **Modularity** and **Convenience.**
- Cooperating processes require an interprocess communication (IPC) mechanism that will allow them to exchange data and information.

There are two fundamental models of interposes communication(IPC).

1. **Shared Memory Model:** under shared memory model, a region of memory that is shared by cooperating processes is established. Processes can then exchange the information by reading and writing data to the shared region.
2. **Message Passing Model:** under message passing model, communication takes place by means of messages exchanged between the cooperating processes.

Under message passing model there are 5 ways by which processes can communicates

1. **Pipe:**
   - By using pipe one process can send message to another process. It is stream based unidirectional communication.
   - Related processes can communicates with each other by using un named pipe. (Processes

that are of same parent process called as related processes), whereas non-related processes can communicates with each other by using named pipe/fifo.

2. **Message Queue:** concurrently running cooperating processes in a same system can communicates by means of sending and receiving packets of messages.
3. **Signals:** A process can send signal to another process or OS can send signal to a process. e.g. SIGKILL, SIGSEGV.
4. **Socket:** a socket is defined as a endpoint for communication. Using socket one process can communicate with another process on same machine or different machine in the network.

   Socket = IP Address + Port.
5. **RPC(Remote Procedure Call):** Use to call method from another process on the same machine or different machine in the network.

Process Synchronization:

- Cooperating processes can directly share data from comman (shared) memary Concurrent access to shared data may result in data inconsistency.
- The mechanism to ensure the orderly execution of cooperating processes that share a comman resource, so that data inconsistency problem can be resolved. is called as ''process synchronozation''

Critical-Section Problem:

- Consider a system consisting of n processes {P1,P2,...,Pn}. Each process **has segment of code, called a critical-section**, in which the process may be changing common variables, updating a table, writing a file, and so on. The important feature of the system is that when one process is executing in its critical-section, no other process is to be allowed to execute in its critical section. That is, no two processes are executing in their critical-sections at the same time. The critical-section designs a protocol that the processes can use to cooperate. Each process must request a permission to enter its critical-section. The section of code implementing this request is the **entry section**. The critical-section may be followed by **exit section**. The remaining code is the remainder section. The general structure of a typical process Pi is shown below

```
do{
        entry section
        critical section
        exit section
        remainder section
}while(TRUE);
```

Race Condition:

- When one or more processes trying to access and manipulate the same resource/data at a time, then race condition occurs. To avoid race condition operating system needs to keep track on ordered execution of the processes and the changes did by last accessed process remain final value.

- **Semaphore:** semaphore is the **synchronization tool** for the solution of critical-section problem. A semaphore **S** is an integer variable that, apart from initialization, is accessed only through two standard atomic operations: **wait()** and **signal().** The **wait()** operation was originally termed **P(to test)**; signal() was originally called **V (increment).**

The definition of wait() is:

```
wait(S)
{
        while( S <= 0 )
        ; //no operation
        S--;
}
```

The definition of signal() is as follows:

```
signal(S)
  {
        S++;
  }
```

- All modifications to the integer value of the semaphore in the wait() and signal() operations must be executed indivisibly. That is, when one process modifies the semaphore value, no other process can simultaneously modify that same semaphore value. In addition, in the case of wait(S), the testing of an integer value of S(S<=0), as well as its possible modification(S--), must be executed without interruption.
- There are two types of semaphores: **Counting Semaphore** can range over an unrestricted domain and **Binary Semaphore** can range only between 0 and 1. On some systems Binary Semaphores are known as **mutex locks**, as they are locks that provide mutual exclusion.

Deadlocks:

- In a multi-programming environment, several processes may compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process enters a waiting state. Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called as **deadlock.**

- **Necessary Conditions to occur deadlock:** A deadlock situation can arise if the following four conditions hold simultaneously in a system
  1. **Mutual Exclusion:** at least one resource must be held in a non-sharable mode; that is, only one process at a time can use the resource
  2. **Hold and Wait:** A process must be holding at least one resource and waiting to acquire

additional resources that are currently being held by other processes.

3. **No Pre-emption:** Resources cannot be pre-empted; that is, a resource can be released only voluntarily by the process holding it, after that process has completed its task.
4. **Circular Wait:** A set {P1, P2, ..., Pn} of waiting processes must exists such that P1 is waiting for a resource held by P2, P2 is waiting for a resource held by P3,..., and Pn is waiting for a resource held by P1.

Resource-Allocation Graph:

- Deadlocks can be described more precisely in terms of a directed graph called a system resource-allocation graph. This graph consists of a set of vertices V and a set of edges E. The set of vertices V is partitioned into two different types of nodes: P = {P1,P2,P3,..Pn}, the set consisting of all active processes in the system, and R = {R1,R2,R3,...,Rm}, the set consisting of all resource types in the system.

- **Request Edge:** A directed edge from process Pi to resource Rj is denoted by **Pi -> Rj**; it signifies that process Pi has requested an instance of resource type Rj and is currently waiting for that resource.

- **Assignment Edge:** A directed edge from resource type Rj to process Pi is denoted by **Rj -> Pi;** it signifies that an instance of resource type Rj has been allocated to process Pi.

- Given the definition of a resource allocation graph, it can be shown that, if the graph contains no cycles, then no process in the system is deadlocked. If the graph does contain a cycle, then a deadlock may exists.

Method of handling deadlock:

1. **Deadlock Prevention:** For deadlock to occur, all four conditions must hold. By ensuring that at least one of these conditions cannot hold, we can prevent the occurrence of a deadlock.

2. **Deadlock Avoidance:** An alternative method to avoid deadlocks is to require additional information about how resources are to be requested.

   There are two algorithms for deadlock-avoidance

   1. **Resource-Allocation Graph Algorithm**

   2. **Banker's Algorithm**

3. **Deadlock Recovery:** When a detection algorithms determines that deadlock exists, several alternatives are available. One possibility is to let system recover from deadlock automatically. There are two options for breaking the deadlock.

1. **Process Termination:** One is simply to abort one or more processes to break the circular wait. The process which got selected for termination is called as **victim process.**
2. **Resource Pre-emption:** To pre-empt some resources from one or more of the deadlocked processes.

Objective Questions :

1. Necessary conditions for deadlock are:
   a. non-pre-emption and circular wait
   b. Mutual exclusion and partial allocation
   c. both (a) and (b)
   d. none of the above

2. In a time-sharing operating system, when the time slot given to process is completed, the process goes from RUNNING state to the
   a. BLOCKED state
   b. READY state
   c. SUSPENDED state
   d. TERMINATED state
3. The dispatcher
   a. actually schedules the tasks into the processor
   b. puts tasks in I/O wait
   c. is always small and simple
   d. never changes task priorities
   e. none of above

4. In Round Robin CPU Scheduling, as the time quantum is increased, the average turnaround time
   a. increases
   b. decreases
   c. remains constants
   d. varies irregularly

5. In which of the following scheduling policies does context switching never take place?
   a. Round Robin
   b. Shortest Job First
   c. Pre-emptive

d. First-Come-First-Served

6. Which of the following statement is true
   I. Shortest remaining time first time scheduling may causes starvation
   II. Pre-emptive scheduling may cause starvation
   III. Round robin is better than FCFS in terms of response time
   a. I only
   b. I and III only
   c. II and III only
   d. I, II and III

7. Which of the following scheduling algorithm gives minimum average waiting time?
   a. FCFS
   b. SJF
   c. Round-Robin
   d. Priority

8. Which of the following features of UNIX may be used for IPC?
   a. Signals
   b. Pipes
   c. Semaphore
   d. Message queues
   e. all of the above

9. The PID of the Kernel process is
   a. undefined
   b. 0
   c. 1
   d. 3

10. Which of the following calls never returns an error?
   a. getpid()
   b. fork()
   c. ioctl()
   d. open()

Memory Management:

- The main purpose of a computer system is to execute programs. These programs, together with the data they access, must be at least partially in a main memory during execution.
- To improve both the utilization of CPU and the speed of its response to users, a general purpose computer must keep several processes in memory.
- An address generated by the CPU is commonly referred as a **logical address,** whereas an address seen by the memory unit-that is the one loaded into the memory address register of the memory-is commonly referred to as a **physical address.**
- The compile-time and load-time address-binding methods generate identical logical and physical addresses. However, the execution-time address-binding scheme results in differing logical and physical addresses.
- The set of all logical addresses generated by a program is a **logical address space;** the set of all physical addresses corresponding theses logical addresses is a **physical address space**.
- The run-time mapping from logical to physical addresses is done by a hardware device called as **memory-management-unit(MMU).**

Swapping:

- A process must be in memory to be executed. A process, however can be swapped temporarily out of memory to a backing store and then brought back into memory for continued execution.
- A variant of this swapping policy is used for priority-based scheduling algorithms. If a higher-priority process arrives and wants service, **the memory manager** can swap out the lower-priority process and then load and execute the higher-priority process. When the higher-priority process finishes, the lower-priority process can be swapped back in and continued. This variants of swapping is sometimes called **roll out, roll in.**

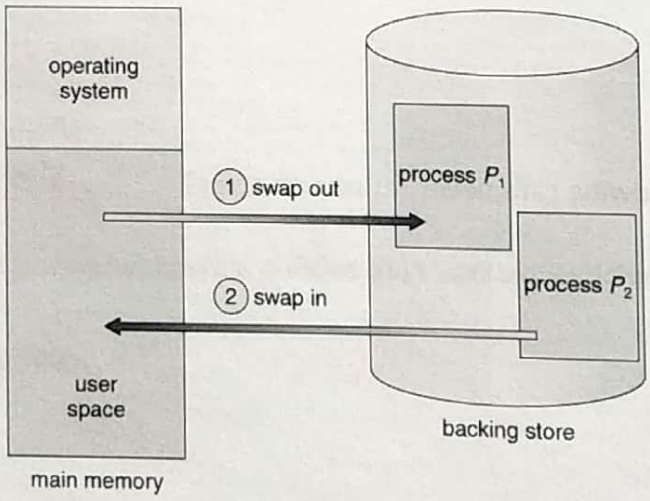


Figure 4.1

- Swapping requires a **backing store/swap area**. The backing store is commonly a fast disk. It must be large enough to accommodate copies of all memory images for all users, and must provide direct access to these memory images. The system maintains a ready queue consisting of all processes whose memory images are on the backing store or in memory and are ready to run.

Contiguous Memory Allocation:

- The main memory must accommodate both the operating system and the various user processes.
- The memory is usually divided into two partitions: one for the resident operating system and one for the user processes. We can place the operating system in either low memory or high memory. The major factor affecting this decision is **the location of interrupt vector.** Since the interrupt vector is often in low memory, programmers usually place the operating system in low memory as well.
- In contiguous memory allocation, each process is contained in a single contiguous section of memory.
- One of the simplest methods for allocating memory is to divide memory into several **fixed-sized** partitions. Each partition can contain exactly one process. Thus the degree of multiprogramming is bound by the number of partitions. In this, multiple partition method, when a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process.
- In the **variable-partition** scheme, the operating system keeps a table indicating which parts of memory are available and which are occupied. Initially, all memory is available for user processes and is considered one large block of available memory, a hole. Eventually, as you will see, memory contains a set of holes of various sizes.
- As process enter the system, they are put into an input queue. The operating system takes into account the memory requirements of each process and the amount of available memory space in determining which processes are allocated memory. When a process is allocated a space, it is loaded into main memory, and it can then compete for CPU time. When a process terminates, its memory is released. This memory can be reused for a new process by operating system.
- At any given time, then, we have a list of available block sizes and an input queue. The operating system can order the input queue according to a scheduling algorithm. Memory is allocated to processes until, the memory requirements of the next process cannot be satisfied i.e. no available block of memory (or hole) is large enough to hold that process. The operating system can then wait until a large enough block is available, or it can skip down the input queue to see whether the smaller memory requirements of some other process can be met.
- The memory blocks available comprise a set of holes of various sizes scattered throughout memory. When a process arrives and needs memory, the system searches the set for a hole that is large enough for this process. If the hole is too large, it is split into two parts. One part is allocated to the arriving process; the other is returned to the set of holes.
- When a process terminates, it releases its block of memory, which is then placed back in the set of holes. If the new hole is adjacent to other holes, these adjacent holes are merged

to form one larger hole. At this point, the system may need to check whether there are processes waiting for memory and whether this newly freed and recombined memory could satisfy the demands of any of these waiting processes.

- This procedure is a particular instance of the general **dynamic storage allocation** problem, which concerns how to satisfy a request of size n from a list of free holes.
- There are many solutions to this problem. The **first fit, best-fit** and **worst-fit** strategies are most commonly used to select a free hole from the set of available holes.
- **First fit:** Allocate the first hole that is big enough. Searching can start either at the beginning of the set of holes or at the location where the previous first-fit search ended. We can stop searching as soon as we find a free hole that is large enough.
- **Best fit:** Allocate the smallest hole that is big enough. We must search the entire list, unless the list is ordered by size. This strategy produces the smallest leftover hole.
- **Worst fit:** Allocate the largest hole. Again, we must search the entire list, unless it is sorted by size. This strategy produces the largest leftover hole, which may be more useful than the smaller leftover hole from a best-fit approach.

External Fragmentation:

- As processes are loaded and removed from memory, the free memory space is broken into little pieces. External fragmentation exists when there is enough total memory space to satisfy a request but the available spaces are not contiguous; storage is fragmented into a large number of small holes.
- Memory fragmentation can be internal as well as external. When unused memory that is internal to the partition then it is called as **internal fragmentation.**
- One solution to the problem of external fragmentation is **Compaction:** to shuffle the memory contents so as to place all free memory together in one large block.
- Compaction is not always possible. If relocation is static and is done at assembly or load time, compaction cannot be done; compaction is possible only if relocation is dynamic and is done at execution time.
- Another possible solution to the **external-fragmentation** problem is to permit the logical address space of the processes to be non-contiguous, thus allowing a process to be allocated physical memory wherever such memory is available. Two complementary techniques achieve this solution: **paging** and **segmentation**.

Paging:

- Paging is a memory-management scheme that permits the physical address space of a process to be non-contiguous.
- Paging avoids external fragmentation and the need for compaction. It also solves the considerable problem of fitting memory chunks of varying sizes onto the backing store; most memory-management schemes used before the introduction of paging suffered from this problem. The problem arises because, when some code fragments or data residing in main memory need to be swapped out, space must be found on the backing store. The backing store has the same fragmentation problems discussed in connection with main memory, but access is much slower, so compaction is impossible. Because of its advantages over earlier methods, paging in its various forms is used in most operating systems.
- The basic method for implementing paging involves breaking physical memory into fixed-

sized blocks called **frames** and breaking logical memory into blocks of the same size called **pages.**

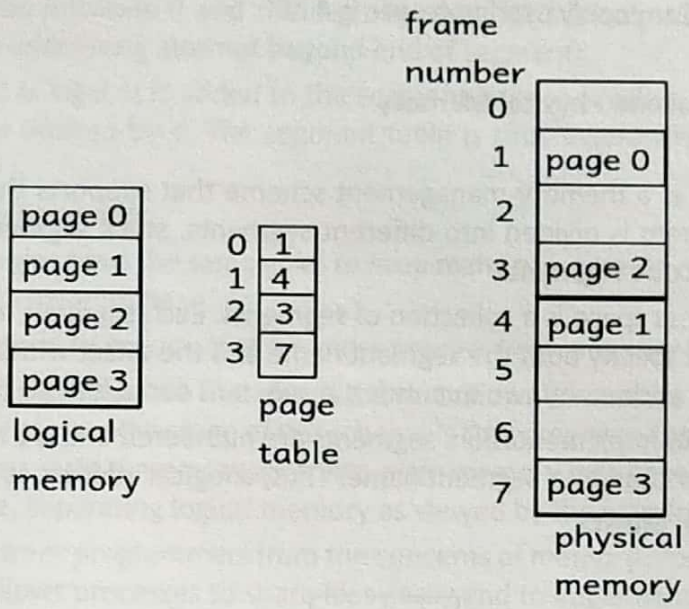


Figure 4.2

- When a process is to be executed, its pages are loaded into any available memory frames from their source (a file system or the backing store). The backing store is divided into fixed-sized blocks that are of the same size as the memory frames.
- Every address generated by the CPU is divided into two parts:

Page number(p): the page number is used as an index into a page table contains the base address of each page in physical memory and,

Page offset(d): this base address is combined with the page offset to define the physical memory address that is sent to the memory unit.

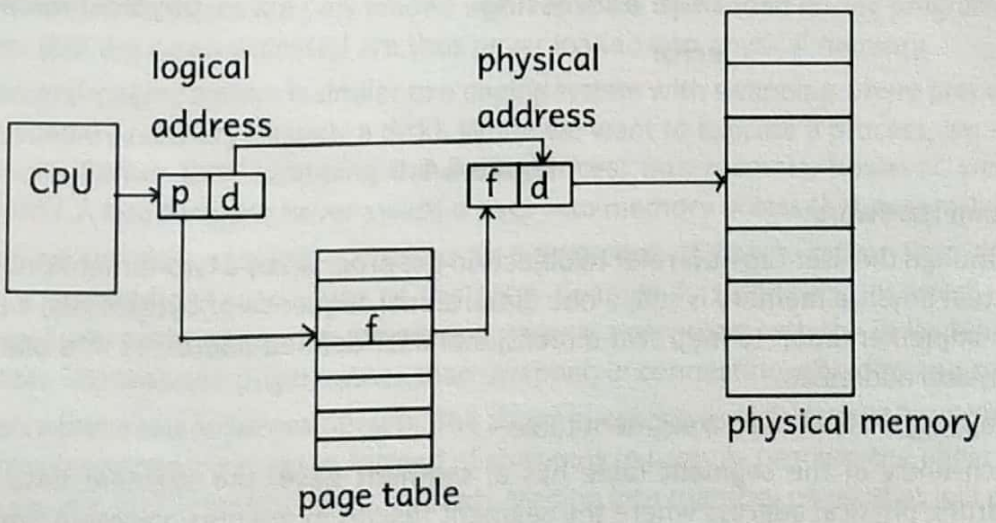


Figure 4.3

- The page size (like the frame size) is defined by the hardware. The size of a page is typically a power of 2, varying between 512 bytes and 16 MB per page, depending on the computer architecture. Commanly used page size is 4 KB.

Paging Model of Logical and Physical Memory

Segmentation:

- Segmentation is a memory-management scheme that supports the user view of memory in which program is divided into different segments, **stack segment, data segment, heap segment** and **code segment**.
- A logical address space is a collection of segments. Each segment has a name and a length. The addresses specify both the segment name and the offset within the segment. The user specifies each address by two quantities: a segment name and an offset.
- For simplicity of implementation, segments are numbered and are referred to by a segment number, rather than by a segment name. Thus, a logical address consists of a two parts :

<segment-number, offset>

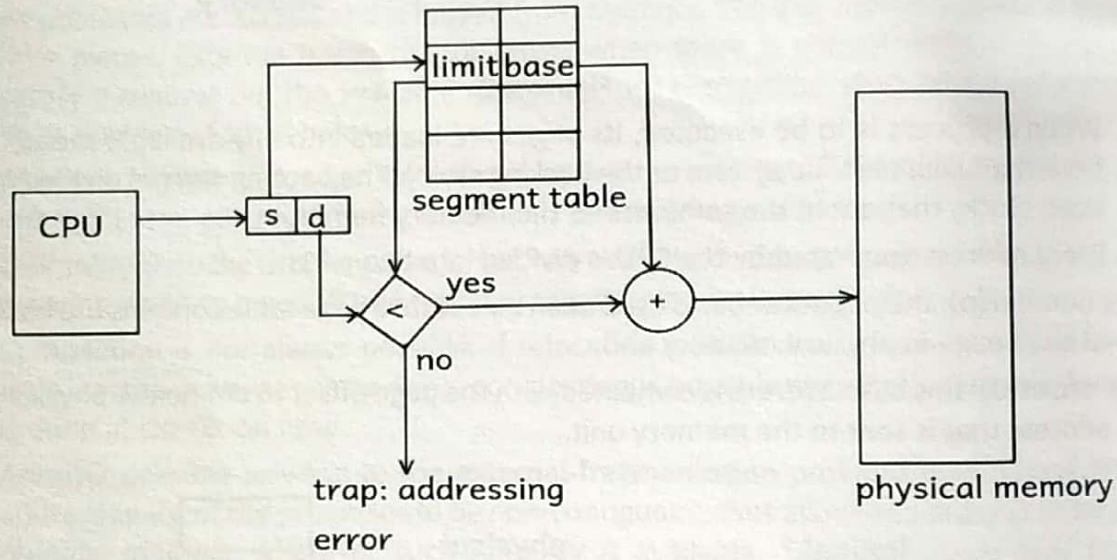


Figure 4.4

Segmentation Hardware:

- Although the user can now refer to objects in the program by a two-dimensional address, the actual physical memory is still, a one dimensional sequence of bytes. Thus, we must define an implementation to map two-dimensional user-defined addresses into one-dimensional physical addresses.

This mapping is effected by a segment table.

- Each entry in the segment table has a, **segment base:** the segment base contains the starting physical address where the segment resides in memory. **segment limit:** it specifies the length of the segment.
- A logical address consists of two parts: a **segment number(s)**, and an **offset(d)** into that

segment.

- The segment number is used as an index to the segment table. The offset d of the logical address must be between 0 and the segment limit. If it is not, we trap to the operating system (logical addressing attempt beyond end of segment).
- When an offset is legal, it is added to the segment base to produce the address in physical memory of the desired byte. The segment table is thus essentially an array of base-limit register pairs.

Virtual Memory Management:

- Above all strategies have the same goal: to keep many processes in memory simultaneously to allow multi-programming.
- However, they tend to require that an entire process be in memory before it can execute.
- Virtual memory is a technique that allows the execution of processes that are not completely in memory. One major advantage of this scheme is that programs can be larger than physical memory. Further, virtual memory abstracts main memory into an extremely large, uniform array of storage, separating logical memory as viewed by the user from physical memory.
- This technique frees programmers from the concerns of memory-storage limitations. Virtual memory also allows processes to share files easily and to implement shared memory.
- In addition, it provides an efficient mechanism for process creation. Virtual memory is not easy to implement, however, and may substantially decrease performance if it is used carelessly.
- Consider how an executable program might be loaded from disk into main memory. One option is to load the entire program in physical memory at program execution time. However, a problem with this approach is that we may not initially need the entire program in memory. Suppose a program starts with a list of available options from which the user is to select. Loading the entire program into memory results in loading the executable code for all options, regardless of whether an option is ultimately selected by the user or not. An alternative strategy is to load pages only as they are needed. This technique is known as demand paging and is commonly used in virtual memory systems. With demand-paged virtual memory, pages are only loaded when they are demanded during program execution; pages that are never accessed are thus never loaded into physical memory.
- A demand-paging system is similar to a paging system with swapping where processes reside in secondary memory (usually a disk). When we want to execute a process, we swap it into memory. Rather than swapping the entire process into memory, however, we use a lazy swapper. A lazy swapper never swaps a page into memory unless that page will be needed.
- Since we are now viewing a process as a sequence of pages, rather than as one large contiguous address space, use of the term swapper is technically incorrect. A swapper manipulates entire processes, whereas a pager is concerned with the individual pages of a process. We thus use pager, rather than swapper, in connection with demand paging.
- When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again. Instead of swapping in a whole process, the pager brings only those pages into memory. Thus, it avoids reading into memory pages that will not be used anyway, decreasing the swap time and the amount of physical memory needed.
- With this scheme, we need some form of hardware support to distinguish between the pages that are in memory and the pages that are on the disk.

- The valid-invalid bit scheme can be used for this purpose. When this bit is set to **valid** the associated page is both legal and in memory. If the bit is set to **invalid** the page either is not valid (that is, not in the logical address space of the process) or is valid but is currently on the disk. The page-table entry for a page that is brought into memory is set as usual but the page-table entry for a page that is not currently in memory is either simply marked invalid or contains the address of the page on disk.
- If process tries to access a page that is not present into memory, then **page fault** occurs.

Page Replacement:

- The operating system could instead swap out a process, freeing all its frames and reducing the level of multiprogramming. Page replacement is a good one in certain circumstances.
- Page replacement follows the following approach, if no frame is free, we find one that is not currently being used and free it. We can free a frame by writing its contents to swap space and changing the page table (and all other tables) to indicate that the page is no longer in memory. We can now use the freed frame to hold the page for which the process faulted
- OS services page fault as follows :

1. Find the location of the desired page on the disk.
2. Find a free frame:

a. if there is a free frame, use it

b. if there is no free frame, use a page-replacement algorithm to select a victim frame.

c. write the victim frame to the disk; change the page and frame tables accordingly.

3. Read the desired page into the newly freed frame; change the page and frame tables.
4. Restart the user process.

- Notice that, if no frames are free, two page transfers (one out and one in) are required. This situation effectively doubles the page-fault service time and increases the effective access time accordingly.
- We can reduce this overhead by using a modify bit(or dirty bit). When this scheme is used, each page or frame has a modify bit associated with it in the hardware. The modify bit for a page is set by the hardware whenever any word or byte in the page is written into, indicating that the page has been modified. When we select a page for replacement, we examine its modify bit. If the bit is set, we know that the page has been modified since it was read in from the disk. In this case, we must write the page to the disk. If the modify bit is not set, however, the page has not been modified since it was read into memory. (but not modified)
- In this case, we need not write the memory page to the disk: it is already there. This technique also applies to read-only pages (for example, pages of binary code). Such pages cannot be modified; thus, they may be discarded when desired.
- This scheme can significantly reduce the time required to service a page fault, since it reduces I/O time by one-half if the page has not been modified.
- **Page replacement is basic to demand paging.** It completes the separation between logical memory and physical memory. With this mechanism, an enormous virtual memory can be provided for programs on a smaller physical memory. With no demand paging, user addresses are mapped into physical addresses, so the two sets of addresses can be different. All the pages of a process still must be in physical memory, however. With demand paging, the size of the logical address space is no longer constrained by physical memory.

- If we have a user process of twenty pages, we can execute it in ten frames simply by using demand paging and using a replacement algorithm. If a page that has been modified is to be replaced, its contents are copied to the disk. A later reference to that page will cause a page fault. At that time, the page will be brought back into memory, perhaps replacing some other page in the process.
- We must solve two major problems to implement demand paging: we must develop a **frame allocation algorithm** and a **page-replacement algorithm.**
- That is, if we have multiple processes in memory, we must decide how many frames to allocate to each process; and when page replacement is required, we must select the frames that are to be replaced.

There are certain page replacement algorithms

1. **FIFO Page Replacement Algorithm:**
   - The simplest page-replacement algorithm is a first-in, first-out (FIFO) algorithm.
   - A FIFO replacement algorithm associates each page with the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen. Notice that it is not strictly necessary to record the time when a page is brought in. We can create a FIFO queue to hold all pages in memory. We replace the page at the head of the queue. When a page is brought into memory, we insert it at the tail of the queue.
   - Usually, as the number of frames increases the page fault rate decreases, but in FIFO page replacement algorithm this assumption was not always true. As we increase the number of frames, sometimes page fault rate increases, this unexpected behaviour is called as **Belady's anomaly.**
2. **Optimal Page Replacement:**
   - It has the lowest page-fault rate of all algorithms and will never suffer from **Belady's anomaly.**
   - It is simply this: Replace the page that will not be used for the longest period of time.
   - Use of this page-replacement algorithm guarantees the lowest possible page fault rate for a fixed number of frames.
   - Unfortunately, the optimal page-replacement algorithm is difficult to implement, because it requires future knowledge of the reference string. As a result, the optimal algorithm is used mainly for comparison studies.
3. **LRU Page Replacement:**
   - If the optimal algorithm is not feasible, perhaps an approximation of the optimal algorithm is possible.
   - In this algorithms we replace the pase that has not been used for the longest period of time. This approach is the **least-recently-used(LRU) algorithm.**
   - LRU replacement associates with each page the time of that page's last use. When a page must be replaced, LRU chooses the page that has not been used for the longest period of time.
4. **Counting Based Page Replacement:**
   - We can keep a counter of the number of references that have been made to each page and develop the following two schemes.
     1. **The least frequently used (LFU) page-replacement algorithm:**

- It requires that the page with the smallest count be replaced. The reason for this selection is that an actively used page should have a large reference count.
- A problem arises, however, when a page is used heavily during the initial phase of a process but then is never used again. Since it was used heavily, it has a large count and remains in memory even though it is no longer needed. One solution is to shift the counts right by 1 bit at regular intervals, forming an exponentially decaying average usage count.

2. **The most frequently used (MFU) page-replacement algorithm:**

- It is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

Thrashing:

High paging activity is called **Thrashing**. A process is thrashing if it is spending more time paging than executing. This reduces Performance of whole system.

Objective Questions :

1. Virtual memory is
   a. an extremely large main memory
   b. an extremely large secondary memory
   c. an illusion of extremely large memory
   d. a type of memory used in super computer

2. Page fault occurs when
   a. the page corrupted by application software
   b. the page is in main memory
   c. the page is not in main memory
   d. one tries to divide a number by 0

3. The page replacement policy that sometimes leads to more page faults when the size of the memory is increased as
   a. FILFO
   b. LRU
   c. no such policy exists
   d. none of the above

4. Fragmentation is
   a. dividing the secondary memory into equal sized fragments
   b. dividing the main memory into equal-sized fragments
   c. fragments of memory words used in a page
   d. fragments of memory words unused in a page.

5. Dirty bit (Modify bit) is used to show the
   a. page with corrupted data
   b. wrong page in the memory
   c. page that is modified after being loaded into cache memory
   d. page that is less frequently accessed

6. Thrashing
   a. reduces page I/O
   b. decreases the degree of multiprogramming(the no. of processes in memory)
   c. implies excessive page I/O
   d. improves the system performance

7. Memory Protection is normally done by
   a. the processor and associated hardware
   b. the operating system
   c. the compiler
   d. the user program
8. What is compaction?
   a. a technique for overcoming internal fragmentation
   b. a paging technique
   c. a technique for overcoming external fragmentation
   d. a technique for overcoming fatal error

9. Memory management technique in which system stores and retrieves data from secondary storage for use in main memory is called
   a. fragmentation
   b. paging
   c. mapping
   d. none of the mentioned

10. Program always deals with
    a. logical address
    b. absolute address
    c. physical address
    d. relative address

File & Storage Management

- File is a named collection of related information that is recorded on secondary storage.
- File is a basic storage unit.
- File is a stream of bytes, bits, lines or records the meaning of which is defined by the file's creator and user.
- File = data + metadata.
- When any new file gets created operating system creates one structure, into which information about that file can be kept, such structure is called as **File Control Block** (in UNIX like operating system's it is called as **iNode**).

- Information about the file also be called as file attributes, such as:
  1. **Name:** the symbolic file name is the only information kept in a human readable form.
  2. **Identifier:** this unique tag, usually a number, identifies the file within the file system; it is non-human readable name for the file.
  3. **Type:** this information needed for the systems that supports different types of files.
  4. **Location:** this information is pointer to a device and to the location of the file on that device.
  5. **Size:** the current size of the file(in bytes, words, or blocks) and possibly the maximum allowed size are included in this attributes.
  6. **Protection:** Access-control information determines who can read, write and execute, and so on.
  7. **Time, date and user identification:** this information may be kept for creation, last modification, and last use. These data can be useful for protection, security, and usage monitoring.

File Operations:

- A file is an abstract data type. To define a file properly, we need to consider the operations that can be performed on files. The operating system can provide system calls to create, write, read, reposition, delete, and truncate files.
- Other common operations include appending new information to the end of an existing file and renaming an existing file. These primitive operations can then be combined to perform other file operations.

- **File System:** It is a way to store data onto the disk in organized manner, such that system can access data from it efficiently and conveniently.
- In computing **file system** is used to control how data is stored and retrieved.

File System Structure:

Boot Block (Boot Sector)	Super Block (Volume Control Table)	Inode List (Master File Table)	Data Blocks

Figure 5.1

- File system structure contains 4 parts: Boot block, Super block, inode List and Data block.
- One disk Partition Cantains one file system. This file system is crested during formatting of that partition

a) **Boot Block:**
- This is the beginning of the file system
- On physical or logical level, this is typically the first sector of logical disk.
- This sector usually contains a small program known as **bootstrap program.**
- Every file system has its own boot block. But if your disk has multiple partitions, only boot block of one partition contains bootstrap program. Remaining partitions boot blocks are kept empty.

b) **Super Block:**
- This block keeps information about: 1) how large the file system is. 2) how many files this file system can store. 3) where is the free space on this file system, etc...

c) **Inode List:**
- Every file has its own, unique Inode.
- Inode is index based(i.e. array), the kernel accesses the Inode by index.
-

d) **data blocks:**
- The data of every file is kept in this data block.

File System Architecture:

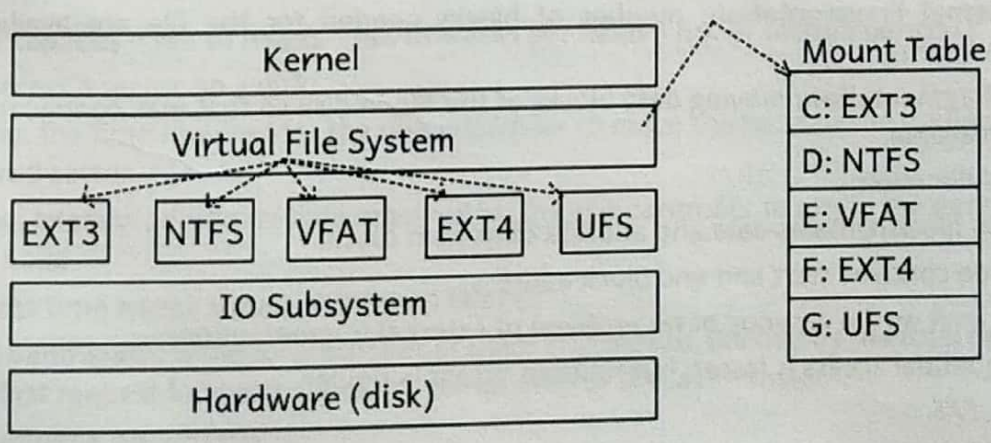


Figure 5.2

+ Virtual File System(VFS): VFS redirect file system request to the appropriate **file system**

manager as per on entry in a mount table.

Each file system has file system manager, which decides how to store data onto the disk.

+ **Mount Table:** contains information about partitions and their filesystems.

e.g. Mount Table

C: FAT

D: NTFS

E: CDFS

+ **Disk Device Driver:** the job of disk device driver is to access (write/read) data on disk.

Step 1: when we call to fwrite() library function in a program it internally gives call to write() system call at kernel level.

Step 2: from inside write() system call, control passed to virtual file system(VFS), VFS refers Mount table which contains information about the partitions and their filesystems.

step 3: VFS redirects file system request to the appropriate file system manager.

step 4: There are dedicated file system managers for each file system. e.g. EXT3 Manager, NTFS Manager, EXT4 Manager, VFAT Manager, etc...

step 5: File System Manager will pass request to IO subsystem, IO subsystem contains **buffer cache** on which operations can be done and from buffer cache data can be read/write by disk device driver, which further communicates with actual disk(hardware).

Disk Space Allocation Mechanisms:

1. **Contiguous Allocation:**
   - Data blocks required for the file are given consecutively.
   - Inode contains starting block number and number of blocks(len).
   - Faster sequential and random access.
   - File cannot grow if blocks after that file are already allocated to another file.
   - **Internal Fragmentation:** file is not using whole data block given to it; rest of data block is wasted. -- can be reduced by decreasing logical block size.
   - **External Fragmentation:** number of blocks needed for the file are available but not contiguous.
   - **Defragmentation:** moving data blocks of the file in disk so that max contiguous free space is available.
2. **Linked Allocation:**
   - Each block contains data and address of its next block.
   - Inode contains start and end block address.
   - No limit on file growing or no problem of external fragmentation.
   - Sequential access is faster; but random access is slower.
   - e.g. FAT
3. **Indexed Allocation:**
   - One special data block is used as index block, which keeps information about remaining data

blocks.

- Inode keeps address of index block.
- Fast sequential and random access. No problem of external fragmentation.
- File can grow easily, but limited by size of index block.
- e.g. UFS, EXT3,...

Types of File Systems:

- Windows: FAT, FAT16, FAT32, NTFS
- Mac OS X: HFS, HFS+
- Android: YAFFSv2 - Yet Another Flash File System, ext4
- iOS: HFSx
- Windows Phone: IMGFS - Image FS, TexFAT – Transaction based extended FAT
- Linux: Ext2/3/4, Reiserfs

Hard Disk: Disk Scheduling:

- Whenever a process needs IO to or from the disk, it issues system call to the operating system. The request specifies several pieces of information:
  a. Operation type read/write
  b. disk address
  c. memary address
  d. number of sectors
- If desired **disk drive** and **controller** are available, the request can be serviced immediately. If the drive or controller is busy, any new requests for service will be placed in the **queue** of pending requests for that drive.
- For a multiprogramming system with many processes, the **disk queue may often have several pending requests. Thus, when one request is completed, the operating system chooses which pending request to service next.**
- Any one of following **disk-scheduling** algorithms can be used to choose next request to be served?
- **disk capacity = no. of heads * no. of tracks per head * no. of sectors per track * 512 bytes.**
- to access a sector on a disk:

1. **seek time**: the time required for the disk controller to move the head to the cylinder containing the desired sector.
2. **rotational latency:** additional time required for the disk controller to rotate the desired sector to the disk head.

disk access time = seek time + rotational latency

- **disk bandwidth:** is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.

Disk Scheduling Algorithms:

to move disk head to the desired cylinders.

1. FCFS Scheduling

2. SSTF Scheduling
3. SCAN – Elevator
4. C-SCAN
5. LOOK – implementation policy of SCAN or C-SCAN.

1. **FCFS Scheduling:**
   - The simplest form of disk scheduling.
   - Generally does not provide the efficient service.

     e.g. a disk queue with requests for I/0 to blocks on cylinders 98,183, 37, 122, 14, 124, 65, 67 in that order.

   Queue = 98, 183, 37, 122, 14, 124, 65, 67 and head is at 53.

   - If the disk head is initially at cylinder 53, it will first move from 53 to 98, then to 183, 37, 122, 14, 124, 65, and finally to 67, for a **total head movement of 640 cylinders.**
   - The **wild swing** from 122 to 14 and then back to 124 illustrates the problem with this schedule.
   - If the requests for cylinders 37 and 14 could be serviced together, before or after the requests for 122 and 124, the total head movement could be decreased substantially, and performance could be thereby improved.
2. **SSTF Scheduling:**
   - It seems reasonable to service all the requests close to the current head position before moving the head far away to service other requests.
   - This assumption is the basis for the shortest seek time first algorithm (SSTF).
   - The SSTF algorithm selects the request with the least seek time from the current head position. Since seek time increases with the number of cylinders traversed by the head, SSTF chooses the pending request closest to the current head position.
   - In the example, request queue, the closest request to the initial head position (53) is at cylinder 65. Once we are at cylinder 65, the next closest request is at cylinder 67. From there, the request at cylinder 37 is closer than the one at 98, so 37 is served next. Continuing, we service the request at cylinder 14, then 98, 122, 124, and finally 183.
   - This scheduling method results in a total head movement of only **236 cylinders**-little more than one-third of the distance needed for FCFS scheduling of this request queue.
   - Clearly, this algorithm gives a substantial improvement in performance.
   - It may cause **starvation** of some requests. Remember that requests may arrive at any time. Suppose that we have two requests in the queue, for cylinders 14 and 186, and while the request from 14 is being serviced, a new request near 14 arrives. This new request will be serviced next, making the request at 186 wait. While this request is being serviced, another request close to 14 could arrive. In theory, a continual stream of requests near one another could cause the request for cylinder 186 to wait indefinitely.
   - Although the SSTF algorithm is a substantial improvement over the FCFS algorithm, **it is not optimal.** In the example, we can do better by moving the head from 53 to 37, even though the latter is not closest, and then to 14, before turning around to service 65, 67, 98, 122,

124, and 183. This strategy reduces the total head movement to **208 cylinders.**

3. **SCAN Scheduling:**
   - In the SCAN algorithm, the disk arm starts at one end of the disk and moves towards the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk. The SCAN algorithm is sometimes called the **elevator algorithm,** since the disk arm behaves just like an elevator in a building, first servicing all the requests going up and then reversing to service requests the other way.

4. **C-SCAN Scheduling:**
   - Circular SCAN is a variant of SCAN designed to provide a more uniform wait time.
   - Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, however, it immediately returns to the beginning of the disk without servicing any requests on the return trip.
   - The C-SCAN scheduling algorithm essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one.

5. **LOOK Scheduling:**
   - As we described them, both SCAN and C-SCAN move the disk head across the full width of the disk. In practice, neither algorithm is often implemented this way.
   - More commonly, the head goes only as far as the final request in each direction. Then, it reverses direction immediately, without going all the way to the end of the disk.
   - Versions of SCAN and C-SCAN that follow this pattern are called **LOOK and C-LOOK scheduling,** because they look for a request before continuing to move in a given direction.

Objective Questions :

1. The system identifies the file by its
   a. name
   b. absolute path
   c. file owner
   d. inode number
2. Which algorithm of disk scheduling selects the request with the least seek time from the current head positions?
   a. SSTF scheduling
   b. FCFS scheduling
   c. SCAN scheduling
   d. LOOK scheduling
3. The time for the disk arm to move the heads to the cylinder containing the desired sector is called
   a. disk time
   b. seek time

c. arm time
d. sector time

4. Which one of the following is not a secondary storage?
   a. magnetic disks
   b. magnetic tapes
   c. RAM
   d. none of the mentioned

5. In which type of allocation method each file occupy a set of contiguous block on the disk?
   a. contiguous allocation
   b. dynamic-storage allocation
   c. linked allocation
   d. indexed allocation

6. What is raw disk?
   a. disk without file system
   b. empty disk
   c. disk lacking logical file system
   d. disk having file system

7. A file control block contains the information about
   a. file ownership
   b. file permissions
   c. location of file contents
   d. all of the mentioned

8. If too little space is allocated to a file,
   a. the file will not work
   b. there will not be any space for the data, as the FCB takes it all
   c. the file cannot be extended
   d. the file cannot be opened

9. In contiguous allocation:
   a. each file must occupy a set of contiguous blocks on the disk
   b. each file is a linked list of disk blocks
   c. all the pointers to scattered blocks are placed together in one location
   d. None of these

10. In linked allocation:
    a. each file must occupy a set of contiguous blocks on the disk
    b. each file is a linked list of disk blocks
    c. all the pointers to scattered blocks are placed together in one location
    d. None of these

ANSWER KEY FOR OBJECTIVE QUESTIONS					
Chapter	**1.**	**2.**	**3.**	**4.**	**5.**
Q.1	b	c	c	c	d
Q.2	a	c	b	c	a
Q.3	d	c	a	a	b
Q.4	d	d	d	d	c
Q.5	c	a	d	c	a
Q.6	b	c	d	c	a
Q.7	c	d	b	a	d
Q.8	b	d	e	c	c
Q.9	d	d	b	b	a
Q.10	c	d	a	a	b