BLOCK I : REVIEW OF COMPUTER ORGANIZATION, MEMORY ARCHITECTURE, CONCURRENT PROCESS AND SCHEDULING

UNIT 1: COMPUTER SYSTEM REVIEW

Space for learners:

Unit Structure:

1.1 INTRODUCTION

Space for learners:

In this unit you will learn about different functional units or sub systems of a computer. A computer system is said to be functional if all the major subsystems work properly. This unit introduces a number of hardware units presents in a computer and give a broad overview and functional aspects of the same. Instruction set is another aspect of a computer systems to be discussed here, where we will learn about different addressing modes and Instruction set architecture of a computer.

1.2 UNIT OBJECTIVES

After going through this unit, you will be able to

- understand the major hardware units of Computer System
- learn about the instruction set architectures
- learn about the various types of addressing modes

1.3 COMPONENTS OF A COMPUTER

An electronic calculating machine that takes digitized information as input, processes the input according to internally stored one or more instructions and produces the output information can be termed as computer.

Digital computer consists of five functionally independent main units

a) Input units

b) Output units

c) Central processing unit

   i. Arithmetic and logic units

   ii. Control Units

   iii. Registers

d) Memory units

Space for learners:

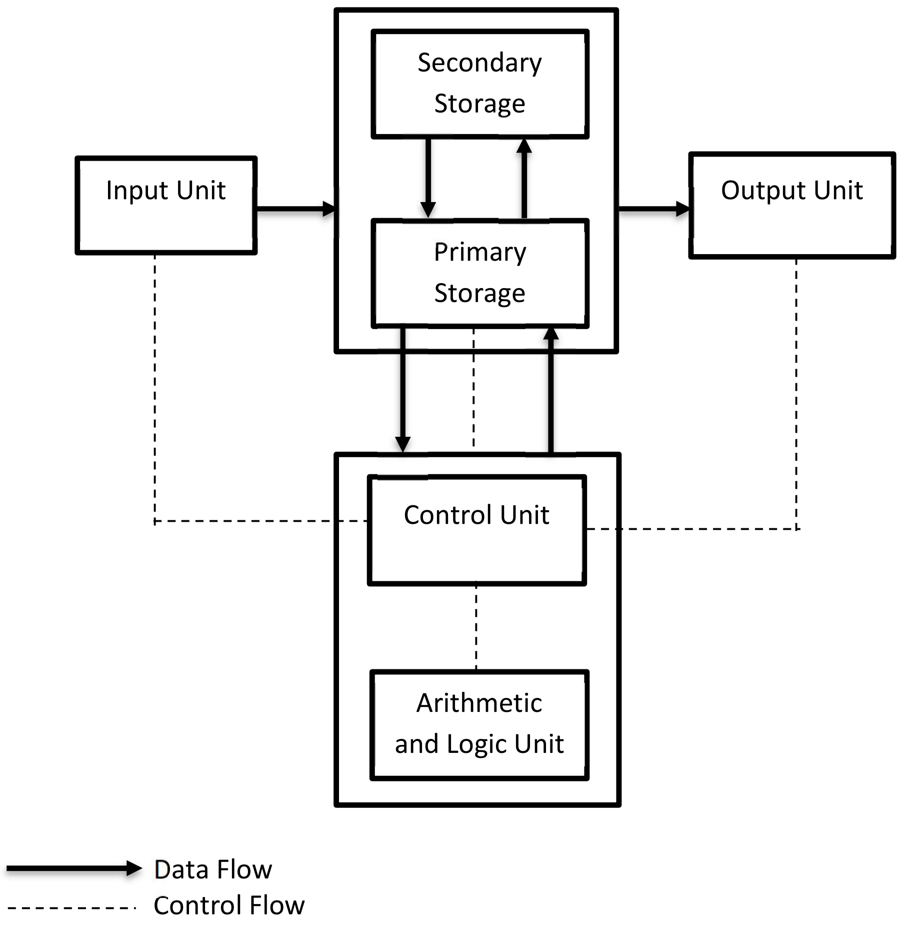


Figure 1: Block Diagram of a Computer

Figure 1 shows the block diagram of a computer introduced by John Von Neumann based on a stored-program concept. In this stored-program concept, programs and data or information are stored in a separate storage unit called memories and are treated the same.

Instructions are the commands that move the information within computer or between different computers and its Input and output (I/O) devices and performs arithmetic and logic operations.

A set of instructions that performs a task is called a program. The processor fetches the instructions from memory, one at a time and performs the desired operations unless there is some interrupt signal occurs.

1.3.1 Functional Units of a Computer

Space for learners:

1.3.1.1 Input Unit

Input unit or device is hardware equipment through which data and control signals are transferred to computer. Input unit converts data and command to computer understandable form. Examples of input devices: keyboards, mouse, scanners, digital cameras, joysticks, digital pen, digitizers, Touch Panel etc.

Figure: Keyboard, Mouse, Scanner, Joystick, Digital Pen

1.3.1.2 Output Unit

An output unit or device is a hardware equipment which converts digital information into human readable or understandable form. Example of output device: Monitor, Printer, Plotters, Speakers etc.

Figure: Monitor, Printer, Plotters, Speaker

CHECK YOUR PROGRESS-I

1. State TRUE or FALSE:

(a) Input device takes inputs from computer

(b) Joysticks is an input device

(c) Printer is used to display Output (True/False)

(d) Through Instruction we can move information within a computer

2. Fill in the Blanks:

(a) Computer use _________Unit to store information.

(b) Computer use _________Unit to do all arithmetic operations.

(c) Instruction are _________used in computer.

(d) Speaker is a ________ Device

1.3.1.3 Central Processing Unit (CPU)

Space for learners:

Central processing Unit; in short CPU is the brain of a computer system. All calculations are made inside the CPU. CPU is responsible for controlling all the devices and maintain communication between them. Arithmetic and logic unit, Control Unit and Registers together referred as central processing unit or processor.

i. *Arithmetic and Logic Unit (ALU):* The arithmetic logic unit is that part of the CPU that handles all the calculations the CPU may need, e.g. Addition, Subtraction, Comparisons. It performs Logical Operations, Bit Shifting Operations, and Arithmetic Operations.

ii. *Control Unit:* The control unit manages and co-ordinates all the operations of computer system through signals. It transfers all input and output flow, fetches instructions and controls data moves around the system.

iii. *Registers*: Registers are small amounts of high-speed memory contained within the CPU used as a temporary storage area. They are used by the processor to store small amounts of data that are needed during processing, such as:

   - Stores the address of the next executing instruction
   - The current instruction being executed
   - The results of calculations

Different processors have different numbers of registers for different purposes, but most have some, or all, of the following:

- *Program Counter:* Program Counter (PC) is used to keep the track of execution of the program. After successful completion of an instruction, PC points to the address of the next instruction to be fetched from the main memory.
- *Memory Data Register (MDR*): Memory Data Register contains data to be read or write from an addressed location.

- *Memory Address Register (MAR):* Memory Address Register is used to hold address of the location to be accessed from memory. The communication between the CPU and the main memory is handled by MAR and MDR.
- *Instruction Register (IR):* The Instruction Register holds the instruction which is just about to be executed. The instruction from PC is fetched and stored in IR. As soon as the instruction in placed in IR, the CPU starts executing the instruction and the PC points to the next instruction to be executed.
- *Accumulator (Acc) :* Accumulator is the frequently used register for storing data taken from memory. It is commonly used as a temporary location for storing data.
- *General Purpose Register:* These are numbered as R0, R1, R2….Rn-1, and used to store temporary data during any ongoing operation.

All the components of CPU are connected to the computer through buses. A bus is a high-speed internal connection. It can be assuming as an electrical wire for connecting and communicating between the units of CPU. Buses are used to send control signals and data between the processor and other components.

Three types of bus are used:

- *Address bus* - carries memory addresses from the processor to other components such as primary memory and input/output devices.
- Data bus - carries the actual data between the processor and other components.
- Control bus - carries control signals from the processor to other components. The control bus also carries the clock's pulses.

STOP TO CONSIDER

Program Counter always points to the address of the next instruction to be fetched from the main memory.
Accumulator is the frequently used register for storing data taken from memory.

Space for learners:

CHECK YOUR PROGRESS-II

3. State TRUE or FALSE:

(a) Control Unit Controls Only Arithmetic and Logic Unit

(b) Registers re used as Temporary Storage

(c) Adress bus is a register

(d) Data bus carries actual data

4. Fill in the Blanks:

(a) Program counter points____________

(b) Memory Data Register contains ____________

(c) Memory Address Register is used to hold _____________

(d) The Instruction Register holds__________

(e) Control Signal is transferred through ______ bus

Space for learners:

1.3.1.4 Memory Units

Memory Units are the storage space for storing program and data. Memory units are used for storing intermediate results and for final results. It has two broad categories.

i. Main Memory or Primary Memory

ii. Secondary Memory

Main Memory or Primary Memory

All computer uses primary memory for storing program and data when computing is running. Primary memory can operate at electronic speeds. When programs are being executed, it must be residing in the main memory. In main memory, a distinct address is mapped with each data location for accessing or manipulating data. Addresses are the numbers that identify successive location

Types of Primary Memory:

- Read Only Memory (ROM)
- Random Access Memory (RAM)
- Cache Memory

Read Only Memory (ROM): ROM is a memory device or storage medium that stores information permanently. It is called read only memory as we can only read the programs and data stored on it but cannot write on it. The manufacturer of ROM fills the programs into the ROM at the time of manufacturing the ROM. After this, the content of the ROM can't be altered, which means you cannot reprogram, rewrite, or erase its content later.

Various types of ROMs:

Programmable Read only Memory (PROM) is a programmable read only memory to store information only once by a user. PROM data cannot be erased.

Erasable Programmable Read Only Memory (EPROM) also a programmable read only memory to store information by a user. Stored information can be erased exposing it to strong ultraviolet light source

Electrically Erasable Programmable Read Only Memory (EEPROM) is a read only memory that can be programmed and can be erased electrically.

Random Access Memory (RAM): RAM provides operating memory for computer, when a program and data is being executed. CPU can access contents from RAM randomly from any location and any order. It is also called as read/write memory, since the information can be written to it as well as read from it. The more processes a computer needs to run at a single time, the more RAM it needs. RAM is as volatile memory. Volatile means information will be lost as soon as the power supply goes off.

Cache Memory: Cache memory is a type of fast, relatively small memory, which computer microprocessors can access more quickly than regular RAM. It is typically directly integrated with the CPU chip, or is placed on a separate chip that can connect CPU and RAM. The main purpose of this type memory is to store program instructions that are frequently used by software during its general operations, this is why fast access is needed as it helps to keep the program running quickly.

STOP TO CONSIDER

Before executing any data or instruction in a processor, it should be residing in RAM.

RAM termed as Random access because any location can be reached randomly with a same amount of time.

Space for learners:

Secondary Memory

Space for learners:

Secondary memory is a non-volatile and persistent computer memory. It enables a user to store data that can be retrieved, transmitted, and utilized by applications and services in real time. Secondary memory is used to store large amount of data or programs permanently.

Some basic characteristics of Secondary Memory

a) It is non-volatile, i.e. it retains data when power is switched off

b) It is large capacities to the tune of terabytes

c) It is cheaper as compared to primary memory. Secondary storage can be broadly divided into three categories

   - Magnetic Storage
   - Optical Storage
   - Solid state storage

- **Magnetic Storage:** Magnetic devices use magnetic fields to magnetise tiny individual sections of a metal spinning disk. Each tiny section represents one bit. A magnetised section represents a binary '1' and a demagnetised section represents a binary '0'. As the disk is spinning, a read/write head moves across its surface. To write data, the head magnetises or demagnetises a section of the disk that is spinning under it. To read data, the head makes a note of whether the section is magnetised or not. Magnetic devices are fairly cheap, high in capacity and durable. Example of Magnetic storage device: Hard Disks, Floppy Disk, magnetic tape

- **Optical storage:** Optical devices use a laser to store and read the stored data from an optical spinning disc made from metal and plastic. The disc surface is divided into tracks, with each track containing many flat areas and hollows. The flat areas are known as lands and the hollows as pits. When the laser shines on the disc surface, lands reflect the light back, whereas pits scatter the laser beam. A sensor looks for the reflected light. Reflected light - land - represents a binary '1', and no reflection - pits - represents a binary '0'. Example of Optical storage: CD-ROM (Compact Disc -Read only

Memory), DVD-ROM (Digital Versatile Disc-Read Only Memory), Blue Ray Disc

- **Solid state storage:** Solid state storage is a special type of storage made from silicon microchips. It can be written to and overwritten like RAM but it is non-volatile. Solid state is also used as external secondary storage. One of the major benefits of solid state storage is that it has no moving parts. Because of this, it is more portable, and produces less heat compared to traditional magnetic storage devices. Example of Solid State Storage: USB memory sticks and solid state drives (SSD)

Space for learners:

STOP TO CONSIDER

- ROM is non-volatile memory
- RAM is volatile memory
- Cache is a volatile memory
- Magnetic Storage device like hard disk is non-volatile memory
- Optical Storage device like CD DVD is non-volatile memory
- Solid state storage device like SSD Hard Disk, pen drive is non-volatile Memory

CHECK YOUR PROGRESS-III

5. State TRUE or FALSE:

(a) Secondary memory is also known as Main Memory

(b) ROM is volatile Memory

(c) RAM is a volatile Memory

(d) DVD is an Optical Media

6. Fill in the Blanks:

(a) ______ provides operating memory for computer

(b) Cache Memory Stores the instruction that are _________used

(c) Magnetic devices use _______ fields to store data

(d) SSD is mode from _________Microchips

1.3.1.5 Units of Memory

Space for learners:

The storage capacity of the memory is expressed in various units of memory. Bit (Binary Digit) is the primary or smallest unit of memory. A microprocessor uses binary digits 0 and 1 to decide the OFF and ON state respectively. The following table shows memory units

Sl	Units	Description
1	Bit	A binary digit is a logical 0 or 1 that indicates whether a component in an electric circuit is in the passive or active state.
2	Nibble	A group of 4 bits is called nibble.
3	Byte	A byte is a collection of 8 bits. The smallest unit that can represent a data item or a character is a byte.
4	Kilobyte (KB)	1 KB = 1024 Bytes
5	Megabyte (MB)	1 MB = 1024 KB
6	GigaByte (GB)	1 GB = 1024 MB
7	TeraByte (TB)	1 TB = 1024 GB
8	PetaByte (PB)	1 PB = 1024 TB

CHECK YOUR PROGRESS-IV

7. Calculate the followings:

(a) 4 Nibble = ________bit

(b) 1 byte = ________bit

(c) 1kilobyte = ________bit

(d) 1024 MB = ________byte

1.4 BASIC INSTRUCTION SETS OF COMPUTER

As mentioned in the previous section Instructions are the commands that move the information within computer or between different computers and its Input and output (I/O) devices and performs arithmetic and logic operations

An instruction set is a collection of machine language commands for a CPU. The term can apply to all of a CPU's potential instructions or a subset of instructions designed to improve performance in specific scenarios.

The instruction set consists of addressing modes, instructions, native data types, registers, memory architecture, interrupt, and exception handling, and external I/O

Machine language is the language, through which computer can understand and communicate. Machine language is made up of instructions and data that are all binary numbers.

An instruction set architecture (ISA), also called computer architecture, is an abstract model of a computer. There are various types of instruction set architecture available and each one has its own usage and advantages. The ISA serves as the boundary between software and hardware.

1.4.1 Instruction Set Architecture

Following are the instruction set architectures based on microprocessor architecture:

- RISC (Reduced Instruction Set Computer)
- CISC (Complex Instruction Set Computer)
- MISC (Minimal Instruction Set Computers)
- VLIW (Very Long Instruction Word)
- EPIC (Explicitly Parallel Instruction Computing)
- OISC (One Instruction Set Computer)
- ZISC (Zero Instruction Set Computer)

1.4.1.1 Reduced Instruction Set Computer (RISC)

Reduced Instruction Set Computer is an instruction set architecture (ISA) with less number of cycles per instruction (CPI) with extremely optimized set of Instruction

Space for learners:

1.4.1.2 Complex Instruction Set Computer (CISC)

Space for learners:

Complex Instruction Set Computer is an instruction set architecture (ISA) with fewer instructions per program than RISC. In CISC, single instructions can execute multiple low-level operations (like an arithmetic operation, load from memory and a memory store) or are capable of multi-step operations

1.4.1.3 Minimal instruction set computers (MISC)

Minimal instruction set computers is a processor architecture which has a very small number of primary instruction operations and corresponding opcodes. So MISC has smaller instruction set, a smaller and faster instruction set decode unit, and faster operation of individual instructions.

1.4.1.4 Very long instruction word (VLIW)

Very long instruction word is an instruction set architectures designed to achieve instruction level parallelism (ILP). Central processing units commonly allow programs to specify instructions to execute in sequence only. A VLIW processor allows programmes to explicitly define concurrent execution of instructions. This design aims to provide higher performance without the complexity inherent in some other designs.

Instruction-level parallelism (ILP) is the parallel or simultaneous execution of a sequence of instructions in a computer program

1.4.1.5 Explicitly parallel instruction computing (EPIC)

Hewlett Packard and Intel collaboratively defined and designed 64-bit microprocessor instruction set, for Explicitly Parallel Instruction Computing. EPIC is an instruction set that allows microprocessors to execute software instructions to control parallel instruction execution using compiler

1.4.1.6 One instruction set computer (OISC)

One instruction set computer is an abstract machine that uses only one instruction where no machine language opcode is used. OISC also well-known as ultimate reduced instruction set computer

(URISC). OISCs have been used as computational models in structural computing research and guides in teaching computer architecture.

1.4.1.7 Zero instruction set computer (ZISC)

A computer architecture based on pattern matching and the absence of micro-instructions is known as a zero instruction set computer (ZISC).

1.4.2 Instruction Set

The instruction set consists of a limited set of unique codes or commands that let the processor know what to do next, along with some basic rules of how to express them.

Instruction of a computer can be express with the followings

- *Instruction length (Length may vary):* Instruction length can range from as little as four bits in certain microcontrollers to hundreds of bits in some very long instruction word systems.
- *Opcodes:* An opcode (operation code) also known as instruction machine code is a command to the central processing unit
- *Operands:* An operand is the part of a computer instruction that specifies data that is to be operating on or manipulated. Basically, a computer instruction describes an operation (add, subtract, and so forth) and the operand or operands on which the operation is to be performed
- *Registers:* A processor register is a quickly accessible location available to a computer's processor.
- *Memory:* It is an external storage for larger and more versatile number of locations, with slower to access

An instruction can vary in length depending on the architecture. In x86 systems, the length of the instruction is normally 1 to 3 bytes (for the opcode), and a number of bytes needed for the operands, depending on the addressing mode.

Space for learners:

CHECK YOUR PROGRESS-V

8. State TRUE or FALSE:

(a) One Instruction set uses only one instruction.

(b) An operand is the part of a computer instruction.

(c) Instruction-level parallelism is the parallel or simultaneous execution of a sequence of instructions in a computer program.

(d) zero instruction set computer uses only one instruction.

9. Fill in the Blanks:

(a) Full form of RISC _________

(b) Full Form of CISC ________

(c) An opcode is a _____ to the central processing unit.

1.4.3 Addressing Mode

An addressing mode provides the way to calculate the effective memory address of an operand by using the information stored in registers and/or constants contained within a machine instruction. The different ways for specifying the locations of instruction operands are known as addressing modes.

In an instruction; the operation field specifies the operation to be performed. The executed operation may have executed on some data that is given explicitly on the instruction or stored in computer registers or memory words. The addressing mode of the instruction decides how the operands to be chosen during program execution. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced.

High-level language like C, C++, Java etc uses local and global variables, arrays, constants and pointers. For translating a high-level language program with human understandable code into assembly language or machine Language, the compiler must be able to implement or use these constructs using the facilities provided in the instruction set of the computer in which the program will be executed.

The ways through which the location of an operand can be found is known as addressing modes. Variables and constants are the simplest data types and are found in almost every computer program. In assembly language, registers or memory locations are used to represent the variable to hold values.

Followings are the different types of Addressing Modes:

Register mode:

CPU register contains the operand and the name of the register is given in the instruction.

Example: Add R2, R3

Absolute mode (Direct Mode):

Here the operand is stored in memory location and the address of the location is given explicitly in the instruction.

Example: Add LOC, R3

Immediate mode:

In this mode, the operand is explicitly given in the instruction without any register or memory location.

Say we want to store value 200 in register R0. Then, using the following immediate instruction we can do that

Move #200, R0

Immediate mode is commonly used to specify the source operand values.

The number sign (#) is used in front of the value to represent as an immediate operand.

Constant values are used frequently in high-level language programs. For example, if we evaluate the expression A = B + 8, where the expression contains the constant value 8. With the assumption that A and B variables have been declared earlier. Memory locations A and B may be accessed using the Absolute mode. The expression A = B + 8 can be expressed in assembly language as follows

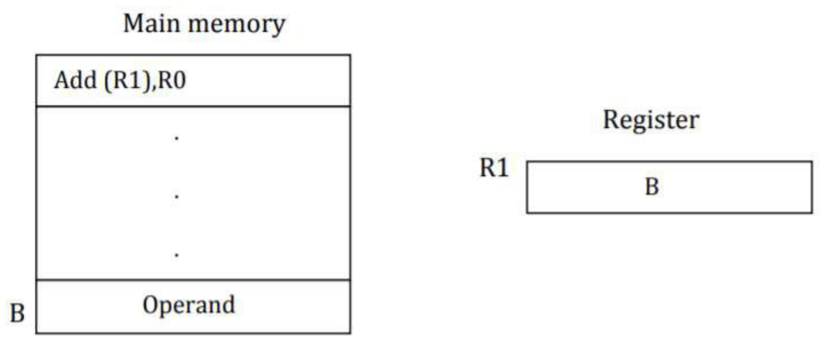
```
Move B, R1
Add #8, R1
Move R1, A
```

Space for learners:

Space for learners:

Indirect mode:

In the addressing mode operand or its address is not explicitly specified in the instruction. Instead, it provides information from which the memory address of the operand can be determined. This address can be referred as effective address (EA) of the operand. So in this mode, the effective address of the operand is the contents of a register or memory location whose address specifies in the instruction. The indirection mode is denoted by placing the name of the register or the memory address in the instruction in parentheses.



For example, consider the instruction, Add (R1), R0. For executing the above Add instruction, the processor fetches the value in register R1 and use as the effective address of the operand. Then the processor starts a read operation from the memory to read the contents of the specified location. The value fetches after read operation is the required operand, which the processor adds to the contents of register R0. The register or memory location that contains the address of an operand is called a pointer. Indirection and the use of pointers are important and powerful concepts in programming.

Index mode:

In this mode, a constant value (displacement) is added to the contents of a register to generate the effective address of the operand. The register used may be any one of the general-purpose registers or a special register for this purpose. In each case, it is referred to as an index register. Index mode is symbolically identified as

Space for learners:

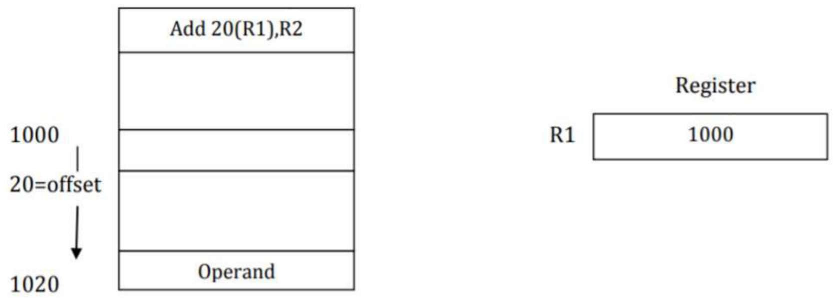
X(Ri)

Where Ri is the name of the register involved and X is the constant value contained in the instruction. The effective address of the operand can be calculated by

EA = X + [Ri].

Square bracket [] indicates the address of that location. Here [Ri] means, address of Ri. During the process of generating effective address, the contents of the index register are not changed.

In an assembly language program, the constant X may be given either as an explicit number or as a symbolic name representing a numerical value. When the instruction is translated into machine language, the constant X is given as a part of the instruction and is usually represented by fewer bits than the word length of the computer.



In the above figure, R1 is the index register that contains the address of a memory location. The value X defines an offset or displacement from the address in index register to the location where the operand is found. According to the above figure; R1 contains address 1000. Program statement is Add 20(R1), R2. So 20 displacements will be added to memory address 1000. So the operand will be found in memory location 1020. Result of the expression will be the addition of the content of operand stored in memory location 1020 and the Register R2.

There are two other variants of index mode;

- Here two register is used for index content. This type of index mode can in write as

(Ri,Rj)

The effective address can be calculated by adding the contents of registers Ri and Rj.

- This type of Index mode uses a constant along with two registers. This mode can be denoted as

  X(Ri,Rj)

  The effective address is the sum of the constant X and the contents of registers Ri and Rj.

Relative mode:

Relative mode is same as index mode. The only difference is that instead of general purpose register, here program counter (PC) for different execution.

Auto increment mode:

In this mode, contents of a register is used as Effective Address of the operand. After accessing the operand, the contents of this register is automatically incremented to point to the next instruction in the list.

Example: (Ri)+

In the above example Ri contains address of the operand. After execution of the instruction, the address contains in Ri will be incremented to point to the next instruction.

Autodecrement mode:

In this mode, contents of a register are used as Effective Address of the operand. After accessing the operand, the contents of this register is automatically decremented to point to the next instruction. Autodecrement mode is be denoted by putting the specified register in parentheses, preceded by a minus sign to indicate that the contents of the register are to be decremented before being used as the effective address

Example : - (Ri)

In the above example Ri contains address of the operand. After execution of the instruction, the address contains in Ri will be decremented to point to the next instruction.

Space for learners:

Space for learners:

CHECK YOUR PROGRESS-VI

10. State TRUE or FALSE:

(a) Absolute mode is also known as indirect mode.

(b) In Immediate mode, the operand is explicitly given in the instruction without any register or memory location

(c) Constant value in Index Mode is also known as displacement.

(d) Relative mode used General Purpose Register.

11. Fill the Blanks:

(a) The ways through which the location of an operand can be found is known as_________

(b) In Register Mode, Operand is stored in ______________

(c) After accessing the operand, the contents of this register is automatically decremented in ________ Addressing mode.

1.5 SUMMING UP

- A computer a fast calculating electronic machine. It has five main functional units; Input, output, Central processing and memory units
- CPU is a combination of these other units called ALU, Control unit and registers
- Program Counter (PC) registers point to the next instruction to be executed next.
- All the components of CPU are connected to the computer through buses. In an ideal computer system three types of bus used; address bus, data bus and control bus
- Before executing a program or instruction it should be stored in main or primary memory. From main memory CPU will fetch and executed the instruction
- RAM (Random access memory) is termed as Random access because any location can be reached randomly in a short and fixed amount of time after specifying its address.

- Cache memory is faster than RAM. And it is placed between RAM and Processor to synchronize the speed of processor and other slow speed devices
- An instruction set is a group of commands for a CPU in machine language
- Machine language is the language, through which computer can understand and communicate.
- An instruction set architecture (ISA), also called computer architecture, is an abstract model of a computer. The ISA serves as the boundary between software and hardware.
- An addressing mode specifies how to calculate the effective memory address of an operand by using information held in registers and/or constants contained within a machine instruction.

Space for learners:

1.6 ANSWERS TO CHECK YOUR PROGRESS

1.
   (a) False
   (b) True
   (c) False
   (d) True
2.
   (a) Memory
   (b) Arithmetic and Logic Unit
   (c) Commands
   (d) Output
3.
   (a) False
   (b) True
   (c) False
   (d) True
4.
   (a) Next Instruction
   (b) Data
   (c) Address
   (d) Instruction
   (e) Control
5.
   (a) False
   (b) False
   (c) True
   (d) True

6.
   (a) RAM
   (b) Frequently
   (c) Magnetic
   (d) Silicon

7.
   (a) 16
   (b) 8
   (c) 8192
   (d) 1073741824

8.
   (a) True
   (b) True
   (c) True
   (d) False

9.
   (a) Reduced Instruction Set Computer
   (b) Complex Instruction Set Computer
   (c) Command

10.
   (a) False
   (b) True
   (c) True
   (d) False

11.
   (a) Addressing modes
   (b) CPU register
   (c) Autodecrement

Space for learners:

1.7 POSSIBLE QUESTIONS

Short answer questions:

1. What is a computer?
2. Give two examples of pointing device?
3. Why we use secondary memory?
4. What is the role of Control Unit?
5. What is the functions of Arithmetic and logic Unit?
6. What is a program?
7. What is an instruction?
8. What do you understand by computer memory?

9. Why RAM is called as Random Access Memory
10. What is a registers?
11. What is a Program counter?
12. What is Memory Data Registers?
13. What is Memory Address Register?
14. What is the use of Instruction register?
15. What are the different types of Primary memory?
16. Convert the followings
    a. 1024 MB to bytes
    b. 1TB to Kilobytes
    c. 1 GB to Megabytes
17. What is addressing modes?
18. What is an opcode?
19. What is an operand?

Long answer questions

1. Mention four features of a computer system
2. Briefly describe the different units of computers.
3. Draw the block diagram of a computer and describe each unit.
4. Write difference between the followings
    a. Input unit and Output Unit
    b. RAM and ROM
    c. Primary Memory and Secondary Memory
5. What is bus? Discuss the different types of bus used in computer
6. What is Optical Storage media? Discuss how Optical media stores data in media.
7. What is Instruction set Architecture (ISA)? Discuss different types of ISA briefly.
8. Discuss different addressing modes use in computer Architecture.

Space for learners:

9. What is index addressing modes? Discuss different index addressing modes with example.
10. Discuss the advantages and disadvantages of secondary memory.

Space for learners:

1.8 REFERENCES AND SUGGESTED READINGS

- V. Carl Hamacher, Zvonko G. Vranesic, Safwat G. Zaky, *Computer Organization ,* McGraw-hill International Editions

UNIT 2: OPERATING SYSTEM OVERVIEW

Unit Structure:

Space for learners:

2.1 INTRODUCTION

We have often come across the term "operating system" and have used different kinds of operating system in our day to day life. For example, we use an operating system when we use a computer, a laptop or a mobile. Operating System can be defined as an interface between the user and the computer hardware. The goal of operating system is to improve the efficiency of a computer system. Different kinds of operating systems have been developed over the decades depending on their uses and new technical advances. Operating system performs various functions in the computer system like program execution, I/O operation, error detection etc.

2.2 UNIT OBJECTIVES

After going through this unit, you will be able to

- define operating system
- describe the history of operating system
- explain the different types of operating systems
- describe the various functions of operating systems

2.3 OPERATING SYSTEM

Space for learners:

The operating system controls and performs a lot of the functions in the computer system. Depending on the function it performs, there are various ways in which the operating system can be defined. However, the operating system can be defined in the following ways:

> "Operating system is a program that manages the computer hardware. It also provides a basis for application programs and acts as an intermediary between the computer user and the computer hardware." – [Ref 1]

Basically, the operation system has two main purposes. The first purpose is to provide a platform that is easier and convenient for the user to access and use the computer hardware. And the second purpose is to efficiently manage the different resources in the computer system.

2.3.1 Operating Systems Goals

The operating system has primarily two main goals. These goals are:

- *Efficiency*
- *Convenience*

Any operating system needs to be efficient in managing the various resources of the computer system. The optimum of the resources like CPU, memory, input/output devices etc. has to be made. The computer user does not directly communicate with the computer hardware. The computer user communicates with the hardware with the use of an operating system. Hence the operating system needs to be convenient for use to the user. Most of the operating systems are designed to be either efficient or convenient and some are designed for both. In addition to these two goals, the operating system should also be able to evolve over the years. Over the years, the user would require newer services and features and these need to be provided to the user. A good operating system should evolve by upgrading to newer versions that have better convenience and efficiency along with updated features.

CHECK YOUR PROGRESS-I

Q1: Define operating system?

Q2. What are the goals of operating system?

Space for learners:

2.4 HISTORY OF OPERATING SYSTEMS AND COMPUTERS

Computers have been in use for many decades now. The first digital computer was developed by Charles Baggage and named the "analytical engine", but it did not have an operating system. Operating systems have evolved a lot over the ages and there is no perfect mapping of operating systems with the different generations of computers. Still, let us look at the history of operating systems that have been developed and in use over the different generations of computers.

First Generation of Computer (1945-55): Vacuum Tubes

The technological advancement in the first generation of computers was the development of vacuum tubes. The machines used in this generation were mostly calculating engines which used mechanical relays. These mechanical relays were replaced by vacuum tubes. Programming was done using machines language in these machines. Assembly and high level programming languages were not used in this generation. Operating systems were also not used in this generation of computers. However, punched cards were introduced in this generation.

Second Generation of Computer (1956-65): Transistors and Batch Systems

The technological advancement in the second generation of computers was the development of transistors. There were now customers for the large sized computers used in this generation that took large rooms and are called as "mainframes". To run a job in these machines, a programmer would write the code and hand it over to the operator present in the input room. Depending on the language used in writing the programming code, the operator would load the compiler for that programming language. For example, if the code was written in FORTRAN, then the operator would search for the

FORTRAN compiler and load it to the computer for execution of that code. If the next job was written in a different programming language, then it required to unload the FORTRAN compiler and load the compiler for that specific language. This caused a lot of wastage of time. Hence, batch operating systems were introduced to reduce this wastage of time. A batch of similar jobs was collected together and then read and loaded one after another. For example, a batch of jobs using written in FORTRAN language. After the completion of one job, the operating system read and loaded the next job run immediately. This process saved the time required for loading and unloading of the compilers of different jobs.

Space for learners:

Third Generation of Computer (1965-1980): ICs and Multiprogramming

The technological advancement in the third generation of computers was the development of integrated circuits (ICs). The integrated circuits replaced the transistors of the second generation computers. The concept of multiprogramming was also developed in this third generation of computers. The CPU till now worked on the one job at hand and executed the CPU burst of instructions for that job. But when there was an I/O set of instructions, the CPU would remain idle since it had to wait for the I/O operation to be completed and this was a major loss of time and resource. The solution to this problem was to partition the computer memory and then have different jobs in these different partitions. The basic idea was that when one job was waiting for I/O operations to be complete, the CPU could be allocated to another job in one of these partitions. This would keep the CPU busy and waste the resource. The concept of time sharing operating system also introduced in this generation used multiprogramming to provide each user with a small portion of a time-shared computer. In the time sharing systems, each user had a terminal and the computer provided fast interactive service to multiple users such that it seemed like many users were using the computer at the same time. The first general-purpose timesharing system was CTSS (Compatible Time Sharing System) and its success led to the development of the MULTICS (MULTiplexed Information and Computing Service) system which was developed to support hundreds of simultaneous users. The MULTICS had an influence in the development of other operating systems like UNIX and Linux.

Fourth Generation of Computer (1980- Present): Personal Computers

Space for learners:

The technological advancement in the fourth generation of computers was the development of large scale integrated circuits (LSI). With LSIs in use now the size of the computer became small now as thousands of transistors could now be fitted into a square centimetre of silicon and thus gave rise to the development of personal computers. Disk Operating System (DOS) was one of the operating systems used in these times. Microsoft developed a new revised system called MS-DOS (Micro Soft Disk Operating System) which was hugely popular. The Apple Macintosh system was also developed during these times and was a success because of the cheap cost and user friendly GUI. Following the success of Macintosh, Microsoft developed their own graphical interface Windows, which was first used as a graphical environment on top of MS-DOS. But in 1995, Windows 95 was launched as a freestanding operating system with MS-DOS as an underlying component for booting and running MS-DOS programs. Over the years many newer versions of Windows were launched like Windows 98, Windows XP, Windows NT, Windows Me and Windows Vista. Windows 7 was one of the prominent operating system launched by Microsoft that had widespread popularity and demand. Later on, other newer versions of Windows were also launched like Windows 8, Windows 10 and Windows 11. UNIX is another popular operating system. LINUX is another alternative operating system that is popular for personal computers. In addition, network operating systems and distributed operating systems were also being developed in this generation of computers.

Fifth Generation of Computer (1990-Present): Mobile Computers

There are many operating systems specially developed for mobiles and smartphones. Symbian operating system was widely used in the early days of smartphones. It was the operating system that was used by major companies like Samsung, Motorola and Nokia. But soon other newly developed operating systems like Blackberry OS and iOS also gave competition to the existing operating systems. In 2011, Nokia introduced their smartphones with Windows platform. After the launch of Android operating system, it has quickly become one of the most popular operating system that is currently used in smartphones. Android is a Linux-based operating system and has the advantage that it is open source and available under a permissive license to evolve and adapt its operating system to cater to today's users' needs and demands. Apple's iOS is another operating system that is widely popular nowadays for smartphones.

Space for learners:

2.5 TYPES OF OPERATING SYSTEMS

Operating systems can be classified into different types. Let us look at some of the different types of operating systems:

- **Mainframe Operating Systems:** The mainframe operating systems are used in heavy processing oriented jobs where huge amounts of data and I/O are processed. There are typically three kinds of services for mainframe systems: batch, transaction processing and timesharing. Batch systems are used in jobs like sales reporting where interactive user are not required. Transaction processing systems are used in jobs that handle a large number of small requests in a short span of time. For example, in airline or train ticket reservation systems. Timesharing systems allow multiple remote users to execute jobs on the computer at the same time. Some mainframe computers perform all of the three functions. OS/390 is an example of mainframe operating system.

- **Server Operating Systems:** Server operating systems have servers which may be large personal computers, workstations or even mainframes. They serve multiple users who are connected over a network. The users can share different hardware and software resources among themselves like printer services, web services etc. Websites use these servers to store web pages and to handle the requests of clients. Some of the server operating systems are Solaris, Linux and Windows Server 201x.

- **Multiprocessor Operating System:** Multiprocessor operating systems are used to increase the computing power of a computer system by connecting multiple CPUs in a single system. Depending on the way these CPUs are connected they can be classified as parallel computers, multicomputer or multiprocessors. With the introduction of multicore chips in personal computers, the number of cores in personal computers like desktop and notebooks are only going to increase further more. Windows and Linux operating systems run on multiprocessors.

Space for learners:

- **Personal Computer Operating System:** Modern personal computer operating systems use multiprogramming to run multiple programs and are designed to support a single user. These are mostly used for simple applications like word processing, games and to access the Internet. Many versions from Linux, Windows and Apple OS are examples for personal computer operating system making these operating systems the most popular in the world.

- **Handheld Computer Operating Systems:** Handled computers or PDA (Personal Digital Assistant) are small computers that can be held in our hand. Smartphones and tablets are some of the examples of handheld devices. Some of the popular operating systems used in these devices are Google's Android and Apple's iOS. These devices have multicore CPUs, camera and other sensors. Third party applications can also be installed and used in these operating systems.

- **Embedded Operating System:** Embedded operating systems are used in devices like washing machines, microwave ovens etc. These devices are generally not thought of as computers. They differ from handheld devices like smartphones in the way that no third party applications can be installed or run in these machines as all the software is pre-installed in the ROM. This makes these devices safe from malicious software and in turn leads to a much less complicated design. Embedded Linux and VxWorks are two examples of embedded operating systems.

- **Sensor - Node Operating System:** Sensor – node operating systems are used in wireless sensor nodes. These sensors are small computers with CPU, RAM, ROM and one or more environmental sensors. It has a small operating system that is used to respond to events like for example detection of fire in a building. Like embedded systems here too the programs are pre-installed and third party applications cannot be installed which makes these devices safe and simpler to design. One of the most popular operating system for sensor node is the TinyOS.

- **Real – Time Operating System:** In real time operating systems, time is a major factor. Depending on the way deadlines are met, real time systems can be divided either into hard real - time systems or soft real – time systems. The hard real – time system must meet the deadlines and the actions need to happen at the exact precise time or else catastrophic events may occur. For example if a welding robot welds the car at wrong time then the car will get ruined. Soft real time systems on the other hand allow small flexibilities in meeting the deadlines provided there is no permanent damage. eCos is an example of a real time operating system. There is often an overlap between the handheld, embedded and real time operating systems.

CHECK YOUR PROGRESS-II

Q3: Name two devices where embedded operating systems are used?

Q4: What are hard and soft real time operating systems?

2.6 FUNCTIONS OF OPERATING SYSTEM

An operating system provides an environment to the user to run application programs and to communicate with the computer hardware. The operating system also needs to perform the jobs requested by the user in an efficient manner and in optimum time. This requires management of a lot of services and collaboration between the different parts of the computer system.

Some of the main functions of the operating system are described below:

- **User interface:** All operating systems have a user interface. This user interface can be a command based interface or a graphical user interface (GUI). In the command-line based interface, the user uses text commands to issue orders to the computer system. In the graphical user interface, instead of commands the user uses a pointing device to choose options from a menu, direct I/O and use a keyboard to enter text. Some systems also provide a combination of both the user interfaces.

__Space for learners:__

- **Program execution:** The operating system must be able to control the execution of the program. The operating system must be able to load the program into the computer memory and then execute it. The program must be able to end either normally or abnormally i.e. with errors.

- **I/O operations:** While a program is running, it may require I/O, which may involve a file or an I/O device. The device requested by the program may be for a printer or scanner or some other specific devices. Some of the I/O devices may require special functions for the use of I/O. Users cannot control the I/O directly and hence the operating systems are used to act as an interface between the user and I/O.

- **File – system manipulation:** Managing file system is an important function of the operating system. Programs need to read and write to files and directories while in executed. There are also other functions to be done on files like creating, deleting and appending a file. File permissions also need to be strictly maintained so that users can only access those files for which they have the required permission and access rights.

- **Communications:** Communications need to be maintained between processes for exchanging of information. These communications can be done through either message passing or through shared memory. The communication can be between processes that are on the same computer and even on different computers that are linked by a computer network.

- **Error detection:** One of the primary functions of operating system is to detect errors and take necessary action. These errors may happen in any part of the computer system like

Space for learners:

the CPU, the memory, the I/O devices or in the program itself. Once the error is detected the operating system should take action to ensure correct computing.

- **Resource allocation:** The operating system needs to have an efficient way to deal with resource allocation for multiple users and their resource needs. Different types of resources are managed in different ways by the operating system based on the type of resource. For example, to allocate a resource like CPU between different processes, CPU-scheduling algorithms may be used based on different strategies like first come first serve or priority based. Similarly, operating system uses different handling and managing mechanisms for other resources also.

- **Protection and security:** Protection and security are important aspects to be considered for operating systems in today's world. Several processes are executed concurrently in the computer system and it should not be possible for one process to interfere with another process. Protection means that all access to system resources should be controlled. Security means that outsider's access to system resources is not allowed. This can be done by authenticating users by means of a password or other tools. The protection and security is maintained for all users and for all resources in the computer system.

CHECK YOUR PROGRESS-III

Q5: What are the different types of user interface provided by operating systems?

Q6: Give an example on how operating system does resource allocation.

2.7 SUMMING UP

- The operating system controls and performs a lot of the functions in the computer system.
- The operating system has primarily two main goals: efficiency and convenience.

- The technological advancement in the first generation of computers was the development of vacuum tubes.
- The technological advancement in the second generation of computers was the development of transistors.
- The technological advancement in the third generation of computers was the development of integrated circuits (ICs) and the concept of multiprogramming.
- The concept of time sharing operating system also introduced in this generation used multiprogramming to provide each user with a small portion of a time-shared computer.
- The technological advancement in the fourth generation of computers was the development of large scale integrated circuits (LSI).
- There are different types of operating systems like mainframe operating system, server operating system, personal computer operating system, multiprocessor operating systems, handheld computer operating system, embedded operating systems, sensor node operating system, real time operating system etc.
- The main functions of the operating system includes providing user interface, program execution, I/O operations, file system manipulation, communication, error detection, resource allocation and to look after the protection and security of the computer systems.

2.8 ANSWERS TO CHECK YOUR PROGRESS

Q1: Operating system is a program that manages the computer hardware. It also provides a basis for application programs and acts as an intermediary between the computer user and the computer hardware

Q2: The operating system has primarily two main goals: efficiency and convenience

Q3: Two devices where embedded operating systems are used are washing machines and microwave ovens.

Q4: The hard real time system must meet the deadlines and the actions need to happen at the exact precise time or else catastrophic

Space for learners:

events may occur. Soft real time systems on the other hand allow small flexibilities in meeting the deadlines provided there is no permanent damage

Q5: This user interface can be a command based interface or a graphical user interface (GUI). In the command-line based interface, the user uses text commands to issue orders to the computer system. In the graphical user interface, instead of commands the user uses a pointing device to choose options from a menu, direct I/O and use a keyboard to enter text. Some systems also provide a combination of both the user interfaces.

Q6: To allocate a resource like CPU between different processes, the operating system uses CPU-scheduling algorithms that are based on different strategies like first come first serve or priority based methods.

2.9 POSSIBLE QUESTIONS

1. What is an operating system?
2. What are the goals of operating system?
3. What are handheld operating systems and personal computer operating systems?
4. What are sensor node operating systems? Give two applications where sensor node operating systems are used.
5. Describe in brief the concept behind batch operating systems.
6. Describe the concept behind multiprogramming and time sharing operating systems.
7. Write a brief note on the different operating systems used in smartphones.
8. Discuss the history of operating system in relation to the different generations of computers.
9. Describe the different types of operating system.
10. Describe the functions of operating system.

Space for learners:

2.10 REFERENCES AND SUGGESTED READINGS

Space for learners:

1. Silberschatz, Abraham, Peter Baer Galvin, and Greg Gagne. *Operating system principles*. John Wiley & Sons, 2006.
2. Tanenbaum, Andrew S., and Herbert Bos. *Modern operating systems*. Pearson, 2015.
3. Tanenbaum, Andrew S., and Albert S. Woodhull. *Operating systems: design and implementation*. Vol. 68. Englewood Cliffs: Prentice Hall, 1997.

UNIT 3: INTRODUCTION TO LINUX

Space for learners:

Unit Structure:

3.1 INTRODUCTION

Like Window and Mac, Linux is also an Operating System. It is Free and Open Source. As of now, Linux is the largest Open-Source Software Projects in the world.

3.2 UNIT OBJECTIVES

After going through this unit, you will be able to:

- know the history of linux,
- understand the Architecture and File System of Linux,
- know different Linux commands.

3.3 HISTORY OF LINUX

Linux is a free open-source secure community used operating system. The operating system is based on Linux Kernel which was released on September 17, 1991, by Linus Torvalds. The source code of the operating system can be modified and distributed to anyone by the Linux community under the GNU General Public License. Earlier, it was used for personal computers and gradually, used in servers, mainframe computers, supercomputers, etc. It is also used in embedded systems, robotic automation, smartwatches, etc. The Androids (operating system) running on a smartphone, smartwatch, and tablets are based on the Linux kernel and are the key success of Linux in the current time. It is generally packaged and distributed in a Linux distribution under GNU.

3.4 LINUX DISTRIBUTIONS

From the very beginning of the development of Linux, the idea followed regarding its distribution were:

- ✓ user can have it for free,
- ✓ user has the source code also for free,
- ✓ user can modify the code and redistribute it for free or priced along with the source code.

The above ideas were then termed **copyleft**. This term was originated from Free Software Foundation. The Free Software Foundation is a non-profit organization and was founded by Richard Stallman in the year 1985.

The distribution of the Linux operating system is made up of Linux kernel software or libraries. The distribution of Linux systems is distributed in different embedding systems or devices, or the personnel computers. A few of the Linux distributions are mentioned below.

i) MX Linux: It is one of the popular OSs which is based on the Debian Linux OS. The OS is more friendly for beginners and intermediates.

ii) Linux Mint: The Linux Mint OS is working as a windows OS more simply and any newcomers can use this OS as like Windows OS.

Space for learners:

iii) Ubuntu: The Ubuntu OS is very simple and easy to use as Mac OS. This OS is based on the Debian OS and hence, it is a stable OS.

iv) Debian: The Debian Linux OS is very stable. It is more complex than other Linux OS and hence, it is not recommended to a new user.

v) Solus: This Linux distribution is developed independently for 64-bit architecture. It is intentionally developed for personal computers where enterprise and server environment-based software are not included.

vi) Fedora: This Linux distribution was developed by the Fedora project, which is similar to RedHat. It is easy to use on laptop and desktop systems. It includes the latest data center technologies.

vii) openSUSE: This Linux OS is a project Linux distribution that serves to promote the use of Free and Open-Source Software(FOSS).

viii) RedHat: This Linux OS is commercial, and its products are freely available. The OS kept their trademark for not distributing their software for being redistributed.

ix) CentOS: CentOS provides an upstream open-source computing platform to the developer to contribute continuously with its upstream source, i.e., Red Hat Linux.

x) Arch Linux: The arch Linux OS is an independent Linux OS that has been developed for 64-bit OS. It provides the latest stable version of the software.

3.5 LINUX ARCHITECTURE

Space for learners:

The Linux Architecture depicted in Fig. 3.1.

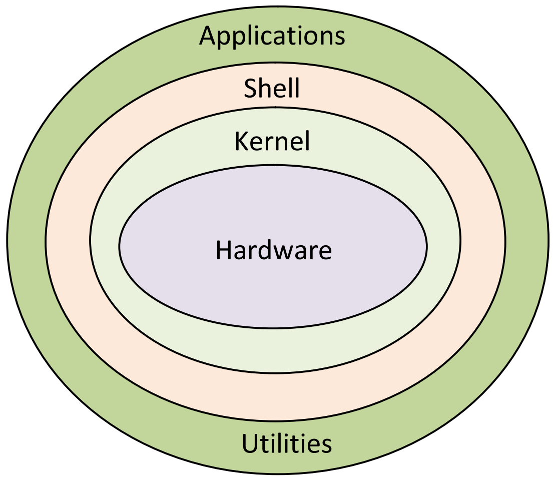


Fig. 3.1: Architecture of Linux

Let's discuss the components, mentioned in fig. 3.1 one by one.

Hardware: This layer, as all of you know, consists of different computer peripherals like ROM, RAM, CPU, Keyboard, Monitor, etc.

Kernel: It is the core/heart of the Linux O/S. The kernel is the core software interface between a computer system's hardware and its processes. It also prevents and mitigates conflicts between different processes. The kernel code is mostly written in C language. When a system boots (in UNIX/Linux), the kernel is loaded into the memory. The types of kernels are:

- Monolithic Kernels
- Hybrid Kernels
- Exo Kernels
- Micro Kernels

The jobs of the kernel are:

- ✓ Process Management (and System Calls)
- ✓ Memory Management
- ✓ Device Drivers

Shell: Shell is a software layer between the Kernel and User Processes like Application, Utilities, and Commands, etc. It is commonly known as Command Interpreter. Thus, whenever a user gives instructions to execute an application or command, the shell interprets them first and then executes them.

Apart from being a Command Interpreter, it is also a scripting language with components like variables, loops, conditional statements, functions, and many more.

Utilities and Applications: Linux OS has System Libraries that are used for different services such as process management, concurrency, memory management, etc. These libraries are implemented for several OS functionalities and need to access the code for the same.

Utilities are the programs that provide almost all the functionalities of an O/S to the users. These perform the specialized level and individual activities of the OS.

The applications, as we all know, are programs that are for different purposes.

CHECK YOUR PROGRESS - I

1. What is a Linux operating system?
2. What do you mean by Linux Mint?
3. What is a debian Linux?
4. What is fedora Linux?
5. What is Linux kernel and library?
6. True or false?
   a. The core part of linux is kernel.
   b. Shell provides the command line interpreter.

3.6 LINUX SHELLS

As discussed above, a shell is a program that acts as an interface between a user and the kernel. It allows a user to give commands to the kernel and receive responses from it. Through a shell, we can execute programs and utilities on the kernel.

There are different types of shells that exist in Linux. Let's know about some commonly used shells.

Bourne Shell: It was developed by Steve Bourne in AT&T Bell Labs and is denoted by "**sh**". In UNIX, the Bourne shell is regarded as the first shell. Due to its compactness and speed, this shell gained tremendous popularity.

Path to the shell: **/bin/sh** and **/sbin/sh**

root User Prompt: **#**

Non **root** User Prompt: **$**

C Shell: It was developed by Bill Joy at the University of California and is denoted by "**bash**". This shell has the support for arithmetic operations with syntax similar to C Programming Language.

Path to the shell: **/bin/csh**

root User Prompt: **#**

Non **root** User Prompt: **%**

Korn Shell: It was developed by David Korn in AT&T Bell Labs and is denoted by "**ksh**". It supports all the features of Bourne Shell and also the arithmetic programming features like C shell.

Path to the shell: **/bin/ksh**

root User Prompt: **#**

Non **root** User Prompt: **$**

GNU Bourne-Again Shell: This shell was developed not only to match with the Bourne shell but also to incorporate the features of C and Korn shells. It is the default shell in Linux.

Path to the shell: **/bin/bash**

root User Prompt: bash-versionNumber**#**

Non **root** User Prompt: bash-versionNumber**$**

The computer which is designed to run the UNIX shell is known as a shell script. It is a list of commands, which are listed in the order of execution. A good shell script will have comments, preceded by the # sign, describing the steps.

Lets, you are writing a shell script. A shell script can be saved as a .sh extension. Before you add anything, you need to start your shell script as follows.

```
#!/bin/sh
```

Space for learners:

This tells the system that the commands that follow are to be executed by the Bourne shell. One can put comments in the script as follows –

```
#!/bin/bash
# University: Gauhati University
# Branch: IDOL
pwd
ls
```

Now, Save the above content and make the script executable form. Before that save your shell as the filename.sh. lets the file of the shell is test.sh

```
$chmod +x test.sh
```

Now, the shell script is ready to be executed and for that type,

```
$./test.sh
```

3.7 LINUX COMMANDS FOR FILE AND DIRECTORY

In the Linux OS, the command is considered a Linux utility, and all the basic and advanced tasks are executed using the Linux commands. The commands are executed in the Linux terminal. Commands in Linux are case-sensitive. To open a terminal, one needs to press the "CTRL + ALT + T" keys together and execute the command by pressing ENTER. Few of the Linux commands are defined as follows.

i) **pwd** :

The pwd directory denotes the current directory of the user. The command gives the absolute path which starts from the root. The root is the base of any Linux system. The path is denoted by the slash(/) and the current user directory is as like below

"/home/username"

ii) **ls** :

Space for learners:

The ls command is used to know what files are in the directory you are in. The user can see all the hidden files by using the command **“ls -a”**.

iii) **cd**:

The cd directory command is used to go to a directory. For example, if the user want o move another directory from the home directory, then the user can type the following

cd directory_name

The command is case-sensitive, so the user needs to type the directory exactly the correct one.

iv) **mkdir & rmdir:**

The **mkdir** command is used to create a new folder or a directory. For example, if a user wants to make a directory IDOL, then the user should type **“mkdir IDOL**”. If the user wants to make a directory in a specific position or under a specific directory, then the user should go to these directories before the creation of a new directory by using the command cd.

The **rmdir** is used to delete an empty directory. But to delete a directory with files, the user should use the rm.

v) **rm** :

The rm command is used to delete files and directories. The user should type "**rm -r**" to delete just the directory. It deletes both the folder and the files it contains when using only the **rm** command.

vi) **touch**:

The **touch** command is used to create a file in the Linux system. The command can be used for anything, from an empty text file to an empty zip file. For example, “**touch idol.txt**”.

vii) **man & --help**:

The **man** command is used to know more about command and how to use it. For example, “**man cd**” shows the manual pages of

Space for learners:

the **cd** command. Typing in the command name and the argument helps it show which ways the command can be used (e.g., **cd –help**).

viii) **cp**:

The cp command is used to copy files through the command line by considering two arguments: The first is the location of the file to be copied, the second is where to copy.

ix) **mv**:

The mv command is used to rename a file. For example, if a user wants to rename the file "**idol1**" to "**idol2**", we can use "**mv idol1 idol2**".

x) **locate**:

The locate command is used to locate a file in a Linux system, just like the search command in Windows. This command is useful when you don't know where a file is saved or the actual name of the file. If you want a file that has the word "idol", it gives the list of all the files in your Linux system containing the word "hello" when you type in "locate -I idol".

STOP TO CONSIDER

Under a directory (in Linux), apart from the entries for files and sub-directories two more entries exists and these are "." and "..".
"." refers to the current working directory and
".." refers to the parent directory of the current working directory

3.8 LINUX COMMANDS FOR PROCESS MANAGEMENT

A process is an instance of a running program. When a user executes a program or executes a command in Linux, it means that the OS creates a process. The Linux operating system creates the five-digit ID for each process which is known as Process ID (PID). Each process has a unique ID. The OS tracks the process through the PID. Pids eventually repeat because all the possible numbers are used up and the next PID rolls or starts over. At any

Space for learners:

point in time, no two processes with the same PID exist in the system because it is the PID that Unix uses to track each process.

The user can start the UNIX process in two ways:

i) Foreground Processes:

Every process that a user runs are in the foreground. The process gets the input from the keyboard and sends the output to the screen. It can be shown using the ls command. The foreground process is also known as the interactive process. These processes are initiated by the user but not by the system. While these processes are running we can not directly initiate a new process from the same terminal.

The process runs in the foreground, the output is directed to the user screen, and if the ls command wants any input (which it does not), it waits for it from the keyboard.

ii) Background Processes

A background process runs without being connected to the user keyboard. If the background process requires any keyboard input, it waits. That's why such kinds of processes are known as non-interactive processes. These processes are initiated by the system itself or by users, though they can be managed by users. These processes have a unique PID or process. The system can initiate other processes also with different PIDs.

The different terms related to the Linux process are presented below.

Space for learners:

i) Listing Running Processes

It is easy to see the processes by running the **ps** (process status) command.

The –f flag is used more commonly along with the ps command for more information such as UID, PID, PPID, C, STIME, TTY, TIM, and CMD. The UID denotes the User ID that this process belongs to. The PID denotes the process ID. The PPID denotes the parent process ID. C is the CPU utilization process. STIME is process time. TTY is the terminal type associated with the process. Time denotes the CPU time taken by the process. CMD denotes the command that started this process.

ii) Stopping Processes

The ending of the process can be done in several different ways. The CTRL + C keystroke will exit the command. This works when the process is running in the foreground mode. If a process is running in the background, the user should get its Job ID using the ps command and then use the kill command to kill the process as follows.

kill job_ID.

iii) Parent and Child Processes

The process of UNIX has two numbers. The first number represents the Process ID (PID) and the second number represents the parent process ID (PID). The user can use the ps –f command for the process ID and the parent process ID.

iv) Zombie and Orphan Processes

Whenever the parent process is killed before its child, then this process is called an orphan process. In this case, the "parent of all processes," the init process, becomes the new PPID (parent process ID).

A Zombie is a process that has completed its task but still, it shows an entry in a process table. Zombie process states always indicated by Z. The zombie process treated as dead they are not used for system processing.

Space for learners:

v) Daemon Processes

The system-related process which is running in the background is known as Daemon Process. The daemon process does not have controlling terminals. If a program runs for a long time, then this process is a daemon process.

vi) The top Command:

The top command is a very useful command in Linux OS which is used to display the Linux process. The real-time view of the Linux system can be viewed by using the top command. The running operation of the system along with the process running in the OS can be viewed using the top command.

3.9 LINUX COMMANDS FOR FILE CONTENT AND USER MANAGEMENT

The operations in Linux OS have been performed on files. The files are handled using directories that are organized in a tree structure. The files of a Linux OS can be divided into 3 categories.

i) Regular Files:

Regular files are the common types of files that include text files, images, binary files, etc. These files can be created using the touch command. The regular file contains ASCII or Human Readable text, executable program binaries, program data, etc.

ii) Directories:

The windows OS represents the directories as folders. But in the Linux operating system, it is known as directories. The directories store the list of file names and the related information. The root directory(/) is the base of the system, /home/ is the default location for the user's home directories, /bin for Essential User Binaries, /boot – Static Boot Files, etc. One can create new directories with the mkdir command

iii) Special Files:

Space for learners:

The real physical devices in the Linux system can be used as special files. The user can use these file systems as ordinary files.

In the Linux operating system, a user is an entity that can manipulate the files and perform different operations in the OS. An ID is assigned to each user in the operating system. The root user ID is 0 whereas the other ID varies from 1 to 999 are assigned for the system user. The other local user IDs start from 1000.

The following commands are used for the user management

i) Using the id command of the Linux OS, one can get the ID of the username.

ii) The command useradd adds a new user to the directory. The user is given the ID automatically depending on which category it falls in.

iii) The password command is used to assign a password to the user. After using this command, the user can update a new password.

iv) To access user configuration file cat /etc/passwd command is used. This command prints the data of the configuration file.

v) The usermod -u new_id username command is used to change the user id.

vi) The command usermod -g new_group_id username is used to modify the group id of the user.

vii) Using the command sudo usermod -l new_login_name old_login_name, one can change the login name.

viii) The command usermod -d new_home_directory_path username is used to change the home directory.

ix) Using the command, userdel -r username, anyone can delete the user information.

Space for learners:

CHECK YOUR PROGRESS - II

7. What is a shell and types of shell?
8. What is command prompt in Linux?
9. How does a shell script start?
10. Give five examples of command.
11. What is Linux process and types?
12. What is daemon process?

Space for learners:

3.10 SUMMING UP

- Linux is a free open-source secure community used operating system.
- The source code of the operating system can be modified and distributed to anyone by the Linux community under the GNU General Public License.
- MX Linux is one of the popular OSs which is based on the Debian Linux OS.
- The Linux Mint OS is working as windows OS more simply and any newcomers can use this OS as like Windows OS.
- The Ubuntu OS is very simple and easy to use as Mac OS. This OS is based on the Debian OS and hence, it is a stable OS.
- The Debian Linux OS is very stable. It is more complex than other Linux OS.
- Solus Linux distribution is developed independently for 64-bit architecture.
- Fedora Linux distribution was developed by the Fedora project, which is similar to RedHat. It is easy to use on laptop and desktop systems.
- openSUSE Linux OS is a project Linux distribution that serves to promote the use of Free and Open-Source Software(FOSS).
- RedHat Linux OS is commercial, and its products are freely available.

Space for learners:

- CentOS provides an upstream open-source computing platform to the developer to contribute continuously with its upstream source, i.e., Red Hat Linux.
- A Shell is an interface that acts as the interface between kernel and user. It collects the input from the user and executes the program based on the user input and displays that output.
- For shell prompt, the Linux user should type prompt, $, i.e called as command prompt.
- In the Unix system, the following shells are available in UNIX.
- Bourne Shell
- C Shell
- The pwd directory denotes the current directory of the user.
- cd command is used the change the directory of the linux system.
- The ls command is used to display the contents of the directory.
- The cat command is used to list the contents of a file. For example, cat idol.txt.
- The mv command is used to move the files from one place to another.
- To rename a file, the Linux system also uses the mv command.
- The mkdir command is used to create a new directory. The rmdir command is used to remove an empty directory.
- The rm is used to delete directories and their contents. For example, rm –r idol. It means that the command deletes all the files and directories recursively.

3.11 ANSWERS TO CHECK YOUR PROGRESS

1) Linux is a free open-source secure community used operating system.
2) The Linux Mint OS is working as windows OS more simply and any newcomers can use this OS as like Windows OS.
3) The Debian Linux OS is very stable. It is more complex than other Linux OS.

4) Fedora Linux distribution was developed by the Fedora project, which is similar to RedHat. It is easy to use on laptop and desktop systems.

5) The Linux kernel is the core part of the Linux operating system. The kernel acts as the core interface between computer hardware and its process, manages the resources between them.

A library is a collection of pre-compiled pieces of code called functions. The library contains common functions and together, they form a package called — a library

6) a) True; b) True

7) A Shell is an interface that acts as the interface between kernel and user. It collects the input from the user and executes the program based on the user input and displays that output.

8) For shell prompt, the Linux user should type prompt, $, i.e., called as command prompt.

9) A shell script starts with #!/bin/sh

10) The following 5 are the commands in the Linux.

    a. The cat command is used to list the contents of a file. For example, cat idol.txt. The command will display the contents of the idol.txt file.

    b. The cp command is used to copy files from one directory to another directory. For example, cp idol.txt /home/username/idolfile.

    c. The mv command is used to move the files from one place to another. For example: mv idol.txt /home/username/idolfile.

    d. To rename a file, the Linux system also uses the mv command. For example, mv idol.txt idol1.txt

    e. The mkdir command is used to create a new directory. For example, mkdir idol

11) A process is an instance of a running program. When a user executes a program or executes a command in Linux, it means that the OS creates a process. The types of the Linux process are

    a. Foreground processes,

    b. Background process.

12) The system-related process which is running in the background is known as Daemon Process.

Space for learners:

3.12 POSSIBLE QUESTIONS

1. What are basic elements or components of Linux?
2. What is Kernel? Explain its functions.
3. What are two types of Linux User Mode?
4. What do you mean by a Process States in Linux?
5. What is Linux Shell? What types of Shells are there in Linux?
6. What is a Zombie Process?
7. What do you mean by Shell Script? Give example.
8. Why /etc/resolv.conf and /etc/hosts files are used?
9. Name some Linux variants.
10. Give some examples of Linux command
11. Difference between Zombie and Orphan Processes.
12. What is Linux file system? Explain the types of Linux file system.

3.13 REFERENCES & SUGGESTED READINGS

- Linux: The Complete Reference, Sixth Edition - Richard Petersen

Space for learners:

UNIT 4: PROCESS MANAGEMENT

Space for learners:

Unit Structure:

4.1 INTRODUCTION

Space for learners:

We know that a program is a set of instructions given to the computer system to do some specific task. 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 like memory and processor. 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

- Scheduling processes and threads on the CPUs.
- Creating and deleting both user and system processes.
- Suspending and resuming processes.
- Providing mechanisms for process synchronization.
- Providing mechanisms for process communication.

A process operates in either user mode or kernel mode. In user mode, a process executes application code with the machine in a non-privileged protection mode. When a process requests services from the operating system with a system call, it switches into the machine's privileged protection mode via a protected mechanism and then operates in kernel mode.

The resources used by a process are similarly split into two parts. The resources needed for execution in user mode are defined by the CPU architecture and typically include the CPU's general-purpose registers, the program counter, the processor-status register, and the stack-related registers, as well as the contents of the memory segments that constitute the FreeBSD notion of a program (the text, data, shared library, and stack segments). Kernel-mode resources include those required by the underlying hardware— such as registers, program counter, and stack pointer—and also by the state required for the FreeBSD kernel to provide system services for a process. This kernel state includes parameters to the current system call, the current process's user identity, scheduling information, and so on.

4.2 UNIT OBJECTIVES

Space for learners:

After going through this unit you will be able to:

- understand the basic concepts of process management of operating system
- know about the process and its different attributes.
- understand various states of a process
- give the basic concept of a thread and how it differ from a process
- know about concept of process scheduling concepts
- define what is virtual processor
- understand about interrupt mechanism of processes.

4.3 PROCESS

A process is basically a program in execution. The execution of a process must progress in a sequential fashion. A process is defined as an entity which represents the basic unit of work to be implemented in the system. To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program. When a program is loaded into the memory and it becomes a process, it can be divided into four sections ─ stack, heap, text and data.

Stack: The process stack contains the temporary data such as method/function parameters, return address, and local variables.

Heap: Heap is a dynamically allocated memory to a process during its runtime.

Text: Text section of a process includes the current activity represented by the value of Program Counter and the contents of the processor's registers.

Data: Data section of any process contains the global and static variables.

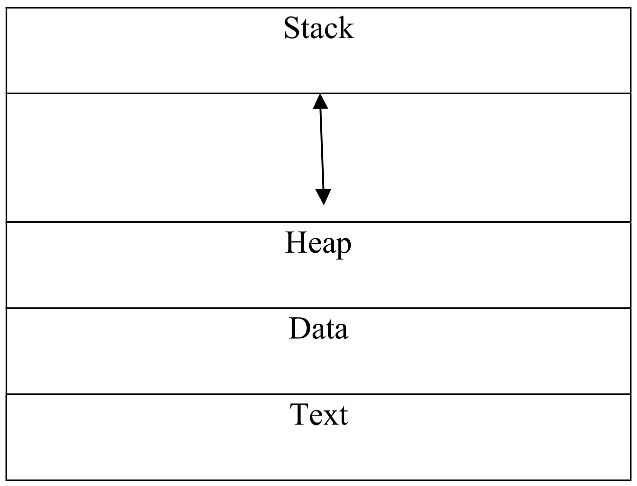


Fig.4.1. The simplified layout of a process in main memory

Space for learners:

4.1.1 Process Control Block (PCB)

A Process Control Block is a data structure maintained by the Operating System for every process. The attributes of the process are used by the Operating System to create the process control block (PCB) for each of them. This is also called context of the process. A PCB keeps all the information needed to keep track of a process with following some important attributes: Process State, Process ID(PID), Process privileges, Pointer, Program Counter, CPU registers, CPU Scheduling Information, Memory management information, Accounting information and IO status information.

Process State: The Process, from its creation to the completion, goes through various states which are new, ready, running and waiting. The current state of the process i.e., whether it is ready, running, waiting, or whatever.

Process ID: The PCB is identified by an integer process ID (PID) which is the unique identification for each of the process in the operating system.
Process privileges This is required to allow/disallow access to system resources.
Pointer A pointer to parent process.

Program Counter: Program Counter is a pointer to the address of the next instruction to be executed for this process. A program counter stores the address of the last instruction of the process on which the process was suspended. The CPU uses this address when the execution of this process is resumed.

Space for learners:

CPU registers: Various CPU registers where process need to be stored for execution for running state.
CPU Scheduling Information: Process priority and other scheduling information which is required to schedule the process.
Memory management information: This includes the information of page table, memory limits, Segment table depending on memory used by the operating system.
Accounting information: This includes the amount of CPU used for process execution, time limits, execution ID etc.
IO status information: This includes a list of I/O devices allocated to the process.

Process ID
Process states
Pointer
Program Counter
Priority
CPU Register
I/O status information
Accounting information
Etc.

Fig.4.2. The Simplified Diagram of a PCB.

The architecture of a PCB is completely dependent on Operating System and may contain different information in different operating systems. The PCB is maintained for a process throughout its lifetime, and is deleted once the process terminates.

4.4 PROCESS STATES

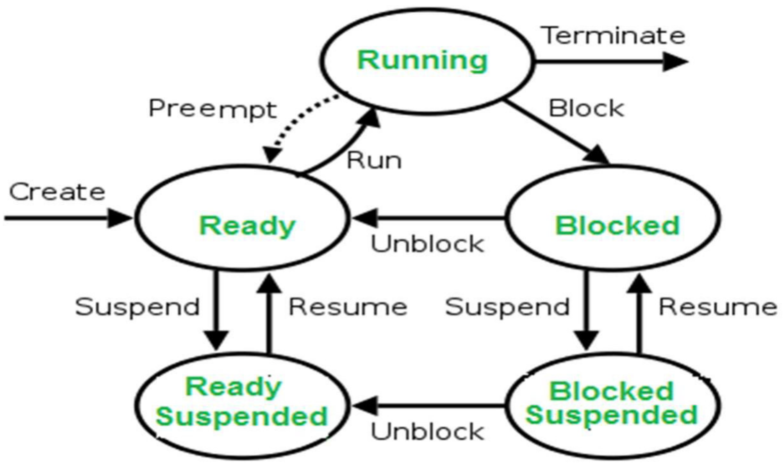


Fig.4.3. States diagram of a process

The process, from its creation to completion, passes through various states. The minimum number of states is five which are New, Ready, Running, Block or Wait, and Termination. Sometimes there are two more states namely suspend ready and suspend wait have been seen for some particular cases of process.

The names of the states are not standardized although the process may be in one of the following states during execution.

(i) **New**

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

(ii) **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.

(iii) **Running**

Space for learners:

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.

(iv**) 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 behaviour 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.

(v**) 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.

(vi) **Ready Suspended**

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. The suspend ready processes remain in the secondary memory until the main memory gets available.

(vii) **Block Suspended**

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.

Space for learners:

4.5 THREAD

A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history.

A thread shares with its peer threads few information like code segment, data segment and open files. When one thread alters a code segment memory item, all other threads see that.

A thread is also called a lightweight process. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead 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 and web server. They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors. The following figure shows the working of a single-threaded and a multithreaded process.

4.5.1. Difference Between Process and Thread

Followings are the differences between processes and threads:

(i) Process is heavy weight or resource intensive. On the other hand thread is lightweight, taking lesser resources than a process.

(ii) Process switching needs interaction with operating system but thread switching does not need to interact with operating system.

(iii) In multiple processing environments, each process executes the same code but has its own memory and file resources. But all threads can share same set of open files, child processes.

(iv) If one process is blocked, then no other process can execute until the first process is unblocked. While one thread is blocked and waiting, a second thread in the same task can run.

(v) Multiple processes without using threads use more resources. On the other hand multiple threaded processes use fewer resources.

(vi) In multiple processes each process operates independently of the others. But in case of thread, one thread can read, write or change another thread's data.

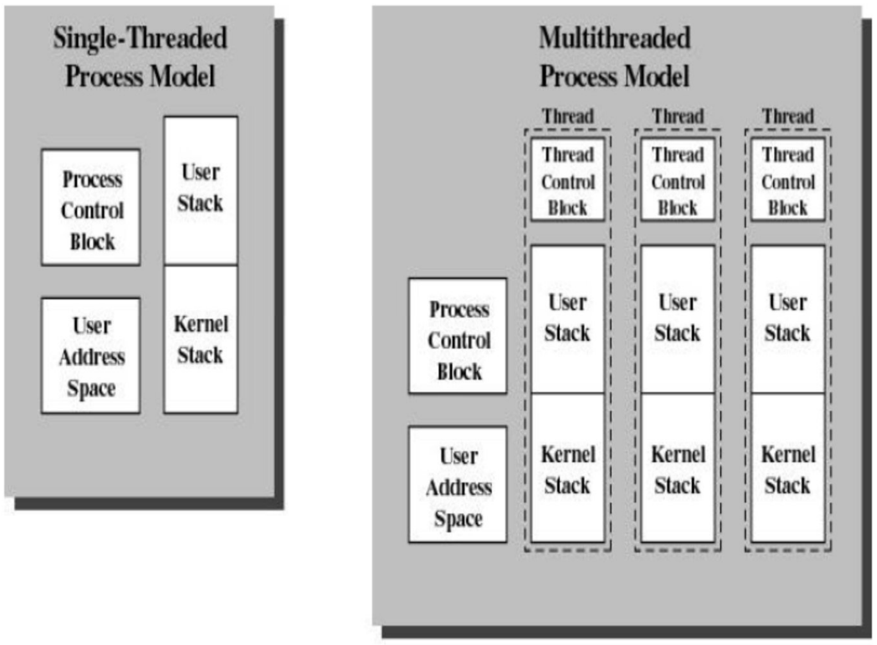


Fig.4.4. Block Diagram for the Single-Threaded and Multithreaded Process Model

4.5.2 Advantages of Thread

(i) Threads minimize the context switching time.

(ii) Use of threads provides concurrency within a process.

(iii) Threads provide efficient communication.

(iv) It is more economical to create and context switch threads.

(v) Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.

Threads are implemented in following two ways:

(i) User Level Threads – These types of threads are user managed threads.

(ii) Kernel Level Threads -- These types of threads are operating system managed threads acting on kernel, an operating system core.

4.5.3 User Level Threads

In this case, the thread management 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 starts with a single thread.

Advantages:

(i) Thread switching does not require Kernel mode privileges.

(ii) User level thread can run on any operating system.

(iii) Scheduling can be application specific in the user level thread.

(iv) User level threads are fast to create and manage.

Disadvantages:

(i) In a typical operating system, most system calls are blocking.

(ii) Multithreaded application cannot take advantage of multiprocessing.

4.5.4 Kernel Level Threads

In this case, thread management is done by the Kernel. There is no thread management code in the application area. Kernel threads are

Space for learners:

supported directly by the operating system. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.

The Kernel maintains context information for the process as a whole and for individual 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:

(i) Kernel can simultaneously schedule multiple threads from the same process on multiple processes.
(ii) If one thread in a process is blocked, the Kernel can schedule another thread of the same process.
(iii) Kernel routines themselves can be multithreaded.

Disadvantages:

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

4.6 OPERATIONS ON THE PROCESS

The following operations are done with a process:

(i) **Creation**

Once the process is created, it will be ready and come into the ready queue (main memory) and will be ready for the execution.

(ii) **Scheduling**

Out of the many processes present in the ready queue, the Operating system chooses one process and start executing it. Selecting the process which is to be executed next, is known as scheduling.

(iii) **Execution**

Once the process is scheduled for the execution, the processor starts executing it. Process may come to the blocked or wait state during the execution then in that case the processor starts executing the other processes.

(iv) **Deletion/killing**

Once the purpose of the process gets over then the OS will kill the process. The Context of the process (PCB) will be deleted and the process gets terminated by the Operating system.

4.6.1 Process Creation

Through appropriate system calls, such as fork or spawn, processes may create other processes. The process which creates other process, is termed the **parent process** of the other process, while the created sub-process is termed its **child** process.

Each process is given an integer identifier, termed as process identifier, or PID. The parent PID (PPID) is also stored for each process.

On a typical UNIX system the process scheduler is termed as sched, and is given PID 0. The first thing done by it at system start-up time is to launch init, which gives that process PID 1. Further Init launches all the system daemons and user logins, and becomes the ultimate parent of all other processes.

A child process may receive some amount of shared resources with its parent depending on system implementation. To prevent runaway children from consuming all of a certain system resource, child processes may or may not be limited to a subset of the resources originally allocated to the parent.

There are two options for the parent process after creating the child:

- Wait for the child process to terminate before proceeding. Parent process makes a wait() system call, for either a specific child process or for any particular child process, which causes the parent process to block until the wait() returns. UNIX shells normally wait for their children to complete before issuing a new prompt.

Space for learners:

- Run concurrently with the child, continuing to process without waiting. When a UNIX shell runs a process as a background task, this is the operation seen. It is also possible for the parent to run for a while, and then wait for the child later, which might occur in a sort of a parallel processing operation.

There are also two possibilities in terms of the address space of the new process:

1. The child process is a duplicate of the parent process.
2. The child process has a program loaded into it.

To illustrate these different implementations, let us consider the **UNIX** operating system. In UNIX, each process is identified by its **process identifier**, which is a unique integer. A new process is created by the **fork** system call. The new process consists of a copy of the address space of the original process. This mechanism allows the parent process to communicate easily with its child process. Both processes (the parent and the child) continue execution at the instruction after the fork system call, with one difference: The return code for the fork system call is zero for the new (child) process, whereas the(non zero) process identifier of the child is returned to the parent.

Typically, the **execlp system call** is used after the fork system call by one of the two processes to replace the process memory space with a new program. The execlp system call loads a binary file into memory - destroying the memory image of the program containing the execlp system call – and starts its execution. In this manner the two processes are able to communicate, and then to go their separate ways.

Below is a **C program** to illustrate forking a separate process using UNIX (using Ubuntu):

```
#include<stdio.h>

void main(int argc,char *argv[])

 {
```

Space for learners:

Space for learners:

```
int pid=fork(); // fork another process
if(pid<0)
  {
    fprintf(stderr, “fork failed”); \\Error occurs
    exit(-1);
   }
 If(pid==0)
    {
       execlp(“/bin/ls”,”ls”,NULL);              //child process
      }
    else
       {
         wait(NULL);                 //parent process
         printf(“Child Complete”);
         exit(0);
        }
     }
```

4.6.2 Process Termination

By making the exit (system call), typically returning an int, processes may request their own termination. This int is passed along to the parent if it is doing a wait(), and is typically zero on successful completion and some non-zero code in the event of any problem.

Processes may also be terminated by the system for a variety of reasons, including :

- The inability of the system to deliver the necessary system resources.
- In response to a kill command or other unhandled process interrupts.
- A parent may kill its children if the task assigned to them is no longer needed i.e. if the need of having a child terminates.
- If the parent exits, the system may or may not allow the child to continue without a parent (In UNIX systems, orphaned processes are generally inherited by init, which then proceeds to kill them.)

When a process ends, all of its system resources are freed up, open files flushed and closed, etc. The process termination status and execution times are returned to the parent if the parent is waiting for the child to terminate, or eventually returned to init if the process already became an orphan.

The processes which are trying to terminate but cannot do so because their parent is not waiting for them are termed **zombies**. These are eventually inherited by init as orphans and killed off.

CPU scheduling is a process that allows one process to use the CPU while the execution of another process is on hold (in waiting state) due to unavailability of any resource like I/O etc, thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast, and fair.

Whenever the CPU becomes idle, the operating system must select one of the processes in the **ready queue** to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects from among the processes in memory that are ready to execute and allocates the CPU to one of them.

Space for learners:

4.7 PROCESS SCHEDULERS

Space for learners:

Process Scheduling is an operating system task that schedules processes of different states like ready, waiting, and running. Process scheduling allows OS to allocate a time interval of CPU execution for each process. Another important reason for using a process scheduling system is that it keeps the CPU busy all the time. This allows you to get the minimum response time for programs. Operating system uses various schedulers for the process scheduling.

4.7.1 Scheduling Objectives

There are some important objectives of Process scheduling

(i) Maximize the number of interactive users within acceptable response times.
(ii) Achieve a balance between response and utilization.
(iii) Avoid indefinite postponement and enforce priorities.
(iv) It also should give reference to the processes holding the key resources.

There are three types of process schedulers we have namely (i) long term scheduler, (ii) short term scheduler and (iii) medium term scheduler.

(i**) Long term scheduler**

Long term scheduler is also known as job scheduler. It chooses the processes from the pool (secondary memory) and keeps them in the ready queue maintained in the primary memory.

Long Term scheduler mainly controls the degree of Multiprogramming. The purpose of long term scheduler is to choose a perfect mix of IO bound and CPU bound processes among the jobs present in the pool.

If the job scheduler chooses more IO bound processes then all of the jobs may reside in the blocked state all the time and the CPU will remain idle most of the time. This will reduce the degree of Multiprogramming. Therefore, the Job of long term scheduler is very critical and may affect the system for a very long time.

(ii) **Short term scheduler**

Short term scheduler is also known as CPU scheduler. It selects one of the Jobs from the ready queue and dispatch to the CPU for the execution.

A scheduling algorithm is used to select which job is going to be dispatched for the execution. The Job of the short term scheduler can be very critical in the sense that if it selects job whose CPU burst time is very high then all the jobs after that, will have to wait in the ready queue for a very long time.

This problem is called starvation which may arise if the short term scheduler makes some mistakes while selecting the job.

(iii)Medium term scheduler

Medium term scheduler takes care of the swapped out processes.If the running state processes needs some IO time for the completion then there is a need to change its state from running to waiting.

Medium term scheduler is used for this purpose. It removes the process from the running state to make room for the other processes. Such processes are the swapped out processes and this procedure is called swapping. The medium term scheduler is responsible for suspending and resuming the processes.

It reduces the degree of multiprogramming. The swapping is necessary to have a perfect mix of processes in the ready queue.

4.7.2 Difference among Schedulers Long-Term Vs. Short Term Vs. Medium-Term

Sl No	Long-Term	Short-Term	Medium-Term
1	Long term is also known as a job scheduler	Short term is also known as CPU scheduler	Medium-term is also called swapping scheduler.
2	It is either absent or minimal in a time-sharing system.	It is insignificant in the time-sharing order.	This scheduler is an element of Time-sharing systems.

Space for learners:

Sl No	Long-Term	Short-Term	Medium-Term
3	Speed of long-term schedulers is less compared to the short term scheduler.	Speed is the fastest compared to the short-term and medium-term scheduler.	It offers medium speed.
4	It allows us to select processes from the loads and pool back into the memory	It only selects processes that are in a ready state of the execution.	It helps you to send process back to memory.
5	It offers full control	It offers less control	It reduces the level of multiprogramming.

Space for learners:

4.8 PROCESS QUEUES

The Operating system manages various types of queues for each of the process states. The PCB related to the process is also stored in the queue of the same state. If the Process is moved from one state to another state then its PCB is also unlinked from the corresponding queue and added to the other state queue in which the transition is made.

There are the following process queues maintained by the Operating system.

(i) Job Queue

In starting, all the processes get stored in the job queue. It is maintained in the secondary memory. The long term scheduler (Job scheduler) picks some of the jobs and put them in the primary memory.

(i) Ready Queue

Ready queue is maintained in primary memory. The short term scheduler picks the job from the ready queue and dispatch to the CPU for the execution.

(ii) Waiting Queue

When the process needs some IO operation in order to complete its execution, OS changes the state of the process from running to waiting. The context (PCB) associated with the process gets stored on the waiting queue which will be used by the Processor when the process finishes the IO.

4.9 VARIOUS TIMES RELATED TO THE PROCESS

(i). Arrival Time

The time at which the process enters into the ready queue is called the arrival time.

(ii). Burst Time

The total amount of time required by the CPU to execute the whole process is called the Burst Time. This does not include the waiting time. It is confusing to calculate the execution time for a process even before executing it hence the scheduling problems based on the burst time cannot be implemented in reality.

(iii). Completion Time

The time at which the process enters into the completion state or the time at which the process completes its execution, is called completion time.

(iv). Turnaround time

The total amount of time spent by the process from its arrival to its completion, is called Turnaround time.

(v**). Response Time**

The difference between the arrival time and the time at which the process first gets the CPU is called Response Time.

4.10 PROCESS SCHEDULING QUEUES

Process Scheduling Queues help us to maintain a distinct queue for each and every process states and PCBs. All the process of the same

Space for learners:

execution state is placed in the same queue. Therefore, whenever the state of a process is modified, its PCB needs to be unlinked from its existing queue, which moves back to the new state queue.

Three types of operating system queues are:

I. **Job queue** – All processes, upon entering into the system, are stored in the Job Queue. It helps us to store all the processes in the system.

II. **Ready queue** – This type of queue helps us to set every process residing in the main memory, which is ready and waiting to execute. Processes in the ready state are placed in the Ready Queue.

III. **Device queues** – It is a process that is blocked because of the absence of an I/O device. Processes waiting for a device to become available are placed in Device Queues. There are unique device queues available for each I/O device.

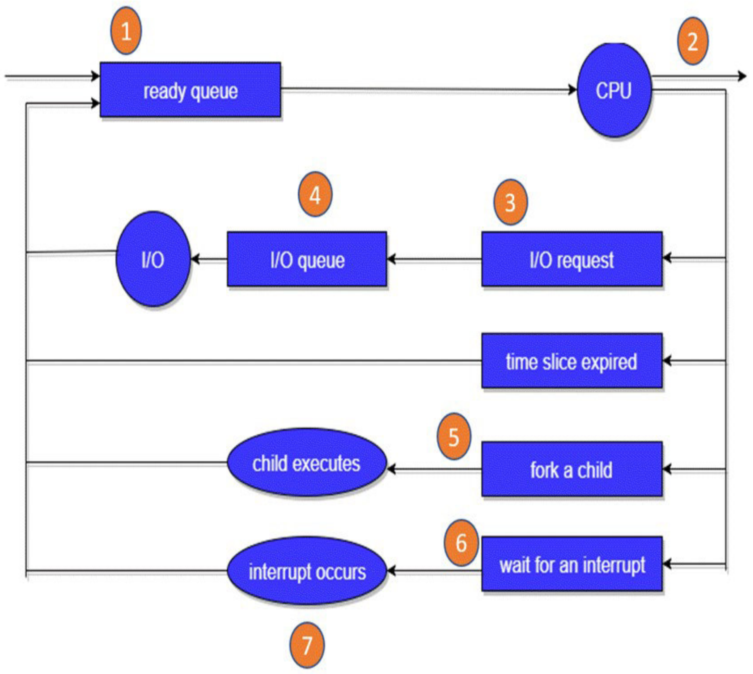


Fig.4.5. Block Diagram of Process Scheduling Queues

Here in the above-given block Diagram of process scheduling queues, we use the rectangle that represents a queue, circle denotes the resource and arrow indicates the flow of the process. Here we discuss the every step from 1 to 7 as follows:

Space for learners:

Space for learners:

1. Every new process first put in the Ready queue .It waits in the ready queue until it is finally processed for execution. Here, the new process is put in the ready queue and wait until it is selected for execution or it is dispatched.
2. One of the processes is allocated the CPU and it is executing
3. The process should issue an I/O request
4. Then, it should be placed in the I/O queue.
5. The process should create a new subprocess
6. The process should be waiting for its termination.
7. It should remove forcefully from the CPU, as a result interrupt. Once interrupt is completed, it should be sent back to ready queue.

The act of determining which process is in the ready state, and should be moved to the running state is known as Process Scheduling.

The prime aim of the process scheduling system is to keep the CPU busy all the time and to deliver minimum response time for all programs. For achieving this, the scheduler must apply appropriate rules for swapping processes IN and OUT of CPU.

4.10.1 Types of CPU Scheduling

Here we observed that CPU scheduling decisions may take place under the following four circumstances:

1. When a process switches from the running state to the waiting state(for I/O request or invocation of wait for the termination of one of the child processes).
2. When a process switches from the running state to the ready state (for example, when an interrupt occurs).
3. When a process switches from the waiting state to the ready state(for example, completion of I/O).
4. When a process terminates.

In circumstances 1 and 4, there is no choice in terms of scheduling. A new process(if one exists in the ready queue) must be selected for execution. There is a choice, however in circumstances 2 and 3.

When Scheduling takes place only under circumstances 1 and 4, we say the scheduling scheme is non-preemptive; otherwise, the scheduling scheme is preemptive.

Space for learners:

4.10.2 Non-Preemptive Scheduling

In non-preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.

It is the only method that can be used on certain hardware platforms because It does not require the special hardware needed for preemptive scheduling.

This scheduling method is used by the Microsoft Windows 3.1 and by the Apple Macintosh operating systems.

In non-preemptive scheduling, it does not interrupt a process running CPU in the middle of the execution. Instead, it waits till the process completes its CPU burst time, and then after that it can allocate the CPU to any other process.

Some Algorithms based on non-preemptive scheduling are: Shortest Job First (SJF basically non-preemptive) Scheduling and Priority (non- preemptive version) Scheduling, etc.

4.10.3 Preemptive Scheduling

In this type of process scheduling, the tasks are usually assigned with priorities. At times it is necessary to run a certain task that has a higher priority before another task although it is running. Therefore, the running task is interrupted for some time and resumed later when the priority task has finished its execution.

Thus this type of scheduling is used mainly when a process switches either from running state to ready state or from waiting state to ready state. The resources (like CPU cycles) are mainly allocated to the process for a limited amount of time and then are taken away, and after that, the process is again placed back in the ready queue in the case if that process still has a CPU burst time remaining. That process stays in the ready queue until it gets the next chance to execute.

Some Algorithms that are based on preemptive scheduling are Round Robin Scheduling (RR), Shortest Remaining Time First (SRTF), Priority (preemptive version) Scheduling, etc.

4.11 SCHEDULING CRITERIA

There are many different criteria to check the best scheduling algorithm, they are respectively:

CPU Utilization

To make out the best use of the CPU and not to waste any CPU cycle, the CPU would be working most of the time (Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)

Throughput

It is the total number of processes completed per unit of time or rather says the total amount of work done in a unit of time. This may range from 10/second to 1/hour depending on the specific processes.

Turnaround Time

It is the amount of time taken to execute a particular process, i.e. The interval from the time of submission of the process to the time of completion of the process(Wall clock time).

Waiting Time

The sum of the periods spent waiting in the ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU.

Load Average

It is the average number of processes residing in the ready queue waiting for their turn to get into the CPU.

Response Time

Amount of time it takes from when a request was submitted until the first response is produced. Remember, it is the time till the first response and not the completion of process execution (final response).

In general CPU utilization and Throughput are maximized and other factors are reduced for proper optimization.

4.12 THE CONCEPTS OF CONTEXT SWITCH

Context switch means switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process. The context of a process is represented in the Process Control Block (PCB) of a process which includes the value of the CPU registers, the process state and memory-management information. When a context switch occurs, the Kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.

Context switch time is pure overhead, because the system does no useful work while switching. Its speed varies from machine to machine, depending on the memory speed, the number of registers that must be copied, and the existence of special instructions (such as a single instruction to load or store all registers). Typical speeds range from 1 to 1000 microseconds.

4.13 INTERRUPT MECHANISM

An interrupt refers to an external event that needs immediate attention from the processor. An interrupt signals the processor, indicating the need of attention, and requires interruption of the current code the processor is executing. As a response, the processor suspends its current activities, saves its state and executes a particular function to service the event that has caused the interruption. Such function is often called an interrupt handler or an interrupt service routine. Once the processor has responded to the interrupt, i.e. after the processor has executed the interrupt handler, the processor resumes its previously saved state and resumes the execution of the same program it was executing before the interrupt occurred. The interrupts are often caused by external devices that

Space for learners:

communicate with the processor (Interrupt-driven I/O). Whenever these devices require the processor to execute a particular task, they generate interrupts and wait until the processor has acknowledged that the task has been performed. To be able to receive and respond to interrupts a processor is equipped with an interrupt port. Through the interrupt port the processor can receive the interrupt request signals and can respond to these requests through the interrupt acknowledge signals.

Space for learners:

Interrupts are important because they give the user better control over the computer. Without interrupts, a user may have to wait for a given application to have a higher priority over the CPU to be run. This ensures that the CPU will deal with the process immediately.

An interrupt is also referred to as an input signal that has the highest priority for hardware or software events that requires immediate processing of an event. During the early days of computing, the processor had to wait for the signal to process any events. The processor should check every hardware and software program to understand if there is any signal to be processed. This method would consume a number of clock cycles and makes the processor busy. Just in case, if any signal was generated, the processor would again take some time to process the event, leading to poor system performance.

A new mechanism was introduced to overcome this complicated process. In this mechanism, hardware or software will send the signal to a processor, rather than a processor checking for any signal from hardware or software. The signal alerts the processor with the highest priority and suspends the current activities by saving its present state and function, and processes the interrupt immediately, this is known as ISR. As it doesn't last long, the processor restarts normal activities as soon as it is processed. Interrupts are classified into two main types.

4.13.1 Hardware Interrupts

An electronic signal sent from an external device or hardware to communicate with the processor indicating that it requires immediate attention. For example, strokes from a keyboard or an action from a mouse invoke hardware interrupts causing the CPU to read and process it. So it arrives asynchronously and during any point of time while executing an instruction.

4.13.2 Software Interrupts

The processor itself requests a software interrupt after executing certain instructions or if particular conditions are met. These can be a specific instruction that triggers an interrupt such as subroutine calls and can be triggered unexpectedly because of program execution errors, known as exceptions or traps.

4.14 VIRTUAL PROCESSOR

A virtual processor is a representation of a physical processor core to the operating system of a logical partition that uses shared processors. This allows the operating system to calculate the number of concurrent operations that it can perform.

A virtual processor is a representation of a physical processor core to the operating system of a logical partition that uses shared processors.

When you install and run an operating system on a server that is not partitioned, the operating system calculates the number of operations that it can perform concurrently by counting the number of processors on the server. For example, if you install an operating system on a server that has eight processors, and each processor can perform two operations at a time, the operating system can perform 16 operations at a time. In the same way, when you install and run an operating system on a logical partition that uses dedicated processors, the operating system calculates the number of operations that it can perform concurrently by counting the number of dedicated processors that are assigned to the logical partition. In both cases, the operating system can easily calculate how many operations it can perform at a time by counting the whole number of processors that are available to it.

However, when you install and run an operating system on a logical partition that uses shared processors, the operating system cannot calculate a whole number of operations from the fractional number of processing units that are assigned to the logical partition. The server firmware must therefore represent the processing power available to the operating system as a whole number of processors. This allows the operating system to calculate the number of concurrent operations that it can perform. A virtual processor is a

Space for learners:

representation of a physical processor to the operating system of a logical partition that uses shared processors.

Advantages of virtual processors

- Virtual processors can share processing.
- Virtual processors save memory and resources.
- Virtual processors can perform parallel processing.
- You can start additional virtual processors and terminate active CPU virtual processors while the database server is running.

CHECK YOUR PROGRESS

Multiple Choice Questions:

1. A program in execution is called
   (A) Process **(B)** Instruction **(C)** Procedure
   (D) Function

2. Which of the following is not a fundamental process state
   (A) ready **(B)** terminated **(C)** executing
   (D) blocked

3. A scheduler which selects processes from secondary storage device is called
   (A) Short term scheduler. **(B)** Long term scheduler.
   (C) Medium term scheduler**.** **(D)** Process scheduler.

4. Program 'preemption' is
   (A) forced de allocation of the CPU from a program which is executing on the CPU.
   (B) release of CPU by the program after completing its task.
   (C) forced allotment of CPU by a program to itself.
   (D) a program terminating itself due to detection of an error.

5. Interval between the time of submission and completion of the job is called
   (A) Waiting time **(B)** Turnaround time
   (C) Throughput **(D)** Response time

4.15 SUMMING UP

- A program is a set of instructions given to the computer system to do some specific task. A program in execution is called a

Space for learners:

process. In order to accomplish its task, process needs the computer resources like memory and processor.

- A process operates in either user mode or kernel mode. In user mode, a process executes application code with the machine in a nonprivileged protection mode.
- When a process requests services from the operating system with a system call, it switches into the machine's privileged protection mode via a protected mechanism and then operates in kernel mode.
- When a program is loaded into the memory and it becomes a process, it can be divided into four sections ─ stack, heap, text and data.
- A Process Control Block is a data structure maintained by the Operating System for every process. The attributes of the process are used by the Operating System to create the process control block (PCB) for each of them. This is also called context of the process.
- The process, from its creation to completion, passes through various states. The minimum number of states is five which are New, Ready, Running, Block or Wait, and Termination.
- A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack.
- A thread is also called a lightweight process. Threads provide a way to improve application performance through parallelism.
- User Level Threads – These types of threads are user managed threads.
- Kernel Level Threads -- These types of threads are operating system managed threads acting on kernel, an operating system core.
- Through appropriate system calls, such as fork or spawn, processes may create other processes. The process which creates other process, is termed the parent process of the other process, while the created sub-process is termed its child process.

Space for learners:

- By making the exit (system call), typically returning an int, processes may request their own termination.
- Process Scheduling is an operating system task that schedules processes of different states like ready, waiting, and running. Process scheduling allows OS to allocate a time interval of CPU execution for each process.
- Long term scheduler is also known as job scheduler. It chooses the processes from the pool (secondary memory) and keeps them in the ready queue maintained in the primary memory.
- Short term scheduler is also known as CPU scheduler. It selects one of the Jobs from the ready queue and dispatch to the CPU for the execution.
- Medium term scheduler takes care of the swapped out processes. If the running state processes needs some IO time for the completion then there is a need to change its state from running to waiting.
- Process Scheduling Queues help us to maintain a distinct queue for each and every process states and PCBs.
- In non-preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.
- In premptive scheduling, the tasks are usually assigned with priorities. At times it is necessary to run a certain task that has a higher priority before another task although it is running.
- Context switch means switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process.
- An interrupt refers to an external event that needs immediate attention from the processor. An interrupt signals the processor, indicating the need of attention, and requires interruption of the current code the processor is executing.
- An electronic signal sent from an external device or hardware to communicate with the processor indicating that it requires immediate attention.
- The processor itself requests a software interrupt after executing certain instructions or if particular conditions are met.

Space for learners:

- A virtual processor is a representation of a physical processor core to the operating system of a logical partition that uses shared processors. This allows the operating system to calculate the number of concurrent operations that it can perform.

Space for learners:

4.16 ANSWERS TO CHECK YOUR PROGRESS

1(A), 2(D), 3(C), 4(A), 5(B)

4.17 POSSIBLE QUESTIONS

Short Type Questions:

1. What is process? How it differ from a program?
2. What do you mean by PCB in a process?
3. What are the different states of a process?
4. What is a thread? How it differ from a process?
5. What is process scheduler? What are its different categories?

Long Answer Type Questions:

1. Explain different attributes found in PCB in a process.
2. Explain the three types of process schedulers with its functions.
3. Explain the different criteria of process scheduling.
4. What do you mean by interrupt? Explain its different categories.

4.18 REFERENCES AND SUGGESTED READINGS

- Avi Silberschatz, Greg Gagne and Peter Baer Galvin, OPERATING SYSTEM CONCEPTS, WILEY PUBLICATION.
- William Stallings, OPERATING SYSTEMS, *INTERNALS AND DESIGN PRINCIPLES*, SEVENTH EDITION, PEARSON PUBLICATION.

UNIT 5: SYSTEM CALLS

Space for learners:

Unit Structure:

5.1 INTRODUCTION

System call is a mechanism through which user programs are offered the services of the operating system. It basically an interface between a process and the Operating System (O/S).

5.2 UNIT OBJECTIVES

After going through this unit, you will be able to:

- know the basics of system call,
- understand the different system calls in Linux O/S.,
- gain a hands-on experience using system calls.

5.3WHAT IS SYSTEM CALL?

System Calls are basically a set of extended instructions provided by the O/S for communications between the user programs and the O/S. These varies form O/S to O/S but the basic concepts are almost similar. System Calls are low level functions of O/S and are basically written in high level language C or C++.Here, in this unit we will basically discuss the System Calls of Linux O/S.

When we call a library function (suppose in C++) to perform certain task the underlying system call(s) is/are invoked and this is illustrated in Fig. 5.1.

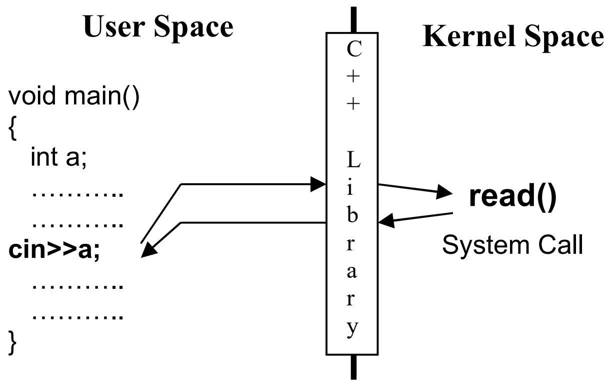


Fig. 5.1: Invocation of System Call

In the above illustration (Fig. 5.1), within the C++ program **cin** (from C++ standard library) statement is used read data. And there is a system call, **read()**, in linux for reading data from source. When the **cin** statement is executed and in turn the **read()** system call gets invoked.

System Calls are mostly used through an interface known as API (Application Programming Interface) rather than direct use. Few examples of API are:

- POSIX API for Unix, Linux, Mac OS
- Win32 API for Windows

STOP TO CONSIDER

In Linux, there are about 60 system calls and most of them are written in C language.

Traditionally, the system calls are divided in to two broad categories and they are:

- System Calls for Processes Management
- System Calls for File System Management

Let's discuss the system calls related to the above two categories.

Space for learners:

Space for learners:

5.4 SYSTEM CALLS FOR PROCESS MANAGEMENT

Before plunging into more detail, let’s understand few basic concepts/terminologies which will be very much related to our discussion taking Linux O/S as an example.

- ✓ A program in execution is termed as **Process**.
- ✓ Each process is assigned with a **Process Id**.
- ✓ A **shell**, also known as **command interpreter**, is basically a process which reads the command issued to a linux terminal.
- ✓ A process can create other processes and these are termed as **Child Process**.A child process is also assigned a process id.

Here, in this section we will discuss the few system calls related to Process Management.

5.4.1 exit() System Call

exit() system call is used to end a process. The syntax for the system call is:

void exit(status);

The status is an integer between 0 to 255 whichis returned to the parent process.It is useful when one process requires to tell its parent that how it ends. The status value ‘0’ means the process have not encountered any problem.In general, the parent of all the processes in linux is **init()** with Process Id **1**.

Program-1: Example of**exit()** system call.

```
int main()
{
        printf(“Program Ends….”);
        exit (0);          //End the Process
}
```

In the above program, the exit value is set to 0.

5.4.2 fork() System Call

fork() system call is used to create a new process. The process from which the fork() system call is invoked is termed as **Parent**

and the new process is termed as **Child**. The syntax of the fork() system call is:

pid_t fork();

The header files for **pid_t** and **fork()** are "**sys/types.h**" and "**unistd.h**" respectively. The points, which are important while working with fork system call, are mentioned below:

- ✓ fork() creates an exact duplicate copy of the original process.
- ✓ The variables, declared before the execution fork(), are also exist in child process.
- ✓ After the execution of fork(), different memory space is allocated forthe child and both of them are executed simultaneously. Thus, the operations performed by both the processes, in spite of having the same content, do not affect each other.
- ✓ The return value of fork(), inside the parent, is the process id of the child. But inside the child it is 0 (zero).
- ✓ Process ids for both the processes, parent and child, are different.

Program-2: Write a C program which creates a child process and then wait for the child to terminate.

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>

int main()
{
      pid_t  process_id;
      process_id = fork();
      if(process_id<0)
      {
          printf("Error in creating child process using fork system call…");
          exit(-1);
      }
      else if(process_id == 0)
      {
          printf("Child Process is running…");
      }
      else
      {
          printf("Parent Process is running…");
          wait();
```

Space for learners:

```
            printf("Child Process terminates…");
            exit(0);
        }
}
```

<u>Ideal Output</u><u>(if **fork()**executes successfully)</u>**:**

```
Child Process is running…

Parent Process is running…

Child Process Terminates…
```

<u>Ideal Output</u> <u>(if **fork()** does not execute successfully)</u>**:**

```
Error in creating child process using fork system call…
```

STOP TO CONSIDER
The output of a program (consisting of both parent & child) depends on the processes' switching.

Explanation:

- ✓ The code within the dotted box is actually the child process' code (if fork executes successfully).
- ✓ The other codes are for the parent process.
- ✓ The variable, process_id, declared before the fork also exists in the child process.
- ✓ When fork executes,
    - o and if successful, it returns an integer greater than 0 (zero).
    - o and if unsuccessful, it returns and integer less than 0 (zero).
- ✓ After successful execution of fork, the values of process_id inside the parent process is an integer and inside child process is 0 (zero).

Now, the Program-2 is executed and suppose the fork executes successfully and hence a child process is created.

- ✓ Considering the ideal situation after fork,
    - o the created child process start execution which displays "Child Process is running…".
    - o Process switching occurs and parent continues its execution, which displays "Parent Process is running…".

Space for learners:

- The wait() executes and parent suspended(blocks) itself until the child terminates.
- Again, process switching occurs and child gets its turn and since there is no other statements to execute, the child exit.
- After receiving the termination signal by the child, the parent resumes and displays "Child Process terminates..." and then exit() function gets executed and parent is terminated.

Now, the Program-2 is executed and suppose the fork executes unsuccessfully and will display **"Error in creating child process using fork system call..."**and the process gets terminated due to the execution of the exit() function call.

5.4.3 wait() System Call

The **wait()** is a very important system call. As already mentioned in the above explanation. It make the parent process to wait for a process (child) to be terminated created by **fork()** system call.

The syntax of the wait() system call is:

pid_t wait(int *status);

If we call **wait()** inside a program without a child then it returns **-1**. But if the process has a child, it will wait for the child to exit and when it happens it will return the child's process id.

The argument, **status** is optional. This is a pointer to the integer where the unix/linux stores the value returned by the child process.

Program-3: Write a C program which creates a child. The child calculates the summation of all the even nos. between 1 and 100 and then displays it. The parent should wait till the child exit.

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>

int main()
{
      pid_t  process_id;
      int i, sum;
      process_id = fork();
      if (process_id<0)
```

Space for learners:

```
        {
            printf("Error in creating child process using fork system call…");
            exit(-1);
        }
        else if (process_id == 0)
        {
            for(i=1, sum=0; i<=100; i++)
            {
                if((i%2) == 0)
                    sum = sum + i;
             }
        printf("The summation = %d", sum);
        }
        else
        {
            printf("Parent Process is running…");
            wait();
            printf("Child Process terminates…");
            exit(0);
        }
}
```

<u>Ideal Output</u> <u>(if **fork()** executes successfully)</u>**:**

```
Parent Process is running…
The summation = 2550
Child Process Terminates…
```

But suppose, a process has more than one child then how will the **wait()** work???In this kind of situation, when **wait()** executes, it will wait for any of the child processes to exit. Thus, when one of the child processes exits the wait ends. And if this is so then what will happen to the remaining child processes as the parent itself dies after the termination of one of its childs??? In this kind of situation, the remaining child processes, termedas Orphan Processes, becomes the child of the **init** process (**process ID 1**).

5.4.4 System Calls for Process Identification: getpid(), getppid()

The getpid() and getppid() system calls are used to get the process ids.As we all know that Processes create Processes and thus every process has their Process Id as well as their Parent Process Id. getpid() system call is used to get the process id of the current

Space for learners:

process and getppid() system call is used to get the process id of the parent process of the currently running process.

Program-4: Write a C program which creates a child. Within the child process itself print the process id of the child and its parent.

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>

int main()
{
      pid_t  process_id, c_pid, p_pid;
      process_id = fork();
      if (process_id<0)
      {
          printf("Error in creating child process using fork system call…");
          exit(-1);
      }
      else if (process_id == 0)
      {
           c_pid = getpid();
           p_pid = getppid();
           printf("Child process id = %u", c_pid);
           printf("\nParent process id = %u", p_pid);
      }
      else
      {
          printf("Parent Process is running…");
          wait();
          printf("Child Process terminates…");
          exit(0);
      }
}
```

5.4.5 The exec System Call

Basically, exec is family of system calls related to process execution. These are basically used to run system commands as separate processes. The library file for these system calls is **unistd.h**.

The system calls fall under the family are:

int execl (const char *path, const char *arg, …, NULL**);**

int execlp (const char *file, const char *arg, …, NULL**);**

int execv (const char *path, char *const argv[]**);**

int execvp (const char *file, char *const argv[]**);**

int execle (const char *path, const char *arg, …, NULL, char * const envp[]**);**

int execve (const char *file, char *const argv[], char *const envp[]**);**

Let's discuss few of the above system calls.

Space for learners:

- **"execl" System Call**

This system call takes the path of the executable file as the 1st and 2nd argument. The arguments that follow the 1st two are also related to the task. And the last argument should be **NULL**. It will return -1 if any error occurs but otherwise will return nothing. For example:

```
execl("/bin/ls","/bin/ls","-al","/idol", NULL);
```

When the above code executes, a detailed list of files and directories under the directory "/idol" will be displayed.

- **"execlp" System Call**

This system call is almost like "execl" except it takes only the name of the executable file since it uses the PATH environment variable to get the path of the executable. The arguments that follow the 1st two are also related to the task. And the last argument should be **NULL**. For example:

```
execl ("ls","ls", "-al", "/idol", NULL);
```

When the above code executes, a detailed list of files and directories under the directory "/idol" will be displayed.

- **"execv" System Call**

This system call takes only two arguments. 1st argument is path of the executable file and the 2nd argument is a list of parameters terminated by NULL. For example:

```
char *arg[] = {"/bin/ls", "-al", "/idol", NULL"};
execv("/bin/ls",arg);
```

The output of the above code will be the same as above.

- **"execvp" System Call**

Space for learners:

The arguments to this system call are same as "execv" but we need to mention only the name of the executable file not the whole path as it uses the PATH environment variable. For example:

```
char *arg[ ] = {"ls", "-al", "/idol", NULL"};
execvp("ls",arg);
```

The output of the above code will be the same as above.

5.5 SYSTEM CALLS FOR FILE MANAGEMENT

These system calls are used for handling the tasks like creating a file/directory, opening a file, reading a file, writing to a file etc. The header files necessary to include are - sys/types.h, sys/stat.h, fcntl.h and unistd.h.

5.5.1 Open System Call

This system call is used to open or creating a file. The syntax is:

```
int open(const char *path, int flags,... /* mode_t mod */);
```

This will return a filed descriptor or will return -1 if fails. The 1st argument is the path of the file to be opened. 2nd argument takes how the file is to be opened such as read-only, write-only etc. These flags are as follows:

O_RDONLY: means Open for reading only,
O_WRONLY: means Open for writing only,
O_RDWR: means Open for both reading and writing.
O_APPEND: means Open and writing will from the end of the file.
O_CREAT: means if file does not exist then Create and then Open.

These flags are defined in **fcntl.h** header file. The 3rd argument is necessary while creating a new file. When file is opened a file pointer is placed at 1st byte of the file except while opening with **O_APPEND** flag where the file pointer is place at the end of the file.

5.5.2 Creat System Call

This system call is used to create a new file. The syntax is:

int creat (const char *path, mode_t mod**);**

This will return a filed descriptor or will return -1 if fails. The 1st argument, path, indicates the name of the file and the 2nd argument, mod, indicates the file access rights.

5.5.3 Read System Call

Using this system call we can read data (no. of bytes) starting from the current position, pointed by the file pointer, in a file. The syntax is:

ssize_t read(int fd, void* buf, size_t noct**);**

This will return no. of bytes read or 0 for EOF (End of File) or -1 if error occurs. The 1st argument, fd, is the File Descriptor of the file to be read. 2nd argument, buf, is the buffer (storage) where the data after the read will be stored and 3rd argument, noct, is the no. of bytes to be read from the file.

5.5.4 Write System Call

Using this system call we can write data (no. of bytes) at the current position, pointed by the file pointer, in to a file. The syntax is:

ssize_t write (int fd, const void* buf, size_t noct**);**

This will return no. of bytes written or -1 if error occurs. The 1st argument, fd, is the File Descriptor of the file where data are to be written. 2nd argument, buf, is the buffer (storage) where the data after the read will be stored and 3rd argument, noct, is the no. of bytes to be written to the file.

5.5.5 Close System Call

This system call is used to close an opened file. The syntax is:

int close (int fd**);**

The only argument to this system call is the descriptor of the file which need to be closed. This returns 0 if successful or -1 if error occurs and also frees the assigned file descriptor.

Space for learners:

5.5.6 lseek System Call

When reading/writing is to be done from/to a particular position in an opened file, lseek system call should be used. In short, it is used to position the file pointer. The syntax of this system call is:

off_t lseek (int fd, off_t offset, int ref**);**

This returns the current position of the file pointer or -1 if error occurs. The file pointer positioning will be performed based on the 3rd argument, ref, which should be one from the following values:

SEEK_SET: positioning relative to the Beginning-of-File (BOF),

SEEK_CUR: positioning relative to the current file pointer position,

SEEK_END: positioning relative to the End-of-File (EOF).

CHECK YOUR PROGRESS

1. What is System Call?
2. When a **cin** statement executed, what system call will be invoked?
3. What is a Process?
4. What is Shell?
5. Write down the syntax of the fork() system call.

State TRUE or FALSE:

6. fork() system call is defined inside the unistd.h header file.
7. Win32 API is for Windows.
8. exit() system call is used to end a process.
9. fork() creates an exact duplicate copy of the original process.
10. The getpid() system call is used to get the process id of the parent process of the current process.

Space for learners:

5.6 SUMMING UP

Space for learners:

- System Calls are basically a set of extended instructions provided by the O/S for communications between the user programs and the O/S.
- System Calls are mostly used through an interface known as API (Application Programming Interface) rather than direct use. For example POSIX, Win32 etc.
- A program in execution is termed as Process and each process is assigned with a Process Id.
- A process can create other processes and these are termed as Child Process.
- exit() system call is used to end a process.
- The parent of all the processes in linux is init() with Process Id 1.
- fork() system call is used to create a new process. It creates an exact duplicate copy of the original process.
- wait() system call makes the parent process to wait for a process (child) to be terminated created by fork() system call.
- getpid() system call is used to get the process id of the current process and getppid() system call is used to get the process id of the parent process of the currently running process.
- The exec is family of system calls related to process execution. They are – execl, execlp, execv, execp, execle, execve.

5.7ANSWERS TO CHECK YOUR PROGRESS

1. System Calls are basically a set of extended instructions provided by the O/S for communications between the user programs and the O/S.

2. When the cin statement is executed and in turn the read() system call gets invoked.

3. A program in execution is termed as Process.

4. A shell, also known as command interpreter, is basically a process which reads the command issued to a linux terminal.

5. The syntax of the fork() system call is:

```
pid_t  fork();
```

6. True

7. False

8. True

9. True

10. False

5.8 POSSIBLE QUESTIONS

1. What is a system call? How is it invoked indirectly? Explain.
2. What is API?
3. Write down the basic difference between a program and a process.
4. Write short notes on:
    a. exit() system call
    b. fork() system call
    c. wait() system call
5. Write a program in C to create a child process which will calculate the length the string "GUIDOL" and displays it. The parent should only terminate when the child completes its task.
6. Discuss the OPEN system call.
7. Write a program in C to illustrate the use of creat, open, read, write and close system calls.

5.9REFERENCES & SUGGESTED READINGS

- Tanenbaum, A.S., BOS, H., *Modern Operating Systems*, PEARSON Publications.

Space for learners:

UNIT 6: PROCESS SCHEDULING ALGORITHMS-I

Space for learners:

Unit Structure:

6.1 INTRODUCTION

In this unit you will learn the basic concept of process scheduling which is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy. Process scheduling is an essential part of Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

The unit will also familiarize you with key terms related to process scheduling like turn-around time, burst time, waiting time etc. You will also learn that scheduling algorithms are divided in to two categories: preemptive and non-preemptive. The unit will thoroughly discuss some important scheduling algorithms like: FCFS, SJF, SRTF etc.

6.2 UNIT OBJECTIVES

After going through this unit, you will be able to:

- understand about CPU scheduling
- know various types of CPU Scheduling
- identify the important CPU scheduling Terminologies
- define CPU Scheduling Criteria
- understand various types of CPU scheduling Algorithm

In a system, there are a number of processes that are present in different states at a particular time. Some processes may be in the waiting state, others may be in the running state and so on. Have you ever thought how CPU selects one process out of some many processes for execution? Yes, you got it right. CPU uses some kind of process scheduling algorithms to select one process for its execution amongst so many processes. The process scheduling algorithms are used to maximize CPU utilization by increasing throughput. In this unit, we will learn about various process scheduling algorithms used by CPU to schedule a process.

6.3 CPU SCHEDULING?

CPU Scheduling is a process of determining which process will own CPU for execution while another process is on hold. The main task of CPU scheduling is to make sure that whenever the CPU remains idle, the OS at least select one of the processes available in the ready queue for execution. The selection process will be carried out by the CPU scheduler. It selects one of the processes in memory that are ready for execution.

6.4 PROCESS SCHEDULING QUEUES

Space for learners:

Process Scheduling Queues help you to maintain a distinct queue for each and every process states and PCBs. All the processes of the same execution state are placed in the same queue. Therefore, whenever the state of a process is modified, its PCB needs to be unlinked from its existing queue, which moves back to the new state queue.

Three types of operating system queues are:

1. **Job queue** – It helps you to store all the processes in the system.
2. **Ready queue** – This type of queue helps you to set every process residing in the main memory, which is ready and waiting to execute.
3. **Device queues** – It is a process that is blocked because of the absence of an I/O device.

6.5 TWO STATE PROCESS MODEL

Two-state process models are:

Running

In the Operating system, whenever a new process is built, it is entered into the system, which should be running.

Not Running

The processes that are not running are kept in a queue, which is waiting for their turn to execute. Each entry in the queue is a point to a specific process.

6.6 TYPE OF PROCESS SCHEDULERS

A scheduler is a type of system software that allows you to handle process scheduling.

There are mainly three types of Process Schedulers:

1. Long Term
2. Short Term
3. Medium Term

Long Term Scheduler

Long term scheduler is also known as a **job scheduler**. This scheduler regulates the program and selects process from the queue and loads them into memory for execution. It also regulates the degree of multi-programing.

However, the main goal of this type of scheduler is to offer a balanced mix of jobs, like processor, I/O jobs that allow managing multiprogramming.

Medium Term Scheduler

Medium-term scheduling is an important part of **swapping**. It enables you to handle the swapped out-processes. In this scheduler, a running process can become suspended, which makes an I/O request.

Short Term Scheduler

Short term scheduling is also known as **CPU scheduler**. The main goal of this scheduler is to boost the system performance according to set criteria. This helps you to select from a group of processes that are ready to execute and allocates CPU to one of them.

6.7 SCHEDULING ALGORITHMS

A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms. There are six popular process scheduling algorithms which we are going to discuss in this unit:

- First-Come, First-Served (FCFS) Scheduling
- Shortest-Job-Next (SJN) Scheduling
- Priority Scheduling
- Shortest Remaining Time
- Round Robin(RR) Scheduling
- Multiple-Level Queues Scheduling

These algorithms are either **non-preemptive or preemptive**. Non-preemptive algorithms are designed so that once a process enters the running state; it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

Preemptive Scheduling is a CPU scheduling technique that works by dividing time slots of CPU to a given process. The time slot given might be able to complete the whole process or might not be able to it. When the burst time of the process is greater than CPU cycle, it is placed back into the ready queue and will execute in the next chance. This scheduling is used when the process switch to ready state.

Algorithms that are backed by preemptive scheduling are round-robin (RR), priority, SRTF (Shortest Remaining Time First).

Non-preemptive Scheduling is a CPU scheduling technique the process takes the resource (CPU time) and holds it till the process gets terminated or is pushed to the waiting state. No process is interrupted until it is completed, and after that processor switches to another process.

Algorithms that are based on non-preemptive Scheduling are non-preemptive priority and Shortest Job First.

Preemptive Vs Non-Preemptive Scheduling

Preemptive Scheduling	Non-Preemptive Scheduling
Resources are allocated according to the cycles for a limited time.	Resources are used and then held by the process until it gets terminated.
The process can be interrupted, even before the completion.	The process is not interrupted until its life cycle is complete.
Starvation may be caused, due to the insertion of priority process in the queue.	Starvation can occur when a process with large burst time occupies the system.
Maintaining queue and remaining time needs storage overhead.	No such overheads are required.

6.7.1 When Scheduling is Preemptive or Non-Preemptive

Space for learners:

To determine if scheduling is preemptive or non-preemptive, consider these four parameters:

1. A process switches from the running to the waiting state.
2. Specific process switches from the running state to the ready state.
3. Specific process switches from the waiting state to the ready state.
4. Process finished its execution and terminated.

Only conditions 1 and 4 apply, the scheduling is called non-preemptive.

All others scheduling are preemptive**.**

6.7.2 Important CPU scheduling Terminologies

Various times related to process are

1. Arrival time
2. Waiting time
3. Response time
4. Burst time
5. Completion time
6. Turn Around Time
7. Gant Chart

1) Arrival Time (AT)

The time when the process arrives into the running state is called as the Arrival time of the process. In simple words, the time at which any process enters the CPU is known as the arrival time.

2) Waiting Time (WT)

It is the time for which a process waits for going into the running state. It is the sum of the time spent by the process in the ready state and the waiting state. Another way of calculating it is as follows:

Waiting Time= Turn Around Time – Burst Time
WT = TAT – BT

3) Response Time

The time difference between the first time a process goes into the running state and the arrival time of the process is called the response time of the process.

4) Burst Time (BT)

The time for which the process needs to be in the running state is known as the burst time of the process. We can also define it as the time which a process requires for execution is the Burst time of the process.

5) Completion Time (CT)

The time when the Process is done with all its execution and it enters the termination state is called as the completion time of the process. It can be also defined as the time when a process ends.

6) Turnaround Time (TAT)

Turn Around time can be defined as the total time the process remains in the main memory of the system. The Ready state, waiting for state and the Running State, together make up the main memory of the system. So, the time for which the process remains in these states is known as the turnaround time of the process. In simple words, it is the time that a process spends after entering the ready state and before entering the termination state.

It can be calculated as follows:

Turn Around Time = Completion Time – Arrival Time

TAT = CT - AT

7) Gant Chart

The Gant chart is used to represent the currently executing process at every single unit of time. This time unit is the smallest unit of time in the processor.

6.7.3 CPU Scheduling Criteria

A CPU scheduling algorithm tries to maximize and minimize the following:

Space for learners:

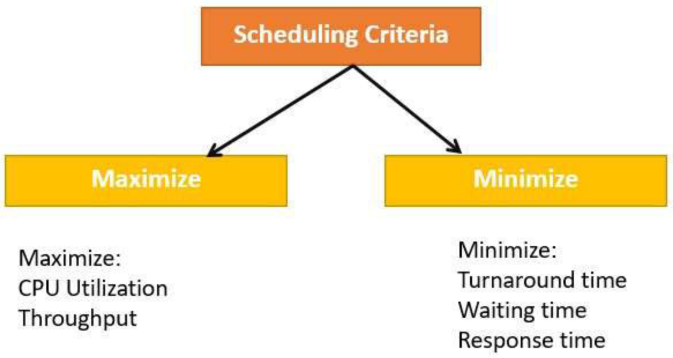


Fig 6.1 Scheduling Criteria

Maximize:

CPU utilization: CPU utilization is the main task in which the operating system needs to make sure that CPU remains as busy as possible. It can range from 0 to 100 percent. However, for the RTOS, it can be range from 40 percent for low-level and 90 percent for the high-level system.

Throughput: The number of processes that finish their execution per unit time is known Throughput. So, when the CPU is busy executing the process, at that time, work is being done, and the work completed per unit time is called Throughput.

Minimize:

Waiting time: Waiting time is an amount that specific process needs to wait in the ready queue.

Response time: It is an amount to time in which the request was submitted until the first response is produced.

Turnaround Time: Turnaround time is an amount of time to execute a specific process. It is the calculation of the total time spent waiting to get into the memory, waiting in the queue and, executing on the CPU. The period between the time of process submission to the completion time is the turnaround time.

6.7.4 First Come First Serve (FCFS)

First Come First Serve (FCFS) is an operating system scheduling algorithm that automatically executes queued requests and processes in order of their arrival. It is the easiest and simplest CPU scheduling algorithm. In this type of algorithm, processes which request the CPU first get the CPU allocation first. This is managed with a FIFO queue. The full form of FCFS is First Come First Serve.

Space for learners:

As the process enters the ready queue, its PCB (Process Control Block) is linked with the tail of the queue and, when the CPU becomes free, it should be assigned to the process at the beginning of the queue. It supports both non-preemptive and pre-emptive scheduling algorithm.

Space for learners:

Example of FCFS scheduling

A real-life example of the FCFS method is buying a movie ticket on the ticket counter. In this scheduling algorithm, a person is served according to the queue manner. The person who arrives first in the queue first buys the ticket and then the next one. This will continue until the last person in the queue purchases the ticket. Using this algorithm, the CPU process works in a similar manner.

Advantages-

- It is simple and easy to understand.
- It can be easily implemented using queue data structure.
- It does not lead to starvation.

Disadvantages-

- It does not consider the priority or burst time of the processes.
- It suffers from convoy effect.

How FCFS Works? Calculating Average Waiting Time

Problem-01: Consider the set of 5 processes whose arrival time and burst time are given below-

Table 6.1

Process Id	Arrival time	Burst time
P1	3	4
P2	5	3
P3	0	2
P4	5	1
P5	4	3

If the CPU scheduling policy is FCFS, calculate the average waiting time and average turnaround time.

Solution- Here, the Gantt Chart-

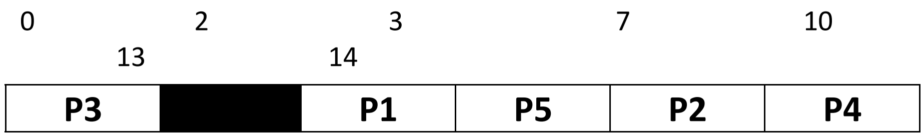


Fig 6.2 Gannt chart

Here, black box represents the idle time of CPU.
Now, we know that-
Turn Around time = Exit time – Arrival time
Waiting time = Turn Around time – Burst time

Table 6.2

Process Id	Exit time	Turn Around time	Waiting time
P1	7	7 – 3 = 4	4 – 4 = 0
P2	13	13 – 5 = 8	8 – 3 = 5
P3	2	2 – 0 = 2	2 – 2 = 0
P4	14	14 – 5 = 9	9 – 1 = 8
P5	10	10 – 4 = 6	6 – 3 = 3

Average Turn Around time = (4 + 8 + 2 + 9 + 6) / 5 = 29 / 5 = 5.8 unit

Average waiting time = (0 + 5 + 0 + 8 + 3) / 5 = 16 / 5 = 3.2 unit

6.7.5 Shortest Job Next (SJN) or Shortest Job First (SJF)

Shortest Job First (SJF) is an algorithm in which the process having the smallest execution time is chosen for the next execution. This scheduling method can be preemptive or non-preemptive. It significantly reduces the average waiting time for other processes awaiting execution. The full form of SJF is Shortest Job First.

There are basically two types of SJF methods:Non-Preemptive SJF and Preemptive SJF.

Characteristics of SJF Scheduling

Space for learners:

- It is associated with each job as a unit of time to complete.
- This algorithm method is helpful for batch-type processing, where waiting for jobs to complete is not critical.
- It can improve process throughput by making sure that shorter jobs are executed first, hence possibly have a short turnaround time.
- It improves job output by offering shorter jobs, which should be executed first, which mostly have a shorter turnaround time.

Advantages-

Preemtive-SJF is optimal and guarantees the minimum average waiting time.

- It provides a standard for other algorithms since no other algorithm performs better than it.

Disadvantages-

It cannot be implemented practically since burst time of the processes cannot be known in advance.

- It leads to starvation for processes with larger burst time.
- Priorities cannot be set for the processes.
- Processes with larger burst time have poor response time.

6.7.5.1.Non-Preemptive SJF

In non-preemptive SJF scheduling, once the CPU cycle is allocated to process, the process holds it till it reaches a waiting state or terminated.

Consider the following five processes each having its own unique burst time and arrival time.

Table 6.3

Process Queue	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

__Space for learners:__

Step 0) At time=0, P4 arrives and starts execution.

Step 1) At time= 1, Process P3 arrives. But, P4 still needs 2 execution units to complete. It will continue execution.

Step 2) At time =2, process P1 arrives and is added to the waiting queue. P4 will continue execution.

Step 3) At time = 3, process P4 will finish its execution. The burst time of P3 and P1 is compared. Process P1 is executed because its burst time is less compared to P3.

Step 4) At time = 4, process P5 arrives and is added to the waiting queue. P1 will continue execution.

Step 5) At time = 5, process P2 arrives and is added to the waiting queue. P1 will continue execution.

Step 6) At time = 9, process P1 will finish its execution. The burst time of P3, P5, and P2 is compared. Process P2 is executed because its burst time is the lowest.

Step 7) At time=10, P2 is executing and P3 and P5 are in the waiting queue.

Step 8) At time = 11, process P2 will finish its execution. The burst time of P3 and P5 is compared. Process P5 is executed because its burst time is lower.

Step 9) At time = 15, process P5 will finish its execution.

Step 10) At time = 23, process P3 will finish its execution.

Step 11) Let's calculate the average waiting time for above example.

Wait time of,

P4= 0-0=0

P1= 3-2=1

P2= 9-5=4

P5= 11-4=7

P3= 15-1=14

Average Waiting Time= 0+1+4+7+14/5 = 26/5 = 5.2

6.7.5.2 Preemptive SJF

In Preemptive SJF Scheduling, jobs are put into the ready queue as they come. A process with shortest burst time begins execution. Even, ifa process with a shorter burst time arrives, the current process is removed or preempted from execution, and the shorter job is allocated CPU cycle**.**

Consider the table 6.3 with the five processes.

Step 0) At time=0, P4 arrives and starts execution.

Step 1) At time= 1, Process P3 arrives. But, P4 has a shorter burst time. It will continue execution.

Step 2) At time = 2, process P1 arrives with burst time = 6. The burst time is more than that of P4. Hence, P4 will continue execution.

Step 3) At time = 3, process P4 will finish its execution. The burst time of P3 and P1 is compared. Process P1 is executed because its burst time is lower.

Step 4) At time = 4, process P5 will arrive. The burst time of P3, P5, and P1 is compared. Process P5 is executed because its burst time is lowest. Process P1 is preempted.

Step 5) At time = 5, process P2 will arrive. The burst time of P1, P2, P3, and P5 is compared. Process P2 is executed because its burst time is least. Process P5 is preempted.

Step 6) At time =6, P2 is executing.

Step 7) At time =7, P2 finishes its execution. The burst time of P1, P3, and P5 is compared. Process P5 is executed because its burst time is lesser.

Step 8) At time =10, P5 will finish its execution. The burst time of P1 and P3 is compared. Process P1 is executed because its burst time is less.

Space for learners:

Step 9) At time =15, P1 finishes its execution. P3 is the only process left. It will start execution.

Step 10) At time =23, P3 finishes its execution.

Step 11) Let's calculate the average waiting time for above example.

Wait time
P4= 0-0=0
P1= (3-2) + 6 =7
P2= 5-5 = 0
P5= 4-4+2 =2
P3= 15-1 = 14
Average Waiting Time = 0+7+0+2+14/5 = 23/5 =4.6

6.7.6. Shortest Remaining Time First (SRTF)

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.

Advantages:
SRTF algorithm makes the processing of the jobs faster than SJN algorithm.

Disadvantages:
The context switch is done a lot more times in SRTF than in SJN, and consumes CPU’s valuable time for processing.

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

Space for learners:

Table 6.4

Process	Burst Time	Arrival Time
P1	7	0
P2	3	1
P3	4	3

The Gantt Chart for SRTF will be:

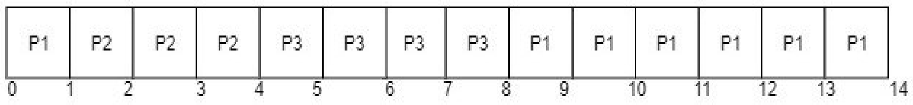


Fig 6.2*Gantt Chart*

Explanation

- At the 0th unit of the CPU, there is only one process that is P1, so P1 gets executed for the 1 time unit.
- At the 1st unit of the CPU, Process P2 arrives. Now, the P1 needs 6 more units more to be executed, and the P2 needs only 3 units. So, P2 is executed first by preempting P1.
- At the 3rd unit of time, the process P3 arrives, and the burst time of P3 is 4 units which is more than the completion time of P2 that is 1 unit, so P2 continues its execution.
- Now after the completion of P2, the burst time of P3 is 4 units that mean it needs only 4 units for completion while P1 needs 6 units for completion.
- So, this algorithm picks P3 above P1 due to the reason that the completion time of P3 is less than that of P1
- P3 gets completed at time unit 8, there are no new process arrived.
- So again, P1 is sent for execution, and it gets completed at the 14th unit.

As Arrival Time and Burst time for three processes P1, P2, P3 are given in the above diagram. Let us calculate Turnaround time, completion time, and waiting time.

Space for learners:

Space for learners:

Table 6.5

Process	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
P1	0	7	14	14-0=14	14-7=7
P2	1	3	5	5-1=4	4-3=1
P3	3	4	8	8-3=5	5-4=1

Average waiting time is calculated by adding the waiting time of all processes and then dividing them by no. of processes.

average waiting time = waiting for time of all processes/ no.of processes

average waiting time=7+1+1=9/3 = **3ms**

CHECK YOUR PROGRESS

A. Multiple Choice Questions:

1. Which of the following scheduling algorithm is non-preemtive?
   a) SJF
   b) FCFS
   c) SRTF
   d) none of the mentioned

2. The processes that are residing in main memory and are ready and waiting to execute are kept on a list called _____________
   a) job queue
   b) ready queue
   c) execution queue
   d) process queue

3. The interval from the time of submission of a process to the time of completion is termed as ____________
   a) waiting time
   b) turnaround time
   c) response time
   d) throughput

4. Which scheduling algorithm allocates the CPU first to the process that requests the CPU first?
a) first-come, first-served scheduling
b) shortest job scheduling
c) priority scheduling
d) none of the mentioned

5. Scheduling algorithms that work on complex:
a). uses few resources
b). uses most resources
c). are suitable for large computers
d). all of the mentioned

6. Scheduling algorithm which allocates the CPU first to the process which requests the CPU first?
a). FCFS scheduling
b). priority scheduling
c). shortest job scheduling
d). none of the mentioned

7. In an operating system, the portion of the process scheduler that forward processes is concerned with :
a). running processes are assigning to blocked queue
b). ready processes are assigning to CPU
c). ready processes are assigning to the waiting queue
d). all of the mentioned

8. CPU performance is measured through ________.

a. Throughput

b. MHz

c. Flaps

d. None of the above

9. FCFS maintains a __________

a. Queue

b. Stack

c. Tree

d. List

10. Full form of FCFS is-

a). First Come First Save

<u>Space for learners:</u>

b). Frequently Come First Save

c). First Come First Serve

d). First Come Final Serve

B. Fill in the Blanks:

1. Waiting Time=Turn Around Time - _________________.
2. _________ Chart is used to represent the currently executing process at every single unit of time.
3. Turn Around Time= Completion Time- ______________.
4. The Time difference between the first time a process goes into the running state and arrival time of the process is called __________________.
5. The OS maintains all PCBs in process scheduling _____________.
6. __________ scheduler determines which programs are admitted to the system for processing.
7. ___________ scheduling method can be managed with a FIFO queue.
8. ___________ is sometimes called SRTF scheduling.
9. __________ is the full of SJF algorithm.
10. __________ method selects the process with the shortest execution time for execution next.

C. State whether TRUE or FALSE

1. CPU scheduling is a process of determining which process will own CPU for execution while another process is on hold.

2. In Preemptive Scheduling, the tasks are mostly assigned with their shortest.

3. In the Non-preemptive scheduling method, the CPU has been allocated to a specific process.

4. Burst time is a time required for the process to wait.

5. CPU utilization is the main task in which the operating system needs to make sure that CPU remains as busy as possible.

6. The number of processes that finish their execution per unit time is known scheduler.

Space for learners:

7. Waiting time is an amount that specific process needs to wait in the ready queue.
8. Waiting time is an amount to complete the execution.
9. Turnaround time is an amount of time to execute a specific process.
10. The CPU uses scheduling not to improve its efficiency,

D. Match Column A with Column B

	Column A		**Column B**
1.	CPU performance is measured through	A.	First Come First Serve
2.	amount of time to execute a specific process	B.	Shortest job next
3.	SNF	C.	turnaround time
4.	SJF	D.	Shortest remaining time first
5.	FCFS	E.	preemtive
6.	can be managed with a FIFO queue	F.	Grantt chart
7.	smallest unit of time in the processor	G.	troughput
8.	______ method is the simplest and Easy to understand	H.	Shortest job first
9.	SRTF	I.	Non-preemtive
10.	No such overheads are required in ________ scheduling	J.	Waiting time

6.8 SUMMING UP

- CPU scheduling is a process of determining which process will own CPU for execution while another process is on hold.
- In Preemptive Scheduling, the tasks are mostly assigned with their priorities.
- In the Non-preemptive scheduling method, the CPU has been allocated to a specific process.
- Burst time is a time required for the process to complete execution. It is also called running time.

Space for learners:

- CPU utilization is the main task in which the operating system needs to make sure that CPU remains as busy as possible
- The number of processes that finish their execution per unit time is known Throughput.
- Waiting time is an amount that specific process needs to wait in the ready queue.
- It is an amount to time in which the request was submitted until the first response is produced.
- Turnaround time is an amount of time to execute a specific process.
- Timer interruption is a method that is closely related to preemption,
- A dispatcher is a module that provides control of the CPU to the process.
- Some popular process scheduling algorithms are: 1) First Come First Serve (FCFS), 2) Shortest-Job-First (SJF) Scheduling 3) Shortest Remaining Time 4) Priority Scheduling etc.
- In the First Come First Serve method, the process which requests the CPU gets the CPU allocation first.
- In the Shortest Remaining time, the process will be allocated to the task, which is closest to its completion.
- In Shortest job first the shortest execution time should be selected for execution next
- The CPU uses scheduling to improve its efficiency.

Space for learners:

6.9 ANSWERS TO CHECK YOUR PROGRESS

A.Answers: 1. (b), 2.(b), 3.(b), 4(a), 5(c), 6(a), 7(b), 8(a), 9(a), 10(c)

B.Answers: 1. Burst time, 2. grantt, 3. arrival, 4. response, 5. queue, 6. Job scheduler, 7. FCFS, 8. Preemtive SJF, 9. Shortest Job First, 10. SJF

C. Answers: 1. True, 2. False, 3. True, 4. False, 5. True, 6. False, 7. True, 8. False, 9. True, 10. false

D.Answers: 1. G, 2. C, 3. B, 4. H, 5. A, 6. A, 7. F, 8. A, 9. D, 10. E

Space for learners:

6.10 POSSIBLE QUESTIONS

Short-Answer Questions:

1. What is process scheduling?
2. What is the need of process scheduling?
3. What is preemptive and non-preemptive scheduling?
4. What are the various scheduling criteria for CPU scheduling?
5. Define throughput.
6. What is turnaround time?
7. What is waiting time in CPU scheduling?
8. What is response time in CPU scheduling?
9. What is Gantt Chart?
10. What are the advantages of FCFS algorithm?

Long-Answer Questions:

1. Discuss the FCFS scheduling algorithm with illustration.
2. Explain SJF scheduling algorithm with illustration.
3. Explain shortest remaining time next scheduling algorithm with illustration.
4. Consider the set of 5 processes whose arrival time and burst time are given below:

Process No	Arrival Time	Burst Time
P1	3	1
P2	1	4
P3	4	2
P4	0	6
P5	2	3

If the CPU scheduling policy is SJF non-preemptive, calculate the average waiting time and average turnaround time.

5. Consider the set of 6 processes whose arrival time and burst time are given below:

Process No	Arrival Time	Burst Time
P1	3	4
P2	5	3

P3	0	2
P4	5	1
P5	4	3
P6	7	5

If the CPU scheduling policy is STRF, calculate the average waiting time and average turnaround time.

6. Discuss the various key terms used in process scheduling.
7. Discuss the criteria for a scheduling algorithm can be preemptive or non-preemptive.
8. Discuss the importance of scheduling algorithms.
9. Explain the various scheduling criteria for CPU scheduling.
10. Compare the preemptive and non-preemptive scheduling algorithms.

6.11 REFERENCES & SUGGESTED READINGS

- lberschatz, Galvin, and Gagne's Operating System Concepts, Seventh Edition.

Space for learners:

UNIT 7: PROCESS SCHEDULING ALGORITHMS-II

Space for learners:

Unit Structure:

7.1INTRODUCTION

Space for learners:

CPU scheduling is a technique that allows one process to use the CPU while another's execution is halted (in a waiting state) due to the lack of a resource such as I/O, allowing the CPU to be fully utilised. I/O and CPU time are both used in a typical procedure. Time spent waiting for I/O in a uni-programming system like MS-DOS is wasted, and CPU is free during this time. One process can use the CPU while another waits for I/O in multiprogramming systems. This is only possible with process scheduling. CPU scheduling is the foundation of a multi-programmed operating system. The OS can make a computer more productive by switching the CPU among the processes. The operating system must choose one of the processes in the ready queue to execute whenever the CPU becomes idle. The short-term scheduler is in charge of the selecting process (or CPU scheduler). The scheduler chooses from among the ready-to-run processes in memory and assigns the CPU to one of them. A multiprogramming system allows multiple processes to run at the same time. When a process must wait, the OS takes the CPU away from that process and assigns it to another. This pattern persists. Multiprogramming's goal is to keep at least one process running at all times in order to maximise CPU utilisation. Only one process can execute at a time on a single processor system; any other processes must wait until the CPU is free and can be rescheduled. CPU scheduling is to make the system more efficient, quick, and fair.

The introduction and objective portion of Process scheduling algorithm were discussed in the previous Process scheduling algorithm Unit 6 with different scheduling algorithm. As a result, another five scheduling algorithms, such as Round Robin Scheduling, Priority CPU Scheduling, Multilevel Queue Scheduling, Multilevel Queue Scheduling, and Scheduling in Real Time System, have been discussed in this Unit 7.

7.2 UNIT OBJECTIVES

After going through this unit, you will be able to:

- understand about Round Robin scheduling
- understand about various types of priority CPU scheduling

- know about Multilevel Queue Scheduling
- explain various issues related to the implementation of concurrency primitives
- explain scheduling in real time system

7.3 ROUND ROBIN SCHEDULING

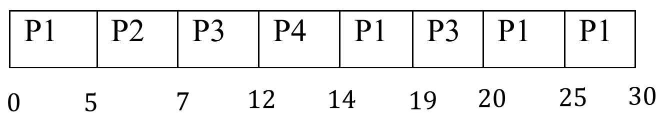
The RR scheduling algorithm was created with time-sharing systems in mind. It's the same as FCFS scheduling, but with the addition of pre-emption to switch between processes. Every process is given a small unit of time called a quantum or time slice. The duration of a time quantum is typically 10 to 100 milliseconds. When a process has completed its execution for the specified amount of time, it is pre-empted and another process executes for the specified amount of time. The CPU scheduler goes around the ready queue, allocating the CPU to every process for 1 time quantum intervals. A circular queue is used to treat the ready queue.

The ready queue is kept as a FIFO queue of processes to execute RR scheduling. New processes are added to the ready queue's tail. The CPU scheduler selects the first process from the ready queue, sets the timer to interrupt after one time quantum, and dispatches it. Then two cases may arise; the process may have a CPU burst of less than 1 time quantum, in which case the process will surrender the CPU voluntarily. After that, the scheduler will move on to the next process in the ready queue. Another scenario is that if the current operating process's CPU burst is longer than one time quantum, the timer will go off, causing an OS interrupt. A context switch is performed, and the process is pushed to the back of the ready queue. The CPU scheduler will then choose the next available process from the ready queue.

For example: suppose time quantum is 5ms and the process P1,P2,P3 and P4 are scheduled by using RR scheduling

Process	Burst Time
P1	20
P2	2
P3	6
P4	2

GANTT chart



Processes	Burst Time	Turn Around Time Turn Around Time = Completion Time – Arrival Time	Waiting Time Waiting Time = Turn Around Time – Burst Time
P1	20	30-0=32	30-20=10
P2	2	7-0=7	7-2=5
P3	6	20-0=21	20-6=14
P4	2	14-0=15	14-2=12

Process P1 receives the first 5 milliseconds, but because it requires another 15 milliseconds, it is pre-empted after the first time quantum, and the CPU is given to process P2. P2 finishes its execution before the 5ms time limit expires. The CPU is subsequently allocated to the third process, P3.it is pre-empted after first time quantum, and the CPU is given to the next process p4. P4 does not require 5ms and exits before reaching its time quantum. The following process, P1, receives the CPU and it is pre-empted after the second time quantum, and the CPU is given to process P3.P3 finishes its execution before the 5ms time limit expires. The following process, P1, receives the CPU.

Average waiting time is calculated by adding the waiting time of all processes and then dividing them by no. of processes.

Average waiting time = waiting time of all processes/ no. of processes

<u>Space for learners:</u>

Average waiting time= (10+5+14+12)/4 = 44/4= 10.25ms

If the ready queue has n processes and the time quantum is q, each process receives 1/n of the CPU time in chunks of at most q time units. Each process must wait (n-1) x q time units before proceeding to the next time quantum.

The magnitude of the time quantum determines the RR policy; if the time quantum is extremely big, the RR policy is the same as FCFS; if the time quantum is extremely tiny, the RR technique is known as processor sharing.

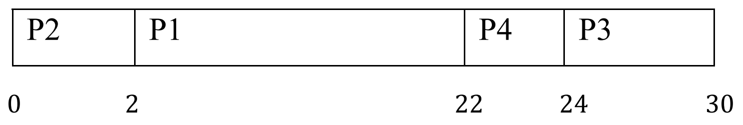
7.4 PRIORITY CPU SCHEDULING

A priority is associated with each process, and the CPU is allocated to the process with the highest priority. Equal priority processes are scheduled in FCFS order. The priority of a process in the Shortest Job First scheduling technique is generally the inverse of the CPU burst time, i.e. the larger the burst time the lower is the priority of that process.

Assume that low numbers indicate high priority in this case.

Process	Burst Time	Priority
P1	20	2
P2	2	1
P3	6	4
P4	2	3

GANTT chart



The average waiting time will be (0+2+22+24)/4=12 ms

Priorities can be established both internally and externally. Internally specified priorities compute the priority of a process using some measurable quantity or quantities. The priority of process, when internally defined, can be decided based on memory requirements, time limits, number of open files, ratio of I/O burst to CPU burst etc.

Space for learners:

External priorities, on the other hand, are determined by factors outside of the operating system, such as the importance of the process, the funds paid for the usage of computer resources, the department sponsoring the activity, and other frequently political concerns. Types of Priority Scheduling Algorithm

Priority scheduling can be of two types:

Space for learners:

7.4.1 Pre-emptive Priority Scheduling

If a new process arrives at the ready queue with a higher priority than the presently running process, the CPU is pre-empted, which means the current process's processing is halted and the incoming new process with the higher priority is given the CPU for execution.

7.4.2 Non-Preemptive Priority Scheduling

If a new process comes with a higher priority than the currently running process in a non-preemptive priority scheduling algorithm, the incoming process is placed at the front of the ready queue, which means it will be executed after the current process has completed.

7.4.3 Problem with Priority Scheduling Algorithm

A major problem with priority scheduling algorithm is indefinite blocking or starvation. A process is considered blocked when it is ready to run but has to wait for the CPU as some other process is running currently.

But in case of priority scheduling if new higher priority processes keeps coming in the ready queue then the processes waiting in the ready queue with lower priority may have to wait for long durations before getting the CPU for execution.

7.4.4 Using Aging Technique with Priority Scheduling

Aging is a solution to the problem of low priority processes being blocked indefinitely. Aging is a method of progressively boosting the priority of processes that have been waiting for a long period in the system.

For example, if we decide the aging factor to be 0.5 for each day of waiting, then if a process with priority 10(which is comparatively low priority) comes in the ready queue. After one day of waiting, its priority is increased to 9.5 and so on.

Space for learners:

7.5 MULTILEVEL QUEUE SCHEDULING

For circumstances when processes can be easily categorised into separate groups, a new family of scheduling algorithms has been developed.

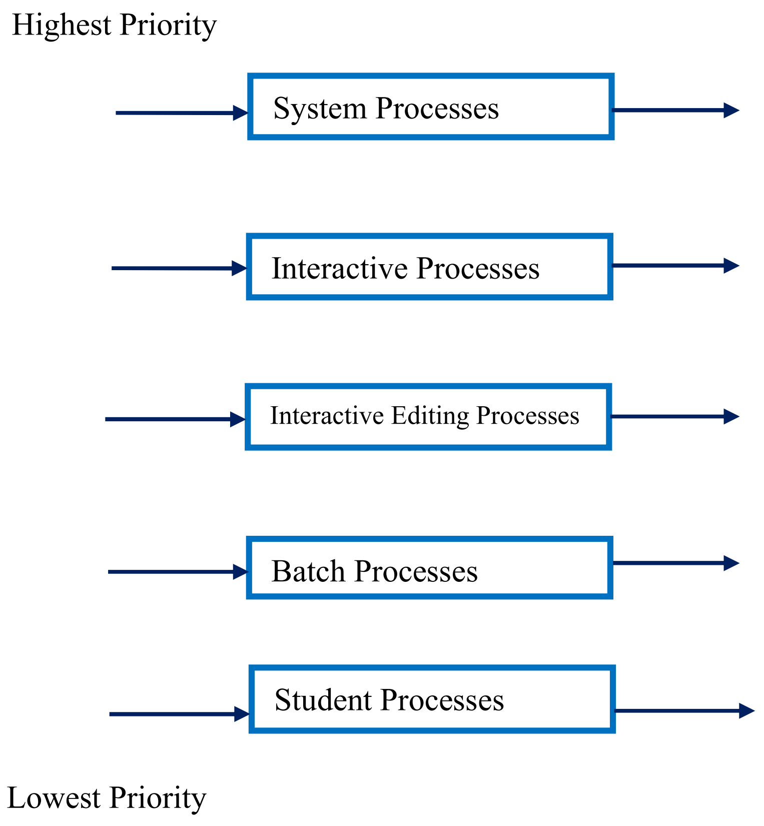
Foreground (or interactive) processes are distinguished from background (or batch) processes. These two processes have varying response times and, as a result, may have different scheduling requirements. Furthermore, foreground processes could take precedence over background processes.

The ready queue is divided into numerous different queues using a multi-level queue scheduling technique. The processes are assigned to one queue indefinitely, usually depending on some property of the process, such as memory size, priority, or kind. Each queue has its own method for scheduling. Separate queues could be used for foreground and background processes, for example. The Round Robin algorithm may be used to schedule the foreground queue, while an FCFS algorithm may be used to schedule the background queue. In addition, the queues must be scheduled, which frequently did using fixed-priority pre-emptive scheduling. The foreground queue, for example, may have absolute precedence over the background queue.

Consider the following example of a five-queue multilevel queue scheduling algorithm:

- System Processes
- Interactive Processes
- Interactive Editing Processes
- Batch Processes
- Student Processes

Each queue has absolute precedence over ones with lower priority. If the queues for system processes, interactive processes, and interactive editing processes were all empty, no process in the batch queue could run. The batch process will be pre-empted if an interactive editing process enters the ready queue while a batch process is running.



Only the processes on the lower priority queues will run if there are no processes on the higher priority queue. Consider the following Example: Once processes on the system queue, the Interactive queue, and Interactive editing queue become empty, only then the processes on the batch queue will run. The processes in the above diagram are described as follows:

- System Process: The operating system has its own set of processes to run, which are referred to as System Processes.
- Interactive Process: The Interactive Process is one in which all participants should participate in the same way.
- Batch Processes: Batch processing is a mechanism in the operating system that gathers programmes and data into a batch before processing begins.
- Student Process: The system process is always given top priority, whereas student processes are always given lowest priority

Space for learners:

Space for learners:

There are numerous processes in an operating system, and we can't put them all in a queue to get the desired outcome; consequently, multilevel queue scheduling is used to overcome this problem. We may use this scheduling to apply various types of scheduling to various types of processes:

For System Processes: First Come First Serve (FCFS) Scheduling.

For Interactive Processes: Shortest Job First (SJF) Scheduling.

For Batch Processes: Round Robin (RR) Scheduling

The problem of starvation for lower-level processes is the fundamental drawback of Multilevel Queue Scheduling. Lower-level processes are either never executed or have to wait a long period due to lower priority or higher priority processes requiring a long time due to starvation.

Example:

Suppose there are three queues.

Q0-RR with a 10-millisecond time quantum

Q1-RR with a 20-millisecond time quantum

Q2-FCFS

Scheduling:

- A new job is added to queue Q0, which is handled by FCFS. Job receives 10 milliseconds when it gains CPU. If it takes longer than 10 milliseconds to complete, the job is pushed to queue Q1.
- In Q1, the work is served FCFS for the second time and is given an additional 20 milliseconds. It gets pre-empted and pushed to queue Q2 if it still does not complete.

7.6 IMPLEMENTATION OF CONCURRENCY PRIMITIVES

Multiple instruction sequences are executed at the same time, which is known as concurrency. This occurs when numerous process threads are running in parallel in the operating system. Message passing or shared memory is used by the running process threads to communicate with one another. Concurrency causes resource sharing, which leads to issues like as deadlocks and resource starvation. It aids with approaches such as coordinating execution of processes, memory allocation, and execution scheduling in order to maximise throughput.

Space for learners:

7.6.1 Problems in Concurrency

- Sharing global resources –If two processes use the same global variable and conduct read and write operations on it, the order in which those operations are performed is critical.
- Optimal allocation of resources –It is difficult for the operating system to manage the allocation of resources optimally.
- Locating programming errors –Because reports are rarely reproducible, finding a programming error might be challenging.
- Locking the channel –The operating system may find it inefficient to simply lock the channel and prohibit other processes from using it.

7.6.2 Advantages of Concurrency

- Running of multiple applications –It allows you to execute many programmes at the same time.
- Better resource utilization –It allows resources that aren't being used by one application to be used by other application.
- Better average response time –Without concurrency, one application must be completed before moving on to the next.
- Better performance –When one application only utilises the processor and another only uses the disc drive, the time it takes to complete both applications concurrently is less than the time it takes to complete each application sequentially.

7.6.3 Drawbacks of Concurrency

- Multiple applications must be protected from each other.
- Additional mechanism is necessary to coordinate various applications.

- Switching between programmes necessitates additional performance overheads and complications in the operating system.
- Sometimes running too many applications concurrently leads to severely degraded performance.

7.6.4 Issues of Concurrency

- Non-atomic –Operations that are non-atomic but interruptible by multiple processes can cause problems.
- Race conditions –A race condition occurs of the outcome depends on which of several processes gets to a point first.
- Blocking –Processes can block waiting for resources. A process could be blocked for long period of time waiting for input from a terminal. If the process is required to periodically update some data, this would be very undesirable.
- Starvation –It occurs when a process does not obtain service to progress.
- Deadlock –It occurs when two processes are blocked and hence neither can proceed to execute.

7.6.5 Process Synchronization

Processes are classified into one of two categories based on their synchronisation:

- Independent Process: Execution of one process does not affect the execution of other processes
- Cooperative Process: The execution of one process has an impact on the execution of others.

Process synchronization problem arises in the case of Cooperative process also because resources are shared in Cooperative processes.

7.6.6 Race Condition

A race condition is an undesirable scenario that arises when a device or system seeks to perform two or more operations at the same time, yet the activities must be performed in the correct sequence due to the nature of the device or system. When several processes access and process the same data at the same time, the outcome is determined by the order in which the access takes place.

A race condition is an occurrence that can happen within a critical section. This occurs, when the result of multiple thread execution in the critical region varies depending on the sequence in which the threads run.

If the critical section is regarded as an atomic instruction, race situations in critical sections can be avoided. Race problems can also be avoided by employing thread synchronisation techniques such as locks or atomic variables.

7.6.7 Critical Section Problem

A critical section is a code segment that only one process can access at a time. In a critical section, atomic action is required, which means that only one process can run in that region at a time. All the other processes have to wait to execute in their critical sections.

The critical section is given as follows:

```
do {
        Entry Section
        Critical Section
        Exit Section
        Remainder Section
} while (TRUE);
```

In the above code, the entry section handles the entry into the critical section. It obtains the resources required for the process's execution. The exit section handles the exit from the critical section. It frees up resources while also informing other processes that a critical section is now available.

The process asks entrance into the Critical Section at the entry section.

Any solution to the problem of the critical section must meet three criteria:

Space for learners:

- Mutual Exclusion: If a process is running in its crucial section, no other processes are permitted to run in that section.
- Progress: If a process isn't using the critical section, it shouldn't prevent other processes from using it.
- Bounded Waiting: Bounded waiting implies that each process must have a set amount of time to wait. It should not have to wait indefinitely to get to the critical section.

Space for learners:

7.6.8 Semaphore

A semaphore is a signalling mechanism and a thread that is waiting on a semaphore can be signalled by another thread. This is different than a mutex as the mutex can be signalled only by the thread that called the wait function.

A semaphore uses two atomic operations, wait and signal for process synchronization.

The wait operation decrements the value of its argument S, if it is positive. If S is negative or zero, then no operation is performed.

```
wait(S){
        while (S<=0);
        S--;
       }
```

The signal operation increments the value of its argument S.

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

There are two types of semaphores: Binary Semaphores and Counting Semaphores

- Binary Semaphores: They can only be either 0 or 1. They are also known as mutex locks, as the locks can provide mutual exclusion. All the processes can share the same mutex semaphore that is initialized to 1. Then, a process has to wait until the lock becomes 0. Then, the process can make the mutex semaphore 1 and start its critical section. When it completes its critical section, it can reset the value of mutex semaphore to 0 and some other process can enter its critical section.

- Counting Semaphores: They can have any value and are not restricted over a certain domain. They can be used to control access to a resource that has a limitation on the number of simultaneous accesses. The semaphore can be initialized to the number of instances of the resource. Whenever a process wants to use that resource, it checks if the number of remaining instances is more than zero, i.e., the process has an instance available. Then, the process can enter its critical section thereby decreasing the value of the counting semaphore by 1. After the process is over with the use of the instance of the resource, it can leave the critical section thereby adding 1 to the number of available instances of the resource.

7.7 SCHEDULING IN REAL TIME SYSTEM

In real-time computing, scheduling analysis refers to the examination and testing of the scheduler system and the algorithms used in real-time applications. Real-time systems are those that do tasks in real time. Real-time scheduling analysis is the examination, testing, and verification of the scheduling system and algorithm used in real-time activities in the field of computer science. A real-time system's performance must be evaluated and certified before it can be used in essential tasks.

The scheduler, clock, and processing hardware components make up a real-time scheduling system. Hard real-time tasks and soft real-time tasks are two types of real-time activities. A hard real-time task must be completed within a certain amount of time, or massive losses may occur. A defined deadline can be missed in soft real-time jobs. This is due to the fact that the task can be rescheduled (or) performed after the deadline.

The scheduler, which is often a short-term task scheduler, is the most significant component in real-time systems. Instead of dealing with the deadline, the main goal of this scheduler is to lower the response time connected with each of the linked processes. If a pre-

Space for learners:

emptive scheduler is employed, the real-time task must wait until the time slice for its related task has finished. Even if the task is given the highest priority, a non-preemptive scheduler must wait until the current task is completed before moving on to the next one. This task may be slow (or) of the low priority, resulting in a lengthier delay.

Combining pre-emptive and non-preemptive scheduling creates a more effective strategy. This can be accomplished by incorporating time-based interrupts into priority-based systems, which implies that the presently operating process is interrupted on a time-based interval, and if a higher priority process exists in a ready queue, it is performed by pre-empting the current process.

Analysis of the algorithm execution times is used to undertake performance verification and execution on a real-time scheduling algorithm. Testing the scheduling algorithm under various test situations, including the worst-case execution time, will be required to verify the performance of a real-time Scheduler. To evaluate the algorithm's performance, these testing scenarios encompass worst-case and unfavourable circumstances.

In a real-time system, different ways can be used to test a scheduling system. Input/output verifications and code analysis are two examples of techniques. One way involves putting each input condition to the test and observing the results. Depending on how many inputs there are, this method could take a lot of effort.A risk-based strategy, in which representative critical inputs are selected for testing, is another faster and more cost-effective alternative. This method is more cost-effective, but if the wrong approach is utilised, it may result in less-than-optimal findings about the system's validity. After changes to the scheduling system, retesting requirements are considered on a case-by-case basis. Real-time system testing and verification should not be restricted to input/output and code verifications, but should also include testing and verification of operating applications employing intrusive and non-intrusive methods.

Space for learners:

CHECK YOUR PROGRESS

Multiple Choice Questions:

1: On receiving an interrupt from an I/O device, the CPU

(A) Halts for predetermined time.

(B) Branches off to the interrupt service routine after completion of the current instruction.

(C) Branches off to the interrupt service routine immediately.

(D) Hands over control of address bus and data bus to the interrupting device.

2: The problem of indefinite blockage oflow-priority jobs in general priority scheduling algorithm can be solved using:

(A) Parity bit

(B) Aging

(C) Compaction

(D) Timer

3: Consider n processes sharing the CPU in round robin fashion. Assuming that each process switch takes s seconds, what must be the quantum size q such that the overhead resulting from process switching is minimized but, at the same time each process is guaranteed to get its turn at the CPU at least every t seconds?

(A) $q \leq \frac{t-ns}{n-1}$

(B) $q \geq \frac{t-ns}{n-1}$

(C) $q \leq \frac{t-ns}{n+1}$

(D) $q \geq \frac{t-ns}{n+1}$

4: A CPU generally handles an interrupt by executing an interrupt service routine

(A) As soon as an interrupt is raised

(B) By checking the interrupt register at the end of fetch cycle

(C) By checking the interrupt register after finishing the executing the current instruction

(D) By checking the interrupt register at fixed time intervals

5: Pre-emptive scheduling is the strategy of temporarily suspending a gunning process

(A) Before the CPU time slice expires

(B) To allow starving processes to run

(C) When it requests I/O

(D) To avoid collision

6: In round robin CPU scheduling as time quantum is increased the average turnaround time

(A) Increases

(B) Decreases

(C) remains constant

(D) Varies irregularly

7: Which of the following scheduling algorithm could result in starvation?

(A)First-come, first-served

(B)Shortest job first

(C) Round robin

(D) Priority

8: Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process. This task is known as a

(A) Swapping

(B) Context switch

(C) Demand paging

(D) Page fault

9: Consider the 3 processes, P1, P2 and P3 shown in the table.

Process	Arrival time	Time Units Required
P1	0	5
P2	1	7
P3	3	4

The completion order of the 3 processes under the policies FCFS and RR2 (round robin scheduling with CPU quantum of 2 time units) are

(A) FCFS: P1, P2, P3 RR2: P1, P2, P3

(B) FCFS: P1, P3, P2 RR2: P1, P3, P2

(C) FCFS: P1, P2, P3 RR2: P1, P3, P2

(D) FCFS: P1, P3, P2 RR2: P1, P2, P3

Space for learners:

7.8 SUMMING UP

- Context Switching: The process of switching the CPU from one process or task to another is known as context switching. The kernel suspends the execution of the process that is in the running state, and the CPU executes another process that is in the ready state.
- Multiprogramming :A computer that can execute multiple programmes at the same time (like running Excel and Firefox simultaneously).
- Multiprocessing: A computer that uses multiple CPUs at the same time.
- Multitasking: Tasks sharing a common resource (like 1 CPU).
- Multithreading: Itis an extension of multitasking.
- Pre-emptive Scheduling: Pre-emptive Scheduling is a style of scheduling in which jobs are largely assigned according to their priority. Even if the lower priority task is still running, it is sometimes necessary to run a higher priority task before a lower priority task. The lower priority task is put on hold for a while and then resumes when the higher priority task is completed.
- Non-preemptive Scheduling: Once the CPU has been allocated to a process in non-preemptive scheduling, the process holds the CPU until it releases it, either by terminating or transitioning to the waiting state. It does not interrupt a process executing on the CPU in the middle of its execution while using non-preemptive scheduling. Instead, it waits until the process has finished its CPU burst period before allocating the CPU to another process.
- Starvation: Starvation is the problem that occurs when high priority processes keep executing and low priority processes get blocked for indefinite time.
- Aging: To prevent starvation of any process, we can use the concept of aging where we keep on increasing the priority of low-priority process based on the its waiting time.
- Round Robin Scheduling: Round Robin is the pre-emptive process scheduling algorithm. Each process is provided a fix

time to execute, it is called a quantum. Once a process is executed for a given time period, it is pre-empted and other process executes for a given time period. Context switching is used to save states of pre-empted processes.

- Priority CPU Scheduling: Priority scheduling is a non-preemptive method that is one of the most widely used in batch systems. A priority is assigned to each process. The process with the highest priority will be carried out first, and so on. On a first-come, first-served basis, processes of the same priority are executed.
- Multilevel Queue Scheduling: The ready queue has been separated into seven different queues by the multilevel queue scheduling method. These processes are permanently assigned to one queue based on their priority, such as memory size, process priority, or process kind. Each queue has its own method for scheduling. Some queues are utilised for the foreground process, while others are used for the background process.
- Scheduling in Real time system: Real-time systems are those that do tasks in real time. Real-time scheduling analysis is the examination, testing, and verification of the scheduling system and algorithm used in real-time activities in the field of computer science. A real-time system's performance must be evaluated and certified before it can be used in essential tasks.
- Throughput: Throughput is the amount of work completed in a unit of time. In other words throughput is the processes executed to number of jobs completed in a unit of time. The scheduling algorithm must look to maximize the number of jobs processed per time unit.
- Turnaround time: The turnaround time is the period between when a process is submitted and when it is completed. The total time spent waiting in the ready queue, executing on the CPU, and performing I/O is the turnaround time.
- Waiting time: The CPU scheduling technique has no effect on the amount of time a process executes or performs I/O; it only impacts the amount of time the ready queue is active. The total amount of time spent waiting in the ready queue is referred to as waiting.

Space for learners:

- Response time: The time it takes from submitting a request to receiving the first response. That is, reaction time refers to the time it takes to initiate a response rather than the time it takes to complete the response.

7.9 ANSWERS TO CHECK YOUR PROGRESS

1.Ans: (B)

Explanation: When the CPU is performing the same job while also receiving an interrupt,

i. It will first complete the current task.

ii. It will branch off to the interrupt service function after the current instruction is completed.

ISR stands for interrupt service routine or also known as an interrupt handler. It is a software process invoked by an interrupt request from a hardware device. It handles the request and sends it to the CPU i.e. interrupting the active process. When the ISR is complete, the process is resumed.

2.Ans: (B)

Aging is a solution to the problem of low-priority processes being blocked indefinitely. Aging is a method of gradually raising the priority of processes that have been waiting for a long time in the system.

3.Ans: (A)

Explanation: Each process will get CPU for q seconds and each process wants CPU again after t seconds.

Thus, there will be (n-1) processes once after current process gets CPU again. Each process takes s seconds for context switch.

(P1)(s)(P2)(s)(P3)(s)(P1)

It can be seen that since P1 left and arrived again, there have been n context switches and (n-1) processes. Thus, equation will be:

$q*(n-1) + n*s <= t$

$q*(n-1) <= t - n*s$

$q <= (t-n.s) / (n-1)$

Space for learners:

4.Ans: (C)

Explanation: A CPU handles interrupt by executing interrupt service subroutine by checking interrupt register after execution of each instruction.

5.Ans: (A)

In preemptive scheduling tasks are usually assigned with priorities. At times it is necessary to run a certain task that has a higher priority before another task although it is running. Therefore, the running task is interrupted for some time and resumed later when the priority task has finished its execution. This is called preemptive scheduling.

In non-preemptive scheduling, a running task is executed till completion. It cannot be interrupted.

6.Ans. (D)

Explanation:-There are few criteria are used for measuring the performance of a particular scheduling algorithm.

The turnaround time is the interval of time between the submission of a process and its completion.

The wait time is the amount of time a process has been waiting in the ready queue.

The response time is the time taken between the process submission and the first response produced.

In RR algorithm, the value of time quantum or the time slice, plays a crucial role in deciding how effective the algorithm is. If the time quantum is too small, there could be lot of context switching happening which could slow down the performance. If the time quantum is too high, then RR behaves like FCFS. If the time quantum is increased, the average response time varies irregularly. If you take any comprehensive material on operating system, you will come across a graph which depicts this behavior. So the answer is option D.

Space for learners:

7.Ans. (B)

Shortest job first could cause starvation. Priority is always given to the shortest job meaning that a job in queue which is long could constantly be starved by arrival of jobs which are shorter than that job.

8.Ans.(B)

In computing, a context switch is the process of storing the state of a process or thread, so that it can be restored and resume execution at a later point. ... In a multitasking context, it refers to the process of storing the system state for one task, so that task can be paused and another task resumed.

Q.9.Ans. (C)

Explanation:

The GANTT chart for the FCFS scheduling algorithm is

P1	P2	P3

0 5 12 16

The completion order for FCFS is P1→P2→P3

The GANTT chart for the RR scheduling algorithm is

P1	P2	P1	P3	P2	P1	P2	P3	P1

0 2 4 6 8 10 11 13 15 16

The completion order for RR is:P1→P3→P2

7.10POSSIBLE QUESTIONS

1. What is round robin scheduling? Explain with an example.
2. Explain Priority CPU scheduling with example.

3. Define Pre-emptive and non-pre-emptive Priority Scheduling.

4. Explain multilevel queue scheduling.

5. What is concurrency? What are problems associated with concurrency. What are the advantages of concurrency? Explain.

6. Define process synchronisation.

7. Define race condition.

8. Explain critical section.

9. Define semaphore.

10. How scheduling is done in real time system. Explain.

Space for learners:

7.11REFERENCES & SUGGESTED READINGS

- lberschatz, Galvin, and Gagne's Operating System Concepts, Seventh Edition.

UNIT 8: CONCURRENT PROCESS MANAGEMENT

Space for learners:

Unit Structure:

8.1 INTRODUCTION

In this unit you will learn about the mechanism of inter-process communication. In inter-process communication two or more processes communicating with each other using shared memory or message passing system. There are many issues associated with a shared memory system. When two processes use shared memory simultaneously then race condition may occur. Mutual exclusion is a way to avoid this race condition. The piece of code by using which a process accesses the shared memory is known as critical region. One can achieve mutual exclusion by restricting the use of this critical region by a process. Different methods to achieve mutual exclusion in shared memory environment have been discussed here. Again, in message passing system processes communicate with each other using two procedures called send() and receive(). The design issues associated with message passing system have been discussed here.

8.2 UNIT OBJECTIVES

Space for learners:

After going through this unit, you will be able to:

- understand the concept of inter-process communication mechanism
- know about shared memory and message passing methods
- learn about race condition, critical region and mutual exclusion
- learn about different ways to achieve mutual exclusion with and without busy waiting
- learn about the variable semaphore

8.3 INTER-PROCESS COMMUNICATION MECHANISM

Inter Process Communication (IPC) is a mechanism that involves communication of one process with another process. A process is independent if it cannot be affected by the other processes executing in the system. A process is cooperating if it can affect or be affected by the other processes executing in the systems. Any process that shares data with other processes is a cooperating process. Cooperating processes need inter-process communication (IPC) mechanism that will allow them to exchange data and information.

In Interposes Communication or IPC, the system has to deals with three issues-

8.3.1First Issue in Inter-Process Communication

The first issue of inter-process commutation deals with how information is passed between processes.

8.3.1.1 Shared Memory

It is a region of memory that is shared by cooperating processes. Processes can change information by reading and writing data to the shared region Shared memory allows multiple processes to share virtual memory space. This is the fastest but not necessarily the easiest way for processes to communicate with one another. In general, one process creates or allocates the shared memory segment. The size and access permissions for the segment are set when it is created. The process then attaches the shared segment, causing it to be mapped into its current data space. If needed, the creating process then initializes the shared memory. Once created, and if permissions permit, other processes can gain access to the shared memory segment and map it into their data space. Each process accesses the shared memory relative to its attachment address. While the data that these processes are referencing is in common, each process uses different attachment address values. For each process involved, the mapped memory appears to be no different from any other of its memory addresses.

8.3.1.2 Message Passing

In message passing system communication takes place by means of messages exchanged between the cooperating processes. - Message Passing is useful for exchanging smaller amounts of data and easier to implement for inter-computer communication.

Message Passing provides a mechanism for processes to communicate and to synchronize their actions without sharing the same address space. This method of inter-process communication uses two primitives, send and receive, which are system calls rather than language constructs. As such, they can easily be put into library procedures, such as

send(destination, &message);

receive(source, &message);

The former call sends a message to a given destination and the latter one receives a message from a given source (or from *ANY*, if the receiver does not care). If no message is available, the receiver could block until one arrives. Alternatively, it could return immediately with an error code.

8.3.2 Second Issue in Inter-Process Communication

The second issue is to proper sequencing of processes when dependencies are present: if process *A* produces data and process

B prints it, *B* has to wait until *A* has produced some data before starting to print.

Space for learners:

8.3.2.1 Race Condition

In some operating systems, processes that are working together may share some common storage that each one can read and write. The shared storage may be in main memory (possibly in a kernel data structure) or it may be a shared file; the location of the shared memory does not change the nature of the communication or the problems that arise.

Let us see how inter-process communication works. Suppose a process wants to print a file in printer spooler. The process enters the file names in a special spooler directory that has a large number of slots, numbered 0, 1, 2, ..., etc to store the file names. Another process printer daemon periodically checks and removes the file name of next file to be printed from the spooler directory.

Suppose there are two shared variables, *out*, which points to the next file to be printed, and *in*, which points to the next free slot in the spooler directory. At a certain instant, slots 0 to 3 are empty (the files have already been printed) and slots 4 to 6 are full (with the names of files to be printed). More or less simultaneously, processes *A* and *B* decide they want to queue a file for printing. Process *A* reads *in* and stores the value, 7, in a local variable called *next_free_slot*. Just then a clock interrupt occurs and the CPU decides that process *A* has run long enough, so it switches to process *B*. Process *B* also reads *in*, and also gets a 7, so it stores the name of its file in slot 7 and updates *in* to be an 8. Then it goes off and does other things. Eventually, process *A* runs again, starting from the place it left off last time. It looks at *next_free_slot*, finds a 7 there, and writes its file name in slot 7, erasing the name that process *B* just put there. Then it computes *next_free_slot*+ 1, which is 8, and sets *in* to 8. The spooler directory is now internally consistent, so the printer daemon will not notice anything wrong, but process *B* will never receive any output. User *B* will hang around the printer room for years, wistfully hoping for output that never comes. Situations like this, where two or more processes are reading or writing some shared data and the final result depends on who runs precisely when, are called **race conditions**.

8.3.3. Third Issue in Inter-Process Communication

Space for learners:

The third issue is to prevent two or more processes from accessing the critical section simultaneously when shared memory is in used.

8.3.3.1 Mutual Exclusion

The key to avoid race condition is prohibiting more than one process from reading and writing the shared data at the same time. To achieve this, we need mutual exclusion mechanism. **Mutual exclusion** is a way to make sure that if one process is using a shared variable or file, the other processes will be excluded from accessing that shared variable or file. That part of the program where the shared memory is accessed is called the **critical region** or **critical section**. Thus if no two processes were ever in their critical regions at the same time, we could avoid race conditions. Although this is a key to avoid race condition, but this is not sufficient for having parallel processes cooperate correctly and efficiently using shared data.

Hence, the necessary and sufficient conditions to hold to have a good solution are-

1. No two processes may be simultaneously inside their critical regions.
2. No assumptions may be made about speeds or the number of CPUs.
3. No process running outside its critical region may block other processes.
4. No process should have to wait forever to enter its critical region.

8.3.3.2 Methods to Achieve Mutual Exclusion with Busy Waiting

In this section we will discuss about various methods for achieving mutual exclusion, so that while one process is updating a shared variable in its critical region, no other processes will enter its critical region.

- **Disabling Interrupts**

Different kinds of interrupts are used to switch the CPU between processes. Therefore, one solution to achieve mutual exclusion is each process disables all interrupts just after entering its critical region and re-enable them just before leaving it. With interrupts turned off the CPU will not be able to switched between processes. But it is not a good idea to give a user process permission to turn off interrupts. Suppose that one of them did, and then never turned them on again? If an interrupt occurred while the list of ready processes, for example, was in an inconsistent state, race conditions could occur. Again in multiprocessor system disabling interrupts in one CPU will not affect other CPUs. Thus disabling interrupt by user process is not an appropriate way for mutual exclusion.

- **Lock Variables**

Consider a shared variable lock which can take the value either 0 or 1. The value of variable lock is 0 means no process is in its critical region and a 1 means some process is in its critical region. Initially the value of the variable lock is set to 0. Before entering critical region, the process checks the value of lock and set it to 1 if it is already 0. Otherwise it will wait until the value of lock becomes 0.

Now suppose one process reads the lock and sees that it is 0. Before it can set the lock to 1, another process is scheduled, runs, and sets the lock to 1. When the first process runs again, it will also set the lock to 1, and two processes will be in their critical regions at the same time.

Again the first process can be reading out the lock value, then checking it again just before storing into it, but that really does not help. The race now occurs if the second process modifies the lock just after the first process has finished its second check.

- **Strict Alternation**

In this approach a spin lock called turn is used whose value initially set to 0. A lock that uses busy waiting is called a spin lock and continuously testing a variable until some value appears is called busy waiting. The variable turn keeps track of whose turn it is to enter the critical region. Initially the value of turn is set to 0. Initially, process 0 examines *turn*, finds it to be 0, and enters its

Space for learners:

critical region. At this time if Process 1 checks the value of turn and finds it to be 0, it will continuously testing *turn* to see when it becomes 1. When process 0 leaves the critical region, it sets *turn* to 1, to allow process 1 to enter its critical region.

```
while (TRUE) {                          while (TRUE) {
while (turn != 0);                        while (turn != 1);
critical_region( );                       critical_region( );
turn = 1;                                  turn = 0;
noncritical_region( );                      noncritical_region( );
    }                                       }
  (a) Process 0                           (b) Process 1.
```

When one of the processes is much slower than the other then this method may not work. Suppose that process 1 finishes its critical region quickly, so both processes are in their noncritical regions, with *turn* set to 0. Now process 0 executes its critical region and leave it by setting *turn* to 1. At this point *turn* is 1 and both processes are executing in their noncritical regions. Now suppose process 0 finishes its noncritical region quickly and tries to enter its critical region. Unfortunately, it is not permitted to enter its critical region now, because *turn* is 1 and process 1 is busy with its noncritical region. This situation violates condition 3 discussed previously: process 0 is being blocked by a process not in its critical region.

- **Peterson's Solution**

Before using the shared variables (i.e., before entering its critical region), each process calls *enter_region*with its own process number, 0 or 1, as the parameter. This call will cause it to wait, if need be, until it is safe to enter. After it has finished with the shared variables, the process calls *leave_region*to indicate that it is done and to allow the other process to enter, if it so desires.

```
#define FALSE 0
#define TRUE 1
#define N 2                          /* number of processes */
int turn;    /* whose turn is it? */
int interested[N];                   /* all values initially 0 (FALSE) */
```

Space for learners:

Space for learners:

```
void enter_region(int process)                    /* process is 0 or 1 */
{
    int other;     /* number of the other process */
    other = 1 −process;                    /* the opposite of process */
    interested[process] = TRUE;
    turn = process;
    while (turn == process && interested[other] == TRUE);
}

void leave_region(int process)          /* process: who is leaving */
 {
    interested[process] = FALSE;
  }
```

Initially, process 0 calls *enter_region* as neither process is in its critical region. It indicates its interest by setting its array element and sets *turn* to 0. Since process 1 is not interested, *enter_region*returns immediately. If process 1 now calls *enter_region*, it will hang there until *interested* [0] goes to *FALSE.* Now consider the situation in which both processes call *enter_region*almost simultaneously. Both processes will store their process number in *turn*. Whichever store it last will reflect in turn; the first one will lost. Suppose that process 1 stores last, so *turn* is 1. When both processes come to the while statement, process 0 executes it zero times and enters its critical region. Process 1 loops and does not enter its critical region.

- **The TSL (Test and Set Lock) Instruction**

Many computers, especially those designed with multiple processors in mind, have an instruction

TSL RX, LOCK

The above Test and Set Lock instruction will read the contents of the memory word *LOCK* into register RX and then stores a nonzero value at the memory address *LOCK*. This *LOCK* is a shared variable. No interrupt will occur during the execution of this instruction. When *LOCK* is 0, any process may set it to 1 using the TSL instruction and then read or write the shared memory. When it is done, the process sets *LOCK* back to 0 using an ordinary move instruction. Now, a process can enter and leave critical region using the following instruction subroutine.

```
enter_region:
 TSL REGISTER, LOCK
  CMP REGISTER,#0             | was LOCK zero?
JNE enter_region| if it was non zero, LOCK was set, so loop
  RET                    | return to caller; critical region entered
leave_region:
   MOVE LOCK,#0                | store a 0 in LOCK
RET                             | return to caller
```

Before entering its critical region, a process calls *enter_region*. In *enter_region,* the first instruction copies the old value of *LOCK* to the register and then sets *LOCK* to 1. Then the old value of *LOCK* is compared with 0. If it is nonzero, the lock was already set, so the program just goes back to the beginning and tests it again. When a process currently in its critical region is done with its critical region it calls *leave_region*, which stores a 0 in *LOCK* and the subroutine returns, with the lock set.

8.3.3.3 Methods to Achieve Mutual Exclusion Without Busy Waiting

Both Peterson's solution and the solution using TSL are correct, but both have the defect of requiring busy waiting. Not only does this approach waste CPU time, but it can also have unexpected effects. Some other situation for achieving mutual exclusion without busy waiting have been discussed below-

Space for learners:

- **Solving Producer consumer problem using Sleep() and Wakeup() system calls**

Instead of wasting CPU time in busy waiting, a process can be blocked when it is not allowed to enter its critical region. The available system calls that can be used for this purpose are- sleep() and wakeup(). The sleep() system call is used to block the caller process and the wakeup() system call is used to wake up a blocked process.

Let us consider the **producer-consumer problem** (also known as the **bounded buffer** problem). Two processes share a common, fixed-size buffer. One of them, the producer, puts information into the buffer, and the other one, the consumer, takes it out.

Suppose the maximum number of items the buffer can hold is *N and* a variable *count* keeps track of the number of items in the buffer. Now what will happen when the producer wants to put a new item in the buffer. The producer will first check if *count* is *N*. If it is, the producer will go to sleep; if it is not, the producer will add an item into the buffer using the procedure insert_item() and increment *count*. Again if consumer wants to remove an item from the buffer then it will first check the value of *count.* If it is 0 then consumer will go to sleep. If it is nonzero then consumer will remove an item from the buffer using the procedure remove_item() and decrement the count. Each of the processes also tests to see if the other should be sleeping, and if not, wakes it up. But this method could lead to race condition, because access to *count* is unconstrained.

```
#define N 100 /* number of slots in the buffer */
int count = 0; /* number of items in the buffer */
void producer(void)
{
int item;
while (TRUE) { /* repeat forever */
item = produce_item( ); /* generate next item */
if (count == N) sleep( ); /* if buffer is full, go to sleep */
insert_item(item); /* put item in buffer */
count = count + 1; /* increment count of items in buffer */
if (count == 1) wakeup(consumer); /* was buffer empty? */
}
}
void consumer(void)
{
int item;
while (TRUE) { /* repeat forever */
if (count == 0) sleep( ); /* if buffer is empty, got to sleep */
item = remove_item( ); /* take item out of buffer */
count = count −1; /* decrement count of items in buffer */
if (count == N −1)
wakeup(producer); /* was buffer full? */
consume_item(item); /* print item */
}
}
```

Space for learners:

Suppose the buffer is empty and the consumer has just read *count* to see if it is 0. At that instant, the scheduler decides to stop running the consumer temporarily and start running the producer. The producer enters an item in the buffer, increments *count*, and notices that it is now 1. Reasoning that *count* was just 0, and thus the consumer must be sleeping, the producer calls *wakeup* to wake the consumer up. Unfortunately, the consumer is not yet logically asleep, so the wakeup signal will have lost. When the consumer next runs, it will test the value of *count* it previously read, find it to be 0, and go to sleep. Sooner or later the producer will fill up the buffer and also go to sleep. Both will sleep forever.

- **Solving Producer consumer problem using Semaphores**

Semaphores are integer variables that are used to solve the critical section problem by using two operations, down and up that are used for process synchronization. To solving synchronization problems and avoiding race conditions all the actions happening inside each of these down and up operations must be done as single atomic action. Hence, once a semaphore operation has started, no other process can access the semaphore until the operation has completed or going to sleep. The operating system briefly disables all interrupts while it is executing down or up operation on a semaphore. If multiple CPUs are being used, each semaphore should be protected by a lock variable, with the TSL instruction used to make sure that only one CPU at a time examines the semaphore.

There are two main types of semaphores-

i. Counting semaphores

ii. Mutexes or Binary semaphores

- ***Counting semaphores***

These are integer value semaphores and have an unrestricted value domain. In producer consumer problem a semaphore could have the value 0, indicating that no wakeups were saved or some positive value if one or more wakeups were pending.

The down operation on a counting semaphore (s) checks to see if the value is greater than 0. If the value is greater than 0 then it decrements the value and continues. If the value is 0, the process is put to sleep or block without completing the down for the moment.

The up operation on the counting semaphore increments the value of the semaphore addressed. If one or more processes were sleeping on that semaphore, unable to complete an earlier down operation, one of them is chosen by the system randomly and is allowed to complete its down.

- ***Mutexes or Binary semaphores***

The mutexes or binary semaphores are like counting semaphores but their value is restricted to 0 and 1. The down operation only works when the semaphore is 1 and the up operation only works when the semaphore is 0.

To solve produce consumer problem this solution uses three semaphores-

full: This semaphore is used for counting the number of slots that are full. *Full* is initially 0. It ensures that the producer stops running when the buffer is full.
empty: This semaphore is used for counting the number of slots that are empty. *empty* is initially equal to the number of slots in the buffer. It ensures that the consumer stops running when the buffer is empty.

Space for learners:

mutex: The *mutex* semaphore is used for mutual exclusion. This semaphore is used to make sure that the producer and consumer do not access the buffer at the same time. *mutex* is initially 1.

If each process does a down just before entering its critical region and an up just after leaving it, mutual exclusion is guaranteed.

```
#define N 100                    /* number of slots in the buffer */
typedef int semaphore;
semaphore mutex = 1;
semaphore empty = N;
semaphore full = 0;
void producer(void)
{
int item;
while (TRUE)
 {
item = produce_item( );    /* generate something to put in buffer */
down(&empty);
down(&mutex
insert_item(item);
up(&mutex);
up(&full);
}
}
void consumer(void)
{
int item;
while (TRUE) {
down(&full);
down(&mutex);
item = remove_item( );
up(&mutex);
up(&empty);
consume_item(item);
}
}
```

- **Monitors**

Space for learners:

A monitor is a collection of procedures, variables, and data structures that are all grouped together in a special kind of module or package. Processes may call the procedures in a monitor whenever they want to, but they cannot directly access the monitor's internal data structures from procedures declared outside the monitor. Figure 2-15 illustrates a piece of code for a monitor.

```
monitor example
integer i;
condition c;
procedure producer(x);
...
end;
procedure consumer(x);
...
end;
end monitor;
```

Monitors have a key property that makes them useful for achieving mutual exclusion: only one process can be active in a monitor at any instant. Monitors are a programming language construct, so the compiler knows they are special and can handle calls to monitor procedures differently from other procedure calls. Typically, when a process calls a monitor procedure, the first few instructions of the procedure will check to see if any other process is currently active within the monitor. If so, the calling process will be suspended until the other process has left the monitor. If no other process is using the monitor, the calling process may enter.

8.3.4 Design Issues for Message Passing Systems

Message passing systems have many challenging problems and design issues that do not arise with semaphores or monitors, especially if the communicating processes are on different machines connected by a network. For example, messages can be lost by the network. To guard against lost messages, the sender and receiver can agree that as soon as a message has been received, the receiver will send back a special **acknowledgement** message. If the sender has not received the acknowledgement within a certain time interval, it retransmits the message.

Now consider what happens if the message itself is received correctly, but the acknowledgement is lost. The sender will retransmit the message, so the receiver will get it twice. It is essential that the receiver can distinguish a new message from the retransmission of an old one. Usually, this problem is solved by putting consecutive sequence numbers in each original message. If the receiver gets a message bearing the same sequence number as the previous message, it knows that the message is a duplicate that can be ignored. Message systems also have to deal with the question of how processes are named, so that the process specified in a send or receive call is unambiguous. **Authentication** is also an issue in message systems: how can the client tell that he is communicating with the real file server, and not with an imposter? At the other end of the spectrum, there are also design issues that are important when the sender and receiver are on the same machine. One of these is performance. Copying messages from one process to another is always slower than doing a semaphore operation or entering a monitor. Much work has gone into making message passing efficient.

CHECK YOUR PROGRESS

State TRUE or FALSE:

1. Mutual exclusion is a way to avoid race condition.
2. Counting semaphore is also known as mutex.
3. In producer consumer problem we can have N producer and N consumer.
4. Both the solutions Peterson's and TSL are correct to achieve mutual exclusion without busy waiting.
5. The primitives of message passing system are-send () and receive ().

Space for learners:

8.4 SUMMING UP

- **Inter-Process Communication** (IPC) is a mechanism that involves communication of one process with another process.
- In inter-process commutation information are passed between processes using shared memory or message passing.
- **Shared memory** is a region of memory that is shared by cooperating processes
- In **message passing system** communication takes place by means of messages exchanged between the cooperating processes. This method of inter-process communication uses two primitives, send and receive.
- When two or more processes are reading or writing some shared data and the final result depends on who runs precisely are called **race conditions**.
- The key to avoid race condition is mutual exclusion.
- **Mutual exclusion** is a way to make sure that if one process is using a shared variable or file, the other processes will be excluded from accessing that shared variable or file.
- The part of the program where the shared memory is accessed is called the **critical region** or **critical section**.
- The necessary and sufficient conditions to hold mutual exclusion are-

  1. No two processes may be simultaneously inside their critical regions.
  2. No assumptions may be made about speeds or the number of CPUs.
  3. No process running outside its critical region may block other processes.
  4. No process should have to wait forever to enter its critical region.
- One solution to achieve mutual exclusion is each process disables all interrupts just after entering its critical region and re-enable them just before leaving it. But disabling interrupt by user process is not an appropriate way for mutual exclusion.

- Another one solution for mutual exclusion is using a shared lock variable. But this solution may sometimes lead to race condition.
- Strict alternation is an another solution to achieve mutual exclusion. It uses a spin lock called turn.
- In Peterson's solution before using the shared variables each process calls *enter_region*with its own process number, 0 or 1, as the parameter. This call will cause it to wait, if need be, until it is safe to enter. After it has finished with the shared variables, the process calls *leave_region*to indicate that it is done and to allow the other process to enter, if it so desires.
- Test and Set Lock (TSL) is a hardware solution to achieve mutual exclusion.
- All the above methods for mutual exclusion have disadvantage of busy waiting. Instead of wasting CPU time in busy waiting, a process can be blocked when it is not allowed to enter its critical region. The available system calls that can be used for this purpose are- sleep() and wakeup().
- In **producer-consumer problem** (also known as the **bounded buffer** problem), two processes share a common, fixed-size buffer. One of them, the producer, puts information into the buffer, and the other one, the consumer, takes it out.
- To achieve mutual exclusion in the producer consumer problem, we can use the system calls sleep() and wakeup(). But this solution may sometimes leads to race condition.
- **Semaphores** are integer variables that are used to solve the critical section problem by using two operations, down and up that are used for process synchronization.
- Another solution to achieve mutual exclusion in producer consumer problem uses semaphore to process synchronization.
- A **monitor** is a collection of procedures, variables, and data structures that are all grouped together in a special kind of module or package.

Space for learners:

8.5 ANSWERS TO CHECK YOUR PROGRESS

1. True.
2. False.
3. True.
4. False.
5. True

Space for learners:

8.6 POSSIBLE QUESTIONS

Short answer type questions:

1. Give the differences between shared memory system verses message passing system.
2. What is mutual exclusion? What are the necessary and sufficient conditions to achieve mutual exclusion?
3. Why disabling interrupt is not a good solution for mutual exclusion?
4. Mention how TSL instruction works.
5. What is producer consumer problem? How sleep() and wakeup() system calls avoid busy waiting in mutual exclusion?
6. What is semaphore? What are the operations that can be applied on a semaphore? Briefly describe about counting semaphore and binary semaphore.
7. Briefly describe about monitor.

Long answer type questions:

1. Briefly describe race condition with an example.
2. Briefly describe about how the following methods achieve mutual exclusion

   a) Lock variable

   b) Strict alternation

   c) Peterson's solution

   d) Test-and-Set Lock instruction

3. Give a solution to producer consumer problem using semaphore.
4. Briefly discussed on the design issues of message passing system.

Space for learners:

8.7 REFERENCES & SUGGESTED READINGS

- “Operating System Concepts” by Avi Silberschatz and Peter Galvin.
- “Operating Systems: Internals and Design Principles” by William Stallings.
- “Operating Systems: A Concept-Based Approach” by D M Dhamdhere.
- “Modern Operating Systems” by Andrew S Tanenbaum.