Operating System and Architecture

**CHAPTER 1: REGISTER TRANSFER LANGUAGE AND MICRO-OPERATIONS …………2**

**CHAPTER 2: ARCHITECTURE OF A SIMPLE PROCESSOR ………...............................18**

**CHAPTER 3: OPERATING SYSTEM, TYPES AND PROCESS…………………………………….34**

**CHAPTER 4: MEMORYMANAGEMENT ……………………………………………………………….62**

**CHAPTER 5: INTER-PROCESS COMMUNICATION AND SYNCHRONIZATION………..78**

**CHAPTER 6: FILE SYSTEM………………………………………………………………………………… 99**

**Chapter 1**

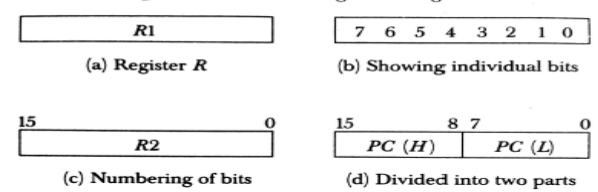
**Register Transfer Language and Micro-operations**

1. Register
2. Register Transfer Language
3. Commonly Used Register
4. Micro-Operation
5. Concept of Bus
6. Data movement among registers
7. Language to represent conditional data transfer, data movement from/to memory.
8. Design of simple Arithmetic, Logic Unit, Control Unit, arithmetic and logical operations.
   1. **BASIC DEFINITIONS:**
   * A digital system is an interconnection of digital hardware modules.
   * The modules are registers, decoders, arithmetic elements, and control logic.
   * The various modules are interconnected with common data and control paths to form a digital computer system.
   * Digital modules are best defined by the registers they contain and the operations that are performed on the data stored in them.
   * The operations executed on data stored in registers are called micro-operations.
   * A micro-operation is an elementary operation performed on the information stored in one or more registers.
   * The result of the operation may replace the previous binary information of a register or may be transferred to another register. ¬ Examples of micro-operations are shift, count, clear, and load.
   * The internal hardware organization of a digital computer is best defined by specifying:
9. The set of registers it contains and their function.
10. The sequence of micro-operations performed on the binary information stored in the registers.
11. The control that initiates the sequence of micro-operations.
    1. **Register**

A Register is a group of flip-flops with each flip-flop capable of storing one bit of information. An n-bit register has a group of n flip-flops and is capable of storing binary information of n-bits.

The flip-flops hold the binary information and gates control when and how new information is transferred into a register. Various types of registers are available commercially. The simplest register is one that consists of only flip-flops with no external gates.

**NOTE:** These days’ registers are also implemented as a register file.



**Fig.1.1 Block Diagram of Register**

**1.2 Register Transfer Language**

The symbolic notation used to describe the micro-operation transfers amongst registers is called Register transfer language. The term register transfer means the availability of hardware logic circuits that can perform a stated micro-operation and transfer the result of the operation to the same or another register. The word language is borrowed from programmers who apply this term to programming languages. This programming language is a procedure for writing symbols to specify a given computational process.

**1.3 Register Transfer**

Information transferred from one register to another is designated in symbolic form by means of replacement operator.

**R2 ← R1**

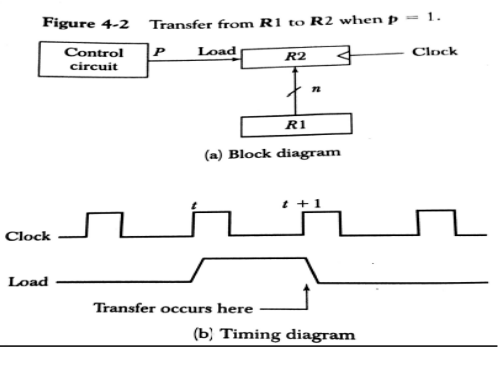
It denotes the transfer of the data from register R1 into R2.

Normally we want the transfer to occur only in predetermined control condition. This can be shown by following if-then statement: if (P=1) then (R2 ← R1)

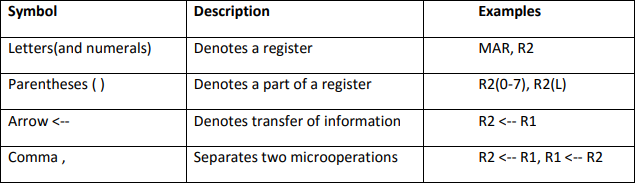
**Following are some commonly used registers:**

1. **Accumulator:** This is the most common register, used to store data taken out from the memory.
2. **General Purpose Registers:** This is used to store data intermediate results during program execution. It can be accessed via assembly programming.
3. **Special Purpose Registers**: Users do not access these registers. These registers are for Computer system.
   1. **MAR:** Memory Address Register is those registers that hold the address for memory unit.
   2. **MBR:** Memory Buffer Register stores instruction and data received from the memory and sent from the memory.
   3. **PC:**Program Counter points to the next instruction to be executed.
   4. **IR:** Instruction Register holds the instruction to be executed.

**Figure 4-2 shows the block diagram that depicts the transfer from R1 to R2.**



* The n outputs of register R1 are connected to the n inputs of register R2.
* The letter n will be used to indicate any number of bits for the register. It will be replaced by an actual number when the length of the register is known.
* Register R2 has a load input that is activated by the control variable P.
* It is assumed that the control variable is synchronized with the same clock as the one applied to the register.
* As shown in the timing diagram, P is activated in the control section by the rising edge of a clock pulse at time t.
* The next positive transition of the clock at time t + 1 finds the load input active and the data inputs of R2 are then loaded into the register in parallel.
* P may go back to 0 at time t+1; otherwise, the transfer will occur with every clock pulse transition while P remains active.
* Even though the control condition such as P becomes active just after time t, the actual transfer does not occur until the register is triggered by the next positive transition of the clock at time t +1.
* A comma is used to separate two or more operations that are executed at the same time.
* The statement T: R2← R1, R1← R2 (exchange operation) denotes an operation that exchanges the contents of two registers during one common clock pulse provided that T=1.
* The basic symbols of the register transfer notation are listed in below table



**1.4 Micro-Operations**

The operations executed on data stored in registers are called micro-operations. A micro- operation is an elementary operation performed on the information stored in one or more registers.

**Example:** Shift, count, clear and load.

**Types of Micro-Operations**

**Micro-Operations**

**Arithmetic micro-operations**

**Logic micro-operations**

**Register transfer micro- operations**

**Shift micro-operations**

**Fig.1.2 Micro operation**

**The Micro-Operations in digital computers are of 4 types:**

1. Register transfer micro-operations transfer binary information from one register to another.
2. Arithmetic micro-operations perform arithmetic operations on numeric data stored in registers.
3. Logic micro-operations perform bit manipulation operation on non-numeric data stored in registers.
4. Shift micro-operations perform shift micro-operations performed on data.

**Arithmetic Micro-operations:**

* The basic arithmetic micro-operations are
* Addition
* Subtraction
* Increment
* Decrement
* Shift
* The arithmetic Micro-operation defined by the statement below specifies the add micro- operation.

**R3 ← R1 + R2**

* It states that the contents of R1 are added to contents of R2 and sum is transferred to R3.
* To implement this statement hardware requires 3 registers and digital component that performs addition
* Subtraction is most often implemented through complementation and addition.
* The subtract operation is specified by the following statement

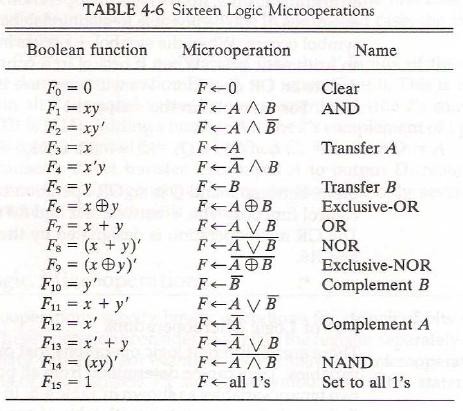
**R3 ← R1 + R2 + 1**

* instead of minus operator, we can write as
* R2 is the symbol for the 1’s complement of R2
* Adding 1 to 1’s complement produces 2’s complement
* Adding the contents of *R1* to the 2's complement of R2 is equivalent to R1-R2.

**Logic Micro-operations:**

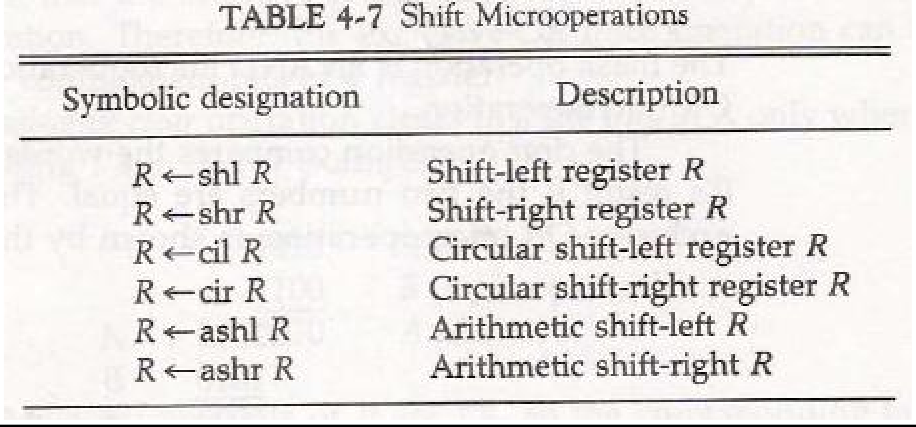
* Logic micro-operations specify binary operations for strings of bits stored in registers.
* These operations consider each bit of the register separately and treat them as binary variables.
* For example, the exclusive-OR micro-operation with the contents of two registers RI and R2 is symbolized by the statement
* It specifies a logic micro-operation to be executed on the individual bits of the registers provided that the control variable P = 1.



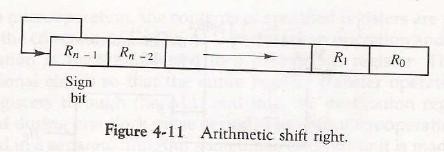


**Shift Micro-operations:**

* Shift micro-operations are used for serial transfer of data.
* The contents of a register can be shifted to the left or the right.
* During a shift-left operation the serial input transfers a bit into the rightmost position.
* During a shift-right operation the serial input transfers a bit into the leftmost position.
* There are three types of shifts: logical, circular, and arithmetic.
* The symbolic notation for the shift micro-operations is shown in Table 4-7.



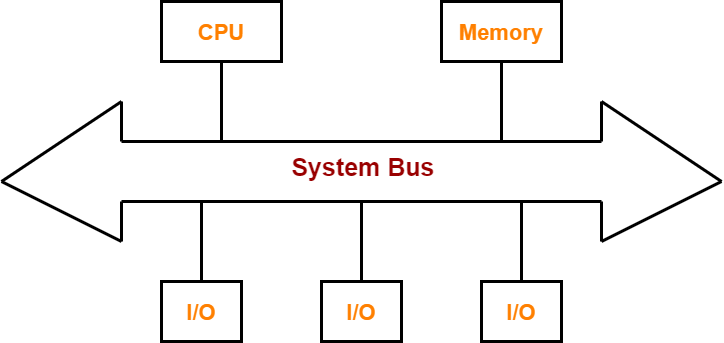
* **Logical Shift:**
* A *logical* shift is one that transfers 0 through the serial input.
* The symbols *shl* and shr for logical shift-left and shift-right microoperations.
* The microoperations that specify a 1-bit shift to the left of the content of register R and a 1-bit shift to the right of the content of register R shown in table 4.7.
* The bit transferred to the end position through the serial input is assumed to be 0 during a logical shift.
* **Circular Shift:**
* The *circular* shift (also known as a *rotate* operation) circulates the bits of the register around the two ends without loss of information.
* This is accomplished by connecting the serial output of the shift register to its serial input.
* We will use the symbols *cil* and *cir* for the circular shift left and right, respectively.
* **Arithmetic Shift:**
* An *arithmetic shift* is a microoperation that shifts a signed binary number to the left or right.
* An arithmetic shift-left multiplies a signed binary number by 2.
* An arithmetic shift-right divides the number by 2.
* Arithmetic shifts must leave the sign bit unchanged because the sign of the number remains the same when it is multiplied or divided by 2.



**1.5 What Is A System Bus?**

 A bus is a set of electrical wires (lines) that connects the various hardware components of a computer system.

* It works as a communication pathway through which information flows from one hardware component to the other hardware component.
* A bus that connects major components (CPU, memory and I/O devices) of a computer system is called as a **System Bus**.



**Fig 1.2 System Bus**

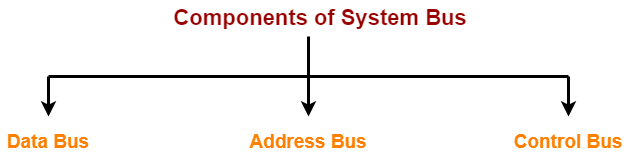
**Need of System Bus**

* A computer system is made of different components such as memory, ALU, registers etc.
* Each component should be able to communicate with other for proper execution of instructions and information flow.
* If we try to implement a mesh topology among different components, it would be really expensive.
* So, we use a common component to connect each necessary component i.e. BUS.

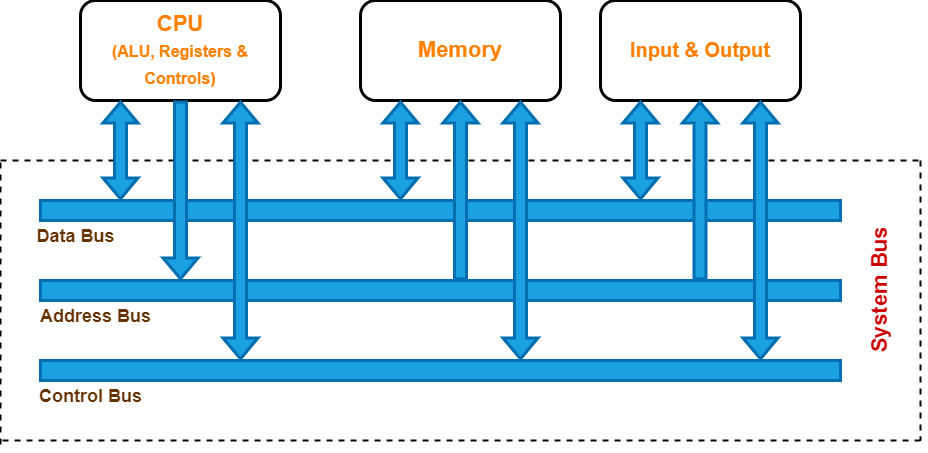
**1.5.1 Components of a System Bus-**

The system bus consists of three major components-

1. Data Bus
2. Address Bus
3. Control Bus



**Fig 1.5 Component of System Bus**



**Fig 1.3 System Bus (Three components)**

## ****Data Bus-****

Data bus is used for transmitting the data / instruction from CPU to memory/IO and vice-versa. It is bi-directional.

### ****Data Bus Width****

* The width of a data bus refers to the number of bits (electrical wires) that the bus can carry at a time.
* Each line carries 1 bit at a time. So, the number of lines in data bus determines how many bits can be transferred parallel.
* The width of data bus is an important parameter because it determines how much data can be transmitted at one time.
* The wider the bus width, faster would be the data flow on the data bus and thus better would be the system performance.

#### ****Examples-****

* A 32-bit bus has thirty two (32) wires and thus can transmit 32 bits of data at a time.
* A 64-bit bus has sixty four (64) wires and thus can transmit 64 bits of data at a time.

## ****Control Bus-****

 As the name suggests, control bus is used to transfer the control and timing signals from one component to the other component.

* The CPU uses control bus to communicate with the devices that are connected to the computer system.
* The CPU transmits different types of control signals to the system components.
* It is bi-directional.

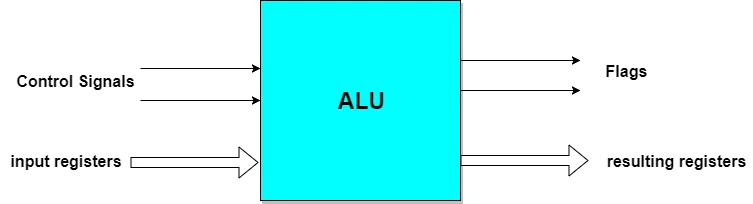
## ****Address Bus-****

The address bus helps to identify the particular location in the memory. Assume that the CPU needs to read data from memory. Then, the address bus helps to identify that specific location. Furthermore, each IO device has a unique ID, and it is the address of that component. The address bus helps to transfer memory addresses of data and IO.

The width of the address bus determines the amount of memory the system can address. When there are ‘n’ address lines, it can directly address 2n memory locations. For example, an 8085 microprocessor has address bus of 16 bits. Therefore, it can access 2 16= 65536 different memory locations.

* Used to carry address signals
* E.g address of memory location or a port (interface – where an input or output device is attached.

**1.6 Arithmetic Logic Unit**

Instead of having individual registers performing the micro-operations, computer system provides a number of registers connected to a common unit called as Arithmetic Logical Unit (ALU). ALU is the main and one of the most important units inside CPU of computer. All the logical and mathematical operations of computer are performed here. The contents of specific register is placed in the in the input of ALU. ALU performs the given operation and then transfer it to the destination register. Fig1.3 depicts.

**Fig.1.**

The ALU is a major component in the processor, the ALU's design and function may be different in the different processors. For case, some ALUs are designed to perform only integer calculations, and some are for floating-point operations. Some processors include a single arithmetic logic unit to perform operations, and others may contain numerous ALUs to complete calculations. The operations performed by ALU are:

* **Logical Operations:** The logical operations consist of NOR, NOT, AND, NAND, OR, XOR, and more.
* **Bit-Shifting Operations**: It is responsible for displacement in the locations of the bits to the right or left by a certain number of places that are known as a multiplication operation.
* **Arithmetic Operations:** Although it performs multiplication and division, this refers to bit addition and subtraction. But multiplication and division operations are more costly to make. In the place of multiplication, addition can be used as a substitute and subtraction for division.

**1.6.1 Configurations of the ALU**

The description of how ALU interacts with the processor is given below. Every arithmetic logic unit includes the following configurations:

* Instruction Set Architecture
* Accumulator
* Stack
* Register to Register
* Register Stack
* Register Memory

**Instruction Set Architecture**

An Instruction Set Architecture (ISA) is part of the abstract model of a computer that defines how the CPU is controlled by the software. The ISA acts as an interface between the hardware and the software, what the processor is capable of doing as well as how it gets done.

The ISA provides the only way through which a user is able to interact with the hardware. It can be viewed as a programmer’s manual because it’s the portion of the machine that’s visible to the assembly language programmer, the compiler writer, and the application programmer.

### The ISA defines the supported data types, the registers, how the hardware manages main memory, key features (such as virtual memory), which instructions a microprocessor can execute, and the input/output model of multiple ISA implementations. The ISA can be extended by adding instructions or other capabilities, or by adding support for larger addresses and data values.

### Accumulator

The intermediate result of every operation is contained by the accumulator, which means Instruction Set Architecture (ISA) is not more complex because there is only required to hold one bit.

### Stack

Whenever the latest operations are performed, these are stored on the stack that holds programs in top-down order, which is a small register. When the new programs are added to execute, they push to put the old programs.

### Register to Register Architecture

It includes a place for 1 destination instruction and 2 source instructions, also known as a 3-register operation machine. This Instruction Set Architecture must be more in length for storing three operands, 1 destination and 2 sources. After the end of the operations, writing the results back to the Registers would be difficult, and also the length of the word should be longer. However, it can be caused to more issues with synchronization if write back rule would be followed at this place.

The MIPS component is an example of the register-to-register Architecture. For input, it uses two operands, and for output, it uses a third distinct component. The storage space is hard to maintain as each needs a distinct memory; therefore, it has to be premium at all times. Moreover, there might be difficult to perform some operations.

### Register - Stack Architecture

Generally, the combination of Register and Accumulator operations is known as for Register - Stack Architecture. The operations that need to be performed in the register-stack Architecture are pushed onto the top of the stack. And its results are held at the top of the stack. With the help of using the Reverse polish method, more complex mathematical operations can be broken down. Some programmers, to represent operands, use the concept of a binary tree. It means that the reverse polish methodology can be easy for these programmers, whereas it can be difficult for other programmers. To carry out Push and Pop operations, there is a need to be new hardware created.

### Register and Memory

In this architecture, one operand comes from the register, and the other comes from the external memory as it is one of the most complicated architectures. The reason behind it is that every program might be very long as they require to be held in full memory space. Generally, this technology is integrated with Register-Register Register technology and practically cannot be used separately.

## Advantages of ALU

ALU has various advantages, which are as follows:

* It supports parallel architecture and applications with high performance.
* It has the ability to get the desired output simultaneously and combine integer and floating-point variables.
* It has the capability of performing instructions on a very large set and has a high range of accuracy.
* Two arithmetic operations in the same code like addition and multiplication or addition and subtraction, or any two operands can be combined by the ALU. For case, A+B\*C.
* Through the whole program, they remain uniform, and they are spaced in a way that they cannot interrupt part in between.
* In general, it is very fast; hence, it provides results quickly.
* There are no sensitivity issues and no memory wastage with ALU.
* They are less expensive and minimize the logic gate requirements.

## Disadvantages of ALU

The disadvantages of ALU are discussed below:

* With the ALU, floating variables have more delays, and the designed controller is not easy to understand.
* The bugs would occur in our result if memory space were definite.
* It is difficult to understand amateurs as their circuit is complex; also, the concept of pipelining is complex to understand.
* A proven disadvantage of ALU is that there are irregularities in latencies.
* Another demerit is rounding off, which impacts accuracy.

**1.7 Micro-processing:** unit is synonymous to central processing unit, CPU used in traditional computer. Microprocessor (MPU) acts as a device or a group of devices which do the following tasks.

* communicate with peripherals devices
* provide timing signal
* direct data flow
* perform computer tasks as specified by the instructions in memory

### ****What Are Control & Timing Signals?****

Control signals are generated in the control unit of CPU. Timing signals are used to synchronize the memory and I/O operations with a CPU clock.

Typical control signals hold by control bus-

* **Memory read –** Data from memory address location to be placed on data bus.
* **Memory write –** Data from data bus to be placed on memory address location.
* **I/O Read –** Data from I/O address location to be placed on data bus.
* **I/O Write –** Data from data bus to be placed on I/O address location.

Other control signals hold by control bus are interrupt, interrupt acknowledge, bus request, bus grant and several others. The type of action taking place on the system bus is indicated by these control signals.

**Example:**

When CPU wants to read or write data, it sends the memory read or memory write control signal on the control bus to perform the memory read or write operation from the main memory. Similarly, when the processor wants to read from an I/O device, it generates the I/O read signal.

## ****Address Bus-****

* As the name suggests, address bus is used to carry address from CPU to memory/IO devices.
* It is used to identify the particular location in memory.
* It carries the source or destination address of data i.e. where to store or from where to retrieve the data.
* It is uni-directional.

**Example-**

 When CPU wants to read or write data, it sends the memory read or memory write control signal on the control bus to perform the memory read or write operation from the main memory and the address of the memory location is sent on the address bus.

If CPU wants to read data stored at the memory location (address) 4, the CPU send the value 4 in binary on the address bus.

### ****Address Bus Width****

* The width of address bus determines the amount of physical memory addressable by the processor.
* In other words, it determines the size of the memory that the computer can use.
* The wider is the address bus, the more memory a computer will be able to use.
* The addressing capacity of the system can be increased by adding more address lines.

**Examples-**

* An address bus that consists of 16 wires can convey 216 (= 64K) different addresses.
* An address bus that consists of 32 wires can convey 232 (= 4G) different addresses.

|  |  |  |
| --- | --- | --- |
| S.N. | **Group** | **Description** |
| 1 | **Address bus** | The 8085 microprocessor has 8 signal lines, A15 - A8 which are uni directional and used as a high order address bus. |
| 2 | **Data bus** | The signal lines AD7 - AD0 are bi-directional for dual purpose. They are used as low order address bus as well as data bus. |
| 3 | **Control signal and Status signal** | **Control Signal**   * **RD bar** − It is a read control signal (active low). If it is active then memory read the data. * **WR bar** − It is write control signal (active low). It is active when written into selected memory.   **Status signal**   * **ALU (Address Latch Enable)** − When ALU is high. 8085 microprocessor use address bus. When ALU is low. 8085 microprocessor is use data bus. * **IO/M bar** − This is a status signal used to differentiate between i/o and memory operations. When it is high, it indicate an i/o operation and when it is low, it indicate memory operation. * **S1 and S0** − These status signals, similar to i/o and memory bar, can identify various operations, but they are rarely used in small system. |
| 4 | **Power supply and frequency signal** | **Vcc** − +5v power supply.  **Vss** − ground reference.  **X, X** − A crystal is connected at these two pins. The frequency is internally divided by two operate system at 3-MHz, the crystal should have a frequency of 6-MHz.  **CLK out** − This signal can be used as the system clock for other devices. |
| 5 | **Externally initiated signal** | **INTR (i/p)** − Interrupt request.  **INTA bar (o/p)** − It is used as acknowledge interrupt.  **TRAP (i/p)** − This is non maskable interrupt and has highest priority.  **HOLD (i/p)** − It is used to hold the executing program.  **HLDA (o/p)** − Hold acknowledge.  **READY (i/p)** − This signal is used to delay the microprocessor read or write cycle until a slow responding peripheral is ready to accept or send data.  **RESET IN bar** − When the signal on this pin goes low, the program counter is set to zero, the bus are tri-stated,& MPU is reset.  **RESET OUT** − This signal indicate that MPU is being reset. The signal can be used to reset other devices.  **RST 7.5, RST 6.5, RST 5.5 (Request interrupt)** − It is used to transfer the program control to specific memory location. They have higher priority than INTR interrupt. |
| 6 | **Serial I/O ports** | The 8085 microprocessor has two signals to implement the serial transmission serial input data and serial output data. |

**Chapter 2**

**Architecture of a simple processor**

1. A simple computer organization and instruction set,
2. Instruction formats
3. Addressing modes
4. Instruction cycle
5. Instruction execution in terms of microinstructions
6. Interrupt cycle, concepts of interrupt and simple I/O organization
7. Synchronous & Asynchronous data transfer
8. Data Transfer Mode
   1. Program Controlled
   2. Interrupt driven
   3. DMA (Direct Memory Access)
9. Pin Diagram of 8086, Architecture of 8086.

**2.1 A simple computer organization and Instruction set**

Computer employer refers back to the manner wherein the additives of a laptop device are prepared and interconnected to carry out particular tasks. One of the maximum essential factors of laptop employer is the set of primary laptop commands that the device can execute.

Basic laptop commands are the primary operations that a laptop device can carry out. These commands are usually divided into 3 categories: statistics motion commands, mathematics and common sense commands, and manipulate commands.

Data motion commands are used to transport statistics among special components of the laptop device. These commands encompass load and shop commands, which circulate statistics among reminiscence and the CPU, and input/output (I/O) commands, which circulate statistics among the CPU and outside devices.

Arithmetic and common sense commands are used to carry out mathematical operations and logical operations on information saved with inside the system. These commands encompass add, subtract, multiply, and divide commands, in addition to common sense commands including AND, OR, and NOT.

## Control commands are used to govern the go with the drift of commands with inside the laptop system. These commands encompass department commands, which switch manipulate to unique elements of this system primarily based totally on distinct conditions, and bounce commands, which switch manipulate to a distinct reminiscence location.

**Computer Organization**: It refers to the operational units and their interconnections that realize the architectural specifications. It describes the function of and design of the various units of digital computer that store and process information. The attributes in computer organization refers to:

* Control signals
* Computer/peripheral interface
* Memory technology

**Computer hardware:** Consists of electronic circuits, displays, magnetic and optical storage media, electromechanical equipment and communication facilities.

**Computer Architecture:** It is concerned with the structure and behavior of the computer. It includes the information formats, the instruction set and techniques for addressing memory. The attributes in computer architecture refers to the:

* Instruction set
* Data representation
* I/O mechanisms
* Addressing techniques.

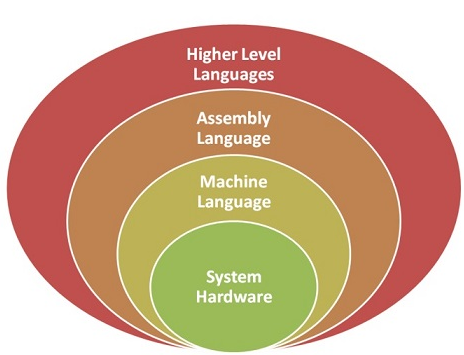
The basic distinction between architecture and organization is: the attributes of the former are visible to programmers whereas the attributes of the later describe show features are implemented in the system.

**An Instruction Set Architecture**

An Instruction Set Architecture (ISA) is part of the abstract model of a computer that defines how the CPU is controlled by the software. The ISA acts as an interface between the hardware and the software, specifying both what the processor is capable of doing as well as how it gets done.

The ISA provides the only way through which a user is able to interact with the hardware. It can be viewed as a programmer’s manual because it’s the portion of the machine that’s visible to the assembly language programmer, the compiler writer, and the application programmer.

The ISA defines the supported data types, the registers, how the hardware manages main memory, key features (such as virtual memory), which instructions a microprocessor can execute, and the input/output model of multiple ISA implementations. The ISA can be extended by adding instructions or other capabilities, or by adding support for larger addresses and data values.



**Fig 2.1 Instruction set Architecture**

## 2.2 Instruction Formats

## The instruction formats are a sequence of bits (0 and 1). These bits, when grouped, are known as fields. Each field of the machine provides specific information to the CPU related to the operation and location of the data.

## The instruction format also defines the layout of the bits for an instruction. It can be of variable lengths with multiple numbers of addresses. These address fields in the instruction format vary as per the organization of the registers in the CPU. The formats supported by the CPU depend upon the Instructions Set Architecture implemented by the processor.

Depending on the multiple address fields, the instruction is categorized as follows:

1. Three address instruction
2. Two address instruction
3. One address instruction
4. Zero address instruction

The operations specified by a computer instruction are executed on data stored in memory or processor registers. The operands residing in processor registers are specified with an address. The registered address is a binary number of k bits that defines one of the 2k registers in the CPU. Thus, a CPU with 16 processors registers R0 through R15 and will have a four-bit register address field.

**Example:** The binary number 0011 will designate register R3.

A computer can have instructions of different lengths containing varying numbers of addresses. The number of address fields of a computer depends on the internal design of its registers. Most of the computers fall into one of three types of CPU organizations:

1. Single accumulator organization.
2. General register organization.
3. Stack organization.

**1. Single Accumulator Organization**

All the operations on a system are performed with an implied accumulator register. The instruction format in this type of computer uses one address field.

For example, the instruction for arithmetic addition is defined by an assembly language instruction ‘ADD.’ Where X is the operand’s address, the ADD instruction results in the operation.

**AC ← AC + M[X]** AC is the accumulator register; M[X] symbolizes the memory word located at address X.

**2. General Register Organization**

The general register type computers employ two or three address fields in their instruction format. Each address field specifies a processor register or a memory. An instruction symbolized by ADD R1, X specifies the operation R1 ← R + M [X].

This instruction has two address fields: register R1 and memory address X.

**3. Stack Organization**

A computer with a stack organization has PUSH and POP instructions that require an address field. Hence, the instruction PUSH X pushes the word at address X to the top of the stack. The stack pointer updates automatically. In stack-organized computers, the operation type instructions don’t require an address field as the operation is performed on the two items on the top of the stack.

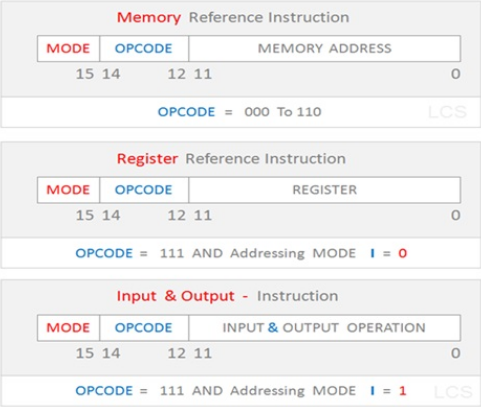


Fig 2.2 Instruction Format

## 2.2.1 Instruction Format types

Each instruction is represented by a sequence of bits within the computer. The instruction is divided into group of bits called field. The way instruction is expressed is known as instruction format. It is usually represented in the form of rectangular box. The instruction format may be of the following types.

## Variable Instruction Formats

These are the instruction formats in which the instruction length varies on the basis of opcode & address specifies. For Example, VAX instructions vary between 1 and 53 bytes while X86 instruction vary between 1 and 17 bytes.

### Format

Variable Instruction Format

**Fig 2.3Variable Instruction Format**

### Advantage

These formats have good code density.

### Drawback

These instruction formats are very difficult to decode and pipeline.

## Fixed Instruction Formats

In this type of instruction format, all instructions are of same size. For Example, MIPS, Power PC, Alpha, ARM.

### Format



**Fig 2.3 Fix Instruction Format**

### Advantage

They are easy to decode & pipeline.

### Drawback

They don't have good code density.

## Hybrid Instruction Formats

In this type of instruction formats, we have multiple format length specified by opcode. For example, IBM 360/70, MIPS 16, Thumb.

### Format



**Fig 2.4 Hybrid Instruction Format**

### Advantage

These compromises between code density & instruction of these type are very easy to decode.

### 2.3 Addressing Modes

* **Direct addressing mode** − In the direct addressing mode, address of the operand is given in the instruction and data is available in the memory location which is provided in instruction. We will move this data in desired location.
* **Indirect addressing mode** − In the indirect addressing mode, the instruction specifies a register which contain the address of the operand. Both internal RAM and external RAM can be accessed via indirect addressing mode.
* **Immediate addressing mode** − In the immediate addressing mode, direct data is given in the operand which move the data in accumulator. It is very fast.
* **Relative addressing mode** − In the relative address mode, the effective address is determined by the index mode by using the program counter instead of general purpose processor register. This mode is called relative address mode.
* **Index addressing mode** − In the index address mode, the effective address of the operand is generated by adding a content value to the contents of the register. This mode is called index address mode.

**Immediate Addressing:**

This is the simplest form of addressing. Here, the operand is given in the instruction.

This mode is used to define constant or set initial values of variables.

The advantage of this mode is that no memory reference other than instruction fetch is required to obtain operand.

The disadvantage is thatthe size ofthe numberis limited to the size ofthe address field because most instruction sets is small compared to word length.

Example: ADD 3 Adds 3 to contents of accumulator and 3 is the operand.



**Fig 2.4 Immediate Addressing**

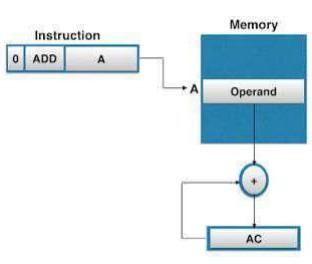
**Direct Addressing:**

* In direct addressing mode, effective address of the operand is given in the address field of the instruction.
* It requires one memory reference to read the operand from the given location and provides only a limited address space.
* Length of the address field is usually less than the word length.
* Example: Move P, Ro

Add Q, Ro

Where P and Q are the address of operand, Ro is any register.

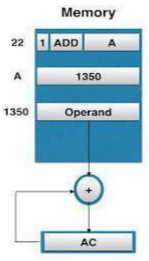
Sometimes Accumulator (AC)is the default register. Then the instruction will look like: Add A

****

**Fig 2.5 Direct Addressing**

**Indirect or Pseudo direct Addressing:**

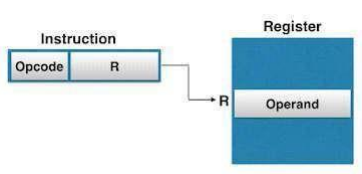
* Indirect addressing mode, the address field of the instruction refers to the address of a word in memory, which in turn contains the full length address of the operand.
* The address field of instruction gives the memory address where on, the operand is stored in memory.
* Control fetches the instruction from memory and then uses its address part to access memory again to read Effective Address.
* The advantage of this mode is that for the word length of N, an address space of 2N can be addressed.
* The disadvantage is that instruction execution requires two memory references to fetch the operand.
* Multilevel or cascaded indirect addressing can also be used.
* Example: Effective Address (EA) = (A). The operand will be present in the memory location A

****

**Fig 2.6 Indirect Addressing**

**Register Addressing:**

* Register addressing mode is similar to direct addressing. The only difference is that the address field of the instruction refers to a register rather than a memory location.
* or 4 bits are used as address field in the instruction to refer 8 to 16 generate purpose registers (GPR).
* The operands are in registers that reside within the CPU.
* The instruction specifies a register in CPU, which contain the operand.
* There is no need to compute the actual address as the operand is in a register and to get operand there is no memory access involved.
* The advantages of register addressing are small address field is needed in the instruction and faster instruction fetch.
* The disadvantages include very limited address space and usage of multiple registers helps in performance but it complicates the instructions.
* Example: MOV AX,BX



**Fig 2.7 Register Addressing**

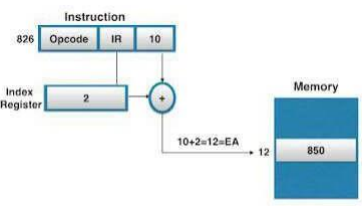
**Indexed addressing:**

The content of Index Register is added to direct address part of instruction to obtaintheeffectiveaddress.Theregisterindirectaddressingfieldofinstructionpointto Index Register, which is a special CPU registerthat contain an Indexed value, and direct addressing field contain base address.

The data array is in memory and each operand in the array is stored in memory relative to base address. The distance between the beginning address and the address of operand is the indexed value stored in indexed register.

Any operand in the array can be accessed with the same instruction, which provided thatthe index register contains the correct index value i.e., the index register can be incremented to facilitate access to consecutive operands.

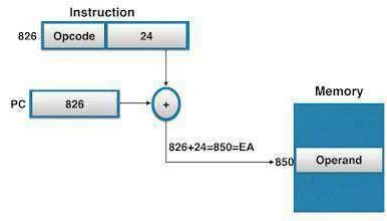
Example: EA=A+ Index



**Fig 2.8 Indexed Addressing**

**Relative addressing:**

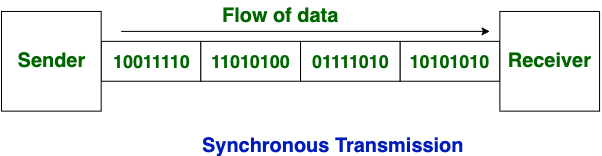
* The content of program counter is added to the address part of instruction to obtain the Effective Address. The address field of the instruction is added to implicitly reference register Program Counter, to obtain effective address.
* Example: EA=A+PC
* Assume that PC contains the value 825 and the address part of instruction contain the value 24, then the instruction at location 825 is read from memory during fetch phase and the Program Counter is then incremented by one to 826. Here both PC and instruction contains address. The effective address computation for relative address mode is826+24=850.



**Fig 2.9 Indexed Addressing**

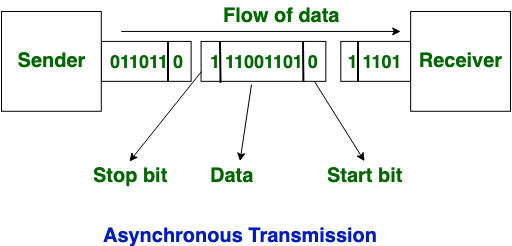
**2.4 Synchronous and Asynchronous Transmission**

Synchronous Transmission, data is sent in form of blocks or frames. This transmission is the full duplex type. Between sender and receiver the synchronization is compulsory. In Synchronous transmission. There is no gap present between data. It is more efficient and more reliable than asynchronous transmission to transfer the large amount of data.



**Fig 2.10 Synchronous Transmission**

Asynchronous Transmission, data is sent in form of byte or character. This transmission is the half duplex type transmission. In this transmission start bits and stop bits are added with data. It does not require synchronization.



**Fig 2.10 Asynchronous Transmission**

**The difference between Synchronous and Asynchronous Transmission**:

|  |  |  |
| --- | --- | --- |
| **S.NO** | **Synchronous Transmission** | **Asynchronous Transmission** |
| 1. | In Synchronous transmission, Data is sent in form of blocks or frames. | In asynchronous transmission, Data is sent in form of byte or character. |
| 2. | Synchronous transmission is fast. | Asynchronous transmission is slow. |
| 3. | Synchronous transmission is costly. | Asynchronous transmission is economical. |
| 4. | In Synchronous transmission, time interval of transmission is constant. | In asynchronous transmission, time interval of transmission is not constant, it is random. |
| 5. | In Synchronous transmission, There is no gap present between data. | In asynchronous transmission, There is present gap between data. |
| 6. | Efficient use of transmission line is done in synchronous transmission. | While in asynchronous transmission, transmission line remains empty during gap in character transmission. |
| 7. | Synchronous transmission needs precisely synchronized clocks for the information of new bytes. | Asynchronous transmissions have no need of synchronized clocks as parity bit is used in this transmission for information of new bytes. |

**2.5 Direct Memory Access (DMA)**

[DMA](https://www.geeksforgeeks.org/direct-memory-access-with-dma-controller-8257-8237/) Controller is a hardware device that allows I/O devices to directly access memory with less participation of the processor. DMA controller needs the same old circuits of an interface to communicate with the CPU and Input/Output devices.

The DMA controller has three registers as follows.

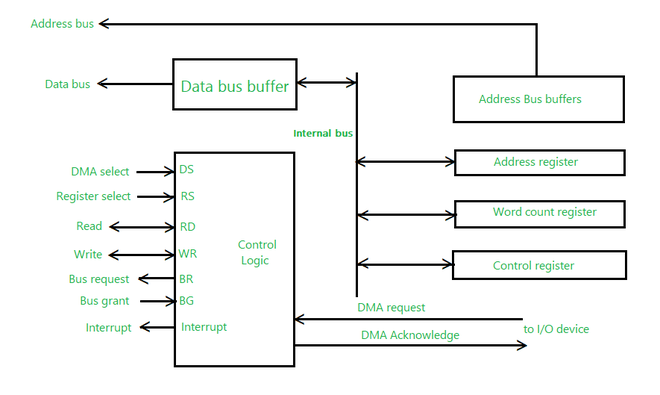
* **Address register –**It contains the address to specify the desired location in memory.
* **Word count register –**It contains the number of words to be transferred.
* **Control register –**It specifies the transfer mode.

**Note:** All registers in the DMA appear to the [CPU](https://www.geeksforgeeks.org/difference-between-cpu-and-gpu/) as I/O interface registers. Therefore, the CPU can both read and write into the DMA registers under program control via the data bus.

**Explanation:**

The CPU initializes the DMA by sending the given information through the [data bus](https://www.geeksforgeeks.org/introduction-of-alu-and-data-path/).

* The starting addresses of the memory block where the data is available (to read) or where data are to be stored (to write).
* It also sends word count which is the number of words in the memory block to be read or write.
* Control to define the mode of transfer such as read or write.
* A control to begin the DMA transfer.



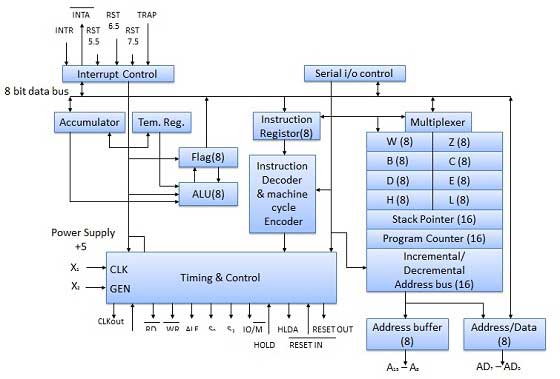
**Fig 2.11 Direct Memory Access (DMA)**

## 

## 2.6 8085 Microprocessor

The 8085 microprocessor is an 8-bit general purpose microprocessor which is capable to address 64k of memory. This processor has forty pins, requires +5 V single power supply and a 3-MHz single-phase clock.

**Block Diagram**

****

## Fig 2.12 8085 Microprocessor

## Flags

Flags are programmable. They can be used to store and transfer the data from the registers by using instruction. The ALU includes five flip-flops that are set and reset according to data condition in accumulator and other registers.

* **S (Sign) flag** − After the execution of an arithmetic operation, if bit D7 of the result is 1, the sign flag is set. It is used to signed number. In a given byte, if D7 is 1 means negative number. If it is zero means it is a positive number.
* **Z (Zero) flag** − The zero flag is set if ALU operation result is 0.
* **AC (Auxiliary Carry) flag** − In arithmetic operation, when carry is generated by digit D3 and passed on to digit D4, the AC flag is set. This flag is used only internally BCD operation.
* **P (Parity) flag** − After arithmetic or logic operation, if result has even number of 1s, the flag is set. If it has odd number of 1s, flag is reset.
* **C (Carry) flag** − If arithmetic operation result is in a carry, the carry flag is set, otherwise it is reset.

## Register section

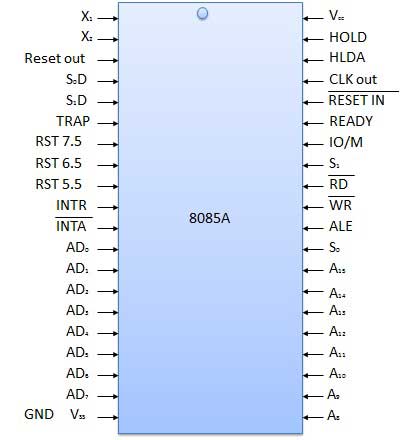
It is basically a storage device and transfers data from registers by using instructions.

* **Stack Pointer (SP)** −The stack pointer is also a 16-bit register which is used as a memory pointer. It points to a memory location in Read/Write memory known as stack. In between execution of program, sometime data to be stored in stack. The beginning of the stack is defined by loading a 16-bit address in the stack pointer.
* **Program Counter (PC)** −This 16-bit register deals with fourth operation to sequence the execution of instruction. This register is also a memory pointer. Memory locations have 16-bit address. It is used to store the execution address. The function of the program counter is to point to memory address from which next byte is to be fetched.
* **Storage registers** −These registers store 8-bit data during a program execution. These registers are identified as B, C, D, E, H, L. They can be combined as register pair BC, DE and HL to perform some 16 bit operations.

## Time and Control Section

This unit is responsible to synchronize Microprocessor operation as per the clock pulse and to generate the control signals which are necessary for smooth communication between Microprocessor and peripherals devices. The RD bar and WR bar signals are synchronous pulses which indicates whether data is available on the data bus or not. The control unit is responsible to control the flow of data between microprocessor, memory and peripheral devices. The entire signal can be classified into six groups.

### PIN diagram

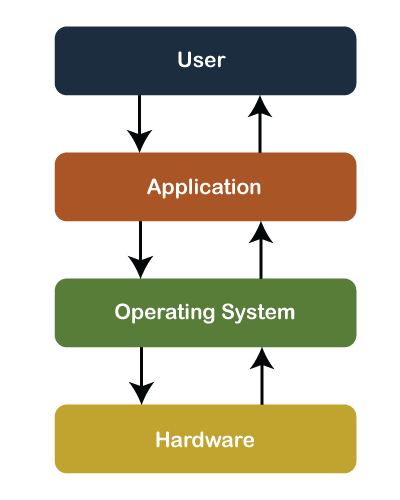


**Chapter 3**

**Operating System, types and Process**

1. Evolution of operating systems (History of evolution of OS with the generations of computers),
2. Types of operating systems,
   1. Multitasking,
   2. Timesharing,
   3. Multithreading,
   4. Multiprogramming and,
   5. Real time operating systems
3. Processes
   1. Process concept
   2. The process control block
   3. Systems programmer's view of processes
   4. Operating system services for process management
   5. Scheduling algorithms
      * 1. First come first serve
        2. Round Robin
        3. Shortest run time next
        4. Highest response ratio next
        5. Multilevel Feedback Queues
4. Performance evaluation of scheduling algorithms stated above.

### 3.1 OPERATING SYSTEM

The operating system is a system program that serves as an interface between the computing system and the end-user. Operating systems create an environment where the user can run any programs or communicate with software or applications in a comfortable and well-organized way.

**Fig 3.1 Operating System**

**Evolution of operating systems**

## The First Generation (1945 - 1955): Vacuum Tubes and Plugboards



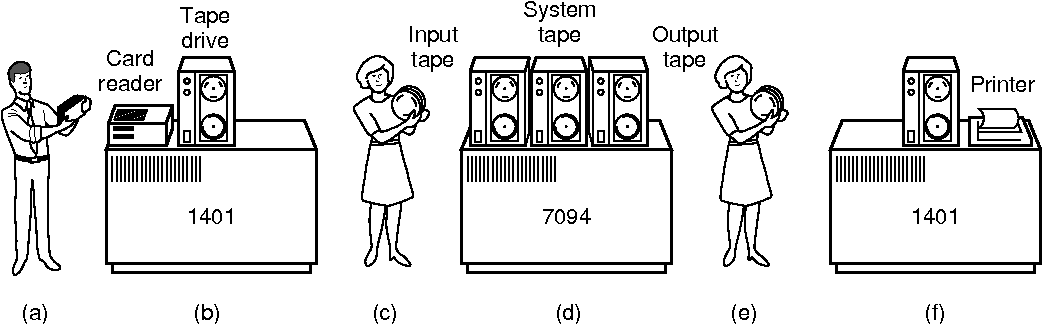
**Fig 3.2 First Generation**

Digital computers were not constructed until the Second World War. Calculating engines with mechanical relays were built at that time. However, the mechanical relays were very slow and were later replaced with vacuum tubes. These machines were enormous but were still very slow.

These early computers were designed, built and maintained by a single group of people. Programming languages were unknown and there were no operating systems so all the programming was done in machine language. All the problems were simple numerical calculations.

By the 1950’s punch cards were introduced and this improved the computer system. Instead of using plugboards, programs were written on cards and read into the system.

## The Second Generation (1955 - 1965): Transistors and Batch Systems



**Fig 3.3 Second Generation**

Transistors led to the development of the computer systems that could be manufactured and sold to paying customers. These machines were known as mainframes and were locked in air-conditioned computer rooms with staff to operate them.

The Batch System was introduced to reduce the wasted time in the computer. A tray full of jobs was collected in the input room and read into the magnetic tape. After that, the tape was rewound and mounted on a tape drive. Then the batch operating system was loaded in which read the first job from the tape and ran it. The output was written on the second tape. After the whole batch was done, the input and output tapes were removed and the output tape was printed.

## The Third Generation (1965 - 1980): Integrated Circuits and Multiprogramming

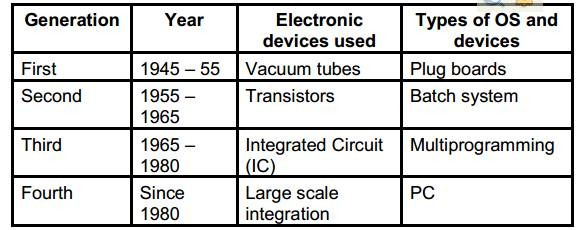
Until the 1960’s, there were two types of computer systems i.e the scientific and the commercial computers. These were combined by IBM in the System/360. This used integrated circuits and provided a major price and performance advantage over the second generation systems.

The third generation operating systems also introduced multiprogramming. This meant that the processor was not idle while a job was completing its I/O operation. Another job was scheduled on the processor so that its time would not be wasted.

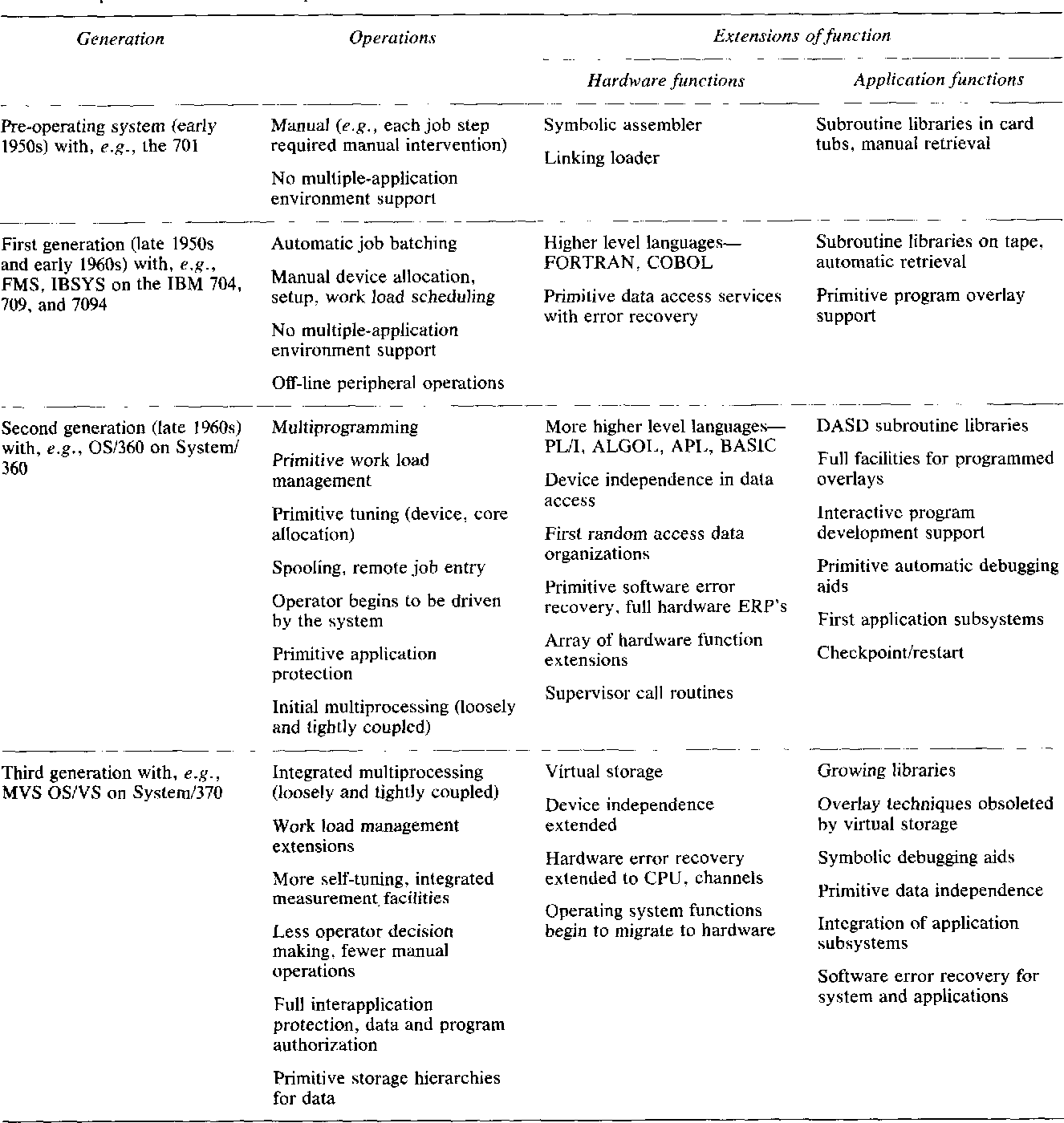
## The Fourth Generation (1980 - Present): Personal Computers

Personal Computers were easy to create with the development of large-scale integrated circuits. These were chips containing thousands of transistors on a square centimeter of silicon. Because of these, microcomputers were much cheaper than minicomputers and that made it possible for a single individual to own one of them.

The advent of personal computers also led to the growth of networks. This created network operating systems and distributed operating systems. The users were aware of a network while using a network operating system and could log in to remote machines and copy files from one machine to another.

****

**Fig 3.4 Fourth Generation**

****

**Fig 3.5 First Generation**

**Types of Operating Systems**

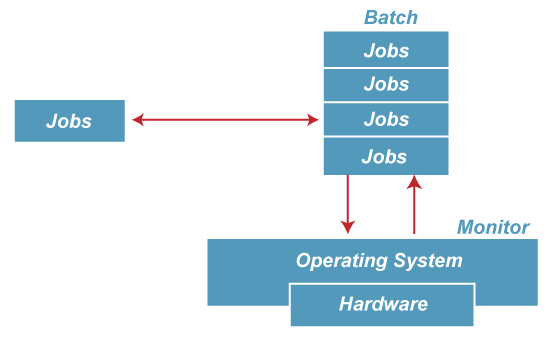
An operating system is a well-organized collection of programs that manages the computer hardware. It is a type of system software that is responsible for the smooth functioning of the computer system.

* Batch OS
* Multiprogramming OS
* Multiprocessing OS
* Multitasking
* Real Time OS
* Network OS
* Distributed OS

**3.2 Types of Operating System**

## Batch Operating System

* In the 1970s, Batch processing was very popular. In this technique, similar types of jobs were batched together and executed in time. People were used to having a single computer which was called a mainframe.
* In Batch operating system, access is given to more than one person; they submit their respective jobs to the system for the execution.
* The system put all of the jobs in a queue on the basis of first come first serve and then executes the jobs one by one. The users collect their respective output when all the jobs get executed.



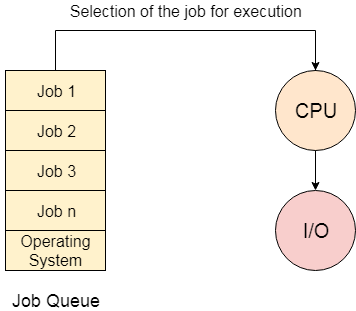
**Fig 3.5 Batch OS**

### Advantages of Batch OS

* The use of a resident monitor improves computer efficiency as it eliminates CPU time between two jobs.

### Disadvantages of Batch OS

**1. Starvation:** Batch processing suffers from starvation. **For Example:**



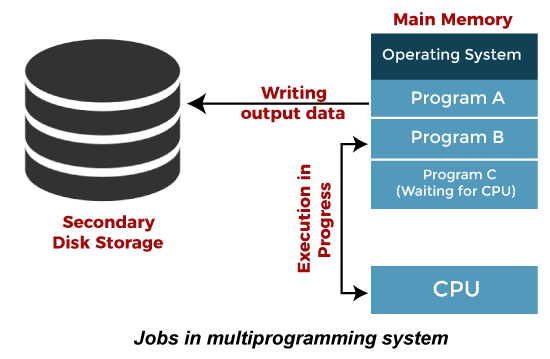
There are five jobs J1, J2, J3, J4, and J5, present in the batch. If the execution time of J1 is very high, then the other four jobs will never be executed, or they will have to wait for a very long time. Hence the other processes get starved.

**2. Not Interactive:** Batch Processing is not suitable for jobs that are dependent on the user's input. If a job requires the input of two numbers from the console, then it will never get it in the batch processing scenario since the user is not present at the time of execution.

**Multiprogramming Operating System**

Multiprogramming is an extension to batch processing where the CPU is always kept busy. Each process needs two types of system time: CPU time and IO time.

In a multiprogramming environment, when a process does its I/O. The CPU can start the execution of other processes. Therefore, multiprogramming improves the efficiency of the system.



**Fig 3.6 Multiprogramming OS**

### Advantages of Multiprogramming OS

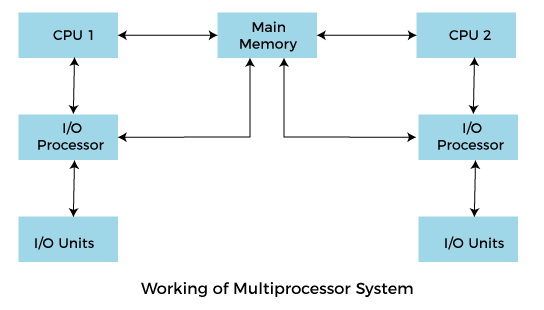
* Throughout the system, it increased as the CPU always had one program to execute.
* Response time can also be reduced.

### Disadvantages of Multiprogramming OS

* Multiprogramming systems provide an environment in which various systems resources are used efficiently, but they do not provide any user interaction with the computer system.

## Multiprocessing Operating System

In Multiprocessing, Parallel computing is achieved. There are more than one processors present in the system which can execute more than one process at the same time. This will increase the throughput of the system.



**Fig 3.7 Multiprocessing OS**

**Advantages of Multiprocessing operating system:**

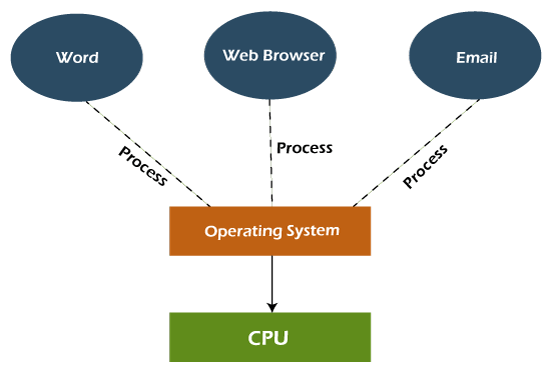
* **Increased reliability:** Due to the multiprocessing system, processing tasks can be distributed among several processors. This increases reliability as if one processor fails, the task can be given to another processor for completion.
* **Increased throughout:** As several processors increase, more work can be done in less.

**Disadvantages of Multiprocessing operating System**

* Multiprocessing operating system is more complex and sophisticated as it takes care of multiple CPUs simultaneously.

### Multitasking Operating System

The multitasking operating system is a logical extension of a multiprogramming system that enables **multiple** programs simultaneously. It allows a user to perform more than one computer task at the same time.



**Fig 3.7 Multitasking OS**

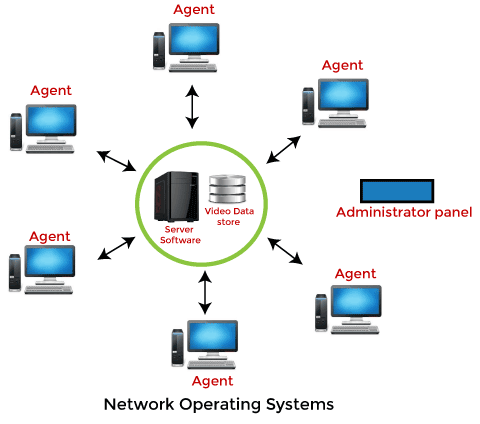
### Advantages of Multitasking operating system

* This operating system is more suited to supporting multiple users simultaneously.
* The multitasking operating systems have well-defined memory management.

### Disadvantages of Multitasking operating system

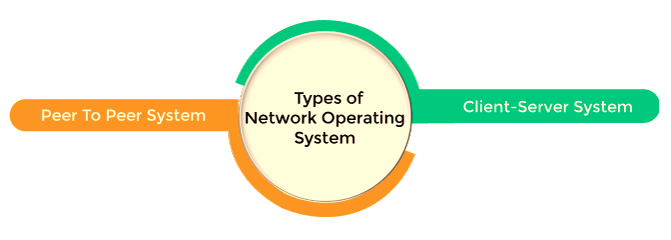
* The multiple processors are busier at the same time to complete any task in a multitasking environment, so the CPU generates more heat.

## Network Operating System



**Fig 3.8 Network OS**

An Operating system, which includes software and associated protocols to communicate with other computers via a network conveniently and cost-effectively, is called Network Operating System.



### Advantages of Network Operating System

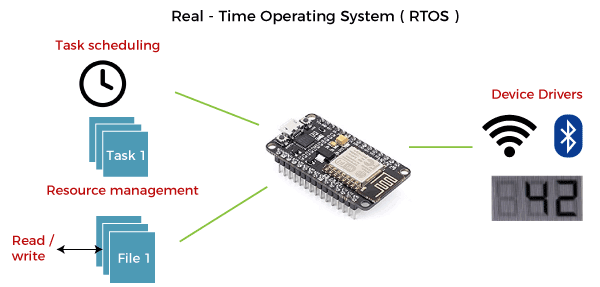
* In this type of operating system, network traffic reduces due to the division between clients and the server.
* This type of system is less expensive to set up and maintain.

### Disadvantages of Network Operating System

* In this type of operating system, the failure of any node in a system affects the whole system.
* Security and performance are important issues. So trained network administrators are required for network administration.

## Real Time Operating System

In Real-Time Systems, each job carries a certain deadline within which the job is supposed to be completed, otherwise, the huge loss will be there, or even if the result is produced, it will be completely useless.



**Fig 3.9 Real Time OS**

### Advantages of Real-time operating system:

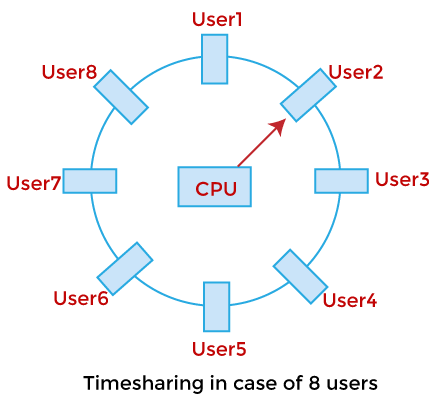
* Easy to layout, develop and execute real-time applications under the real-time operating system.
* In a Real-time operating system, the maximum utilization of devices and systems.

### Disadvantages of Real-time operating system:

* Real-time operating systems are very costly to develop.
* Real-time operating systems are very complex and can consume critical CPU cycles.

## Time-Sharing Operating System

In the Time Sharing operating system, computer resources are allocated in a time-dependent fashion to several programs simultaneously. Thus it helps to provide a large number of user's direct access to the main computer. It is a logical extension of multiprogramming. In time-sharing, the CPU is switched among multiple programs given by different users on a scheduled basis.



**Fig 3.10 Time Sharing OS**

A time-sharing operating system allows many users to be served simultaneously, so sophisticated CPU scheduling schemes and Input/output management are required.

Time-sharing operating systems are very difficult and expensive to build.

### Advantages of Time Sharing Operating System

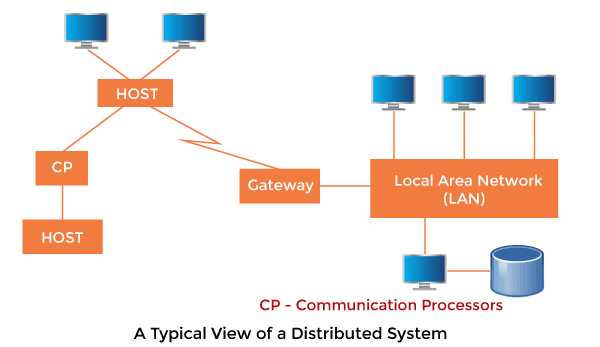
* The time-sharing operating system provides effective utilization and sharing of resources.
* This system reduces CPU idle and response time.

### Disadvantages of Time Sharing Operating System

* Data transmission rates are very high in comparison to other methods.
* Security and integrity of user programs loaded in memory and data need to be maintained as many users access the system at the same time.

## Distributed Operating System

The Distributed Operating system is not installed on a single machine, it is divided into parts, and these parts are loaded on different machines. A part of the distributed Operating system is installed on each machine to make their communication possible. Distributed Operating systems are much more complex, large, and sophisticated than Network operating systems because they also have to take care of varying networking protocols.



**Fig 3.11 Network OS**

### Advantages of Distributed Operating System

* The distributed operating system provides sharing of resources.
* This type of system is fault-tolerant.

### Disadvantages of Distributed Operating System

* Protocol overhead can dominate computation cost.

**3.3 Process Management**

A Program does nothing unless its instructions are executed by a CPU. A program in execution is called a process. In order to accomplish its task, process needs the computer resources.

There may exist more than one process in the system which may require the same resource at the same time. Therefore, the operating system has to manage all the processes and the resources in a convenient and efficient way.

Some resources may need to be executed by one process at one time to maintain the consistency otherwise the system can become inconsistent and deadlock may occur.

The operating system is responsible for the following activities in connection with Process Management

1. Scheduling processes and threads on the CPUs.
2. Creating and deleting both user and system processes.
3. Suspending and resuming processes.
4. Providing mechanisms for process synchronization.
5. Providing mechanisms for process communication.

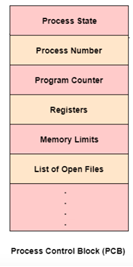
**3.3.1 Process Control Block**

Process Control Block is a data structure that contains information of the process related to it. The process control block is also known as a task control block, entry of the process table, etc.

It is very important for process management as the data structuring for processes is done in terms of the PCB. It also defines the current state of the operating system.

**Structure of the Process Control Block**

The process control stores many data items that are needed for efficient process management. Some of these data items are explained with the help of the given diagram –



**Fig 3.12 PCB**

The following are the data items −

### Process State: This specifies the process state i.e. new, ready, running, waiting or terminated.

### Process Number: This shows the number of the particular process.

### Program Counter: This contains the address of the next instruction that needs to be executed in the process.

### Registers: This specifies the registers that are used by the process. They may include accumulators, index registers, stack pointers, general purpose registers etc.

**List of Open Files:** These are the different files that are associated with the process

**CPU Scheduling Information:** The process priority, pointers to scheduling queues etc. is the CPU scheduling information that is contained in the PCB. This may also include any other scheduling parameters.

**Memory Management Information:** The memory management information includes the page tables or the segment tables depending on the memory system used. It also contains the value of the base registers, limit registers etc.

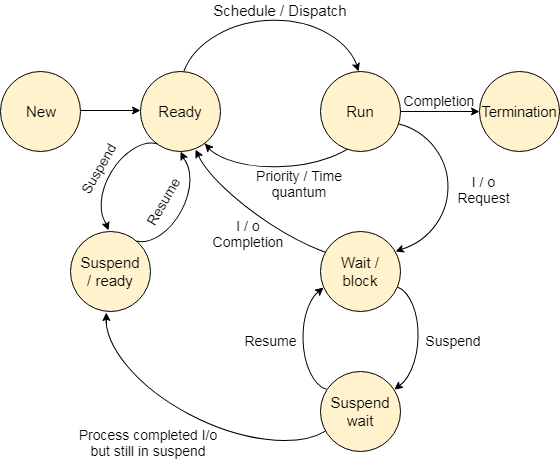
**I/O Status Information:** This information includes the list of I/O devices used by the process, the list of files etc.

**Accounting information:** The time limits, account numbers, amount of CPU used, process numbers etc. are all a part of the PCB accounting information.

**Location of the Process Control Block**

The process control block is kept in a memory area that is protected from the normal user access. This is done because it contains important process information. Some of the operating systems place the PCB at the beginning of the kernel stack for the process as it is a safe location.

**3.3.2 Process States**

****

**Fig 3.13 Process States**

### 1. New :

A program which is going to be picked up by the OS into the main memory is called a new process.

### 2. Ready

Whenever a process is created, it directly enters in the ready state, in which, it waits for the CPU to be assigned. The OS picks the new processes from the secondary memory and put all of them in the main memory.

The processes which are ready for the execution and reside in the main memory are called ready state processes. There can be many processes present in the ready state.

### 3. Running

One of the processes from the ready state will be chosen by the OS depending upon the scheduling algorithm. Hence, if we have only one CPU in our system, the number of running processes for a particular time will always be one. If we have n processors in the system then we can have n processes running simultaneously.

### 4. Block or wait

From the Running state, a process can make the transition to the block or wait state depending upon the scheduling algorithm or the intrinsic behavior of the process.

When a process waits for a certain resource to be assigned or for the input from the user then the OS move this process to the block or wait state and assigns the CPU to the other processes.

### 5. Completion or termination

When a process finishes its execution, it comes in the termination state. All the context of the process (Process Control Block) will also be deleted the process will be terminated by the Operating system.

### 6. Suspend ready

A process in the ready state, which is moved to secondary memory from the main memory due to lack of the resources (mainly primary memory) is called in the suspend ready state.

If the main memory is full and a higher priority process comes for the execution then the OS have to make the room for the process in the main memory by throwing the lower priority process out into the secondary memory. Suspend ready processes remain in the secondary memory until the main memory gets available.

### 7. Suspend wait

Instead of removing the process from the ready queue, it's better to remove the blocked process which is waiting for some resources in the main memory. Since it is already waiting for some resource to get available hence it is better if it waits in the secondary memory and make room for the higher priority process. These processes complete their execution once the main memory gets available and their wait is finished.

# 3.4 Scheduling Algorithms

There are various algorithms which are used by the Operating System to schedule the processes on the processor in an efficient way.

## The Purpose of a Scheduling algorithm

1. Maximum CPU utilization
2. Fare allocation of CPU
3. Maximum throughput
4. Minimum turnaround time
5. Minimum waiting time
6. Minimum response time

**There are the following algorithms which can be used to schedule the jobs.**

### 1. First Come First Serve

It is the simplest algorithm to implement. The process with the minimal arrival time will get the CPU first. The lesser the arrival time, the sooner will the process gets the CPU. It is the non-preemptive type of scheduling.

### 2. Round Robin

In the Round Robin scheduling algorithm, the OS defines a time quantum (slice). All the processes will get executed in the cyclic way. Each of the process will get the CPU for a small amount of time (called time quantum) and then get back to the ready queue to wait for its next turn. It is a preemptive type of scheduling.

### 3. Shortest Job First

The job with the shortest burst time will get the CPU first. The lesser the burst time, the sooner will the process get the CPU. It is the non-preemptive type of scheduling.

### 4. Shortest remaining time first

It is the preemptive form of SJF. In this algorithm, the OS schedules the Job according to the remaining time of the execution.

### 5. Priority based scheduling

In this algorithm, the priority will be assigned to each of the processes. The higher the priority, the sooner will the process get the CPU. If the priority of the two processes is same then they will be scheduled according to their arrival time.

### 6. Highest Response Ratio Next

In this scheduling Algorithm, the process with highest response ratio will be scheduled next. This reduces the starvation in the system.

### FCFS Scheduling

**First come first serve** (FCFS) scheduling algorithm simply schedules the jobs according to their arrival time. The job which comes first in the ready queue will get the CPU first. The lesser the arrival time of the job, the sooner will the job get the CPU. FCFS scheduling may cause the problem of starvation if the burst time of the first process is the longest among all the jobs.

## Advantages of FCFS

* Simple
* Easy
* First come, First serve

## Disadvantages of FCFS

* The scheduling method is non preemptive, the process will run to the completion.
* Due to the non-preemptive nature of the algorithm, the problem of starvation may occur.
* Although it is easy to implement, but it is poor in performance since the average waiting time is higher as compare to other scheduling algorithms.

### Example

Let's take an example of The FCFS scheduling algorithm. In the Following schedule, there are 5 processes with process ID **P0, P1, P2, P3 and P4**. P0 arrives at time 0, P1 at time 1, P2 at time 2, P3 arrives at time 3 and Process P4 arrives at time 4 in the ready queue. The processes and their respective Arrival and Burst time are given in the following table.

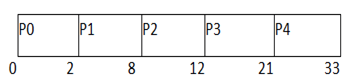
The Turnaround time and the waiting time are calculated by using the following formula.

1. Turn Around Time = Completion Time - Arrival Time
2. Waiting Time = Turnaround time - Burst Time

The average waiting Time is determined by summing the respective waiting time of all the processes and divided the sum by the total number of processes.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Process ID** | **Arrival Time** | **Burst Time** | **Completion Time** | **Turn Around Time** | **Waiting Time** |
| 0 | 0 | 2 | 2 | 2 | 0 |
| 1 | 1 | 6 | 8 | 7 | 1 |
| 2 | 2 | 4 | 12 | 10 | 6 |
| 3 | 3 | 9 | 21 | 18 | 9 |
| 4 | 6 | 12 | 33 | 29 | 17 |

**Avg Waiting Time= 31/5**



# Shortest Job First (SJF) Scheduling

Till now, we were scheduling the processes according to their arrival time (in FCFS scheduling). However, SJF scheduling algorithm, schedules the processes according to their burst time.

In SJF scheduling, the process with the lowest burst time, among the list of available processes in the ready queue, is going to be scheduled next.

However, it is very difficult to predict the burst time needed for a process hence this algorithm is very difficult to implement in the system.

## Advantages of SJF

1. Maximum throughput
2. Minimum average waiting and turnaround time

## Disadvantages of SJF

1. May suffer with the problem of starvation
2. It is not implementable because the exact Burst time for a process can't be known in advance.

There are different techniques available by which, the CPU burst time of the process can be determined. We will discuss them later in detail.

### Example

In the following example, there are five jobs named as P1, P2, P3, P4 and P5. Their arrival time and burst time are given in the table below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **PID** | **Arrival Time** | **Burst Time** | **Completion Time** | **Turn Around Time** | **Waiting Time** |
| 1 | 1 | 7 | 8 | 7 | 0 |
| 2 | 3 | 3 | 13 | 10 | 7 |
| 3 | 6 | 2 | 10 | 4 | 2 |
| 4 | 7 | 10 | 31 | 24 | 14 |
| 5 | 9 | 8 | 21 | 12 | 4 |

Since, No Process arrives at time 0 hence; there will be an empty slot in the **Gantt chart** from time 0 to 1 (the time at which the first process arrives).

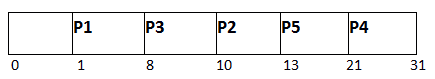
According to the algorithm, the OS schedules the process which is having the lowest burst time among the available processes in the ready queue.

Till now, we have only one process in the ready queue hence the scheduler will schedule this to the processor no matter what is its burst time.

This will be executed till 8 units of time. Till then we have three more processes arrived in the ready queue hence the scheduler will choose the process with the lowest burst time.

Among the processes given in the table, P3 will be executed next since it is having the lowest burst time among all the available processes.

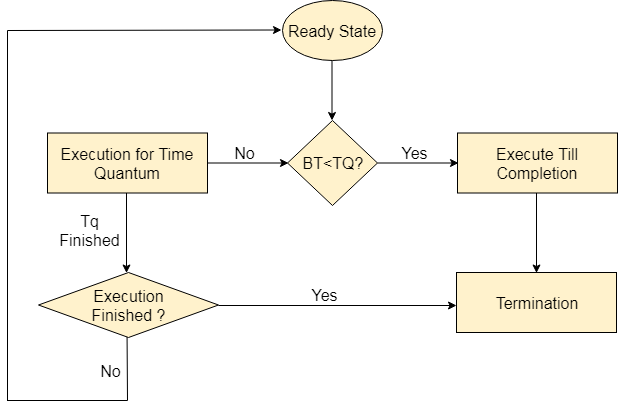
So that's how the procedure will go on in **shortest job first (SJF)** scheduling algorithm.



**Avg Waiting Time = 27/5**

**Round Robin Scheduling Algorithm**

Round Robin scheduling algorithm is one of the most popular scheduling algorithm which can actually be implemented in most of the operating systems. This is the **preemptive version** of first come first serve scheduling. The Algorithm focuses on Time Sharing. In this algorithm, every process gets executed in a **cyclic way**. A certain time slice is defined in the system which is called time **quantum**. Each process present in the ready queue is assigned the CPU for that time quantum, if the execution of the process is completed during that time then the process will **terminate** else the process will go back to the **ready queue** and waits for the next turn to complete the execution.

 Fig 3.14 Process of RR

## Advantages

1. It can be actually implementable in the system because it is not depending on the burst time.
2. It doesn't suffer from the problem of starvation or convoy effect.
3. All the jobs get a fare allocation of CPU.

## Disadvantages

1. The higher the time quantum, the higher the response time in the system.
2. The lower the time quantum, the higher the context switching overhead in the system.
3. Deciding a perfect time quantum is really a very difficult task in the system.

# Shortest Remaining Time First (SRTF) Scheduling Algorithm

This Algorithm is the **preemptive version** of **SJF scheduling**. In SRTF, the execution of the process can be stopped after certain amount of time. At the arrival of every process, the short term scheduler schedules the process with the least remaining burst time among the list of available processes and the running process.

Once all the processes are available in the **ready queue**, No preemption will be done and the algorithm will work as **SJF scheduling**. The context of the process is saved in the **Process Control Block** when the process is removed from the execution and the next process is scheduled. This PCB is accessed on the **next execution** of this process.

### Example

In this Example, there are five jobs P1, P2, P3, P4, P5 and P6. Their arrival time and burst time are given below in the table.

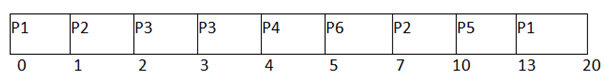
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Process ID** | **Arrival Time** | **Burst Time** | **Completion Time** | **Turn Around Time** | **Waiting Time** | **Response Time** |
| 1 | 0 | 8 | 20 | 20 | 12 | 0 |
| 2 | 1 | 4 | 10 | 9 | 5 | 1 |
| 3 | 2 | 2 | 4 | 2 | 0 | 2 |
| 4 | 3 | 1 | 5 | 2 | 1 | 4 |
| 5 | 4 | 3 | 13 | 9 | 6 | 10 |
| 6 | 5 | 2 | 7 | 2 | 0 | 5 |

Avg Waiting Time = 24/6

The Gantt chart is prepared according to the arrival and burst time given in the table.

1. Since, at time 0, the only available process is P1 with CPU burst time 8. This is the only available process in the list therefore it is scheduled.
2. The next process arrives at time unit 1. Since the algorithm we are using is SRTF which is a preemptive one, the current execution is stopped and the scheduler checks for the process with the least burst time.
3. Till now, there are two processes available in the ready queue. The OS has executed P1 for one unit of time till now; the remaining burst time of P1 is 7 units. The burst time of Process P2 is 4 units. Hence Process P2 is scheduled on the CPU according to the algorithm.
4. The next process P3 arrives at time unit 2. At this time, the execution of process P3 is stopped and the process with the least remaining burst time is searched. Since the process P3 has 2 unit of burst time hence it will be given priority over others.
5. The Next Process P4 arrives at time unit 3. At this arrival, the scheduler will stop the execution of P4 and check which process is having least burst time among the available processes (P1, P2, P3 and P4). P1 and P2 are having the remaining burst time 7 units and 3 units respectively.

P3 and P4 are having the remaining burst time 1 unit each. Since, both are equal hence the scheduling will be done according to their arrival time. P3 arrives earlier than P4 and therefore it will be scheduled again.



The Next Process P5 arrives at time unit 4. Till this time, the Process P3 has completed its execution and it is no more in the list. The scheduler will compare the remaining burst time of all the available processes. Since the burst time of process P4 is 1 which is least among all hence this will be scheduled.

1. The Next Process P6 arrives at time unit 5, till this time, the Process P4 has completed its execution. We have 4 available processes till now, that are P1 (7), P2 (3), P5 (3) and P6 (2). The Burst time of P6 is the least among all hence P6 is scheduled. Since, now, all the processes are available hence the algorithm will now work same as SJF. P6 will be executed till its completion and then the process with the least remaining time will be scheduled.

Once all the processes arrive, No preemption is done and the algorithm will work as SJF.

**Highest Response Ratio Next (HRRN) Scheduling**

Highest Response Ratio Next (HRNN) is one of the most optimal scheduling algorithms. This is a non-preemptive algorithm in which, the scheduling is done on the basis of an extra parameter called Response Ratio. A Response Ratio is calculated for each of the available jobs and the Job with the highest response ratio is given priority over the others.

**Response Ratio is calculated by the given formula.**

1. Response Ratio = (W+S)/S

**Where,**

1. W → Waiting Time
2. S → Service Time or Burst Time

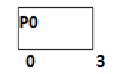
If we look at the formula, we will notice that the job with the shorter burst time will be given priority but it is also including an extra factor called waiting time. Since,

**Example**

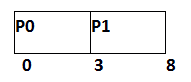
In the following example, there are 5 processes given. Their arrival time and Burst Time are given in the table.

|  |  |  |
| --- | --- | --- |
| **Process ID** | **Arrival Time** | **Burst Time** |
| 0 | 0 | 3 |
| 1 | 2 | 5 |
| 2 | 4 | 4 |
| 3 | 6 | 1 |
| 4 | 8 | 2 |

At time 0, The Process P0 arrives with the CPU burst time of 3 units. Since it is the only process arrived till now , hence this will get scheduled immediately.



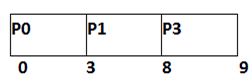
P0 is executed for 3 units, meanwhile, only one process P1 arrives at time 3. This will get scheduled immediately since the OS doesn't have a choice.



P1 is executed for 5 units. Meanwhile, all the processes get available. We have to calculate the Response Ratio for all the remaining jobs.

1. RR (P2) = ((8-4) +4)/4 = 2
2. RR (P3) = (2+1)/1 = 3
3. RR (P4) = (0+2)/2 = 1

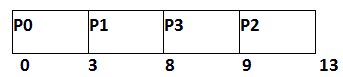
**Since, the Response ratio of P3 is higher hence P3 will be scheduled first.**



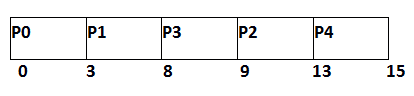
P3 is scheduled for 1 unit. The next available processes are P2 and P4. Let's calculate their Response ratio.

1. RR ( P2) = (5+4)/4 = 2.25
2. RR (P4) = (1+2)/2 = 1.5

The response ratio of P2 is higher hence P2 will be scheduled.



Now, the only available process is P4 with the burst time of 2 units, since there is no other process available hence this will be scheduled.



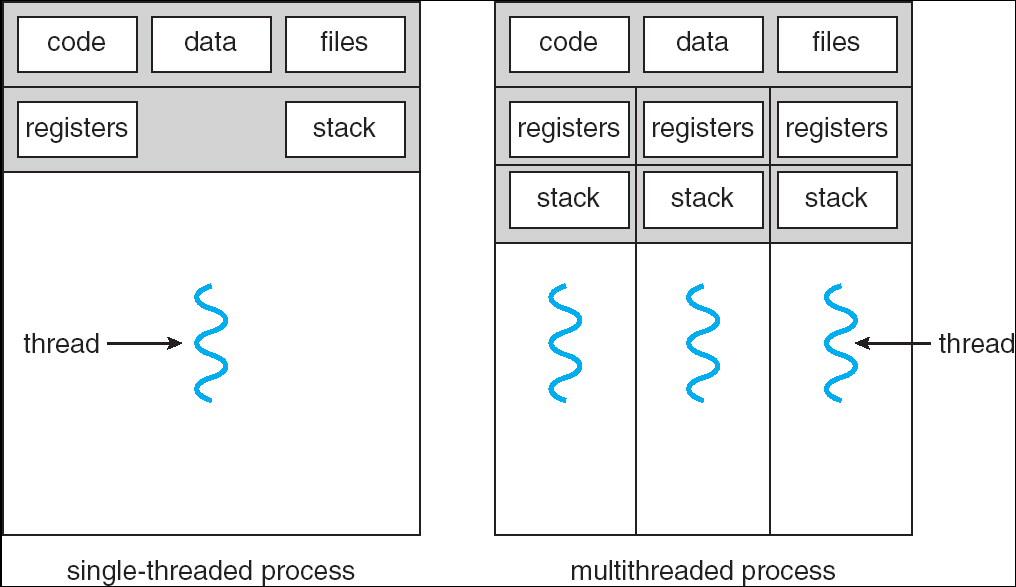
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Process ID** | **Arrival Time** | **Burst Time** | **Completion Time** | **Turn Around Time** | **Waiting Time** |
| 0 | 0 | 3 | 3 | 3 | 0 |
| 1 | 2 | 5 | 8 | 6 | 1 |
| 2 | 4 | 4 | 13 | 9 | 5 |
| 3 | 6 | 1 | 9 | 3 | 2 |
| 4 | 8 | 2 | 15 | 7 | 5 |

**Average Waiting Time = 13/5**

# Introduction of Thread

* A thread is a flow of execution through the process code, with its own program counter, system registers and stack. Threads are a popular way to improve application performance through parallelism. A thread is sometimes called a light weight process.
* Threads represent a software approach to improving performance of operating system by reducing the over head thread is equivalent to a classical process. Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control.
* Threads have been successfully used in implementing network servers. They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors.

**Fig.3.15 Shows the single and multithreaded process.**

****

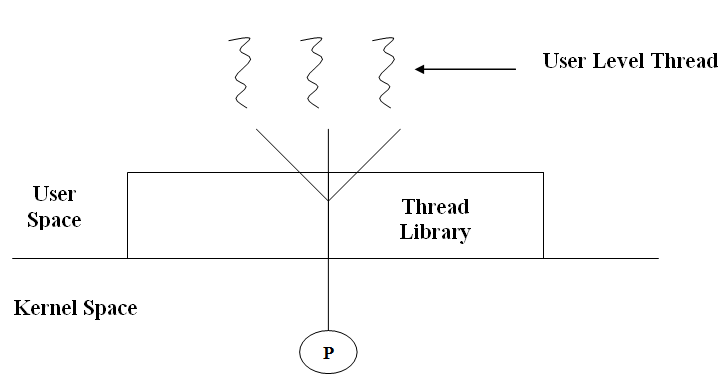
# Types of Thread

Threads is implemented in two ways:

1. UserLevel
2. KernelLevel
   * 1. **User Level Thread**

In a user thread, all of the work of thread management is done by the application and the kernel is not aware of the existence of threads. The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts. The application begins with a single thread and begins running in that thread.

Fig. 4.2 shows the user level thread.



User level threads are generally fast to create and manage.

### Advantage of user level thread over Kernel level thread:

* + - 1. Thread switching does not require Kernel mode privileges.
      2. User level thread can run on any operating system.
      3. Scheduling can be application specific.
      4. User level threads are fast to create and manage.

### Disadvantages of user level thread:

* + - 1. In a typical operating system, most system calls are blocking.
      2. Multithreaded application cannot take advantage of multiprocessing.
    1. **Kernel Level Threads**
* In Kernel level thread, thread management done by the Kernel. There is no thread management code in the application area. Kernel threads are supported directly by the operating system.
* Any application can be programmed to be multithreaded. All of the threads with in an application are supported within a single process. The Kernel maintains context information for the process as a whole and for individuals threads within the process.
* Scheduling by the Kernel is done on a thread basis. The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.

### Advantages of Kernel level thread:

* + - 1. Kernel can simultaneously schedule multiple threads from the same process on multiple process.
      2. If one thread in a process is blocked, the Kernel can schedule another thread of the same process.
      3. Kernelroutinesthemselvescanmultithreaded.

### Disadvantages:

1. Kernel threads are generally slower to create and manage than the user threads.
2. Transfer of control from one thread to another within same process requires a mode switch to the Kernel.

## Advantages of Thread

1. Thread minimize context switching time.
2. Use of threads provides concurrency with in a process.
3. Efficient communication.
4. Economy-It is more economical to create and context switch threads.
5. Utilization of multiprocessor architectures– The benefits of multithreading can be greatly increased in a multiprocessor architecture.
   1. **Difference between User Level & Kernel Level Thread**

|  |  |  |
| --- | --- | --- |
| Sr. No | User Level Threads | Kernel Level Thread |
| 1 | User level thread are faster to create  And manage. | Kernel level thread are slower to  Create and manage. |
| 2 | Implemented by a thread library at  The user level. | Operating system support directly to  Kernel threads. |
| 3 | User level thread can run on any  Operating system. | Kernel level threads are specific to  The operating system. |
| 4 | Support provided at the user level  Called user level thread. | Support may be provided by kernel  Is called Kernel level threads. |
| 5 | Multithread application cannot take  Advantage of multiprocessing. | Kernel routines themselves can be  Multi threaded. |

* 1. **Difference between Process and Thread**

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Process** | **Thread** |
| 1 | Process is called heavy weight process. | Thread is called light weight process. |
| 2 | Process switching needs interface with operating system. | Thread switching does not need to call a operating system and cause an interrupt to the Kernel. |
| 3 | In multiple process implementation each process executes the same code but has its  Own memory and file resources. | All threads can share same set of open files, child processes. |
| 4 | If one server process is blocked no other server process can execute until the first process unblocked. | While one server thread is blocked and waiting, second thread in the same task could run. |
| 5 | Multiple redundant process uses more resources than multiple threaded. | Multiple threaded process uses fewer resources than multiple redundant process. |
| 6 | In multiple processes each process operates independently of the others. | One thread can read, write or even completely wipe out another threads stack. |

**Chapter 4**

**Memory Management:**

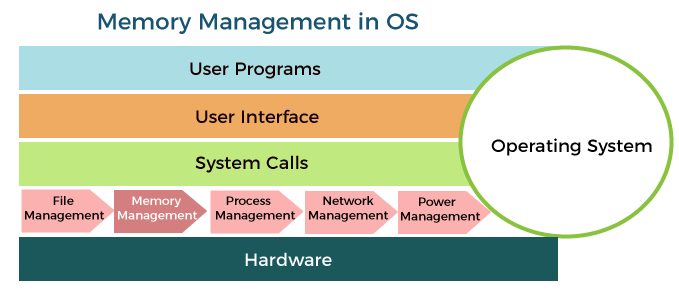
1. Memory management without swapping or paging,
2. Concepts of swapping and paging,
3. Page replacement algorithms

* Least recently used
* Optimal page replacement
* Most recently used
* Clock page replacement
* First in First out (This includes discussion of Belady’s anomaly and the category of Stack algorithms), Modeling paging algorithms,

1. Design issues for paging system,
2. Segmentation,
3. Segmented Paging,
4. Paged Segmentation.

**Memory Management**

Memory is the important part of the computer that is used to store the data. Its management is critical to the computer system because the amount of main memory available in a computer system is very limited. At any time, many processes are competing for it. Moreover, to increase performance, several processes are executed simultaneously. For this, we must keep several processes in the main memory, so it is even more important to manage them effectively.

****

### Memory management plays several roles in a Computer System

Memory manager is used to keep track of the status of memory locations, whether it is free or allocated. It addresses primary memory by providing abstractions so that software perceives a large memory is allocated to it.

Memory manager permits computers with a small amount of main memory to execute programs larger than the size or amount of available memory. It does this by moving information back and forth between primary memory and secondary memory by using the concept of swapping.

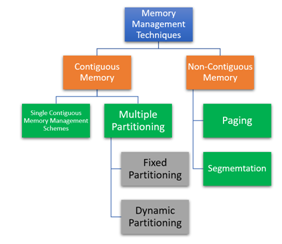
The memory manager is responsible for protecting the memory allocated to each process from being corrupted by another process. If this is not ensured, then the system may exhibit unpredictable behavior.

Memory managers should enable sharing of memory space between processes. Thus, two programs can reside at the same memory location although at different times.

**Memory management Techniques:**

The Memory management Techniques can be classified into following main categories:

* Contiguous memory management schemes
* Non-Contiguous memory management schemes

****

**Classification of memory management schemes**

### Contiguous memory management schemes:

In a Contiguous memory management scheme, each program occupies a single contiguous block of storage locations, i.e., a set of memory locations with consecutive addresses.

### Single contiguous memory management schemes:

The Single contiguous memory management scheme is the simplest memory management scheme used in the earliest generation of computer systems. In this scheme, the main memory is divided into two contiguous areas or partitions. The operating systems reside permanently in one partition, generally at the lower memory, and the user process is loaded into the other partition.

**Advantages of Single contiguous memory management schemes:**

* Simple to implement.
* Easy to manage and design.
* In a Single contiguous memory management scheme, once a process is loaded, it is given full processor's time, and no other processor will interrupt it.

**Disadvantages of Single contiguous memory management schemes:**

* Wastage of memory space due to unused memory as the process is unlikely to use all the available memory space.
* The CPU remains idle, waiting for the disk to load the binary image into the main memory.
* It cannot be executed if the program is too large to fit the entire available main memory space.
* It does not support multiprogramming, i.e., it cannot handle multiple programs simultaneously.

### Multiple Partitioning

The single Contiguous memory management scheme is inefficient as it limits computers to execute only one program at a time resulting in wastage in memory space and CPU time. The problem of inefficient CPU use can be overcome using multiprogramming that allows more than one program to run concurrently. To switch between two processes, the operating systems need to load both processes into the main memory. The operating system needs to divide the available main memory into multiple parts to load multiple processes into the main memory. Thus multiple processes can reside in the main memory simultaneously.

**The multiple partitioning schemes can be of two types**

* Fixed Partitioning
* Dynamic Partitioning

### Fixed Partitioning

The main memory is divided into several fixed-sized partitions in a fixed partition memory management scheme or static partitioning. These partitions can be of the same size or different sizes. Each partition can hold a single process. The number of partitions determines the degree of multiprogramming, i.e., the maximum number of processes in memory. These partitions are made at the time of system generation and remain fixed after that.

**Advantages of Fixed Partitioning Memory Management Schemes**

* Simple to implement.
* Easy to manage and design.

**Disadvantages of Fixed partitioning memory management schemes**

* This scheme suffers from internal fragmentation.
* The number of partitions is specified at the time of system generation.

### Dynamic Partitioning

The dynamic partitioning was designed to overcome the problems of a fixed partitioning scheme. In a dynamic partitioning scheme, each process occupies only as much memory as they require when loaded for processing. Requested processes are allocated memory until the entire physical memory is exhausted or the remaining space is insufficient to hold the requesting process. In this scheme the partitions used are of variable size, and the number of partitions is not defined at the system generation time.

**Advantages of Dynamic Partitioning memory management schemes:**

* Simple to implement.
* Easy to manage and design.

**Disadvantages of Dynamic Partitioning memory management schemes:**

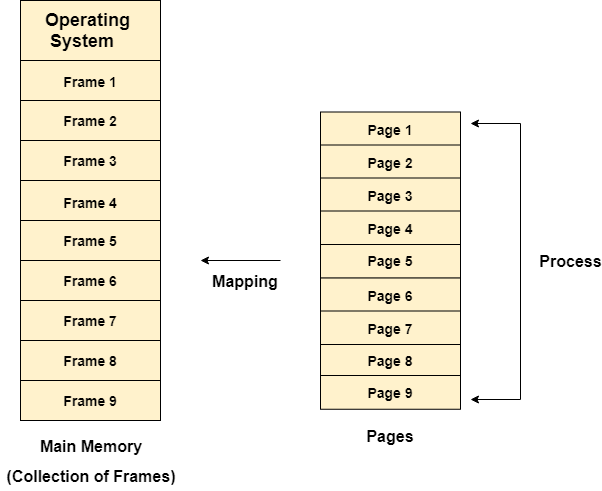
* This scheme also suffers from internal fragmentation.
* The number of partitions is specified at the time of system segmentation.

### Non-Contiguous memory management schemes:

In a Non-Contiguous memory management scheme, the program is divided into different blocks and loaded at different portions of the memory that need not necessarily be adjacent to one another. This scheme can be classified depending upon the size of blocks and whether the blocks reside in the main memory or not.

**Paging**

* In Operating Systems, Paging is a storage mechanism used to retrieve processes from the secondary storage into the main memory in the form of pages.
* The main idea behind the paging is to divide each process in the form of pages. The main memory will also be divided in the form of frames.
* One page of the process is to be stored in one of the frames of the memory. The pages can be stored at the different locations of the memory but the priority is always to find the contiguous frames or holes.
* Pages of the process are brought into the main memory only when they are required otherwise they reside in the secondary storage.



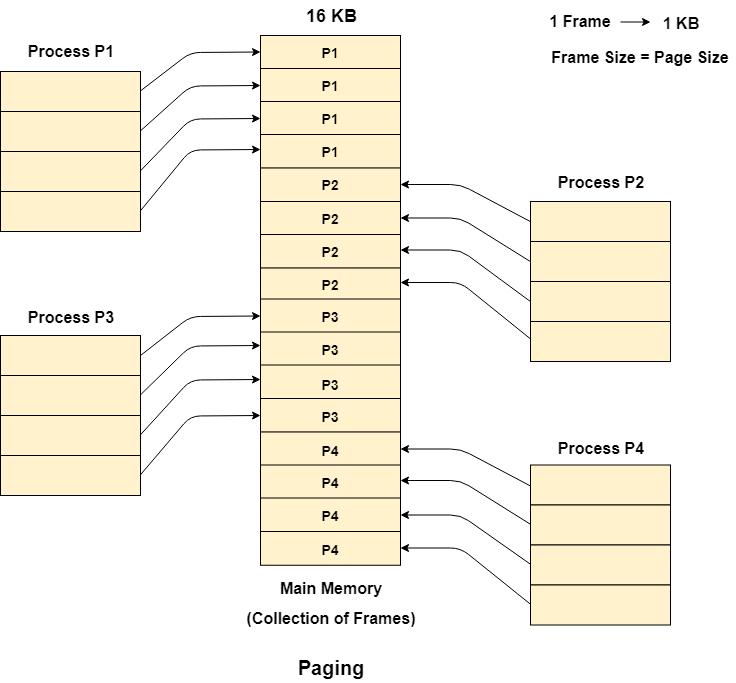
### Example

Let us consider the main memory size 16 Kb and Frame size is 1 KB therefore the main memory will be divided into the collection of 16 frames of 1 KB each.

There are 4 processes in the system that is P1, P2, P3 and P4 of 4 KB each. Each process is divided into pages of 1 KB each so that one page can be stored in one frame.

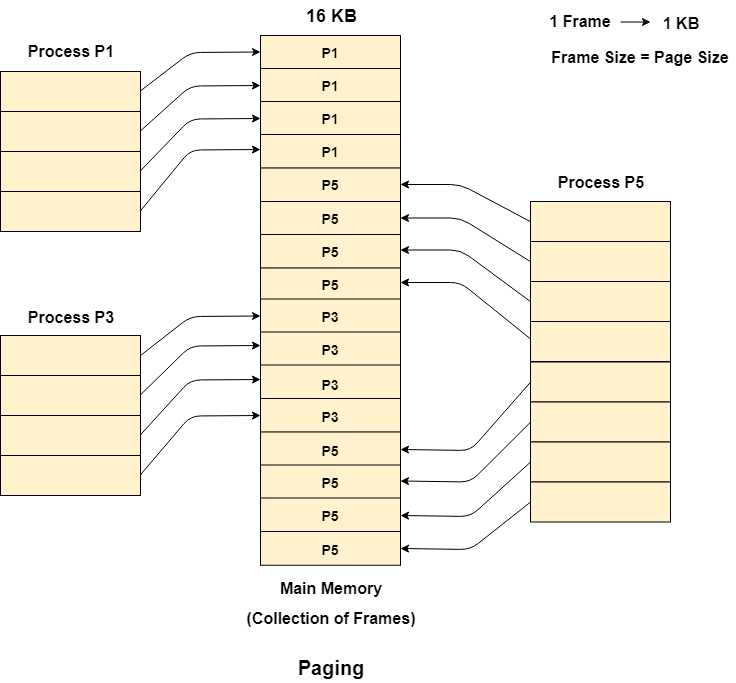
Initially, all the frames are empty therefore pages of the processes will get stored in the contiguous way.

Frames, pages and the mapping between the two is shown in the image below.

Let us consider that, P2 and P4 are moved to waiting state after some time. Now, 8 frames become empty and therefore other pages can be loaded in that empty place. The process P5 of size 8 KB (8 pages) is waiting inside the ready queue.

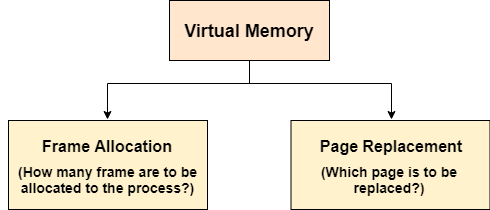
Given the fact that, we have 8 non contiguous frames available in the memory and paging provides the flexibility of storing the process at the different places Therefore, we can load the pages of process P5 in the place of P2 and P4.

.



**Page Replacement Algorithms**

The page replacement algorithm decides which memory page is to be replaced. The process of replacement is sometimes called swap out or write to disk. Page replacement is done when the requested page is not found in the main memory (page fault).



There are two main aspects of virtual memory, Frame allocation and Page Replacement. It is very important to have the optimal frame allocation and page replacement algorithm. Frame allocation is all about how many frames are to be allocated to the process while the page replacement is all about determining the page number which needs to be replaced in order to make space for the requested page.

### What If the algorithm is not optimal?

1. if the number of frames which are allocated to a process is not sufficient or accurate then there can be a problem of thrashing. Due to the lack of frames, most of the pages will be residing in the main memory and therefore more page faults will occur. However, if OS allocates more frames to the process then there can be internal fragmentation.

2. If the page replacement algorithm is not optimal then there will also be the problem of thrashing. If the number of pages that are replaced by the requested pages will be referred in the near future then there will be more number of swap-in and swap-out and therefore the OS has to perform more replacements then usual which cause performance deficiency.

Therefore, the task of an optimal page replacement algorithm is to choose the page which can limit the thrashing.

### Types of Page Replacement Algorithms

There are various page replacement algorithms. Each algorithm has a different method by which the pages can be replaced.

1. **Optimal Page Replacement algorithm →** this algorithms replaces the page which will not be referred for so long in future. Although it can not be practically implementable but it can be used as a benchmark. Other algorithms are compared to this in terms of optimality.
2. **Least recent used (LRU) page replacement algorithm →** this algorithm replaces the page which has not been referred for a long time. This algorithm is just opposite to the optimal page replacement algorithm. In this, we look at the past instead of staring at future.
3. **FIFO →** in this algorithm, a queue is maintained. The page which is assigned the frame first will be replaced first. In other words, the page which resides at the rare end of the queue will be replaced on the every page fault.

**Belady's Anomaly**

In the case of LRU and optimal page replacement algorithms, it is seen that the number of page faults will be reduced if we increase the number of frames. However, Balady found that, In FIFO page replacement algorithm, the number of page faults will get increased with the increment in number of frames.

This is the strange behavior shown by FIFO algorithm in some of the cases. This is an Anomaly called as Belady'sAnomaly.

Let's examine such example:

The reference String is given as 0 1 5 3 0 1 4 0 1 5 3 4. Let's analyze the behavior of FIFO algorithm in two cases.

### Case 1: Number of frames = 3

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Request | 0 | 1 | 5 | 3 | 0 | 1 | 4 | 0 | 1 | 5 | 3 | 4 |
| Frame 3 |  |  | 5 | 5 | 5 | 1 | 1 | 1 | 1 | 1 | 3 | 3 |
| Frame 2 |  | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 5 | 5 | 5 |
| Frame 1 | 0 | 0 | 0 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 4 | 4 |
| Miss/Hit | Miss | Miss | Miss | Miss | Miss | Miss | Miss | Hit | Hit | Miss | Miss | Hit |

**Number of Page Faults = 9**

### Case 2: Number of frames = 4

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Request | 0 | 1 | 5 | 3 | 0 | 1 | 4 | 0 | 1 | 5 | 3 | 4 |
| Frame 4 |  |  |  | 3 | 3 | 3 | 3 | 3 | 3 | 5 | 5 | 5 |
| Frame 3 |  |  | 5 | 5 | 5 | 5 | 5 | 5 | 1 | 1 | 1 | 1 |
| Frame 2 |  | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 4 |
| Frame 1 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 4 | 4 | 4 | 3 | 3 |
| Miss/Hit | Miss | Miss | Miss | Miss | Hit | Hit | Miss | Miss | Miss | Miss | Miss | Miss |

Number of Page Faults = 10

Therefore, in this example, the number of page faults is increasing by increasing the number of frames hence this suffers from Belady's Anomaly.

### Segmentation

In Operating Systems, Segmentation is a memory management technique in which the memory is divided into the variable size parts. Each part is known as a segment which can be allocated to a process.

The details about each segment are stored in a table called a segment table. Segment table is stored in one (or many) of the segments.

Segment table contains mainly two type of information about segment:

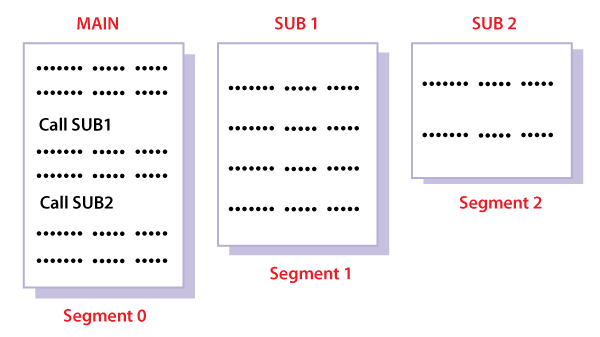
1. Base: It is the base address of the segment
2. Limit: It is the length of the segment.

## Why Segmentation is required?

Till now, we were using Paging as our main memory management technique. Paging is more close to the Operating system rather than the User. It divides all the processes into the form of pages regardless of the fact that a process can have some relative parts of functions which need to be loaded in the same page.

Operating system doesn't care about the User's view of the process. It may divide the same function into different pages and those pages may or may not be loaded at the same time into the memory. It decreases the efficiency of the system.

It is better to have segmentation which divides the process into the segments. Each segment contains the same type of functions such as the main function can be included in one segment and the library functions can be included in the other segment.



## Translation of Logical address into physical address by segment table

CPU generates a logical address which contains two parts:

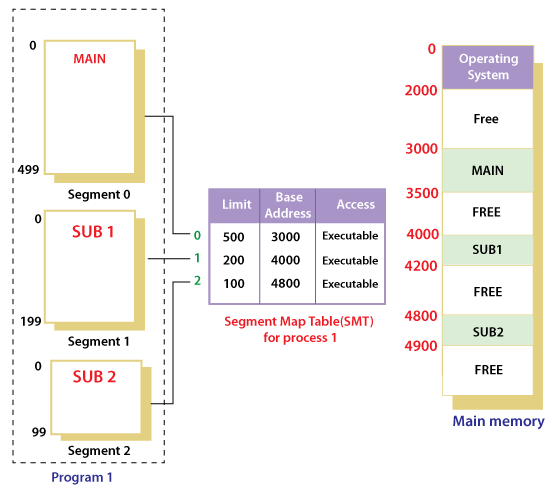
1. Segment Number
2. Offset

**For Example:**

Suppose a 16 bit address is used with 4 bits for the segment number and 12 bits for the segment offset so the maximum segment size is 4096 and the maximum number of segments that can be refereed is 16.

When a program is loaded into memory, the segmentation system tries to locate space that is large enough to hold the first segment of the process, space information is obtained from the free list maintained by memory manager. Then it tries to locate space for other segments. Once adequate space is located for all the segments, it loads them into their respective areas.

The operating system also generates a segment map table for each program.



With the help of segment map tables and hardware assistance, the operating system can easily translate a logical address into physical address on execution of a program.

The **Segment number** is mapped to the segment table. The limit of the respective segment is compared with the offset. If the offset is less than the limit then the address is valid otherwise it throws an error as the address is invalid.

In the case of valid addresses, the base address of the segment is added to the offset to get the physical address of the actual word in the main memory.

The above figure shows how address translation is done in case of segmentation.

## Advantages of Segmentation

1. No internal fragmentation
2. Average Segment Size is larger than the actual page size.
3. Less overhead
4. It is easier to relocate segments than entire address space.
5. The segment table is of lesser size as compared to the page table in paging.

## Disadvantages

1. It can have external fragmentation.
2. it is difficult to allocate contiguous memory to variable sized partition.
3. Costly memory management algorithms.

**Paging VS Segmentation**

|  |  |  |
| --- | --- | --- |
| **Sr No.** | **Paging** | **Segmentation** |
| 1 | Non-Contiguous memory allocation | Non-contiguous memory allocation |
| 2 | Paging divides program into fixed size pages. | Segmentation divides program into variable size segments. |
| 3 | OS is responsible | Compiler is responsible. |
| 4 | Paging is faster than segmentation | Segmentation is slower than paging |
| 5 | Paging is closer to Operating System | Segmentation is closer to User |
| 6 | It suffers from internal fragmentation | It suffers from external fragmentation |
| 7 | There is no external fragmentation | There is no external fragmentation |
| 8 | Logical address is divided into page number and page offset | Logical address is divided into segment number and segment offset |
| 9 | Page table is used to maintain the page information. | Segment Table maintains the segment information |
| 10 | Page table entry has the frame number and some flag bits to represent details about pages. | Segment table entry has the base address of the segment and some protection bits for the segments. |

**Segmented Paging**

Pure segmentation is not very popular and not being used in many of the operating systems. However, Segmentation can be combined with Paging to get the best features out of both the techniques.

In Segmented Paging, the main memory is divided into variable size segments which are further divided into fixed size pages.

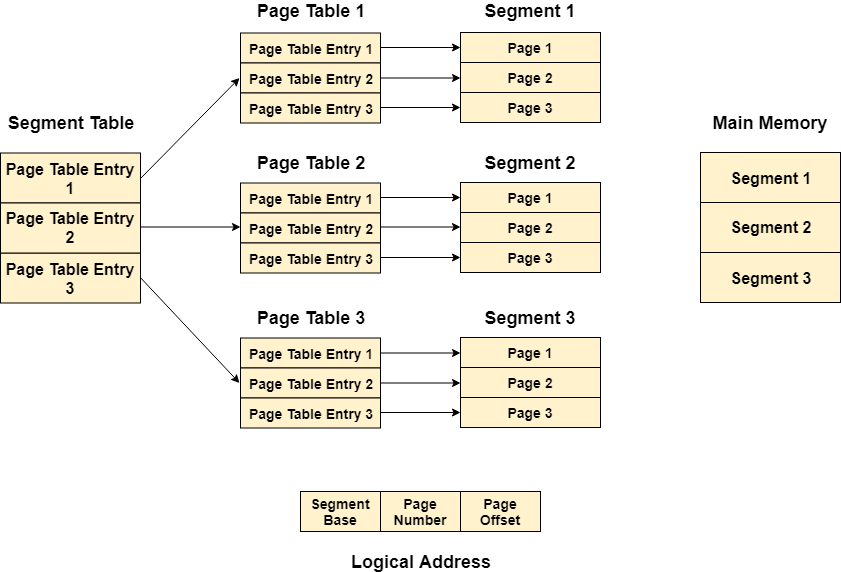
1. Pages are smaller than segments.
2. Each Segment has a page table which means every program has multiple page tables.
3. The logical address is represented as Segment Number (base address), Page number and page offset.

**Segment Number →** It points to the appropriate Segment Number.

**Page Number →** It Points to the exact page within the segment

**Page Offset →** Used as an offset within the page frame

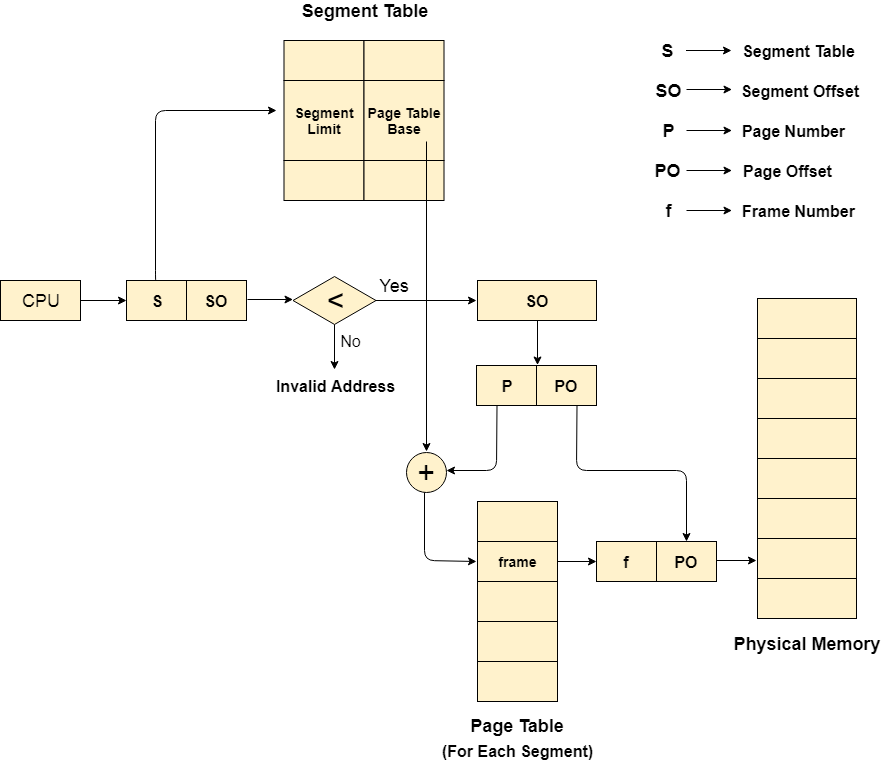
Each Page table contains the various information about every page of the segment. The Segment Table contains the information about every segment. Each segment table entry points to a page table entry and every page table entry is mapped to one of the page within a segment.



## Translation of logical address to physical address

The CPU generates a logical address which is divided into two parts: Segment Number and Segment Offset. The Segment Offset must be less than the segment limit. Offset is further divided into Page number and Page Offset. To map the exact page number in the page table, the page number is added into the page table base.

The actual frame number with the page offset is mapped to the main memory to get the desired word in the page of the certain segment of the process.



## Advantages of Segmented Paging

1. It reduces memory usage.
2. Page table size is limited by the segment size.
3. Segment table has only one entry corresponding to one actual segment.
4. External Fragmentation is not there.
5. It simplifies memory allocation.

## Disadvantages of Segmented Paging

1. Internal Fragmentation will be there.
2. The complexity level will be much higher as compare to paging.
3. Page Tables need to be contiguously stored in the memory.

**Chapter 5**

**Inter-process Communication and Synchronization:**

1. The need for inter-process synchronization,
2. Concept of mutual exclusion,
3. Binary and counting semaphores,
4. Hardware support for mutual exclusion, queuing implementation of semaphores,
5. Classical problems in concurrent programming,

* Dining Philosopher’s problem,
* Bounded Buffer Problem,
* Sleeping Barber Problem,
* Readers and Writers problem,

1. Critical section,

* Critical region and conditional critical region,
* Monitors and messages

1. Deadlocks:

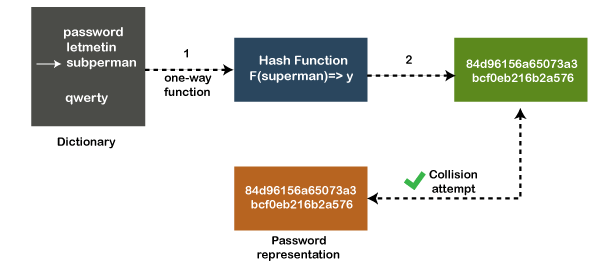
* Concepts of deadlock detection,
* deadlock prevention,
* Deadlock avoidance.
* Banker’s Algorithm.

**5.1 Inter Process Communication**

Inter Process Communication is a type of mechanism usually provided by the operating system (or OS). The main aim or goal of this mechanism is to provide communications in between several processes. In short, the intercommunication allows a process letting another process know that some event has occurred.

"Inter-process communication is used for exchanging useful information between numerous threads in one or more processes (or programs)."

To understand inter process communication, you can consider the following given diagram that illustrates the importance of inter-process communication:

****

**Fig. 5.1 Interprocess communication**

### 5.1.2 Role of Synchronization in Inter Process Communication

It is one of the essential parts of inter process communication. Typically, this is provided by interprocess communication control mechanisms, but sometimes it can also be controlled by communication processes.

These are the following methods that used to provide the synchronization:

1. **Mutual Exclusion**
2. **Semaphore**
3. **Barrier**
4. **Spinlock**

**Mutual Exclusion:-**

It is generally required that only one process thread can enter the critical section at a time. This also helps in synchronization and creates a stable state to avoid the race condition.

**Semaphore:-**

Semaphore is a type of variable that usually controls the access to the shared resources by several processes. Semaphore is further divided into two types which are as follows:

1. Binary Semaphore
2. Counting Semaphore

**Barrier:-**

A barrier typically not allows an individual process to proceed unless all the processes does not reach it. It is used by many parallel languages, and collective routines impose barriers.

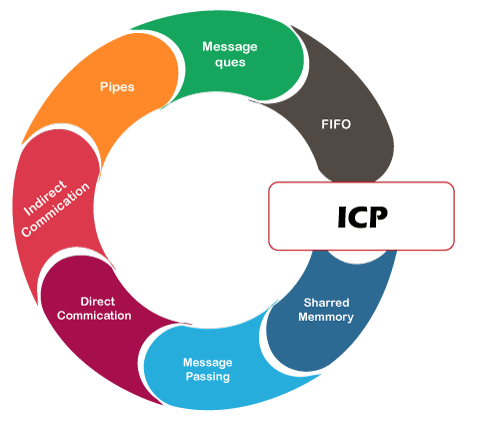
**Spinlock:-**

Spinlock is a type of lock as its name implies. The processes are trying to acquire the spinlock waits or stays in a loop while checking that the lock is available or not. It is known as busy waiting because even though the process active, the process does not perform any functional operation (or task).

### 5.1.3 Approaches to Inter-process Communication

These are a few different approaches for Inter- Process Communication:

* Pipes
* Shared Memory
* Message Queue
* Direct Communication
* Indirect communication
* Message Passing
* FIFO



**Fig. 5.2 ICP Approaches**

**Pipe**

The pipe is a type of data channel that is unidirectional in nature. It means that the data in this type of data channel can be moved in only a single direction at a time. Still, one can use two-channel of this type, so that he can able to send and receive data in two processes. Typically, it uses the standard methods for input and output. These pipes are used in all types of POSIX systems and in different versions of window operating systems as well.

**Shared Memory:-**

It can be referred to as a type of memory that can be used or accessed by multiple processes simultaneously. It is primarily used so that the processes can communicate with each other. Therefore the shared memory is used by almost all POSIX and Windows operating systems as well.

**Message Queue:-**

**In general, several different messages are allowed to read and write the data to the message queue.**

**In the message queue, the messages are stored or stay in the queue unless their recipients retrieve them.**

**In short,** we can also say that the message queue is very helpful in inter-process communication and used by all operating systems.

**Message Passing:-**

It is a type of mechanism that allows processes to synchronize and communicate with each other. However, by using the message passing, the processes can communicate with each other without restoring the hared variables.

Usually, the inter-process communication mechanism provides two operations that are as follows:

* send (message)
* received (message)

**Direct Communication:-**

In this type of communication process, usually, a link is created or established between two communicating processes. However, in every pair of communicating processes, only one link can exist.

**Indirect Communication**

Indirect communication can only exist or be established when processes share a common mailbox, and each pair of these processes shares multiple communication links. These shared links can be unidirectional or bi-directional.

**FIFO:-**

It is a type of general communication between two unrelated processes. It can also be considered as full-duplex, which means that one process can communicate with another process and vice versa.

### 5.1.4 Need of inter-process communication

There are numerous reasons to use inter-process communication for sharing the data. Here are some of the most important reasons that are given below:

* It helps to speedup modularity
* Computational
* Privilege separation
* Convenience
* Helps operating system to communicate with each other and synchronize their actions as well.

**5.2 The Critical Section Problem**

Critical Section is the part of a program which tries to access shared resources. That resource may be any resource in a computer like a memory location, Data structure, CPU or any IO device.

The critical section cannot be executed by more than one process at the same time; operating system faces the difficulties in allowing and disallowing the processes from entering the critical section.

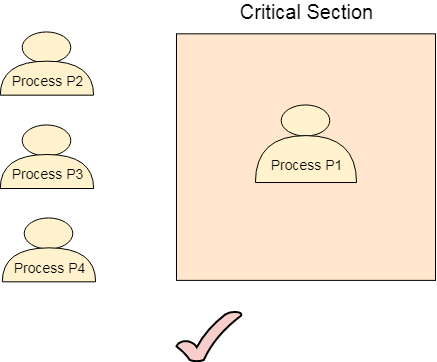
The critical section problem is used to design a set of protocols which can ensure that the Race condition among the processes will never arise.

In order to synchronize the cooperative processes, our main task is to solve the critical section problem. We need to provide a solution in such a way that the following conditions can be satisfied.

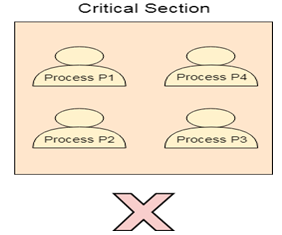
## 5.2.1 Requirements of Synchronization mechanisms

**Primary**

1. **Mutual Exclusion**

Our solution must provide mutual exclusion. By Mutual Exclusion, we mean that if one process is executing inside critical section then the other process must not enter in the critical section.

**Fig. 5.3 Mutual Exclusion**

****

**Fig. 5.4 Critical Section**

1. **Progress**

Progress means that if one process doesn't need to execute into critical section then it should not stop other processes to get into the critical section.

### Secondary

1. **Bounded Waiting**

We should be able to predict the waiting time for every process to get into the critical section. The process must not be endlessly waiting for getting into the critical section.

1. **Architectural Neutrality**

Our mechanism must be architectural natural. It means that if our solution is working fine on one architecture, then it should also run on the other ones as well.

**5.3 Introduction to Semaphore**

To get rid of the problem of wasting the wake-up signals, Dijkstra proposed an approach which involves storing all the wake-up calls. Dijkstra states that, instead of giving the wake-up calls directly to the consumer, producer can store the wake-up call in a variable. Any of the consumers can read it whenever it needs to do so.

Semaphore is the variables which stores the entire wake up calls that are being transferred from producer to consumer. It is a variable on which read, modify and update happens automatically in kernel mode.

**NOTE**: Semaphore cannot be implemented in the user mode because race condition may always arise when two or more processes try to access the variable simultaneously. It always needs support from the operating system to be implemented.

**According to the demand of the situation, Semaphore can be divided into two categories.**

1. Counting Semaphore
2. Binary Semaphore or Mutex

**5.31 Counting Semaphore**

There are the scenarios in which more than one processes need to execute in critical section simultaneously. However, counting semaphore can be used when we need to have more than one process in the critical section at the same time.

**Problem on Counting Semaphore**

### A Counting Semaphore was initialized to 12. Then 10P (wait) and 4V (Signal) operations were computed on this semaphore. What is the result?

**S = 12 (initial)**

**10 p (wait) :**

**SS = S -10 = 12 - 10 = 2**

**then 4 V :**

**SS = S + 4 =2 + 4 = 6**

**Hence, the final value of counting semaphore is 6.**

**5.3.2 Binary Semaphore or Mutex**

In counting semaphore, Mutual exclusion was not provided because we has the set of processes which required to execute in the critical section simultaneously.

However, Binary Semaphore strictly provides mutual exclusion. Here, instead of having more than 1 slots available in the critical section, we can only have at most 1 process in the critical section. The semaphore can have only two values, 0 or 1.

**5.4 Classical Problems of Synchronization**

The following three problems:

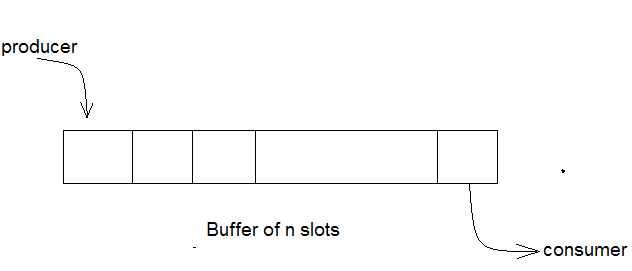
1. Bounded Buffer (Producer-Consumer) Problem
2. Dining Philosophers Problem
3. The Readers Writers Problem

**5.4.1 Bounded Buffer Problem**

Bounded buffer problem, which is also called producer consumer problem, is one of the classic problems of synchronization. Let's start by understanding the problem here, before moving on to the solution and program code.

**Problem Statement:**

There is a buffer of n slots and each slot is capable of storing one unit of data. There are two processes running, namely, producer and consumer, which are operating on the buffer.



**Fig. 5.5 Bounded Buffers**

## Solution:

One solution of this problem is to use semaphores. The semaphores which will be used here are:

* m, a binary semaphore which is used to acquire and release the lock.
* empty, a counting semaphore whose initial value is the number of slots in the buffer, since, initially all slots are empty.
* full, a counting semaphore whose initial value is 0.

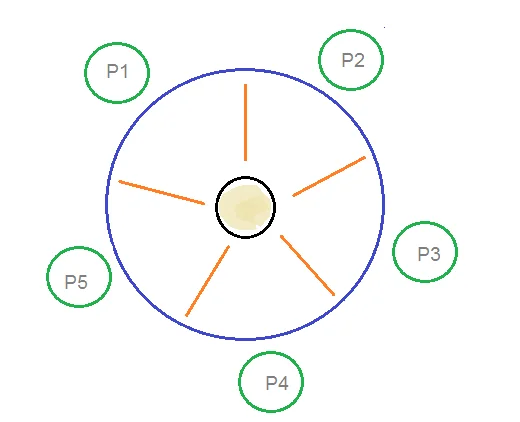
At any instant, the current value of empty represents the number of empty slots in the buffer and full represents the number of occupied slots in the buffer.

**5.4.2 Dining Philosophers Problem**

The dining philosophers’ problem is another classic synchronization problem which is used to evaluate situations where there is a need of allocating multiple resources to multiple processes.

### Problem Statement:

Consider there are five philosophers sitting around a circular dining table. The dining table has five chopsticks and a bowl of rice in the middle as shown in the below figure.

****

**Fig. 5.6** Dining philosophers’ problem

At any instant, a philosopher is either eating or thinking. When a philosopher wants to eat, he uses two chopsticks - one from their left and one from their right. When a philosopher wants to think, he keeps down both chopsticks at their original place.

## Solution:

From the problem statement, it is clear that a philosopher can think for an indefinite amount of time. But when a philosopher starts eating, he has to stop at some point of time. The philosopher is in an endless cycle of thinking and eating.

An array of five semaphores, stick[5], for each of the five chopsticks.

**5.4.3 Readers Writer Problem**

Reader’s writer’s problem is another example of a classic synchronization problem. There are many variants of this problem, one of which is examined below.

### Problem Statement:

There is a shared resource which should be accessed by multiple processes. There are two types of processes in this context. They are reader and writer. Any number of readers can read from the shared resource simultaneously, but only one writer can write to the shared resource. When a writer is writing data to the resource, no other process can access the resource. A writer cannot write to the resource if there are non zero number of readers accessing the resource at that time.

### 

### Solution:

From the above problem statement, it is evident that readers have higher priority than writer. If a writer wants to write to the resource, it must wait until there are no readers currently accessing that resource.

Here, we use one mutex m and a semaphore w. An integer variable read\_count is used to maintain the number of readers currently accessing the resource. The variable read\_count is initialized to 0. A value of 1 is given initially to m and w.

Instead of having the process to acquire lock on the shared resource, we use the mutex m to make the process to acquire and release lock whenever it is updating the read\_count variable.

**5.5 DEADLOCK**

Every process needs some resources to complete its execution. However, the resource is granted in a sequential order.

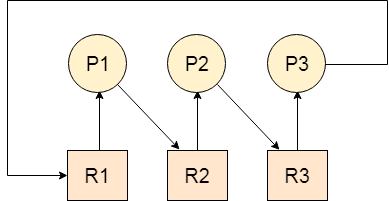
1. The process requests for some resource.
2. OS grant the resource if it is available otherwise let the process waits.
3. The process uses it and release on the completion.

A Deadlock is a situation where each of the computer process waits for a resource which is being assigned to some another process. In this situation, none of the process gets executed since the resource it needs, is held by some other process which is also waiting for some other resource to be released.

Let us assume that there are three processes P1, P2 and P3. There are three different resources R1, R2 and R3. R1 is assigned to P1, R2 is assigned to P2 and R3 is assigned to P3.

After some time, P1 demands for R1 which is being used by P2. P1 halts its execution since it can't complete without R2. P2 also demands for R3 which is being used by P3. P2 also stops its execution because it can't continue without R3. P3 also demands for R1 which is being used by P1 therefore P3 also stops its execution.

In this scenario, a cycle is being formed among the three processes. None of the process is progressing and they are all waiting. The computer becomes unresponsive since all the processes got blocked.



**Fig. 5.7.** Deadlock

### Difference between Starvation and Deadlock

|  |  |  |
| --- | --- | --- |
| **Sr.** | **Deadlock** | **Starvation** |
| 1 | Deadlock is a situation where no process got blocked and no process proceeds | Starvation is a situation where the low priority process got blocked and the high priority processes proceed. |
| 2 | Deadlock is an infinite waiting. | Starvation is a long waiting but not infinite. |
| 3 | Every Deadlock is always a starvation. | Every starvation need not be deadlock. |
| 4 | The requested resource is blocked by the other process. | The requested resource is continuously be used by the higher priority processes. |
| 5 | Deadlock happens when Mutual exclusion, hold and wait, No preemption and circular wait occurs simultaneously. | It occurs due to the uncontrolled priority and resource management. |

## 5.5.1 Necessary conditions for Deadlocks

1. **Mutual Exclusion**

A resource can only be shared in mutually exclusive manner. It implies, if two process cannot use the same resource at the same time.

1. **Hold and Wait**

A process waits for some resources while holding another resource at the same time.

1. **No preemption**

The process which once scheduled will be executed till the completion. No other process can be scheduled by the scheduler meanwhile.

1. **Circular Wait**

All the processes must be waiting for the resources in a cyclic manner so that the last process is waiting for the resource which is being held by the first process.

# 5.5.2 Strategies for handling Deadlock

## 1. Deadlock Ignorance

Deadlock Ignorance is the most widely used approach among all the mechanism. This is being used by many operating systems mainly for end user uses. In this approach, the Operating system assumes that deadlock never occurs. It simply ignores deadlock. This approach is best suitable for a single end user system where User uses the system only for browsing and all other normal stuff.

There is always a tradeoff between Correctness and performance. The operating systems like Windows and Linux mainly focus upon performance. However, the performance of the system decreases if it uses deadlock handling mechanism all the time if deadlock happens 1 out of 100 times then it is completely unnecessary to use the deadlock handling mechanism all the time.

In these types of systems, the user has to simply restart the computer in the case of deadlock. Windows and Linux are mainly using this approach.

## 2. Deadlock prevention

Deadlock happens only when Mutual Exclusion, hold and wait, No preemption and circular wait holds simultaneously. If it is possible to violate one of the four conditions at any time then the deadlock can never occur in the system.

## 3. Deadlock avoidance

In deadlock avoidance, the operating system checks whether the system is in safe state or in unsafe state at every step which the operating system performs. The process continues until the system is in safe state. Once the system moves to unsafe state, the OS has to backtrack one step.

In simple words, The OS reviews each allocation so that the allocation doesn't cause the deadlock in the system.

## 4. Deadlock detection and recovery

This approach let the processes fall in deadlock and then periodically check whether deadlock occur in the system or not. If it occurs then it applies some of the recovery methods to the system to get rid of deadlock.

**5.5.3 Deadlock Prevention**

If we simulate deadlock with a table which is standing on its four legs then we can also simulate four legs with the four conditions which when occurs simultaneously, cause the deadlock.

However, if we break one of the legs of the table then the table will fall definitely. The same happens with deadlock, if we can be able to violate one of the four necessary conditions and don't let them occur together then we can prevent the deadlock.

Let's see how we can prevent each of the conditions.

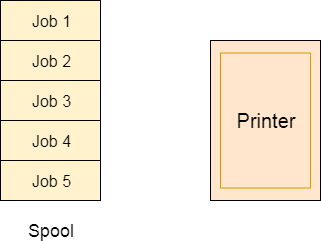
## 1. Mutual Exclusion

Mutual section from the resource point of view is the fact that a resource can never be used by more than one process simultaneously which is fair enough but that is the main reason behind the deadlock. If a resource could have been used by more than one process at the same time then the process would have never been waiting for any resource.

However, if we can be able to violate resources behaving in the mutually exclusive manner then the deadlock can be prevented.

### Spooling

For a device like printer, spooling can work. There is a memory associated with the printer which stores jobs from each of the process into it. Later, Printer collects all the jobs and print each one of them according to FCFS. By using this mechanism, the process doesn't have to wait for the printer and it can continue whatever it was doing. Later, it collects the output when it is produced.

****

**Fig. 5.7** Spooling

Although, Spooling can be an effective approach to violate mutual exclusion but it suffers from two kinds of problems.

1. This cannot be applied to every resource.
2. After some point of time, there may arise a race condition between the processes to get space in that spool.

We cannot force a resource to be used by more than one process at the same time since it will not be fair enough and some serious problems may arise in the performance. Therefore, we cannot violate mutual exclusion for a process practically.

## 2. Hold and Wait

Hold and wait condition lies when a process holds a resource and waiting for some other resource to complete its task. Deadlock occurs because there can be more than one process which are holding one resource and waiting for other in the cyclic order.

However, we have to find out some mechanism by which a process either doesn't hold any resource or doesn't wait. That means, a process must be assigned all the necessary resources before the execution starts. A process must not wait for any resource once the execution has been started.

**!(Hold and wait) = !hold or !wait (negation of hold and wait is, either you don't hold or you don't wait)**

This can be implemented practically if a process declares all the resources initially. However, this sounds very practical but can't be done in the computer system because a process can't determine necessary resources initially.

Process is the set of instructions which are executed by the CPU. Each of the instruction may demand multiple resources at the multiple times. The need cannot be fixed by the OS.

The problem with the approach is:

1. Practically not possible.
2. Possibility of getting starved will be increases due to the fact that some process may hold a resource for a very long time.

## 3. No Preemption

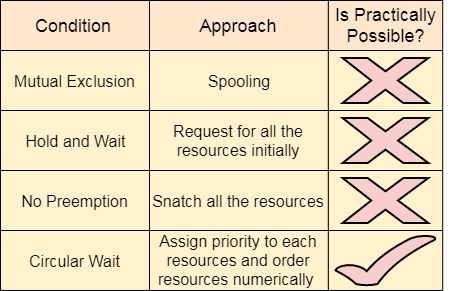
Deadlock arises due to the fact that a process can't be stopped once it starts. However, if we take the resource away from the process which is causing deadlock then we can prevent deadlock.

This is not a good approach at all since if we take a resource away which is being used by the process then all the work which it has done till now can become inconsistent.

Consider a printer is being used by any process. If we take the printer away from that process and assign it to some other process then all the data which has been printed can become inconsistent and ineffective and also the fact that the process can't start printing again from where it has left which causes performance inefficiency.

## 4. Circular Wait

To violate circular wait, we can assign a priority number to each of the resource. A process can't request for a lesser priority resource. This ensures that not a single process can request a resource which is being utilized by some other process and no cycle will be formed.

****

**Fig. 5.8** Deadlock Prevention

**5.6 Deadlock avoidance**

In deadlock avoidance, the request for any resource will be granted if the resulting state of the system doesn't cause deadlock in the system. The state of the system will continuously be checked for safe and unsafe states.

In order to avoid deadlocks, the process must tell OS, the maximum number of resources a process can request to complete its execution.

The simplest and most useful approach states that the process should declare the maximum number of resources of each type it may ever need. The Deadlock avoidance algorithm examines the resource allocations so that there can never be a circular wait condition.

Safe and Unsafe States

The resource allocation state of a system can be defined by the instances of available and allocated resources, and the maximum instance of the resources demanded by the processes.

A state of a system recorded at some random time is shown below.

#### Resources Assigned

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Process** | **Type 1** | **Type 2** | **Type 3** | **Type 4** |
| A | 3 | 0 | 2 | 2 |
| B | 0 | 0 | 1 | 1 |
| C | 1 | 1 | 1 | 0 |
| D | 2 | 1 | 4 | 0 |

#### Resources still needed

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Process** | **Type 1** | **Type 2** | **Type 3** | **Type 4** |
| A | 1 | 1 | 0 | 0 |
| B | 0 | 1 | 1 | 2 |
| C | 1 | 2 | 1 | 0 |
| D | 2 | 1 | 1 | 2 |

1. E = (7 6 8 4)
2. P = (6 2 8 3)
3. A = (1 4 0 1)

Above tables and vector E, P and A describes the resource allocation state of a system. There are 4 processes and 4 types of the resources in a system. Table 1 shows the instances of each resource assigned to each process.

Table 2 shows the instances of the resources, each process still needs. Vector E is the representation of total instances of each resource in the system.

Vector P represents the instances of resources that have been assigned to processes. Vector A represents the number of resources that are not in use.

A state of the system is called safe if the system can allocate all the resources requested by all the processes without entering into deadlock.

If the system cannot fulfill the request of all processes then the state of the system is called unsafe.

The key of Deadlock avoidance approach is when the request is made for resources then the request must only be approved in the case if the resulting state is also a safe state.

## 5.6.1 Banker’s Algorithm

**Banker’s Algorithm** is used majorly in the banking system to avoid deadlock. It helps you to identify whether a loan will be given or not.

This algorithm is used to test for safely simulating the allocation for determining the maximum amount available for all resources. It also checks for all the possible activities before determining whether allocation should be continued or not.

**For example, there are X number of account holders of a specific bank, and the total amount of money of their accounts is G.**

**When the bank processes a car loan, the software system subtracts the amount of loan granted for purchasing a car from the total money ( G+ Fixed deposit + Monthly Income Scheme + Gold, etc.) that the bank has.**

**It also checks that the difference is more than or not G. It only processes the car loan when the bank has sufficient money even if all account holders withdraw the money G simultaneously.**

## Banker’s Algorithm Notations

Here is an important notation used in Banker’s algorithm:

* X: Indicates the total number of processes of the system.
* Y: Indicates the total number of resources present in the system.

### Available

[I: Y] indicate which resource is available.

### Max

[l:X,l: Y]: Expression of the maximum number of resources of type j or process i

### Allocation

[l:X,l:Y]. Indicate where process you have received a resource of type j

### Need

Express how many more resources can be allocated in the future

## Example of Banker’s algorithm

Assume that we have the following resources:

* 5 Pen drives
* 2 Printers
* 4 Scanners
* 3 Hard disks

Here, we have created a vector representing total resources: Available = (5, 2, 4, 3).

Assume there are four processes. The available resources are already allocated as per the matrix table below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Process Name** | **Pen Drives** | **Printer** | **Scanner** | **Hard disk** |
| P | 2 | 0 | 1 | 1 |
| Q | 0 | 1 | 0 | 0 |
| R | 1 | 0 | 1 | 1 |
| S | 1 | 1 | 0 | 1 |
| Total | 4 | 2 | 2 | 3 |

Here, the allocated resources are the total of these columns:

Allocated = (4, 2, 2, 3).

We also create a Matrix to display the number of each resource required for all the processes. This matrix is called **Need**=(3,0,2,2)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Process Name** | **Pen Drives** | **Printer** | **Scanner** | **Hard disk** |
| P | 1 | 1 | 0 | 0 |
| Q | 0 | 1 | 1 | 2 |
| R | 2 | 1 | 0 | 0 |
| S | 0 | 0 | 1 | 0 |

The available vector will be :

Available=Available- Allocated

= (5, 2, 4, 3) -(4, 2, 2, 3)

=(1, 0, 2, 0)

### Resource Request Algorithm

Resource request algorithm enables you to represent the system behavior when a specific process makes a resource request.

Let understand this by the following steps:

**Step 1)** When a total requested instance of all resources is lesser than the process, move to step 2.

**Step 2)** When a requested instance of each and every resource type is lesser compared to the available resources of each type, it will be processed to the next step. Otherwise, the process requires to wait because of the unavailability of sufficient resources.

**Step 3)** Resource is allocated as shown in the below given Pseudocode.

Available = Available – Request (y)

Allocation(x) = Allocation(x) + Request(x)

Need(x) = Need(x) - Request(x)

This final step is performed because the system needs to assume that resources have been allocated. So that there should be less resources available after allocation.

## Characteristics of Banker’s Algorithm

Here are important characteristics of banker’s algorithm:

* Keep many resources that satisfy the requirement of at least one client
* Whenever a process gets all its resources, it needs to return them in a restricted period.
* When a process requests a resource, it needs to wait
* The system has a limited number of resources
* Advance feature for max resource allocation

## Disadvantage of Banker’s algorithm

Here, are cons/drawbacks of using banker’s algorithm

* Does not allow the process to change its Maximum need while processing
* It allows all requests to be granted in restricted time, but one year is a fixed period for that.
* All processes must know and state their maximum resource needs in advance.

**Chapter 6**

**File System:**

1. File systems, directories,
2. File system implementation, security protection mechanisms.
3. Input/output: Principles of I/O Hardware:

I/O devices,

Device controllers,

direct memory access.

Principles of I/O software:

1. Goals interrupt handlers, device drivers, and device independent I/O software.
2. User space I/O Software.
3. Disks:

* Disk hardware,
* Disk scheduling algorithms
  + - First come first serve,
    - shortest seek time first,
    - SCAN,
    - C-SCAN,
    - LOOK and
    - C-LOOK algorithms

1. Error handling, track-at-a-time caching,
2. RAM Disks.
3. Clocks: Clock hardware, memory-mapped terminals, I/O software.

**FILE SYSTEM**

A file is a collection of correlated information which is recorded on secondary or non-volatile storage like magnetic disks, optical disks, and tapes. It is a method of data collection that is used as a medium for giving input and receiving output from that program.

In general, a file is a sequence of bits, bytes, or records whose meaning is defined by the file creator and user. Every File has a logical location where they are located for storage and retrieval.

## Objective of File management System

## The main objectives of the file management system:

* It provides I/O support for a variety of storage device types.
* Minimizes the chances of lost or destroyed data
* Helps OS to standardized I/O interface routines for user processes.
* It provides I/O support for multiple users in a multiuser systems environment.

## Properties of a File System

Here, are important properties of a file system:

* Files are stored on disk or other storage and do not disappear when a user logs off.
* Files have names and are associated with access permission that permits controlled sharing.
* Files could be arranged or more complex structures to reflect the relationship between them.

## File structure

A File Structure needs to be predefined format in such a way that an operating system understands. It has an exclusively defined structure, which is based on its type.

Three types of files structure in OS:

* A text file: It is a series of characters that is organized in lines.
* An object file: It is a series of bytes that is organized into blocks.
* A source file: It is a series of functions and processes.

## File Attributes

A file has a name and data. Moreover, it also stores Meta information like file creation date and time, current size, last modified date, etc. All this information is called the attributes of a file system.

Here, are some important File attributes used in OS:

* **Name:** It is the only information stored in a human-readable form.
* **Identifier**: Every file is identified by a unique tag number within a file system known as an identifier.
* **Location:** Points to file location on device.
* **Type:** This attribute is required for systems that support various types of files.
* **Size**. Attribute used to display the current file size.
* **Protection**. This attribute assigns and controls the access rights of reading, writing, and executing the file.
* **Time, date and security:** It is used for protection, security, and also used for monitoring

## File Type

It refers to the ability of the operating system to differentiate various types of files like text files, binary, and source files. However, Operating systems like MS\_DOS and UNIX has the following type of files:

### Character Special File

It is a hardware file that reads or writes data character by character, like mouse, printer, and more.

### Ordinary files

* These types of files stores user information.
* It may be text, executable programs, and databases.
* It allows the user to perform operations like add, delete, and modify.

### Directory Files

* Directory contains files and other related information about those files. Its basically a folder to hold and organize multiple files.

### Special Files

* These files are also called device files. It represents physical devices like printers, disks, networks, flash drive, etc.

## Functions of File

* Create file, find space on disk, and make an entry in the directory.
* Write to file, requires positioning within the file
* Read from file involves positioning within the file
* Delete directory entry, regain disk space.
* Reposition: move read/write position.

## Commonly used terms in File systems

### Field:

This element stores a single value, which can be static or variable length.

### DATABASE:

Collection of related data is called a database. Relationships among elements of data are explicit.

### FILES:

Files are the collection of similar record which is treated as a single entity.

### RECORD:

A Record type is a complex data type that allows the programmer to create a new data type with the desired column structure. It groups one or more columns to form a new data type. These columns will have their own names and data type.

## File Access Methods

File access is a process that determines the way that files are accessed and read into memory. Generally, a single access method is always supported by operating systems. Though there are some operating system which also supports multiple access methods.

**Three file access methods are:**

* Sequential access
* Direct random access
* Index sequential access

### Sequential Access

In this type of file access method, records are accessed in a certain pre-defined sequence. In the sequential access method, information stored in the file is also processed one by one. Most compilers access files using this access method.

### Random Access

The random access method is also called direct random access. This method allow accessing the record directly. Each record has its own address on which can be directly accessed for reading and writing.

### Sequential Access

This type of accessing method is based on simple sequential access. In this access method, an index is built for every file, with a direct pointer to different memory blocks. In this method, the Index is searched sequentially, and its pointer can access the file directly. Multiple levels of indexing can be used to offer greater efficiency in access. It also reduces the time needed to access a single record.

## Space Allocation

In the Operating system, files are always allocated disk spaces.

Three types of space allocation methods are:

* Linked Allocation
* Indexed Allocation
* Contiguous Allocation

### Contiguous Allocation

In this method,

* Every file users a contiguous address space on memory.
* Here, the OS assigns disk address is in linear order.
* In the contiguous allocation method, external fragmentation is the biggest issue.

### Linked Allocation

In this method,

* Every file includes a list of links.
* The directory contains a link or pointer in the first block of a file.
* With this method, there is no external fragmentation
* This File allocation method is used for sequential access files.
* This method is not ideal for a direct access file.

### Indexed Allocation

In this method,

* Directory comprises the addresses of index blocks of the specific files.
* An index block is created, having all the pointers for specific files.
* All files should have individual index blocks to store the addresses for disk space.

## File Directories

A single directory may or may not contain multiple files. It can also have sub-directories inside the main directory. Information about files is maintained by Directories. In Windows OS, it is called folders.

**Following is the information which is maintained in a directory:**

* **Name** The name which is displayed to the user.
* **Type**: Type of the directory.
* **Position**: Current next-read/write pointers.
* **Location**: Location on the device where the file header is stored.
* **Size**: Number of bytes, block, and words in the file.
* **Protection**: Access control on read/write/execute/delete.
* **Usage**: Time of creation, access, modification

### DEVICE DRIVERS

* In computing, a device driver or software driver is a computer program allowing higher-level computer programs to interact with a hardware device.
* A driver typically communicates with the device through the computer bus or communications subsystem to which the hardware connects. When a calling program invokes a routine in the driver, the driver issues commands to the device.
* Once the device sends data back to the driver, the driver may invoke routines in the original calling program. Drivers are hardware-dependent and operating-system-specific.
* They usually provide the interrupt handling required for any necessary asynchronous time-dependent hardware interface.

### DISK STRUCTURE

* Disk provides bulk of secondary storage of computer system. The disk can be considered the one I/O device that is common to each and every computer. Disks come in many size and speeds, and information may be stored optically or magnetically. Magnetic tape was used as an early secondary storage medium, but the access time is much slower than for disks. For backup, tapes are currently used.
* Modern disk drives are addressed as large one dimensional array of logical blocks, where the logical block is the smallest unit of transfer. The actual details of disk I/O operation depend on the computer system, the operating system and the nature of the I/O channel and disk controller hardware.
* The basic unit of information storage is a sector. The sectors are stored on a flat, circular, media disk. This media spins close to one or more read/write heads. The heads can move from the inner portion of the disk to the outer portion.
* When the disk drive is operating, the disk is rotating at constant speed. To read or write, the head must be positioned at the desired track and at the beginning of the desired sector on that track. Track selection involves moving the head in a movable head system or electronically selecting one head on a fixed head system. These characteristics are common to floppy disks, hard disks, CDROM and DVD.

### DISK PERFORMANCE PARAMETERS

* When the disk drive is operating, the disk is rotating at constant speed. To read or write, the head must be positioned at the desired track and at the beginning of the desired sector on that track.
* Track selection involves moving the head in a movable-head system or electronically selecting one head on a fixed-head system. On a movable-head system, the time it takes to position the head at the track is known as seek time.
* When once the track is selected, the disk controller waits until the appropriate sector rotates to line up with the head. The time it takes for the beginning of the sector to reach the head is known as rotational delay, or rotational latency. The sum of the seek time, if any, and the rotational delay equals the access time, which is the time it takes to get into position to read or write.
* Once the head is in position, the read or write operation is then performed as the sector moves under the head; this is the data transfer portion of the operation; the time required for the transfer is the transfer time.
* Seek Time Seek time is the time required to move the disk arm to the required track. It turns out that this is a difficult quantity to pin down. The seek time consists of two key components: the initial startup time and the time taken to traverse the tracks that have to be crossed once the access arm is up to speed. Ts = m x n + s
* Rotational Delay Disks, other than floppy disks, rotate at speeds ranging from 3600 rpm up to, as of this writing, 15,000 rpm; at this latter speed, there is one revolution per 4 ms. Thus, on the average, the rotational delay will be 2 ms. Floppy disks typically rotate at between 300 and 600 rpm. Thus the average delay will be between 100 and 50 ms.
* Transfer Time The transfer time to or from the disk depends on the rotation speed of the disk in the following fashion:

**T= b/rN**

where T = transfer time

b = number of bytes to be transferred

N = number of bytes on a track

r = rotation speed, in revolutions per second

Thus the total average access time can be expressed as Ta = Ts + where Ts is the average seek time.

### Disk Scheduling

As we know, a process needs two types of time, CPU time and IO time. For I/O, it requests the Operating system to access the disk.

However, the operating system must be fare enough to satisfy each request and at the same time, operating system must maintain the efficiency and speed of process execution.

The technique that operating system uses to determine the request which is to be satisfied next is called disk scheduling.

### Seek Time: Seek time is the time taken in locating the disk arm to a specified track where the read/write request will be satisfied.

### Rotational Latency: It is the time taken by the desired sector to rotate itself to the position from where it can access the R/W heads.

### Transfer Time: It is the time taken to transfer the data.

### Disk Access Time: Disk access time is given as,

**Disk Access Time = Rotational Latency + Seek Time + Transfer Time**

### Disk Response Time: It is the average of time spent by each request waiting for the IO operation.

### Purpose of Disk Scheduling: The main purpose of disk scheduling algorithm is to select a disk request from the queue of IO requests and decide the schedule when this request will be processed.

### Goal of Disk Scheduling Algorithm

* Fairness
* High throughout
* Minimal traveling head time

### Disk Scheduling Algorithms

The list of various disks scheduling algorithm is given below. Each algorithm is carrying some advantages and disadvantages. The limitation of each algorithm leads to the evolution of a new algorithm.

* FCFS scheduling algorithm
* SSTF (shortest seek time first) algorithm
* SCAN scheduling
* C-SCAN scheduling
* LOOK Scheduling
* C-LOOK scheduling

### FCFS Scheduling Algorithm

The simplest form of scheduling is first-in-first-out (FIFO) scheduling, which processes items from the queue in sequential order. This strategy has the advantage of being fair, because every request is honored and the requests are honored in the order received. With FIFO, if there are only a few processes that require access and if many of the requests are to clustered file sectors, then we can hope for good performance.

* Priority With a system based on priority (PRI), the control of the scheduling is outside the control of disk management software.
* Last In First Out in transaction processing systems, giving the device to the most recent user should result. In little or no arm movement for moving through a sequential file. Taking advantage of this locality improves throughput and reduces queue length.

# Disadvantages

* The scheme does not optimize the seek time.
* The request may come from different processes therefore there is the possibility of inappropriate movement of the head.

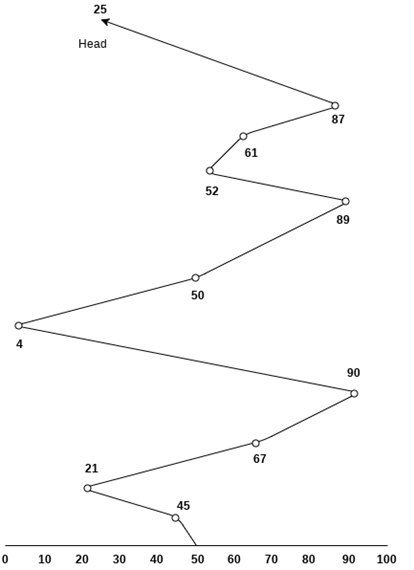
### Example

Consider the following disk request sequence for a disk with 100 tracks

45, 21, 67, 90, 4, 50, 89, 52, 61, 87, 25

Head pointer starting at 50 and moving in left direction. Find the number of head movements in cylinders using FCFS scheduling.

### Solution



Number of cylinders moved by the head

= (50-45)+(45-21)+(67-21)+(90-67)+(90-4)+(50-4)+(89-50)+(61-52)+(87-61)+(87-25)

= 5 + 24 + 46 + 23 + 86 + 46 + 49 + 9 + 26 + 62

= 376

# Shortest Seek Time First (SSTF) Scheduling Algorithm

* Shortest seek time first (SSTF) algorithm selects the disk I/O request which requires the least disk arm movement from its current position regardless of the direction. It reduces the total seek time as compared to FCFS.
* It allows the head to move to the closest track in the service queue.
* The choice should provide better performance than FCFS algorithm.
* Under heavy load, SSTF can prevent distant request from ever being serviced. This phenomenon is known as starvation. SSTF scheduling is essentially a form of shortest job first scheduling.

## Disadvantages

* It may cause starvation for some requests.
* Switching direction on the frequent basis slows the working of algorithm.
* It is not the most optimal algorithm.

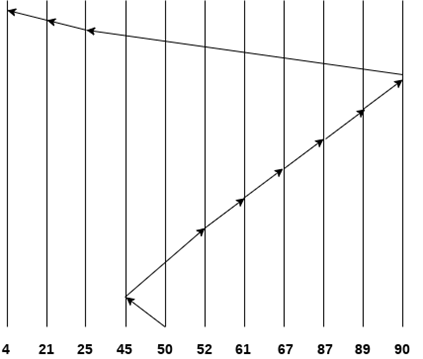
### Example

Consider the following disk request sequence for a disk with 100 tracks

45, 21, 67, 90, 4, 89, 52, 61, 87, 25

Head pointer starting at 50. Find the number of head movements in cylinders using SSTF scheduling.

**Solution:**



Number of cylinders = 5 + 7 + 9 + 6 + 20 + 2 + 1 + 65 + 4 + 17 = 136

## SCAN and C-SCAN algorithm

The scan algorithm has the head start at track 0 and move towards thehighest numbered track, servicing all requests for a track as it passes the track. The service direction is then reserved and the scan proceeds in the opposite direction, again picking up all requests in order

## OS SCAN and C-SCAN algorithm

## 

## 

## SCAN Algorithm

It is also called as Elevator Algorithm. In this algorithm, the disk arm moves into a particular direction till the end, satisfying all the requests coming in its path, and then it turns backand moves in the reverse direction satisfying requests coming in its path.

It works in the way an elevator works, elevator moves in a direction completely till the last floor of that direction and then turns back.

The scan algorithm has the head start at track 0 and move towards the highest numbered track, servicing all requests for a track as it passes the track. The service direction is then reserved and the scan proceeds in the opposite direction, again picking up all requests in order.

SCAN algorithm is guaranteed to service every request in one complete pass through the disk. SCAN algorithm behaves almost identically with the SSTF algorithm. The SCAN algorithm is sometimes called elevator algorithm.

### Example

Consider the following disk request sequence for a disk with 100 tracks

98, 137, 122, 183, 14, 133, 65, 78

Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using SCAN scheduling.

Number of Cylinders = 40 + 14 + 65 + 13 + 20 + 24 + 11 + 4 + 46 = 237

# C-SCAN algorithm

In C-SCAN algorithm, the arm of the disk moves in a particular direction servicing requests until it reaches the last cylinder, then it jumps to the last cylinder of the opposite direction without servicing any request then it turns back and start moving in that direction servicing the remaining requests.

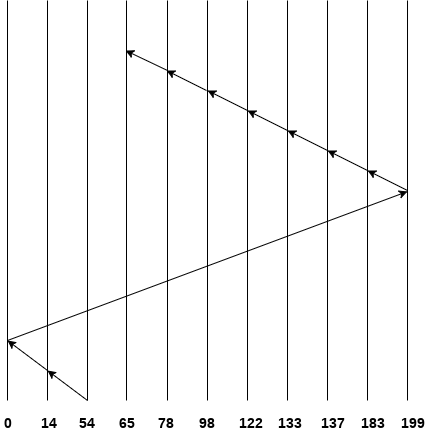
The C-SCAN policy restricts scanning to one direction only. Thus, when the last track has been visited in one direction, the arm is returned to the opposite end of the disk and the scan begins again. This reduces the maximum delay experienced by new requests.

### Example

Consider the following disk request sequence for a disk with 100 tracks

98, 137, 122, 183, 14, 133, 65, 78

Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using C-SCAN scheduling.



No. of cylinders crossed = 40 + 14 + 199 + 16 + 46 + 4 + 11 + 24 + 20 + 13 = 387

# Look Scheduling

It is like SCAN scheduling Algorithm to some extant except the difference that, in this scheduling algorithm, the arm of the disk stops moving inwards (or outwards) when no more request in that direction exists. This algorithm tries to overcome the overhead of SCAN algorithm which forces disk arm to move in one direction till the end regardless of knowing if any request exists in the direction or not.

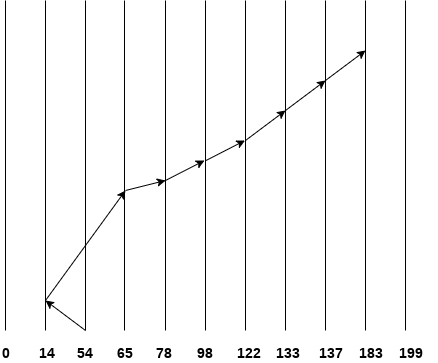
Start the head moving in one direction. Satisfy the request for the closest track in that direction when there is no more request in the direction, the head is traveling, reverse direction and repeat. This algorithm is similar to innermost and outermost track on each circuit.

### Example

Consider the following disk request sequence for a disk with 100 tracks

98, 137, 122, 183, 14, 133, 65, 78

Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using LOOK scheduling.



Number of cylinders crossed = 40 + 51 + 13 + +20 + 24 + 11 + 4 + 46 = 209

# C LOOK Scheduling

C Look Algorithm is similar to C-SCAN algorithm to some extent. In this algorithm, the arm of the disk moves outwards servicing requests until it reaches the highest request cylinder, then it jumps to the lowest request cylinder without servicing any request then it again start moving outwards servicing the remaining requests.

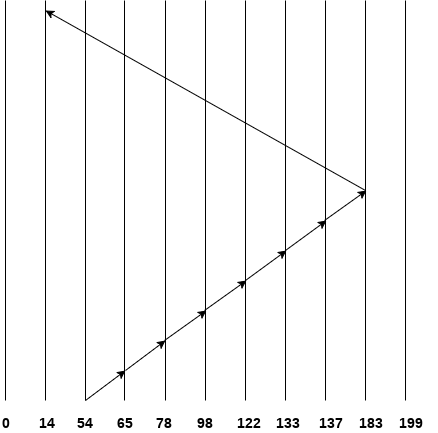
It is different from C SCAN algorithm in the sense that, C SCAN force the disk arm to move till the last cylinder regardless of knowing whether any request is to be serviced on that cylinder or not.

### Example

Consider the following disk request sequence for a disk with 100 tracks

98, 137, 122, 183, 14, 133, 65, 78

Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using C LOOK scheduling.



Number of cylinders crossed = 11 + 13 + 20 + 24 + 11 + 4 + 46 + 169 = 298

**DISK MANAGEMENT**

Operating system is responsible for disk management. Following are some activities discussed.

* Disk Formatting Disk formatting is of two types.

a) Physical formatting or low level formatting.

b) Logical Formatting

**Physical Formatting**

* Disk must be formatted before storing data.
* Disk must be divided into sectors that the disk controllers can read/write.
* Low level formatting files the disk with a special data structure for each sector.
* Data structure consists of three fields: header, data area and trailer.
* Header and trailer contain information used by the disk controller.
* Sector number and Error Correcting Codes (ECC) contained in the header and trailer.
* For writing data to the sector – ECC is updated.
* For reading data from the sector – ECC is recalculated.
* Low level formatting is done at factory

**Logical Formatting**

• After disk is partitioned, logical formatting used.

• Operating system stores the initial file system data structures onto the disk.

**Boot Block**

* When a computer system is powered up or rebooted, a program in read only memory executes.
* Diagnostic check is done first.
* Stage 0 boot program is executed.
* Boot program reads the first sector from the boot device and contains a stage-1 boot program.
* May be boot sector will not contain a boot program.
* PC booting from hard disk, the boot sector also contains a partition table.
* The code in the boot ROM instructs the disk controller to read the boot blocks into memory and then starts executing that code.
* Full boot strap program is more sophisticated than the bootstrap loader in the boot ROM.

**DISK RELIABILITY**

* Good performance means high speed, another important aspect of performance is reliability.
* A fixed disk drive is likely to be more reliable than a removable disk or tape drive.
* An optical cartridge is likely to be more reliable than a magnetic disk or tape.
* A head crash in a fixed hard disk generally destroys the data, whereas the failure of a tape drive or optical disk drive often leaves the data cartridge unharmed.

BOOKS:

1. Milenkovic, M., "Operating Systems - concepts and Design" McGraw Hill International EditionComputer Science series 1992.

2. Galvin P., J.L. Abraham Silberschatz. "Operating System Concepts". John Wiley & Sons Company, 1989. 4. Tanenbaum, A.S. "Modern Operating System", Prentice Hall of India Pvt.

Ltd.1995.

3. William Stallings “Operating Systems” , Prentice Hall of India Pvt. Ltd.

4. M. Morris Mano, “Computer System Architecture”, PHI, 3rd edition, 1993

5. Liu and Gibson, “8086/8088 Micro processor Assembly Language”