MSc-IT

GAUHATI UNIVERSITY
Institute of Distance and Open Learning

Semester- I

MSc-IT
Paper: INF 1036
(under CBCS)

Operating System

www.idolgu.in

OPERATING SYSTEM

M.Sc.-IT-19-I-1036

GAUHATI UNIVERSITY

Institute of Distance and Open Learning

First Semester

(under CBCS)

M.Sc.-IT

Paper: INF-1036

OPERATING SYSTEM

Contents:

Contributors:

Dr. Mirzanur Rahman Asstt. Prof., Dept. of IT Gauhati University	(Block I : Unit- 1)
Dr. Sruti Sruba Bharali Asstt. Prof., HCB School of Science and Technology. KKHSOU	(Block I : Unit- 2)
Dr. Khurshid Alam Borbora Asstt. Prof., GUIDOL Gauhati University	(Block I : Units- 3 & 5)
Dr. Kshirod Sarmah Asstt. Prof., Dept. of Computer Science PDUAM, Goalpara, Assam	(Block I : Unit- 4)
Dr. Sisir Kumar Rajbongshi Asstt. Prof., Dept. of Computer Science PDUAM, Goalpara, Assam	(Block I : Unit- 6)
Mr. Hem Chandra Das Asstt. Prof., Dept. of Computer Science & Technology Bodoland University Kokrajhar(BTAD), Assam	(Block I : Unit- 7)
Dr. Irani Hazarika Asstt. Prof., Dept. of Computer Science Gauhati University, Assam	(Block I : Unit- 8/ Block II: Units: 3 & 4)
Dr. Pranab Das Asstt. Prof. (Sr.), Dept. of Computer Applications Assam Don Bosco University	(Block II : Unit- 1)
Mr. Dipankar Dutta Asstt. Prof., Dept. of Computer Science NERIM, Guwahati, Assam	(Block II : Unit- 2)
Mrs. Pinky Saikia Dutta Asstt. Prof., Dept. of Computer Science & Engineering, GIMT, Guwahati	(Block II : Unit- 5)
Dr. Utpal Barman Asstt. Prof., Dept. of Computer Science & Engineering, GIMT, Guwahati	(Block II : Unit- 6)
Ms. Mala Ahmed Asstt. Prof., Dept. of Computer Science & Engineering, GIMT, Guwahati	(Block II : Unit- 7)
Dr. Manoj Kumar Deka Asstt. Prof., Dept. of Computer Science & Technology Bodoland University, Kokrajhar(BTAD)	(Block II : Unit- 8)

Content Editor:

Dr. Ridip Dev Choudhury
Associate Professor and Director(i/c),
Hiranya Chandra Bhuyan School of Science and Technology,
KKHSOU, Guwahati

Course Coordination:

Prof. Dandadhar Sarma	Director, IDOL, Gauhati University
Prof. Anjana Kakoti Mahanta	Prof., Dept. Computer Science, G.U.

Cover Page Designing:

Bhaskar Jyoti Goswami	IDOL, Gauhati University

May, 2022

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

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 works 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:

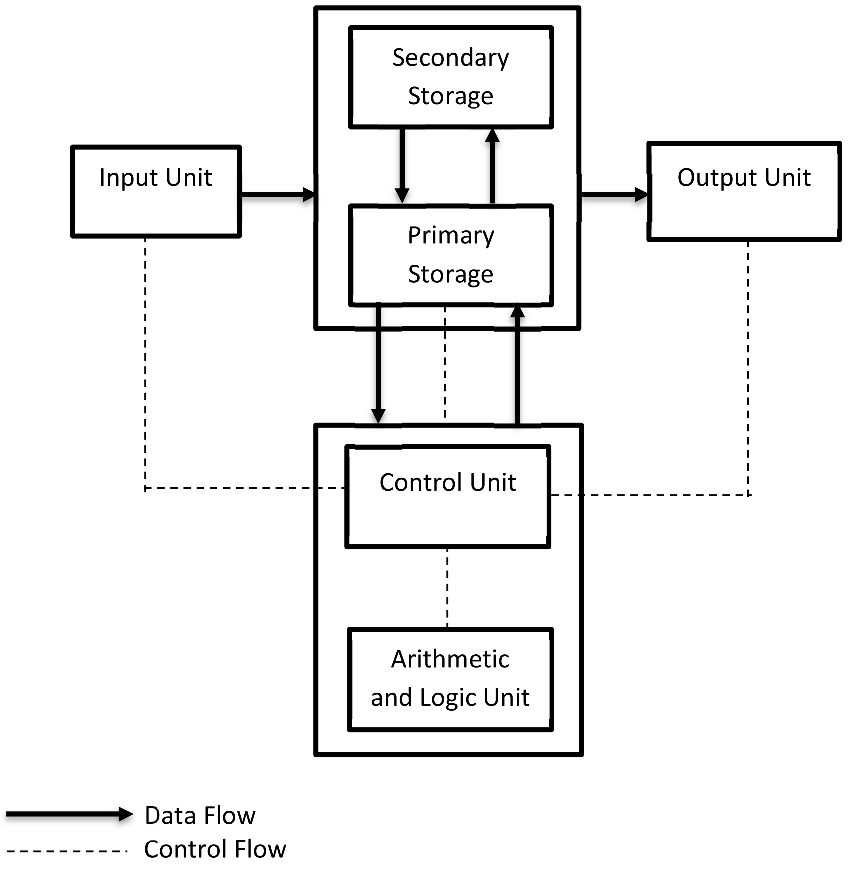


Figure 1: Block Diagram of a Computer

Space for learners:

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

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

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

Space for learners:

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

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 category
    - 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 is 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)

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

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

Space for learners:

1.3.1.5 Units of Memory

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

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

Space for learners:

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

1.4.1.2 Complex Instruction Set Computer (CISC)

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

Space for learners:

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.

Space for learners:

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.

CHECK YOUR PROGRESS

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.

Space for learners:

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.

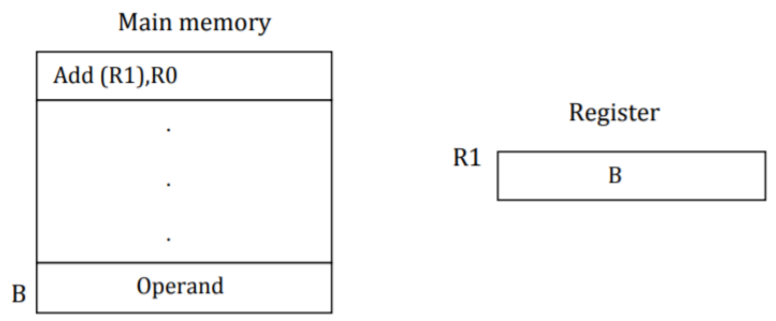
Constant values are used frequently in high-level language programs. For example, if we evaluate the expression A = B + 8, where the expression contains the constant value 8. With the assumption that A and B variables have been declared earlier. Memory locations A and B may be accessed using the Absolute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

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:

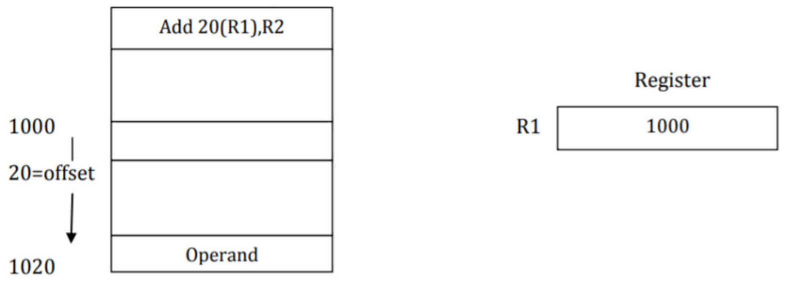
X(Ri)

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.

CHECK YOUR PROGRESS

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.

Space for learners:

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.
   1) False
   2) True
   3) False
   4) 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.
(e) Addressing modes
(f) CPU register
(g) 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?

Space for learners:

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.
9. What is index addressing modes? Discuss different index addressing modes with example.

10. Discuss the advantages and disadvantages of secondary memory.

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

Space for learners:

Unit Structure:

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

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

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.

Space for learners:

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

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 (MicroSoft 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.

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 the 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 acan 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

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

Space for learners:

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.

- **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

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.

Space for learners:

- 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.

Space for learners:

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 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.

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

Space for learners:

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.

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 Linus OS that has been developed for 64-bit OS. It provides the latest stable version of the software.

Space for learners:

3.5 LINUX ARCHITECTURE

Space for learners:

The Linux Architecture depicted in Fig. 3.1.

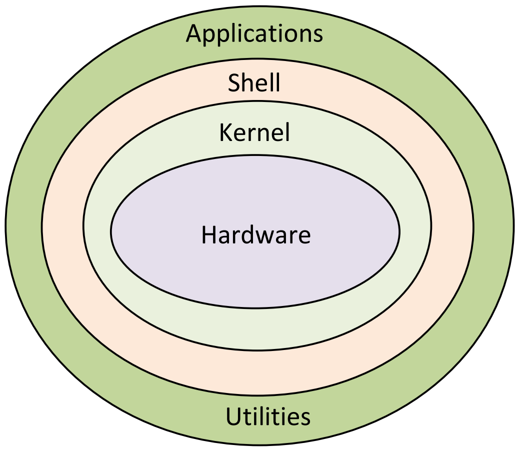


Fig. 3.1: Architecture of Linux

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
```

<u>Space for learners:</u>

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 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.

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.

Space for learners:

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:

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

Space for learners:

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.

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.

- 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.

Space for learners:

3.11 ANSWER 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 AND 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

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.

Space for learners:

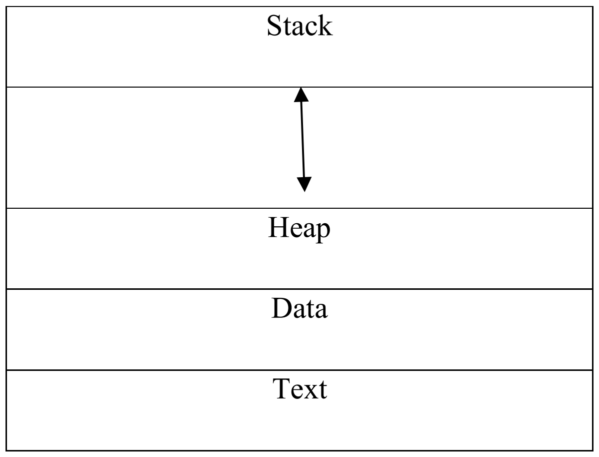


Fig.4.1. The simplified layout of a process in main memory

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.

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

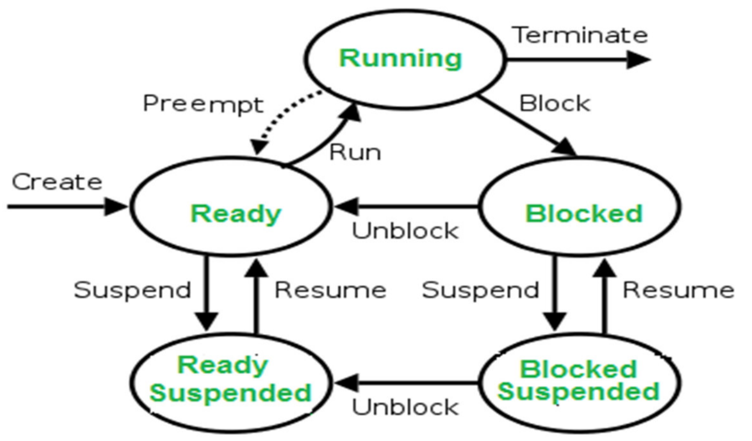


Fig.4.3. States diagram of a process

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

Space for learners:

complete their execution once the main memory gets available and their wait is finished.

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

Space for learners:

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.

Space for learners:

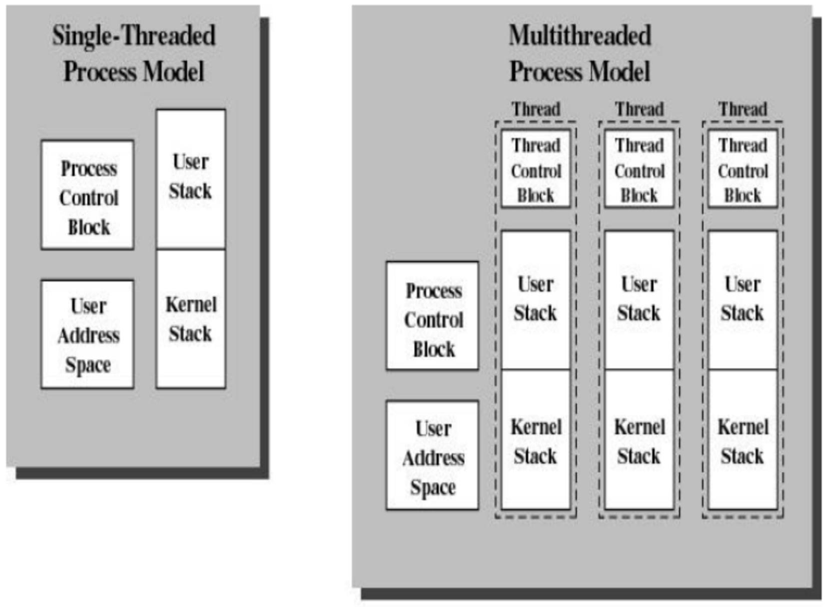


Fig.4.4. Block Diagram for the Single-Threaded and Multithreaded Process Model

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.

- 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.

Space for learners:

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[])

 {
```

```
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);
    }
  }
```

Space for learners:

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.

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.

Space for learners:

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 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.

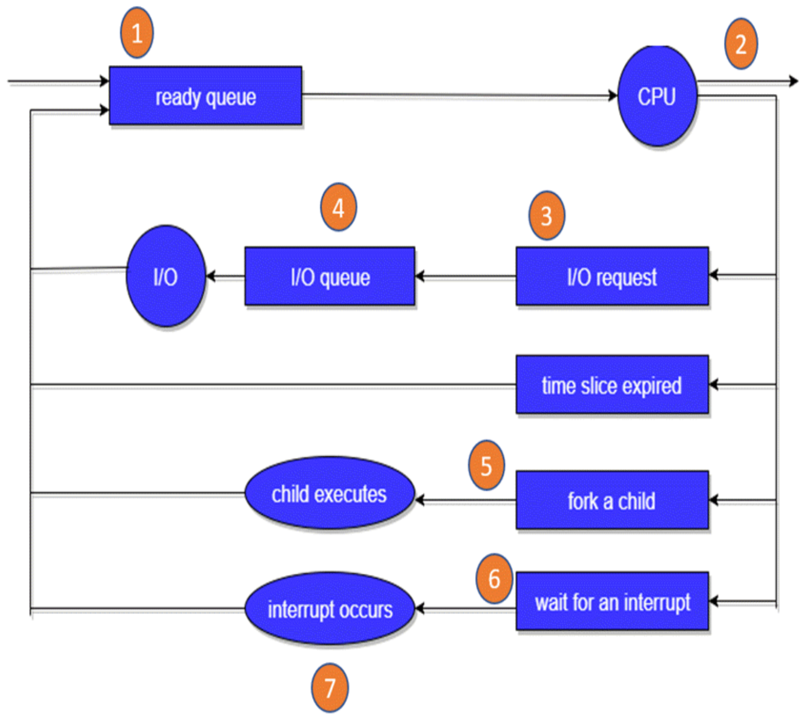


Fig.4.5. Block Diagram of Process Scheduling Queues

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:

1. Every new process first put in the Ready queue .It waits in the ready queue until it is finally processed for execution.

Space for learners:

Space for learners:

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.

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.

Space for learners:

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

Space for learners:

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.

Space for learners:

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 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.

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.

Space for learners:

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 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.

Space for learners:

- 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 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.

__Space for learners:__

- 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.
- 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.

Space for learners:

- 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.
- 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.

Space for learners:

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.

5.3 WHAT 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.

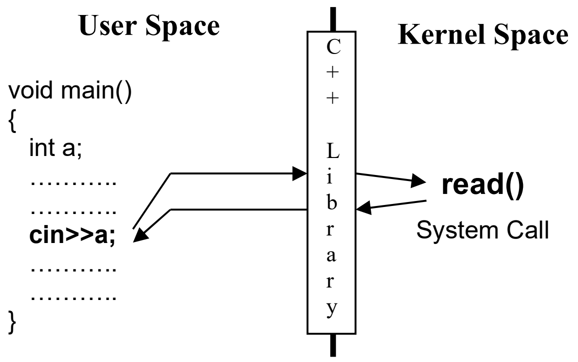


Fig. 5.1: Invocation of System Call

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 which is 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 for the 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
```

Space for learners:

```
        {
            printf("Parent Process is running…");
            wait();
            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…".

<u>Space for learners:</u>

- Process switching occurs and parent continues its execution, which displays “Parent Process is running…”.
- 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;
```

Space for learners:

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

Ideal Output (if **fork()** executes successfully)**:**

```
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, termed as 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.

Space for learners:

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.

Space for learners:

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.

- **"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:

__Space for learners:__

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.

Space for learners:

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.

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 - I

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.

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.

5.7 ANSWERS 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.

5.9 REFERENCES AND 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.

Space for learners:

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

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

Space for learners:

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.

Space for learners:

Space for learners:

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:

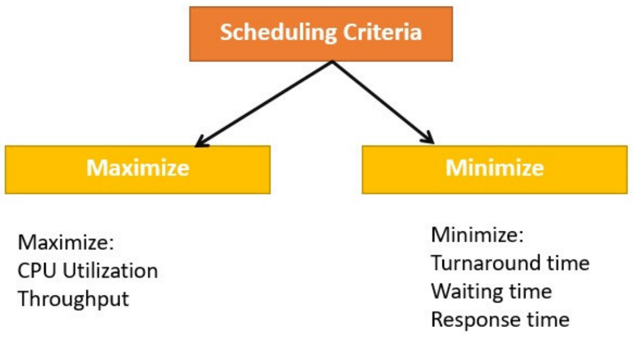


Fig 6.1 Scheduling Criteria

Space for learners:

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.

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.

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

Space for learners:

P5	4	3

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

Solution- Here, the Gantt Chart-

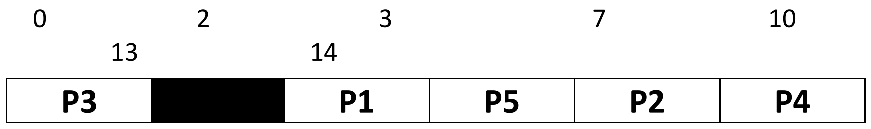


Fig 6.2 Gannt chart

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

Space for learners:

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

- 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.

Space for learners:

Table 6.3

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

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, if a 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 table5.3 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.

Space for learners:

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.

Table 6.4

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

The Gantt Chart for SRTF will be:

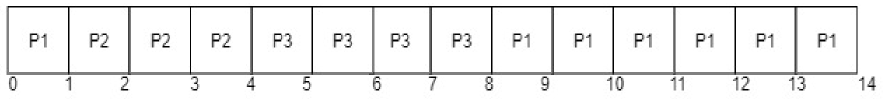


Fig 6.2 *Gantt Chart*

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

Space for learners:

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.

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.

Space for learners:

6.11 REFERENCES AND SUGGESTED READINGS

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

UNIT 7: PROCESS SCHEDULING ALGORITHM-II

Space for learners:

Unit Structure:

7.1 INTRODUCTION

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 VI 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 VII.

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

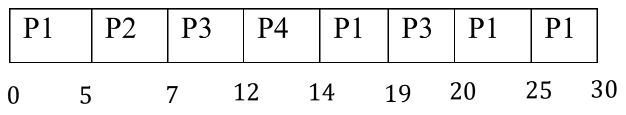
The RR scheduling algorithm was created with time-sharing systems in mind. It's the same as FCFS scheduling, but with the addition of pre-emption to switch between processes. Every process is given a small unit of time called a quantum or time slice. The duration of a time quantum is typically 10 to 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	Waiting Time
		Turn Around Time = Completion Time – Arrival 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

Space for learners:

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.

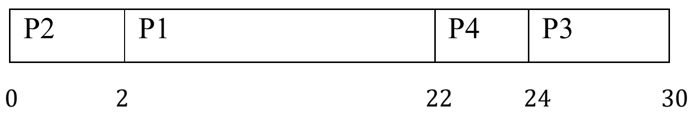
7.4 PRIORITY CPU SCHEDULING

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.

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.

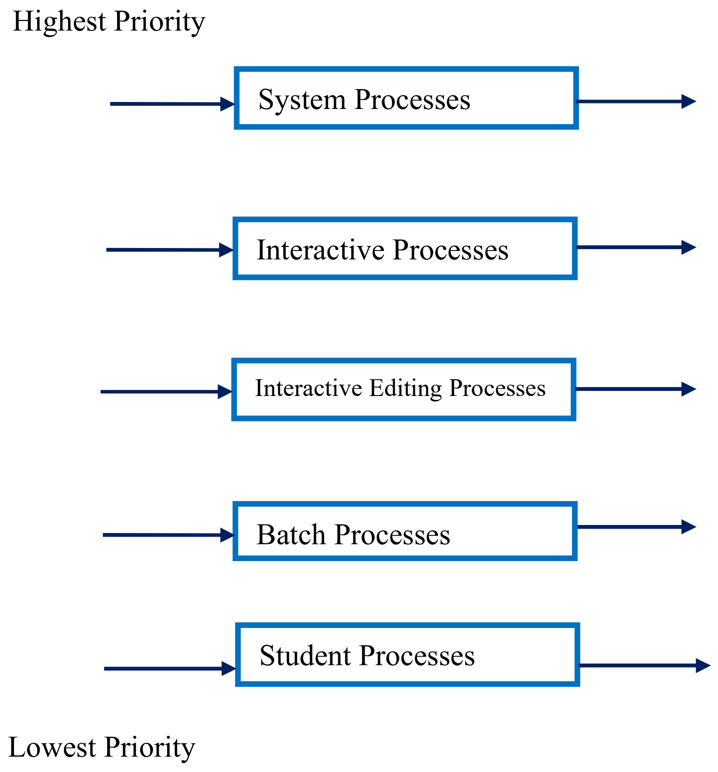
Foreground (or interactive) processes are distinguished from background (or batch) processes. These two processes have varying response times and, as a result, may have different scheduling requirements. Furthermore, foreground processes could take precedence over background processes.

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:

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

Space for learners:

starvation. It aids with approaches such as coordinating execution of processes, memory allocation, and execution scheduling in order to maximise throughput.

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.

Space for learners:

- 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.

Space for learners:

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:

- 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.

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

Space for learners:

Multiple Choice Questions:

Q1: 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.

Q2: The problem of indefinite blockage of low-priority jobs in general priority scheduling algorithm can be solved using:

(A) Parity bit

(B) Aging

(C) Compaction

(D) Timer

Q3: 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) $q \geq \frac{t-ns}{n+1}$

Q4: 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

Q5: Pre-emptive scheduling is the strategy of temporarily suspending a gunning process

(A) Before the CPU time slice expires

Space for learners:

(B) To allow starving processes to run

(C) When it requests I/O

(D) To avoid collision

Q6: In round robin CPU scheduling as time quantum is increased the average turnaround time

(A) Increases

(B) Decreases

(C) remains constant

(D) Varies irregularly

Q7: Which of the following scheduling algorithm could result in starvation?

(A) First-come, first-served

(B)Shortest job first

(C) Round robin

(D) Priority

Q8: 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

Q9: 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:

Q10: Consider the following set of processes, with the length of the CPU burst given in milliseconds:

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

The processes are assumed to have arrived in order P1, P2, P3, P4, P5 all at time 0.

a. Draw two Gantt chart that illustrates the execution of these processes using the following scheduling algorithms: non-primitive priority (a smaller priority number implies a higher priority) and RR (quantum=1).
b. What is the turnaround time of each process for each of the scheduling algorithms in part a?
c. What is the waiting time of each process for each of the scheduling algorithm in part a?
d. What is the algorithm in part a results in the minimum average waiting time (overall processes)?

Q11. Consider a system implementing multilevel queue scheduling. What strategy can a computer user employ to maximize the amount of CPU time allocated to the user's process?

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: It is 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

__Space for learners:__

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.
- 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.

Space for learners:

7.9 ANSWERS TO CHECK YOUR PROGRESS

Space for learners:

Q1.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.

Q2.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.

Q.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:

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

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

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

Q.4.Ans: (C)

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

Q.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.

Q.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:

Q.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.

Q.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

7.10 POSSIBLE 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.

Space for learners:

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:

7.11 REFERENCES AND 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-

8.3.1 First 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.

Space for learners:

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.

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

Space for learners:

```
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 */
}
}
```

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

Space for learners:

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

Space for learners:

- ***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.
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**

Space for learners:

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 - I

1. What is IPC?
2. What is race condition?
3. What is critical region?
4. What is semaphore?
5. What is monitor?

State TRUE or FALSE:

6. Mutual exclusion is a way to avoid race condition.
7. Counting semaphore is also known as mutex.
8. In producer consumer problem we can have N producer and N consumer.
9. Both the solutions Peterson’s and TSL are correct to achieve mutual exclusion without busy waiting.
10. The primitives of message passing system are-send () and receive ()

Space for learners:

8.4 SUMMING UP

Space for learners:

- **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

Space for learners:

State TRUE or FALSE:

6. True.
7. False.
8. True.
9. False.
10. True

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

Space for learners:

3. Give a solution to producer consumer problem using semaphore.
4. Briefly discussed on the design issues of message passing system.

8.7 REFERENCES AND 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.

BLOCK II:

MEMORY AND I/O MANAGEMENT, SYSTEM DEADLOCK AND MULTIPROGRAMMING SYSTEM

UNIT 1: MEMORY MANAGEMENT

Space for learners:

Unit Structure:

1.1 INTRODUCTION

The unit deals with management of main memory during process execution. One of the most important functions of operating system is memory management that includes the hardware support in processor for paging, virtual memory and segmentation. Virtual memory allows a program with memory space larger than the size of the main memory available in the system. This is possible by allowing only that section of the code that is active at that point of time without the need of having all instructions and data of the process being present in main memory at the same time. The concept of paging and segmentation eliminates the need of allocating main memory to the process in contiguous manner. Also if the overall memory requirement exceeds the physical memory limit, pages from memory may need to be replaced to make room for new pages. Various page replacement algorithms like FIFO, LRU and Optimal are used in such case.

Space for learners:

1.2 UNIT OBJECTIVES

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

- explain memory hierarchy, cache memory and associative memory
- explain the working of memory address protection.
- explain the paging memory management scheme.
- analyze and solve problems on paging.
- explain the working of paging hardware.
- explain the concept of segmentation and solve problems on segmentation.
- describe the benefits of Virtual memory management system
- explain and solve problems on different page replacement algorithms.

1.3 HIERARCHY OF MEMORY TYPES

The memory in a computer system can be divided into a hierarchy as shown in Figure 1.1. The hierarchy is based on access time, speed, cost and capacity of the memory. The five memory types in the hierarchy are registers at the top followed by the cache memory, main memory, hard disk and magnetic tapes. The first three memory types register, cache and main memory are volatile memories that is they lose their stored data in absence of power supply. The last two memory types, hard disk and magnetic tape keeps the stored data permanently even in the absence of power.

In the Figure 1.1, capacity that is the volume of information the memory can store increases as we move from top to bottom in the hierarchy.

Access time that is time required to perform read/write request increases as we move from top to bottom in the hierarchy. Similarly, the speed gap between CPU and memory decreases as we move from bottom to top of the hierarchy and finally cost per bit increases going from bottom to top of the hierarchy.

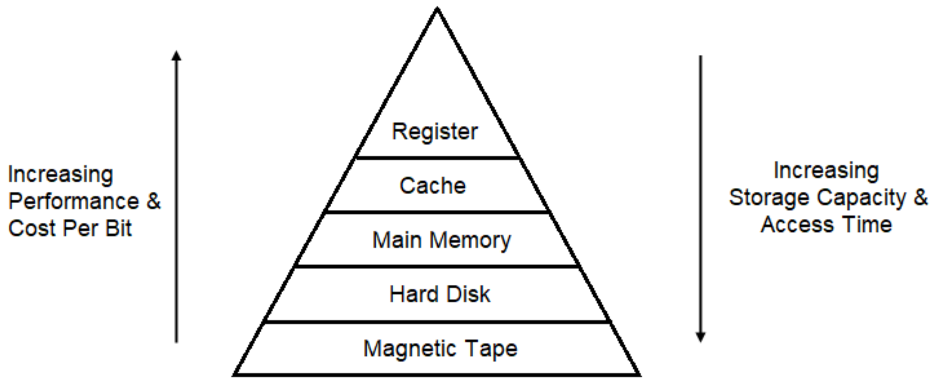

Increasing Performance & Cost Per Bit
Register
Cache
Main Memory
Hard Disk
Magnetic Tape
Increasing Storage Capacity & Access Time

Figure 1.1: Memory Hierarchy

1.4 CACHE MEMORY

Cache Memory is a type of memory that operates at a very high speed. It's used to boost performance and synchronize with a high-speed CPU. Although cache memory is more expensive than main memory but it is less expensive than CPU registers. Cache memory acts as a buffer between the main memory and the CPU as shown in Figure 1.2. Cache memory stores frequently requested instructions and data so that they may be accessed quickly by the CPU. It smaller and faster memory that reduces the average access time of main memory by storing copies of most frequently used data.

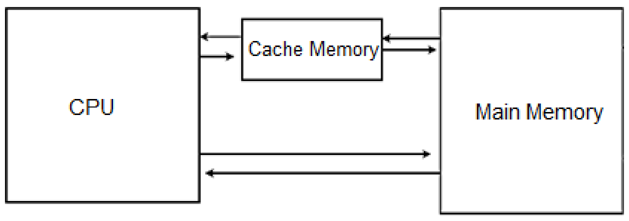


Figure 1.2: Cache Memory acting as a buffer between CPU and Main Memory.

1.5 ASSOCIATIVE MEMORY

Space for learners:

Associative memory is also known as Translation Lookaside Buffer (TLB) is a special type of memory that is optimized to perform parallel searches on data, in contrast to sequential search of data.

Operating system provides support for storing page table of a process. Generally, a page table can be stored in following ways:

- Set of dedicated registers
- In main memory
- Associative Memory or Translation lookaside buffer (TLB)

The feasibility of the first approach using a set of dedicated registers is that the page table should be reasonably smaller in size like 256 entries. With the second approach page table can be very large like millions of entries can be stored in the main memory with a pointer to the starting address of the page table for referencing. However, in this case the time required to access the page table is slower by a factor of two as it involves first accessing memory for the page table to locate the frame number which is combined with the displacement to get the physical address and then a second memory access to read the byte.

The solution to the disadvantages of the first two approaches is resolved using a fast lookup hardware support called Associative memory. Associative memory or TLB is a small, expensive but very fast associative memory. It can store entries in the range of 64 to 1024. Associative memory has two parts: a tag and a value. When a page/key needs to be searched the key is compared simultaneously with all the tags of the in the associative memory.

1.6 ADDRESS PROTECTION

In a main memory there can be several user process and operating system running at a time. To protect the address space of the operating systems as well as user processes, so that they do not run into to each other's address space, the concept of hardware address protection is introduced. Address protection is implemented with the help of two registers, the *base register* and the *limit register*. The base register holds the starting address of the process address and the limit register specifies the range. For example, in Figure 1.3, the base register holds the starting address *5500* of *Process-2* in the main memory and the limit register specifies range of *750* meaning that the range of legal address of *Process-2* is from *5500* to *6249* (inclusive).

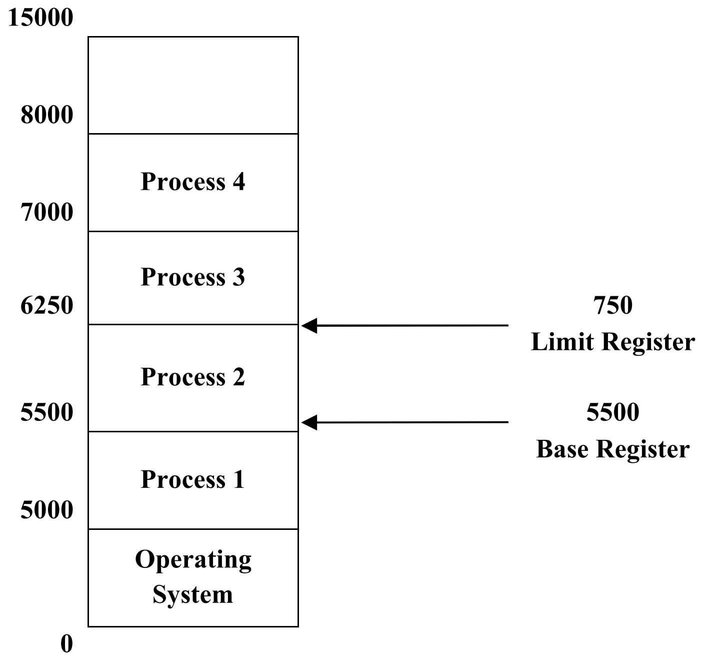


Figure 1.3: A logical address defined by base and limit register.

The Memory address protection is accomplished with the help of hardware support as shown in Figure1.4. The hardware checks that the CPU generated address is within the range specified by *base register* and *base + limit register.* A memory access attempted outside the valid range, results in trap or a fatal error.

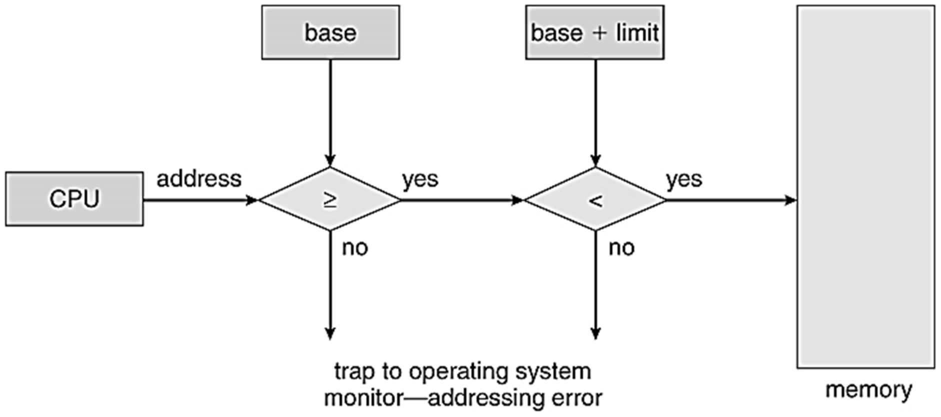


Figure 1.4: Memory address protection using base and limit register [1].

Space for learners:

1.7 PAGING

To understand the concept of Paging we have to go through the following concepts:

- Process: It is a program in execution or a program placed in main memory for execution.
- Logical Address: It is the address that is generated by the CPU for a program while it is running. As the address does not exist physically it is also called virtual address. The hardware unit of memory known as memory management unit (MMU) maps logical address to physical address.
- Physical Address: A physical address is the actual address in the main memory.

Paging is a memory management scheme that is used to map CPU generated logical address of a process to physical address in main memory. A process consists of fixed size blocks; Figure 1.5 shows an example of a process with 4 blocks each of size 1 kilobyte. Size of a block depend upon architecture of the computer and varies between 512 bytes to 16 megabytes.

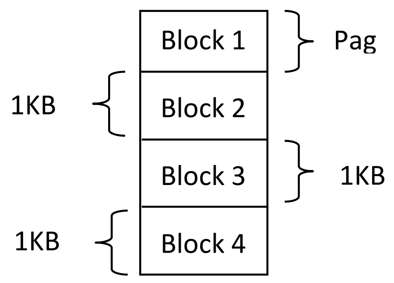


Figure 1.5: A Process with 4 blocks each of size 1 kilobyte.

The paging technique divides the logical memory to blocks of the fixed size known as Pages and divides physical memory into blocks of fixed-size known as Frames. Figure 1.6 shows an example of pages and frames in logical and physical memory respectively.

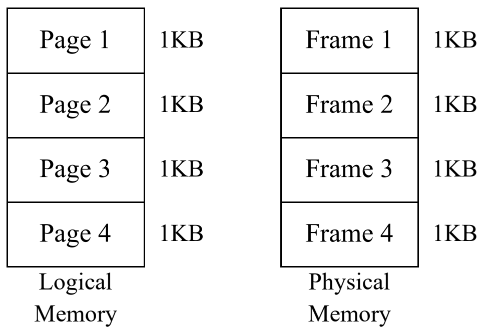


Figure 1.6: A Process with 1KB block size in logical and physical memory.

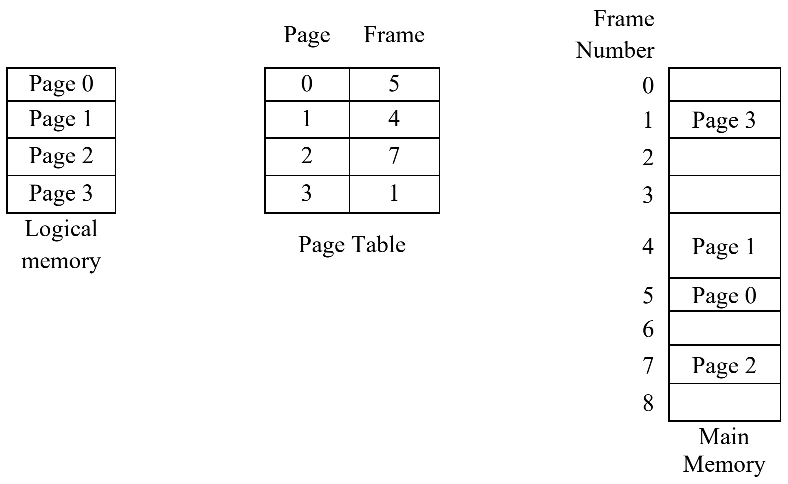


Figure 1.7: Paging model of physical and logical memory.

Space for learners:

Paging scheme allows a process to be stored in the main memory in noncontiguous manner. It also solves the problem of searching and fitting blocks of different sizes in main memory by having all block of same size. One more advantage of the paging scheme is that it prevents from external fragmentation that is if the main memory blocks are of varying sizes and the size of the free blocks are smaller than the size of the pages, then the operating will be required to merge two or more blocks into a single block large enough to fit a page. By keeping block of equal sizes for both pages and frames, such problems are resolved. The Figure 1.7 shows paging model of physical and logical memory. A page table is used for mapping between logical addresses and physical addresses. A page table resides in the main memory. The Figure 1.7 shows noncontiguous allocation of a process in main memory. The mapping of logical address to physical address is achieved using the page table.

The hardware support for paging is demonstrated using an example in Figure 1.8. The logical address generated by the CPU is divided into two parts namely *page number* and *displacement* with the page. The page number is used as an index in the page table to search for the corresponding frame number. The displacement is combined with frame number to get the physical address. In the Figure 1.8, the logical address having *page number 3* is searched for the corresponding frame number in the page table which is *frame number 15*. The *frame number 15* is combined with the *displacement 7* to form the physical address.

Space for learners:

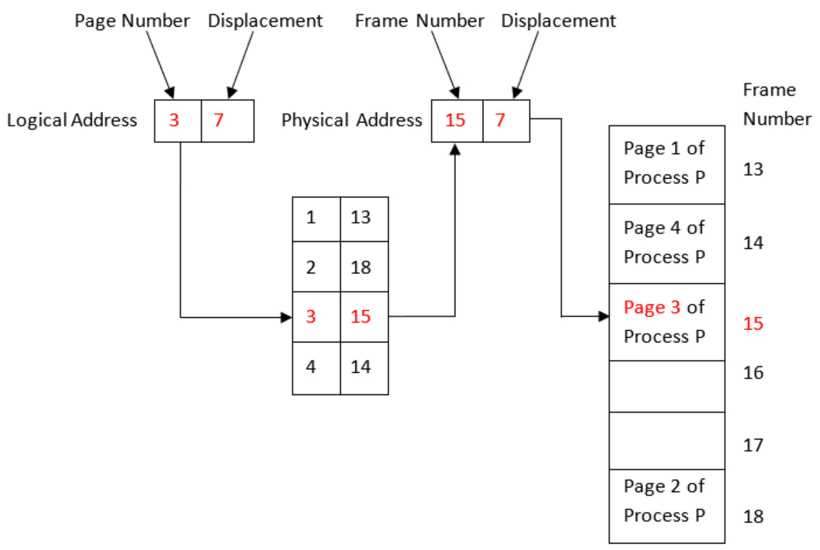


Figure 1.8: Paging hardware support.

If the size of the logical address space is 2^m and size of a page is 2^n bytes/words, then "*m-n*" bits of a logical address designate the page number the "*n*" bits designate the displacement or offset. Therefor the logical address is:

Page Number	Displacement
p	d
m - n	n

Paging Example -1:

Assume a page size of 1K and a 15-bit logical address space. How many pages are in the system?

Solution:

Page size = 1K = 2^{10} i.e. displacement, n=10 bits

No. of bits in logical address = 15, i.e. m=15 bits.

Therefore, no. of bits used for page number is, m - n = 5 bits

Total no. of pages in the system is 2^5 =32.

Paging Example -2:

Assume that a CPU has a 15-bit logical address space with 8 logical pages. How large are the pages?

Solution:

Space for learners:

There are 8 logical pages, that means 3 bits are required to address 8 logical pages ($2^3 = 8$).
Therefore, m - n=3 bits
Logical address is 15 bits, m=15 bits
Displacement = 15 -3 = 12 bits.
So, the pages are of size $2^{12} = 4096$ = 4K bytes

1.7.1 Paging Hardware Support

The operating system provides hardware support for quick search in the form of Associative memory or TLB. There are possibly two cases for a page search in TLB, Figure 1.9 illustrates the paging hardware with Translation Look aside Buffer for these two cases:

- If the search key/page is found it is called as a *TLB hit* and corresponding value/frame is returned from the TLB. Displacement is combined with frame number and the physical address is accessed.
- If the search key/page is not found it is called as a *TLB miss* and the page is searched in the page table stored in main memory. The frame number corresponding to the search page is combined with the displacement to access the address in the physical memory. Also the page number and frame number is added to the TLB so that if the same page is referred next time it is found quickly. In case the TLB is full, operating system selects a page replacement algorithm to replace an existing page with the new entry.

The percentage of times that a particular page number is found in the TLB is called the *hit ratio*. If the hit ratio is 60% that means 60 times out of 100 references the page will be found in TLB and remaining 40 times the page is found in the page table.

Space for learners:

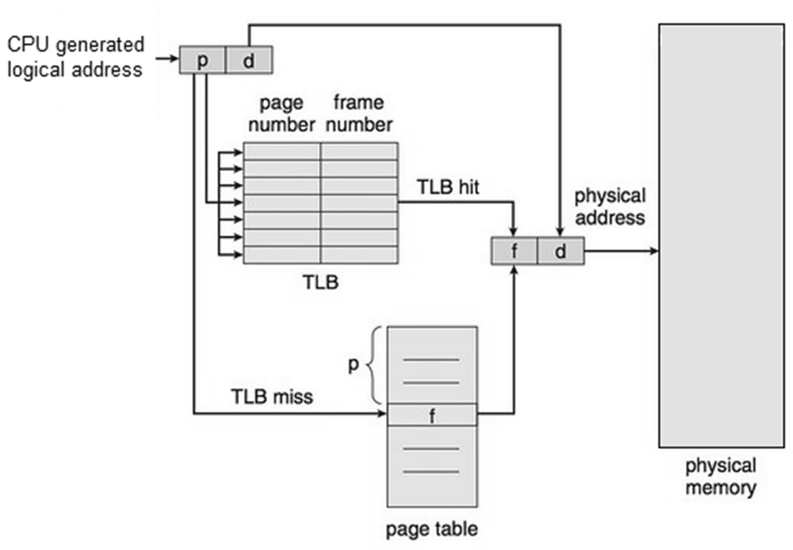


Figure 1.9: Paging hardware with Translation Look aside Buffer [1].

Paging Example -3:
If it takes 25 nanoseconds to search the TLB and 75 nanoseconds to access memory. If the hit ratio is 70%, calculate effective memory access time.

Solution:
If the page is in the TLB, time taken to access the physical address
= Time taken to search the TLB + Time taken to access memory
= 25 +75 =100 nanoseconds

If the page is in not in the TLB, time taken the physical address
= Time taken to search the TLB + Time taken to access page table + Time taken to access memory
= 25 +75 +75
= 175 nanoseconds

Hit ratio is 70%, therefore
Effective access time = 0.70 X 100 + 0.30 X 175 =122.5 nanoseconds.

Space for learners:

1.8 SEGMENTATION

Space for learners:

Segmentation is a memory management scheme similar to paging that allows a process to be stored in the main memory in noncontiguous manner. Unlike paging where all the pages or frames are of fixed size, segmentation allows blocks or segments of variable size. Segmentation maps the user's view of a program onto the physical memory. Looking at the user's view in Figure 1.10, a program contains several variable size segments, such as the main program, subroutine, symbol table, methods etc. It also includes data structures like arrays, objects, variables, stacks etc. These segments and data structures are referred by their name without concerning about the address these segments are stored in memory. Users are not concerned about the order in which the segments are stored in the memory.

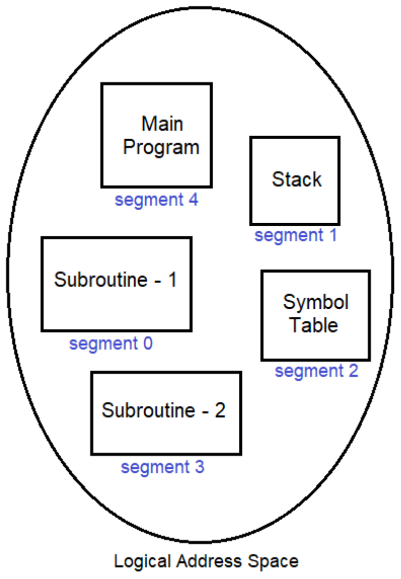


Figure 1.10: User's view of a program

The logical address space is a group of segments. Each segment has a name and a length. From the implementation point of view, segments are numbered instead of using name and the logical address is represented using the *two tuple:*

Segment-number	Displacement

1.8.1 Segmentation Hardware

The mapping of the logical address <segment-number, displacement> to the physical address is achieved with the help of segment table and the segmentation hardware as shown in Figure 1.11. Each entry of the segment table has a segment limit and segment base. The base represents the starting address of the segment in the main memory and the limit specifies the length of the segment. The segment table is indexed on the segment number.

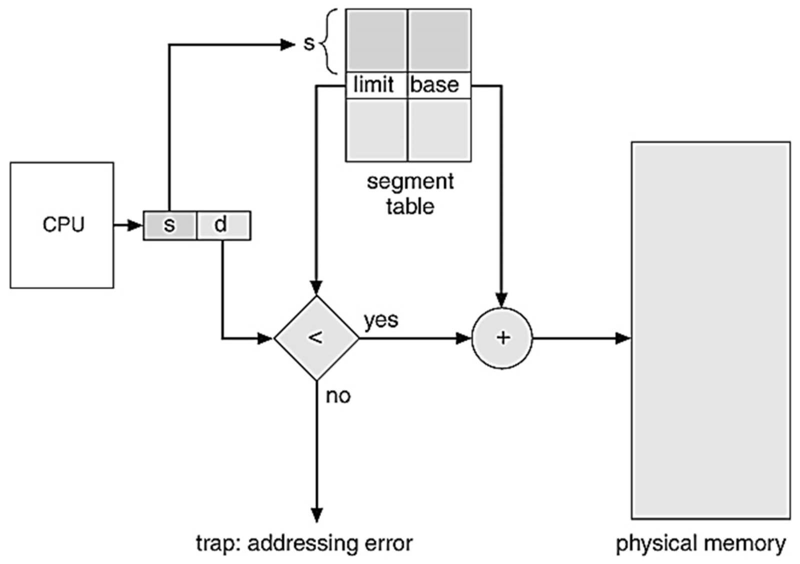


Figure 1.11: Segmentation Hardware [1].

The working of segmentation hardware starts by first identifying the segment number, *s* and the displacement, *d* of the logical address. The segment number is used to search the segment table, which is indexed on the segment number. The displacement, *d* of the logical address should be between 0 and limit. If the condition is not satisfied, it means that the logical address is going beyond the segment limit and a trap interrupt is initiated which is handled by the operating system.

A segmentation example is shown in Figure 1.12. There are 5 segments numbered from 0 through 4. The segments are stored in physical memory in noncontiguous manner. Also no specific ordering is followed for storing the segments as can be observed in the example. The segment table has an entry for each of the segment, the starting address of the segment mentioned as *base* and the length of the segment mentioned as *limit*. For example, segment 0 begins at address 5100 and length of the segment is limited to 500 bytes. Therefore, a reference to byte 17 of segment 0 is mapped to 5100 (base of segment 0) + 17 = 5117. Similarly, a reference to byte 88 of segment 4 is mapped to 7300 + 88 = 7388. A trap interrupt will be called if byte 1700 of segment 4 is referenced as the limit is 1500.

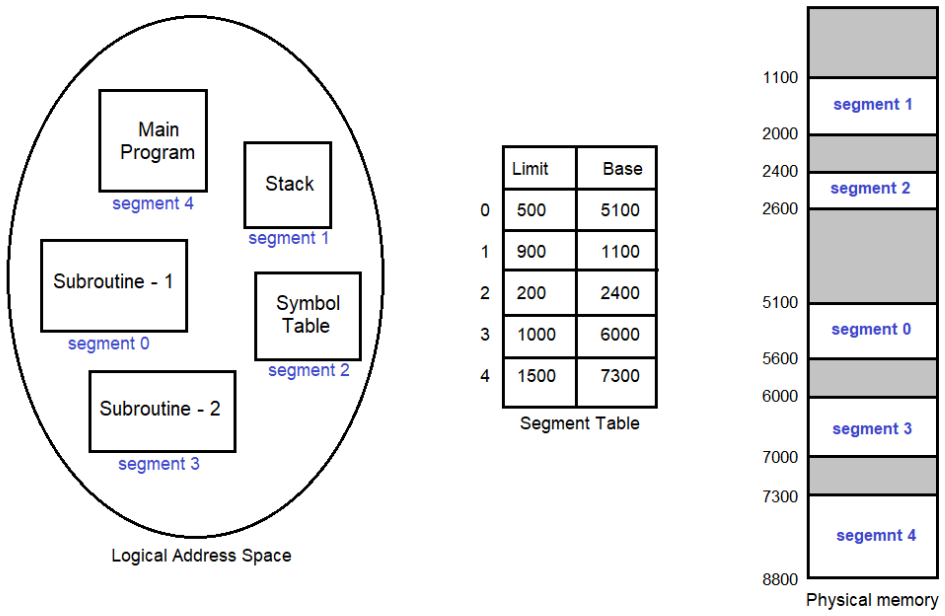


Figure 1.12: Example of Segmentation.

1.9 VIRTUAL MEMORY

The memory management scheme discussed in previous section requires the entire process to be in the main memory for execution. Most of the times there can be a requirement of many processes to be in the memory simultaneously for execution. This situation can prevent simultaneous execution of multiple processes due to the size of the main memory, which may not be large enough to hold all the processes. So, a concept of virtual memory was introduced.

A virtual memory management scheme allows execution of a process even if it is not completely in memory. That is, it requires only that

Space for learners:

section of the process's code to be in the memory that will be executed. Generally, a process contains several functions or procedures and not all the functions are required to be in the memory at the same time. So the function or the procedure that will be executed needs to be in the main memory, while the other functions or procedures can be placed in the secondary memory and wait for their turn of execution. So whenever a function is not available in the main memory, it is brought from the secondary memory to main memory for execution. The main advantage of this scheme is that a program larger than main memory can still run on a smaller physical memory. This is how a games like *Need for speed* or *Call of Duty* which require respectively 30 GB and 90 GB of memory can still run on a system having 6 GB RAM with sufficient hard disk space. Also, as only a section of the code of a process needs to be in memory so many process can be there in memory simultaneously. Thereby increasing CPU utilization and throughput.

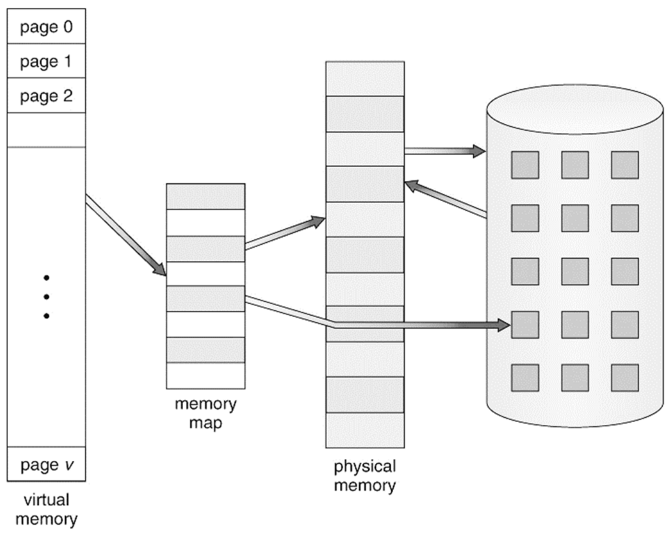


Figure 1.13: Example showing virtual memory larger than physical memory [1].

Figure 1.13 shows an example of a larger virtual memory than physical memory. The programmer thus need not have to worry about the size of the main memory available, thus can concentrate on the problem to be programmed. As can be seen in the Figure 1.13, pages from the large virtual memory address space is stored in the secondary memory and the pages are brought back to main memory whenever a call to those pages are required. If the main memory does not have any free slot for the pages, then some page replacement algorithms are used to replace the pages in main memory with the pages from secondary memory.

Figure 1.14 shows dynamic memory allocation, where the stack grows upward and the heap grows downward. The gap shown in the figure between the heap and the stack is the part of virtual address space and will require physical memory space only if either heap or stack grows or both of them grows.

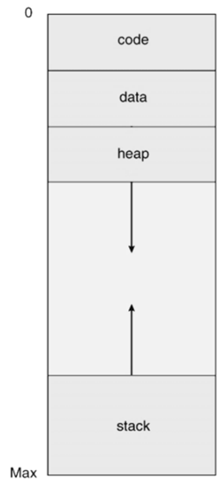


Figure 1.14: Virtual memory address space.

1.9.1 Demand Paging

Suppose a user wants to run a program, so the entire program is loaded to main memory from the secondary memory. However, if the program runs one option/case out of the several cases based on the user input, it is impractical to load the code for all the cases, other cases my never be called for execution. So a virtual memory technique known as demand paging is used to load only those pages of the process when they are required or whenever there is a demand for the page occurs during the program execution.

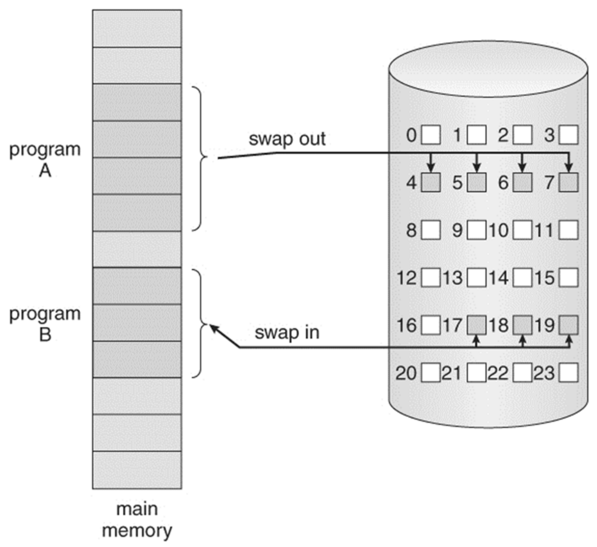


Figure 1.15: Example showing Demand Paging [1].

In Figure 1.15 shows and example of demand paging where pages 4, 5, 6 and 7 of Program A is swapped out of memory and pages 17, 18 and 19 of Program B is moved in to the memory because of the demand for the pages 17, 18 and 19. The method is implemented by a *pager* program responsible for demand paging.

1.10 PAGE REPLACEMENT ALGORITHMS

Since operating system allows virtual memory to be larger than the main memory, as a result a page fault may occur. A page fault occurs when a running process tries to accesses a memory page that is not loaded in main memory. In the event of a page fault, the operating system may have to replace an existing page with the new page. A page replacement algorithm is required in an operating system that utilizes paging for memory management. It determines which page has to be replaced when a new page arrives. Different page replacement algorithms offer various methods for determining which pages to replace. All methods have the same goal: to decrease page faults.

Space for learners:

1.10.1 FIFO Page Replacement

This is the most basic algorithm for replacing pages. The operating system uses this technique to maintain track of all memory pages in a queue, with the oldest page at the top. When a page has to be replaced, the first page in the queue is removed.

For example, consider the reference string 4, 0, 2, 5, 3, 5, 4, 0, 4, 5, 2 as shown in Figure 1.16 that is the order in which the memory references for the pages will be made. Assume that the memory which can accommodate three frames/pages at a time. The replacement algorithm uses the FIFO approach that is the first page moved to memory will be the first one to be replaced, this is followed by replacing second page, third page and so on with a new page. Initially, all the frames are empty so first three references (4, 0, 2) will result in page fault and are brought into the empty frames. The next reference 5 will replace the page 4 as it was the first page moved to the memory. Similarly, reference 3 will replace page 0 as it was the second page moved to memory. The next reference 5 is already in memory so no page fault and hence no page replacement. The process continues until all the page request in the reference string are processed. The total number of page faults using FIFO page replacement algorithm is 9.

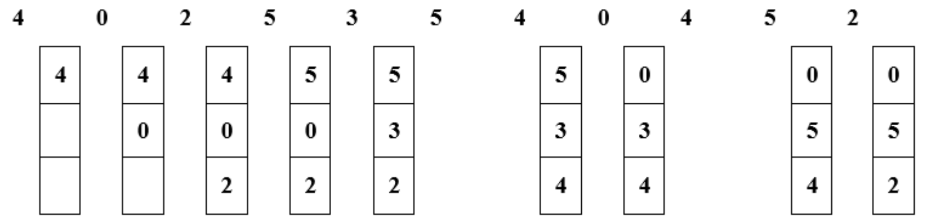


Figure 1.16: FIFO page replacement algorithm.

1.10.2 LRU Page Replacement

Least Recently Used (LRU) page replacement algorithm is a Greedy algorithm where the page to be replaced is the page which has not been used for the longest duration of time in the past. LRU keeps track of page usage over a period of time. It is based on the assumption that the pages that have been extensively utilized in the past will also be heavily used in the future.

For example, consider the reference string 4, 0, 2, 5, 3, 5, 4, 0, 4, 5, 2 as shown in Figure 1.17 that is the order in which the memory references for the pages will be made. Assume that the memory which can accommodate three frames/pages at a time. Initially, all the frames are empty so first three references (4, 0, 2) will result in page faults and are brought into the empty frames. The next reference 5 will replace the page 4 as on scanning left starting at reference 5, we find that among the pages (4, 0, 2), page 4 is the least recently used page. Similarly, the next reference 3 will replace page 0 as on scanning left starting at reference 3, we find that among the pages (5, 0, 2), page 0 is the least recently used page. The next reference 5 is already in memory so no page fault and hence no page replacement. The process continues until all the page request in the reference string are processed. The total number of page faults using LRU page replacement algorithm is 8.

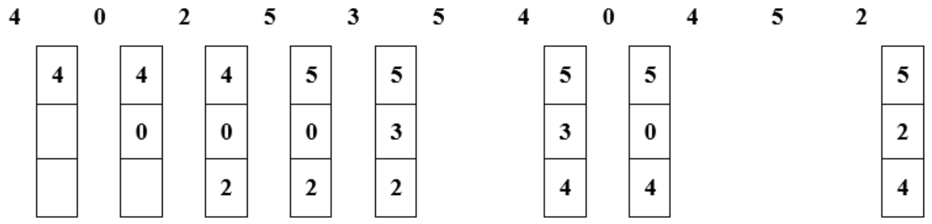


Figure 1.17: LRU page replacement algorithm.

1.10.3 Optimal Page Replacement

The best page replacement algorithm is the Optimal Page Replacement algorithm, which produces the fewest page faults. This method replaces pages that will not be utilized for the longest period of time in the future. The algorithm is difficult to implement because it requires future knowledge of the pages referenced pages.

For example, consider the reference string 4, 0, 2, 5, 3, 5, 4, 0, 4, 5, 2 as shown in Figure 1.18 that is the order in which the memory references for the pages will be made. Assume that the memory which can accommodate three frames/pages at a time. Initially, all the frames are empty so first three references (4, 0, 2) will result in page faults and are brought into the empty frames. The next reference 5 will replace the

Space for learners:

page 2 as on scanning right starting at reference 5, we find that among the pages (4, 0, 2), page 2 is not used for the longest duration of time. Similarly, the next reference 3 will replace page 0 as on scanning right starting at reference 3, we find that among the pages (4, 0, 5), page 0 is not used for the longest duration of time. The next reference 5 is already in memory so no page fault and hence no page replacement. The process continues until all the page request in the reference string are processed. The total number of page faults using LRU page replacement algorithm is 7.

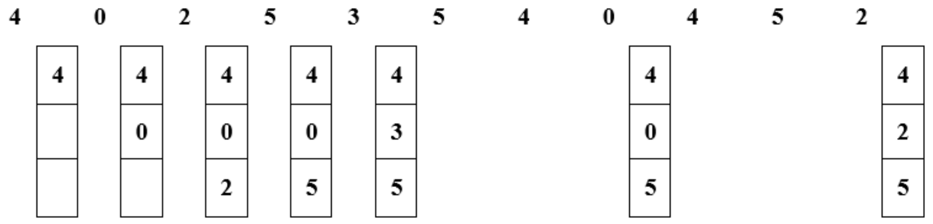


Figure 1.18: Optimal page replacement algorithm.

Space for learners:

CHECK YOUR PROGRESS

1. A memory buffer used to minimize the speed difference between CPU and Main memory is called ____________.
   a) Main memory
   b) Cache memory
   c) register
   d) disk buffer

2. Increasing the RAM improves performance because of ________________
   a) Increase in Virtual memory
   b) Bigger RAMs are faster
   c) Less page faults occur
   d) All of the above

3. Page fault occurs when
   a) Exception is thrown
   b) Requested page is not in memory
   c) Page is corrupted
   d) Requested page is in memory

4. Each logical address must be _______ than the value in limit register.
   a) less than
   b) equal to
   c) Not equal to
   d) greater than

5. Which one is the fastest memory
   a) Cache Memory
   b) Associative Memory
   c) Main Memory
   d) Secondary memory

6. Fixed-sized blocks in physical memory is called ________
   a) Block
   b) Frame
   c) Pages
   d) Segment

7. In paging CPU generated logical address has two parts _____________and _____________.
   a) Page offset & frame bit

__Space for learners:__

b) Page number & Page offset
c) Frame offset & displacement
d) Frame number & page offset

8. Paging does not suffer from ________.
   a) Internal Fragmentation
   b) External Fragmentation
   c) Both a) and b)
   d) None of the above

9. If it takes 10 milliseconds to search the TLB and 80 milliseconds to access the physical memory. If the TLB hit ratio is 0.6, the effective memory access time (in milliseconds) is _________.
   a) 120
   b) 122
   c) 134
   d) 124

10. The displacement 'd' in a logical address must be _____________
    a) Greater than segment limit
    b) Greater than the segment number
    c) Between 0 and the segment number
    d) Between 0 and segment limit

11. In segmentation, each address is specified by _____________
    a) A key and value
    b) A displacement and value
    c) A segment number & displacement
    d) A value and segment number

12. The virtual memory manager loads only those component of a program during execution as a when required is known as
    a) Segmentation
    b) Swapping

Space for learners:

c) Virtual memory
d) Demand Paging

Space for learners:

1.11 SUMMING UP

- The five memory types in the hierarchy are register, cache, main memory, hard disk and magnetic tapes based on access time, speed, cost and capacity of the memory.
- Cache memory acts as a buffer between the main memory and the CPU.
- Associative memory is also known as Translation look aside Buffer (TLB) is a special type of memory that is optimized to perform parallel searches on data.
- Address protection is implemented with the help of two registers, the base register and the limit register.
- Paging is a memory management scheme that is used to map CPU generated logical address of a process to physical address in main memory.
- Logical Address is the address that is generated by the CPU for a running program.
- A physical address is the actual address in the main memory.
- Paging is a memory management scheme that is used to map CPU generated logical address of a process to physical address in main memory.
- The logical address generated by the CPU is divided into two parts namely page number and displacement with the page.
- Translation look aside Buffer is a small, expensive but very fast associative memory.
- In a translation look aside buffer, if the search page is found it is called as an TLB hit if the page is not found it called as TLB miss.
- The percentage of times that a particular page number is found in the TLB is called the hit ratio.

- Segmentation is a memory management scheme similar to paging that allows a process to be stored in the main memory in noncontiguous manner.
- The mapping of the logical address <segment-number, displacement> to the physical address is achieved with the help of segment table and the segmentation hardware.
- A virtual memory management scheme allows execution of a process even if it is not completely in memory.
- A virtual memory technique known as demand paging is used to load only those pages of the process when they are required or whenever there is a demand for the page occurs during the program execution.
- A page fault occurs when a running process tries to accesses a memory page that is not loaded in main memory.
- A page replacement algorithm is required in an operating system that utilizes paging for memory management. It determines which page has to be replaced when a new page arrives.

Space for learners:

1.12 ANSWERS TO CHECK YOUR PROGRESS

i. b	ii. c	iii. b	iv. a	v. b
vi. b	vii. b	viii. b	ix. b	x. d
xi. c	xii. d			

1.13 POSSIBLE QUESTIONS

1. What is an associative memory? Why it is used?
2. How does the operating system ensure that two or more processes do not use the same address space?
3. Explain Paging memory management scheme.
4. What is hit ratio? Why page should be replaced in the memory?
5. Consider a logical address space of 16 pages of 512 words each, mapped on to a physical memory of 64 frames. How many bits are

there in the logical address? How many bits are there in the physical address?

6. If it takes 125 nanoseconds to search the TLB and 500 nanoseconds to access memory. If the hit ratio is 90%, calculate effective memory access time.
7. Assume a page size of 4K and an 18-bit logical address space. How many pages are in the system?
8. Assume that a CPU has a 16-bit logical address space with 4 logical pages. How large are the pages?
9. What is segmentation? Explain.
10. Define a virtual memory. With a neat diagram, explain the working of a virtual memory. What are the benefits of a virtual memory?
11. What is demand paging? Explain.
12. Consider logical address 1025 and the following
13. page table for some process P0. Assume a 15-bit address space with a page size of 1K. What is the physical address to which logical address 1025 will be mapped?

6
2
3

14. Consider the following segment table:

Segment	Base	Length
34	100	100
21	2500	200
0	1200	50
90	1700	300
7	500	500
2	600	50
99	650	200

Space for learners:

What are the physical address for the following logical address?

i. 0,25
ii. 2,89
iii. 90,345
iv. 34,50
v. 99,201

15. Consider the reference string 0, 3, 0, 4, 5, 3, 2, 0, 5, 4, 6, 7, 3, 4

    Find the number of Page faults in each of the following cases assuming that memory can accommodate 4 pages/frames at a time.

    i. FIFO Page Replacement
    ii. LRU Page Replacement
    iii. Optimal Page Replacement

Space for learners:

1.14 REFERENCES AND SUGGESTED READINGS

- Operating System Principles 8th edition by Abraham Silberschatz, Greg Gagne, and Peter Baer Galvin, Willey
- Operating Systems: Internals and Design Principles 9th edition by William Stallings, Pearson Education
- Madnik and Donovan, Operating systems, McGraw Hill.
- Andrew, S. Tannenbaum, Modern operating system, PHI.

UNIT 2: INPUT-OUTPUT ORGANIZATION

Space for learners:

Unit Structure:

2.1 INTRODUCTION

Input and output peripherals are the key components of a computer system. The main task of a computer system can be categorized as Input/output and processing. In a computer system, the operating system (OS) is used to manage and control the input/output devices and perform operations on the data receives from I/O devices and output it. In this chapter, we will discuss the basic input/output hardware, Input/output services and interface provided by OS, how

OS bridges the gap between Input/output hardware interface and Input/output application interface, interrupts handling, etc.

2.2 UNIT OBJECTIVES

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

- know about the I/O devices.
- understand different implementation issues related to I/O devices.
- explain how the I/O interface manages the gap between I/O devices and other units of a computer system.
- understand How to I/O devices are connected to a computer.
- learn about the I/O device controller.
- Learn how to access the I/O devices.
- explain how an OS handled interrupts and its different cases.
- understand how DMA is used to improve the throughput of a system.
- learn about the bus organization of a computer system.
- know about the functionalities of the kernel of an OS.

2.3 INPUT/OUTPUT PERIPHERALS

A computer system consists of four basic building blocks such as ALU, control unit, memory unit, and input/output unit. An input device can be defined as a hardware unit used to provide inputs into a system. The inputs may be a piece of data, information, control instruction, control signal, etc. The data or information can be of a different format – text, graphics, signals, etc., which is converted into a machine-understandable format by the input devices. The output hardware used in a computer system is keyboard, mouse, joystick, scanner, electronic pen, microphone, sensor devices, CCTV, light pen, trackball, graphic tablet, etc. The hardware peripherals used to get the output from the processor, project them or reproduce them in a human-understandable format can be defined as an output device/hardware. Output hardware's are monitor, printer, headphones, speaker, sound card, video card, plotter, screen

Space for learners:

projector, speech synthesizer, GPS, etc. input/output hardware are may be wired or wireless.

To communicate with a machine, the I/O devices are connected with connection points of a machine known as a port. The I/O devices may use a common set of wires to transfer data/signals/addresses are known as a bus. A bus system in a computer can have three different types such as – to transfer data, data bus, control bus for transferring control signals, and address bus for transferring addresses in between processor and I/O devices or memory units.

A controller is used to control the I/O devices, system buses, and ports. A processor can send data and commands to the controller for performing I/O transfer. The controller has one or more registers to hold the data and commands. To read and write the device control registers the processor uses a set of standard data-transfer instructions and thus executes the I/O requests. The I/O device ports also have four registers namely – status, control, data in, and data out registers. The bits contained in the **status register** are used to depicting the states of completeness, availability of a byte of information to be read from the **data-in** register and occurring of a device error. To start a control command or to change the mode of a device, the bits in the control register are used. Reading the contents from the data-in register a host can access the inputs and by writing into the data-out register, a host can send output. The I/O device port registers are typically 1 – 4 bytes in size.

STOP TO CONSIDER

1. Certain bits in a control register of a serial port are used to choose a communication between full-duplex and half-duplex.
2. Other bits are used to check the parity.
3. Third bits are used to set the word-length between 7-8 bits.
4. Fourth bit is used choose the supported speed of the serial port.

2.4 ACCESSING I/O DEVICES

The computer system uses a common line to connect all the I/O devices with it called the bus. The bus system allows the I/O devices to exchange information as shown in **figure 2.1.** The bus system

Space for learners:

typically consists of three different sets of lines: to transfer addresses – address

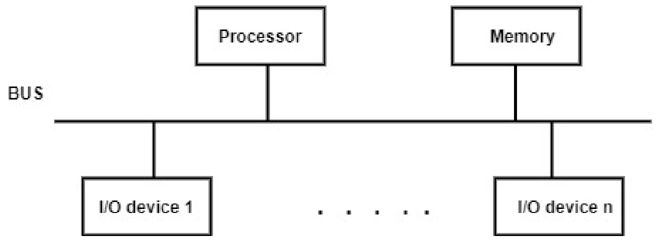


Figure 2.1: Bus structure of a computer system

bus, data – data bus, and control signals – control bus. Each I/O device has a unique set of specified addresses. When the processor placed a request by placing an address into the address line, any one of the connected devices will recognize it and respond to the command issued on the control line. During execution the processor request either a read or write command through the command line and transferred over the data bus. If the I/O device and the memory shared the same address space then the mechanism will be known as memory-mapped I/O.

The accessing speed of the I/O devices is varied from device to device and with the CPU also. The speed of the CPU is very high in comparison to the I/O devices. The CPU can execute millions of instructions when a user supplies input through an input device such as the keyboard. The CPU can execute the input character received from the keyboard, only after available in the input buffer of the keyboard interface.

The I/O interface for each I/O device provides a platform to connect between the buses and the devices such that it can communicate data to and fro between the CPU and I/O devices. It has been depicted in figure 2.2. The interfaces are consists of four different set of registers – data in, data out, status and control registers.

Space for learners:

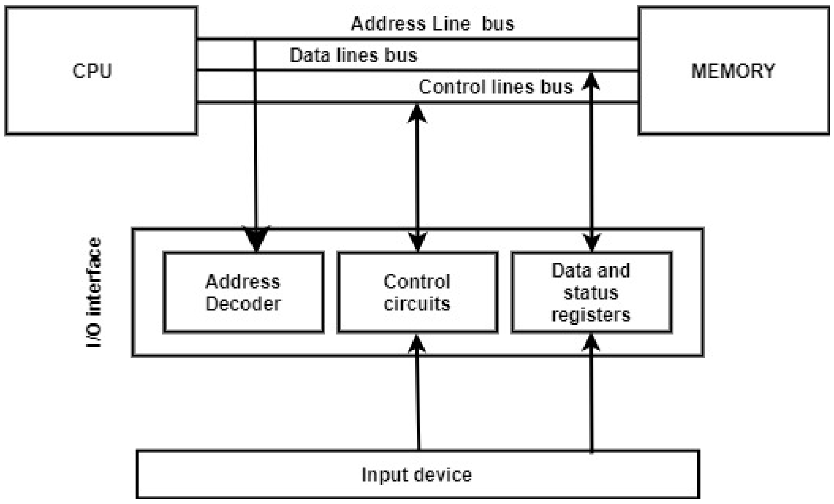


Figure 2.2: Communication of data between CPU and I/O devices

Space for learners:

An input device interface for the keyboard sets the SIN bit of the status register as 1 if it has to perform an input operation. If the data is read by the processor for execution, it will be reset to 0. The SIN bit is regularly checked and read the data register. When the processor repeatedly checks for the status flag to get the required synchronization between the I/O device and the processor, is known as *program-controlled I/O*. Two other techniques to control the I/O device are interrupt-driven I/O and Direct memory access (DMA). Interrupt-driven I/O devices use to send an interrupt request over the bus to indicate that the device is ready for transferring data. In DMA, the I/O devices can directly communicate with the memory without intervening with the CPU frequently.

2.5 POLLING

The controller can announce its status using the **busy-bit** in the **status register**. The busy bit is set to 1 to state the status of the controller as busy and set to 0 when it is ready to accept the next command. The host machines announce the availability of commands to be executed by setting the **command-ready** bit. To place a command, a host repeatedly checks the busy bit until it becomes clear. The host sets the write bit into the command register and writes a byte into the data-out register. Then the host set the command bit ready. When the controller gets the command-ready as 1, immediately it sets the busy bit. The controller will read the command register and notice the write command. The controller reads the data-out register to get the bytes of information and do the input/output to the device. After completion, the controller clears the command-ready bit, clears the error bit in the status register indicating the successful completion of the I/O operation, and clears the busy bit to indicate that the task is finished. When the host checks for the busyness of the controller by checking the status register repeatedly until the busy bit becomes clear or 0 is known as busy-waiting or polling. If the duration of a wait is long, the host may switch to another task. In many computer architectures, three CPU instruction cycles such as read a device register, logical-and to extract a status bit, and branch if not zero are sufficient to poll a device. Polling may be inefficient if the host repeatedly attempted for busy-bit but does not find any device ready for service due to the involvement of the CPU with incomplete processing. To overcome this problem, the hardware controller should inform the CPU, when the device becomes ready for service, rather than require the CPU to poll repeatedly for an I/O completion. The hardware mechanism that enables a device to notify the CPU is called an interrupt.

2.6 INTERRUPTS

When an I/O device is busy performing a task for a long time, and the processor repeatedly asks for the status of the device, the processor may not execute any necessary computation within this time and seat idle. But the CPU can perform some other necessary computations while waiting for the I/O device to become ready. It is possible by sending a signal called an ***interrupt*** to the processor. Using the concept of interrupt the waiting cycle can be eliminated and increase the throughput of the system.

Interrupts is a hardware mechanism where the CPU senses the interrupt-request line after the execution of each instruction. Interrupts can take place at any time. The I/O device controller sets the interrupts-request line to get the CPU cycle at any time. If the CPU detects an interrupt in the interrupts request line, it immediately saves the current state on a processor stack and jumps to the interrupts handler routine. Suppose the CPU executing the ***ith*** instruction during execution while the interrupt request arrives as shown in **figure 2.3**. The processor will complete the execution of the instruction ***"i"*** and load the program counter by the first instruction of the interrupt service routine. The address of the next instruction ***i+1*** will be stored onto the processor stack and after completion of the interrupt service routine, the PC will load the instruction ***i+1***.

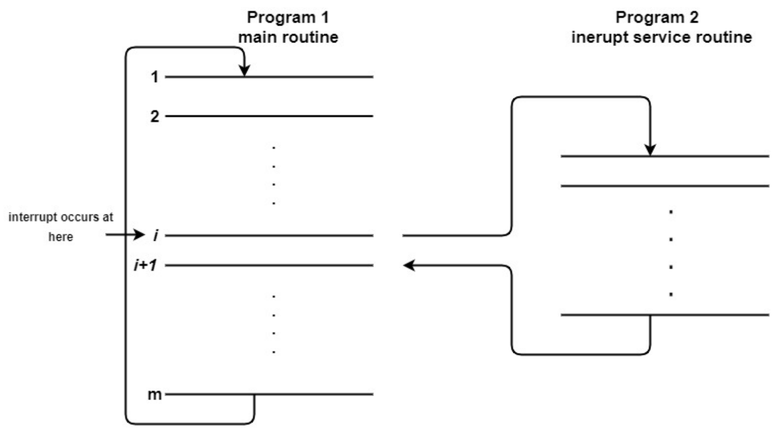
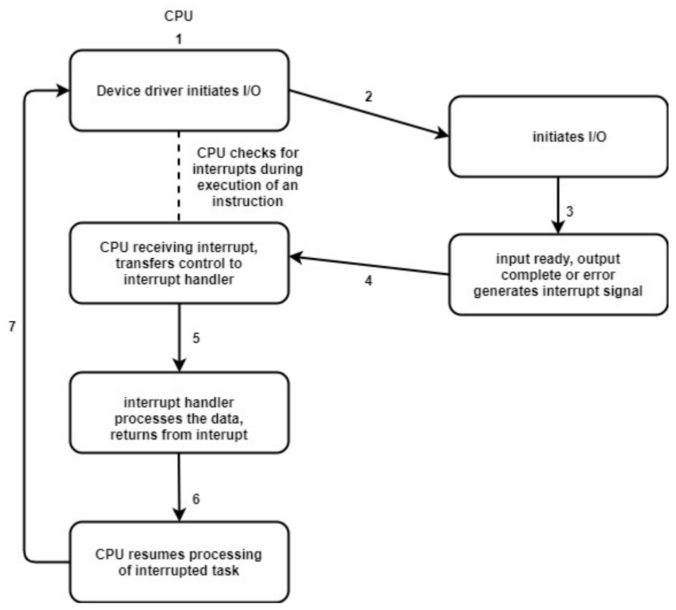


Figure 2.3: Execution of a program instruction by CPU

The interrupt handler routine determines the cause of the interrupt, performs the required operations, and executes the return-from operation to resume the CPU states before the interrupt. The following **figure 2.4** depicts a complete interrupts-driven input/output cycle. Input/output operations can be classified as synchronous and asynchronous.



CPU
1
Device driver initiates I/O
2
initiates I/O
CPU checks for interrupts during execution of an instruction
3
CPU receiving interrupt, transfers control to interrupt handler
4
input ready, output complete or error generates interrupt signal
7
5
interrupt handler processes the data, returns from interupt
6
CPU resumes processing of interrupted task

Figure 2.4: Complete interrupt driven input/output cycle

Space for learners:

In the case of a synchronous scheme, the CPU execution has to wait when the I/O devices are proceeds. But in the case of the asynchronous scheme, I/O operations can proceeds simultaneously with the CPU execution. The basic interrupt mechanism of a system allows the CPU to respond to an asynchronous event. To have an efficient input/output, the interrupt-controller hardware provides some more sophisticated features such as – ability to handle interrupt during complex processing tasks, the need to know about the interrupt initiating device without polling all the devices, and should support multilevel interrupt.

Two request lines – maskable and non-maskable are used by the CPU to identify the interrupt request type. In the case of a non-maskable interrupt, the CPU has to respond immediately by switching itself from its current execution. In case of a maskable interrupt, the CPU can turn off it such that the current execution is not interrupted.

2.6.1 Handling Multiple Devices

A computer system may be connected with several I/O devices. There is no definite order in which I/O devices can request an interrupt, as the I/O devices are operationally independent. More than one device can activate the interrupt request line at the same time. This can raise some difficulties or issues in the system:

i. How a processor can recognize the interrupt generating device.

ii. If more than one device generates interrupt and maintain different interrupt service routine, then how the processor can recognize the starting address of the appropriate routine.

iii. Is it necessary to allow another device to set the interrupt request line, while one device is already being served its interrupt service routine?

iv. How to handle more than one interrupt generated exactly at the same time.

It is possible to handle more than one interrupt generated at the same time by breaking the tie and select any one of the two for service. After completion of the service routine of the selected device, the second one can be served. Some of the ways to handle multiple interrupts are:

2.6.1.1 Polling Scheme

Space for learners:

A device can indicate its interrupt request by placing 1 in one of the bits of the status register. The particular bit in the status register is known as IRQ. To identify the interrupt requested device, the interrupt service routine has to poll all the I/O devices connected with the bus system. For servicing, a dedicated subroutine will call a device, which encountered its IRQ bit set at first and is being served. It is easy to implement but wastes notable time during interrogating the IRQ bits of all the devices which are not requesting any service. To overcome this problem vectored interrupt mechanism has been used.

2.6.1.2 Vectored Interrupt

To reduce the time used in interrogating IRQ bits of each device in the polling scheme, vectored interrupt mechanism is used. Here the I/O device itself has to inform the CPU directly about its interrupt request. The interrupt requesting device sends the first address of its interrupt service routine to the processor over the bus as an indication of generating an interrupt. Then the processor can start the execution of the corresponding interrupt service routine from the specified starting address send by the device. This system enables the processor to recognize the interrupt request if any I/O device even activates a single interrupt request line. The location shared by the device with the processor has to be considered as the starting address of the interrupt service routine. The CPU loads the starting address onto the program counter which is known as the ***interrupt vector***. Interrupt vector typically sends by the I/O devices over the data bus and the address length ranges in between 4 – 8 bits.

The processor may not respond to the interrupt vector immediately after requesting. The interrupts requesting devices have to wait to get the acknowledgment from the processor until the completion of the current execution. The interrupt requesting device can load its interrupt vector onto the bus if the CPU is ready to read it. While the processor is ready to read the interrupt vector, it enables the interrupt acknowledgment (INTA) line. The I/O device responds to the CPU by placing the interrupt vector onto the bus and turned off the INTR signal.

2.6.1.3 Priority Interrupt

Space for learners:

When more than one device is involved in requesting an interrupt, the processor may have an arrangement to not allow other devices when one interrupt service routine is already is in process. Once an interrupt service routine is started to serve by the processor, it continues until completion of it and before the processor accepts an interrupt request from a second device. i.e. the second device has to wait until the completion of the current interrupt service routine execution. Sometimes the delay of execution may lead to an erroneous operation. Some of the waiting interrupt requests may have more priority than the executing one. To overcome this situation, the I/O devices have to arrange in a priority-based structure. Here the processor will accept the higher priority device request while servicing a device with lower priority.

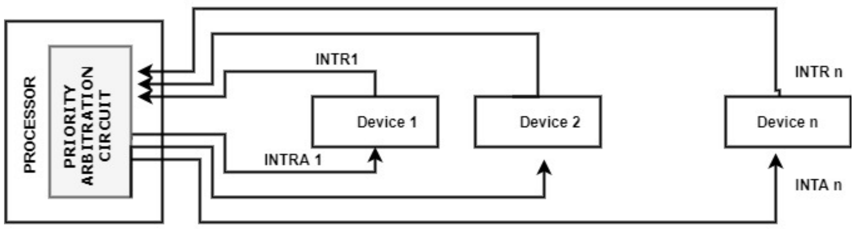


Figure 2.5: priority interrupt

If the processor accepts interrupt requests from other devices during the execution of an interrupt service routine; the accepting device will be selected based on device priority. This type of arrangement of I/O devices is known as a multiple-level priority organization. During the execution of a device request, the processor can accept interrupt requests from other devices which have a higher priority level than the current one. Once the processor has started to serve an interrupt service routine of a particular priority level, it disables interrupts from devices that have the same or lower priority level. The interrupt from higher priority devices may continue and be accepted. A multiple priority scheme can be implemented easily by using separate INTR and INTA lines for each device as shown in **figure 2.5**. In the diagram, each INTR line is assigned a different level of priority.

2.6.1.4 Daisy Chain

Space for learners:

If more than one device generates interrupts simultaneously, in a multiple-level priority organization it is clear that the processor will serve the device with the highest priority. But in the case of vectored interrupt one device is select to send the interrupt request. Another efficient mechanism to solve this problem is the ***daisy chain***. In this widely used scheme, all the I/O devices are shared a common interrupt request line for sending interrupt requests. But the processor uses only one interrupt acknowledgment (INTA) line to acknowledge the devices. The INTA line is connected in a daisy chain fashion, such as it passes through all the I/O devices as shown in **figure 2.6**. When the several I/O devices are activated the INTR line, the processor responds to it by enabling the INTA line. At first, the CPU serves device 1. If device 1 has a pending request, it will hold the INTA signal line until the operations have been completed. In the daisy chain arrangement, the device which is electronically closest to the processor has the highest priority. The second device along the chain has the second-highest priority and so on. In daisy chain arrangement the requirement of wires is less in comparison to the priority base structure as shown in **figure 2.6**.

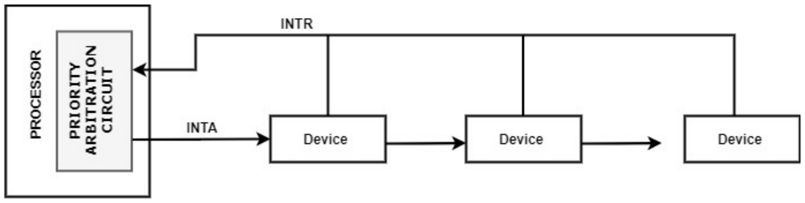


Figure 2.6: Daisy chain

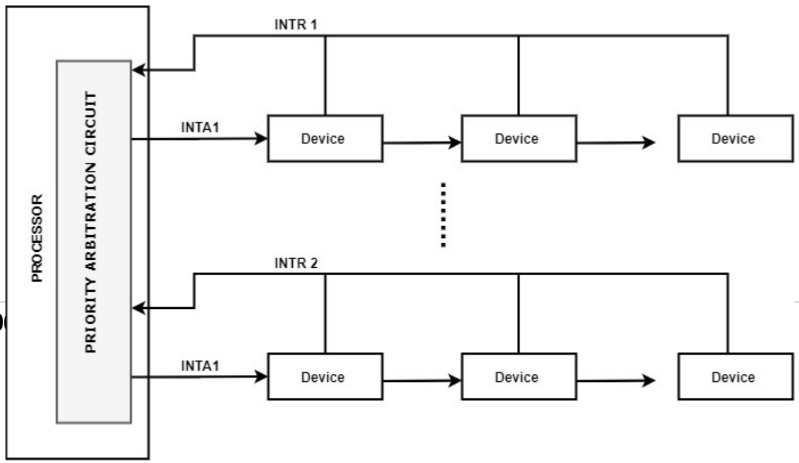


Figure 2.7: Daisy chain with priority based structure

Combining both the priority interrupt and daisy chain mechanism, a more general and useful interrupt handling mechanism can be achieved. An example of such a hybrid structure is depicted in the following diagram 2.7. Here the devices are combined to form a group of a particular priority level. Within the group, devices are connected in a daisy chain fashion.

2.8 DIRECT MEMORY ACCESS

It has been observed from the previous sections that the I/O operations are mainly concentrated on the transfer of data between the processor and I/O devices. The processor can perform this by polling a device or the device itself can send an interrupt request to the processor. During this process involvement of the processor is very high. When the processor served an interrupt service routine, several program instructions have to be executed for each data word transfer. Additionally, the processor may be busy with polling the status register of each device, instruction to increment the memory address and keep a record of the word count. When an interrupt occurred, the additional overhead associated with the saving of currently executed instruction address into a stack, load the program counter by starting address of the interrupt service routine and again resume the previous execution. For transferring a large block of data directly in between the I/O device and the main memory, a different technique may use known as Direct Memory Access (DMA). In DMA the continuous intervention of the CPU is reduced by allowing the I/O devices to access the memory unit directly under the control of a special control unit. The control unit is a part of the device interface and performed the DMA transfer. This control circuit is known as the DMA controller. In general, the I/O devices are accessing the main memory through the processor. But in the case of DMA, the role of the processor is replaced by the DMA controller. The DMA controller is responsible for providing the required memory addresses and control signals needed for data transfer.

The controller unit performs the data transfer operation without interrupting the CPU, but the complete operation is under the control of the main program executed by the processor. To start an operation, the CPU sends the starting address, data words in the block, and the data flow direction to the controller. Once the DMA controller receives this information, it started to perform the requested operation. After completion of the data transferring, the CPU is informed by the DMA through an interrupt signal and the processor removes the control from the DMA controller.

STOP TO CONSIDER

1. DMA transfers may have several attributes such as: Source address, destination address, transfer length, transfer type, block size, line stride, line length, etc.
2. DMA transfers can be categorized into two forms based on the hardware design and the involved peripheral devices, such as – single cycle DMA and burst transfer.

During the data transfer using DMA, if the current execution cannot continue by the CPU, then CPU can switch the operation to some other which is ready in the ready queue. After receiving the interrupt signal from the DMA controller, the processor can return to the process requested for data transfer.

The entire operations of input/output are always performed by the operating system. OS is responsible for suspending a program executed by the processor and starting another one. Initiation of a DMA is also a task of the OS.

For example, to transfer a data block from the main memory to disk, a dedicated program writes the starting address and the word count of the data block into the corresponding registers of the disk controller. The DMA controller performs this operation independently without intervening in the CPU. After completion of the transfer of the data block, the done bit of the status and the control register are set. Simultaneously the controller sends an interrupt request to the CPU and sets the IRQ bit. The status register is used to store the information about proper transferring of the data block or if there occurred any error during data transfer.

The priority of demanding the bus system by DMA devices is always more than the processor. Among the DMA devices, the highest speed devices are getting higher priority than the others. Memory accesses by the processor and the DMA devices are interwoven. In a computer system, most of the memory access requests are generated by the processor itself. Thus it can be said that the DMA devices are stealing the memory cycles from the CPU. This mechanism is known as ***cycle stealing***. It can be stated that the DMA devices are allowed to access the memory of a computer system exclusively to transfer a block of data without interruption and it can be defined as ***block*** or ***burst*** mode. But if the processor and the DMA controller or two DMA controllers request the main memory at the same time, then a conflict may arise. As a remedy or to resolve the conflict, a mechanism is used by the bus system to coordinate among the memory accessing devices is known as ***bus arbitration***.

2.7.1 Bus Arbitration

A device known as a ***bus-master*** is used to control the initiation of the data transfer through the bus system at any time. When the bus master relinquishes control of the bus, another device may acquire it immediately. But using the bus arbitration mechanism, the next device which is going to be the bus master will be selected and the bus mastership will be transferred. There are two arbitration processes namely centralized and distributed arbitration. A single bus arbiter is used to perform the required arbitration in centralized arbitration. In the case of distributed bus arbitration, all the devices participating in the selection process of the next bus master.

2.8 BUSES

The prime units of a computer system are interconnected through a common bus system. The common bus is used to transfer data, addresses, and control signals among the prime computer units such as memory, processor, I/O devices, and the control unit. The line required in bus arbitration and interrupt are also included in this common bus system. During transferring information a set of rules have to be followed by the buses known as protocols. A bus protocol can be defined as a set of rules to govern the behaviour of interconnected devices. There are three kinds of buses available in a system. To transfer data – data bus, to transfer addresses – address bus, and to transfer control signals – control bus.

In the control lines, a single R/W signal is used to indicate, either read or write operation to be performed on memory. If the signal bit is set to 1 means a read operation, and 0 indicates a write operation.

These lines are also used to carry time information, at what time a device will perform the read/write operation i.e. at what time a device will place data onto the bus or at what time receives data from the bus. Based on the timing of data transfer over a bus, two different categories can be obtained – synchronous and asynchronous bus systems.

In an asynchronous bus system, all the devices derived the time from a common clock. Equal time duration is assigned for each device in the synchronous bus. Each of the time intervals of equal size is known as the bus cycle. One word of data can transfer in a bus cycle.

In an asynchronous bus system, the common clock is replaced by two-time control lines such as ***Master–ready*** and ***Slave–ready***. This method is based on the use of a handshake between the master and slave. Here at first, the master indicates about the data whether it is ready for transmission or not, and second, the slave will respond to it. According to the handshaking protocol – the master will place the command information and addresses on the bus. It indicates the activation of the master–ready line and it is received by all the interconnected devices. At this point, all the devices have to decode their addresses. The slave line performs the required operation and informs the CPU by activating the slave–ready line. A full handshaking method can provide the highest degree of flexibility and reliability.

2.9 APPLICATION I/O INTERFACE

I/O interfaces are used to enable the I/O devices and treat them in a standard and uniform manner. I/O interfaces can be customized with a layer known as the device drivers. Device drivers are used to hiding the differences between the device controllers from the I/O subsystem of the kernel. The use of the driver application encapsulates the behaviour of the devices in a few generic classes that hide the hardware differences from applications. It makes the operating system (OS) independent of the hardware and simplifies the job of the OS developers. This process restricts the device manufacturer either to manufacture a product that is compatible with the existing host controller interface of the OS or write a device driver to interface the new hardware to facilitate the OS. Thus a new device can be added to a computer through an OS. For different OS

<u>Space for learners:</u>

types, the device drivers may vary. For example, a graphics driver for MS-DOS may not be supported by the OS, MS- Windows 2000, or in MAC OS.

The devices can be categorized based on the data transfer style. The ***character-stream*** device transfers the data byte by byte whereas the ***block device*** transfers a block of bytes as a unit at a time. The keyboard is an example of a ***character stream*** interface. In a ***sequential device*** data transfer occurred in a fixed order determined by the device, whereas a ***random access device*** can instruct the device to search data on any available memory location randomly. Some of the devices perform data transfer within a predictable response time known as ***synchronous devices***, whereas some of them show irregularity or in-predictable response time known as an ***asynchronous device***. A ***sharable device*** can be accessed by several processes or threads but a ***dedicated device*** cannot. Some of the devices can perform both ***read/write*** operations, but some of them can perform either read or write operations i.e. transfer of data in only one direction.

STOP TO CONSIDER

1. The device drivers are always operated within the kernel of an OS.
2. Kernel is the core part of an OS, which has direct access to the computer hardwares.
3. Device drivers are splitted into two layers such as – logical and physical layer.
4. Broad classification catrgories of device drivers are: kernel device drivers and user mode device drivers.

The block device interface will collect all the related information for accessing disk drivers and other block-oriented devices. These devices are expecting commands like *read()* or ***write()***. Random access devices can expect to have a ***seek ()*** command to locate the address of the next block to be transferred. To interact with the network devices, most of the OS including UNIX, Windows NT have used a network socket interface.

The computer system has ***clock*** and ***timer*** hardware to provide some basic functions such as current time, elapsed time, and a timer to perform trigger operations. The hardware used to maintain the

<u>Space for learners:</u>

trigger and the elapsed time is known as a ***programmable interval timer***. The OS provides an interface to the user to control the timer. During the power cut or shut-down mode of a system, a CMOS cell is used to supply power to the clock and timer.

2.10 KERNEL I/O SUBSYSTEM

The kernel of an OS provides lots of functionalities related to input/output such as – scheduling, caching, buffering, spooling, device reservation, error handling, etc. The kernel also protects the system from malicious software and errant processes.

2.10.1 I/O Scheduling

The kernel subsystem scheduled the I/O request such that the devices can perform their operations in an ordered manner. I/O scheduling can improve the system performance by fairly distributing the devices among the processes and thus improve the average waiting time. To implement I/O scheduling a wait queue containing I/O requests for the devices to be maintained. If an application is issued a blocking I/O system call, then the I/O request will be kept in the wait queue for that particular device. The I/O scheduler may rearrange the contents of the wait queue to improve the system performance and average access time experienced by the applications. In the case of an asynchronous I/O, the I/O scheduling has to keep track of many I/O requests simultaneously. The efficiency of a computer system can be improved by using other techniques that use storage in main memory or a disk via buffering, caching, and spooling.

STOP TO CONSIDER

1. Different scheduling algorithms are used by an OS to scheduled the operations of I/O devices.
2. First Come First Serve (FCFS), Shortest Job First (SJF), Priority scheduling, Round robin etc. are the prime scheduling algorithms.

2.10.2 Buffering

Before transferring data from a device to another device or device to application, maybe store it in a memory area temporarily known as a buffer. Buffering is done due to three reasons. First, cope up with the speed mismatch between the speed of the producer and the consumer. The second, to provide adaptation for devices that have different data transfer sizes. A third use of buffering is to copy semantics for application I/O. Copying of data between kernel buffer and application data space is common in the operating system despite the overhead that this operation introduces, because of the clean semantics.

2.10.3 Caching

Cache memory is a faster memory placed in between the processor and the main memory. Caching is used to reduce the speed compatibility of the processor and the memory access time. It stores a block of data word into it which is being used by the processor shortly. The difference between buffering and caching is that in buffering an existing copy of data is hold whereas in cache a copy of data items can be store that can remain elsewhere.

2.10.4 Spooling

The output stream of data has to store in a buffer before going to an output device. It is known as spooling. A printer can be used to print the output of a process at a time, but many applications can request the printer at a time to print their output concurrently without mixing the outputs. The OS allows this by intercepting all the outputs to the printer. The output of each application is spooled into a separate disk file. Once the current printing process is being over, spooling system copied the next output to be print from the queue. It can copy one output from the queue to the printer at a time.

2.10.5 Error Handling

Using protected memory, an OS can protect a system from loss of data or information due to any errors that occurred in hardware or at the application level. Thus a system can protect from small mechanical faults. Devices and I/O data transfer may fail due to

Space for learners:

several reasons. It may be either transient such as when a network becomes overloaded or for permanent reasons such as when a disk controller becomes defective. An OS can handle transient kinds of failures effectively.

Space for learners:

CHECK YOUR PROGRESS

A. Choose the correct options for the following questions:

1. Which of the following is a major part of the time taken when accessing data on the disk?

   A. Settle time

   B. Rotational latency

   C. Seek time

   D. Waiting time

2. How does the hardware trigger an interrupt?

   A. Sending signals to CPU through the system bus

   B. Executing a special program called interrupt program

   C. Executing a special program called system program

   D. Executing a special operation called system call

3. Which operation is performed by an interrupt handler?

   A. Saving the current state of the system

   B. Loading the interrupt handling code and executing it

   C. Once done handling, bringing back the system to the original state it was before the interrupt occurred

   D. All of these

4. Which of the following is an example of the spooled device?

   A. A graphic display device

   B. A line printer used to print the output of several jobs

   C. A terminal used to enter input data to a running program

   D. A secondary storage device in a virtual memory system

5. An application loads 100 libraries at start-up. Loading each library requires exactly one disk access. The seek time of the disk to a random location is given as 10 ms. The rotational speed of the disk is 6000 rpm. If all 100 libraries are loaded from random locations on the disk, how long does it take to load all libraries? (The time to

transfer data from the disk block once the head has been positioned at the start of the block may be neglected)

A. 0.50 s

B. 1.50 s

C. 1.25 s

D. 1.00 s

6. Consider the following table of arrival time and burst time for three processes P0, P1, and P2.

Process	Arrival time	Burst Time
P0	0 ms	9 ms
P1	1 ms	4 ms
P2	2 ms	9 ms

7. The pre-emptive shortest job first scheduling algorithm is used. Scheduling is carried out only at the arrival or completion of processes. What is the average waiting time for the three processes?

A. 5.0 ms

B. 4.33 ms

C. 6.33 ms

D. 7.33 ms

8. Let the time taken to switch between user and kernel modes of execution be t1 while the time taken to switch between two processes be t2. Which of the following is TRUE? (GATE CS 2011)

A. t1 > t2
B. t1 = t2
C. t1 < t2

D. Nothing can be said about the relation between t1 and t2

9. A set of wires and a rigidly defined protocol that specifies a set of messages that can be sent on the wires.

A. Port

B. Node

C. Bus

D. None of these

Space for learners:

10. The _________ presents a uniform device-access interface to the I/O subsystem, much as system calls provide a standard interface between the application and the operating system.

A. Devices

B. Buses

C. Device drivers

D. I/O systems

11. The interrupt vector contains

A. The interrupts

B. the memory addresses of specialized interrupt handlers

C. the identifiers of interrupts

D. the device addresses

Space for learners:

2.11 SUMMING UP

- An input device can be defined as a hardware unit used to provide inputs into a system.
- The hardware peripherals used to get the output from the processor, project them or reproduce them in a human-understandable format can be defined as an output device/hardware.
- To communicate with a machine, the I/O devices are connected with connection points of a machine known as a port.
- A controller is used to control the I/O devices, system buses, and ports.
- The controller has one or more registers to hold the data and commands.
- The I/O device ports have four registers namely – status, control, data in, and data out registers.
- The I/O device port registers are typically 1 – 4 bytes in size.
- The computer system uses a common line to connect all the I/O devices with it called the bus.
- If the I/O device and the memory shared the same address space then the mechanism will be known as memory-mapped I/O.

- The CPU can execute the input character received from the keyboard, only after available in the input buffer of the keyboard interface.
- When the processor repeatedly checks for the status flag to get the required synchronization between the I/O device and the processor, is known as *program-controlled I/O*.
- The controller can announce its status using the **busy-bit** in the **status register**.
- Using the concept of interrupt the waiting cycle can be eliminated and increase the throughput of the system.
- In the case of a synchronous scheme, the CPU execution has to wait when the I/O devices are proceeds.
- In the case of the asynchronous scheme, I/O operations can proceeds simultaneously with the CPU execution.
- Two request lines – maskable and non-maskable are used by the CPU to identify the interrupt request type.
- In the case of a non-maskable interrupt, the CPU has to respond immediately by switching itself from its current execution.
- A device can indicate its interrupt request by placing 1 in one of the bits of the status register.
- The CPU loads the starting address onto the program counter which is known as the interrupt vector.
- In DMA the continuous intervention of the CPU is reduced by allowing the I/O devices to access the memory unit directly under the control of a special control unit.
- The controller unit performs the data transfer operation without interrupting the CPU, but the complete operation is under the control of the main program executed by the processor.
- The priority of demanding the bus system by DMA devices is always more than the processor.
- A device known as a bus-master is used to control the initiation of the data transfer through the bus system at any time.
- Before transferring data from a device to another device or device to application, maybe store it in a memory area temporarily known as a buffer.

Space for learners:

2.12 ANSWERS TO CHECK YOUR PROGRESS

Space for learners:

A.

1. C

2. A

3. D

4. B

5.

Answer B

Explanation: Since transfer time can be neglected, the average access time is the sums of average seek time and average rotational latency. The average seeks time for a random location time is given as 10 ms. The average rotational latency is half of the time needed for a complete rotation. It is given that 6000 rotations need 1 minute. So one rotation will take 60/6000 seconds which is 10 ms. Therefore average rotational latency is half of 10 ms, which is 5ms.

Average disk access time = seek time + rotational latency

= 10 ms + 5 ms

= 15 ms

For 100 libraries, the average disk access time will be 15*100 ms

6.

Answer A

Explanation: Process P0 is allocated processor at 0 ms as there is no other process in the ready queue. P0 is preempted after 1 ms as P1 arrives at 1 ms and burst time for P1 is less than the remaining time of P0. P1 runs for 4ms. P2 arrived at 2 ms but P1 continued as the burst time of P2 is longer than P1. After P1 completes, P0 is scheduled again as the remaining time for P0 is less than the burst time of P2.

P0 waits for 4 ms, P1 waits for 0 ms, and P2 waits for 11 ms. So average waiting time is (0+4+11)/3 = 5.

7.

Answer C

Explanation: Process switching involves a mode switch. Context switching can occur only in kernel mode.

8. C
9. C
10. B

2.13 POSSIBLE QUESTIONS

A. Answer the following questions:

1. What is an interrupt?
2. What is the use of the INTA line in a processor?
3. What is a device driver?
4. What is a socket?
5. What is the function of an interrupt handler?
6. What is the role of a scheduler in an OS?
7. What types of errors can detect and correct by an OS?
8. What is a vectored interrupt?
9. What do you mean by polling?
10. What is spooling?

B. Answer the following questions:

1. Explain different types of interrupts.
2. Explain the process how the CPU identifies an interrupt requesting device.
3. Prepare a list of devices with its priority value for a particular operating system.
4. Explain the working principle of daisy chain.
5. Explain at least two computer operations in details where DMA transfer is required.
6. How DMA transfer can be used in storing data explain.

Space for learners:

7. How DMA transfers take place when transfer a large block of data explain.
8. Explain different categories of device drivers with example.
9. How the device drivers work?
10. Explain the architectures of device drivers.
11. Explain the concept of virtual device driver.
12. Explain round robin and SJF scheduling algorithm with example.
13. How buffering is used in printing process explain briefly.
14. What do you mean by bus arbitration? Explain distributed bus arbitration with a block diagram.
15. Explain the bus structure of a computer system.
16. How daisy chain is used to handle interruptions? Explain the procedure to make a priority base daisy chain with a block diagram.
17. Explain how DMA is used in a system.
18. Explain the term memory-mapped I/O and program-controlled I/O.
19. Explain the process of accessing I/O devices.
20. What is an interrupt? Explain how multiple I/O devices can be handle by an OS?
21. Explain the concepts of bus arbitration.
22. Differentiate between polling and vectored interrupt.

Space for learners:

2.14 REFERENCES AND SUGGESTED READINGS

- Operating System Principles 8th edition by Abraham Silberschatz, Greg Gagne, and Peter Baer Galvin, Willey
- Operating Systems: Internals and Design Principles 9th edition by William Stallings, Pearson Education
- Madnik and Donovan, Operating systems, McGraw Hill.
- Andrew, S. Tannenbaum, Modern operating system, PHI.

UNIT 3: INTRODUCTION TO DEADLOCK

Space for learners:

Unit Structure:

3.1 INTRODUCTION

In this unit you will learn about basic of deadlock. Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process. There are two types of resources available preemtable and non-preemptable. Deadlock can be resource deadlock or communication deadlock. There are four conditions those must hold to occur deadlock. The resource deadlock can be modelled using resource graphs. If the resource graph contains a cycle, then it means that deadlock present. There are different strategies to deal with deadlock which will be discussed in next chapter. Starvation is another process closely related to deadlock.

3.2 UNIT OBJECTIVES

Space for learners:

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

- Understand the concept of deadlock
- Know about different types of resources
- Learn about different types of deadlock
- Learn about the four condition those must be hold to occur deadlock
- Learn about resource graph
- Learn how to modelled deadlock for single resource type
- Know about different strategies to deal with deadlock
- Learn about starvation

3.3 DEFINITION OF DEADLOCK

Computer systems are full of resources that can be used only by one process at a time. Common examples include printers, tape drives etc. If a set of processes, try to simultaneously access the same resources then sometimes situation like deadlock may arise. Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process. Because all the processes are waiting, none of them will ever cause any event that could wake up any of the other members of the set, and all the processes continue to wait forever.

Deadlock can be defined formally as follows:

A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause.

Deadlocks can also occur across machines. For example, many offices have a local area network with many computers connected to it. Often devices such as scanners, Blu-ray/DVD recorders, printers, and tape drives are connected to the network as shared resources, available to any user on any machine. If these devices can be reserved remotely, deadlocks can occur. Again for example, in a database system a program may have to lock several records it is using, to avoid race conditions. If process *A* locks record *R1* and process *B* locks record *R2* and then each process tries to lock the other one's record, then the system will be deadlock. Thus, deadlocks can occur on hardware resources or on software resources.

3.4 TYPES OF RESOURCES

A major class of deadlocks involves resources to which some process has been granted exclusive access. A computer will normally have many different resources that a process can acquire. A resource can be a hardware device (e.g. a Blu-ray drive) or a piece of information (e.g., a record in a database). A resource is anything that must be request, used, and released over the course of time.

If the resource is not available when it is requested, the requesting process has to wait. The process may wait a little while and try again or it may automatically have blocked and awakened when it becomes available. Usually a request or open system calls are provided to allow processes to explicitly ask for resources.

The resources mainly classified into two types-

a) Preemptable resources

b) Non-preemptable resources

A *preemptable resource* is the resource that can be taken away from its current owner (and given back later) without causing any effect. One preemptable resource is memory. For example, a system has 1 GB of memory, one printer and two 1-GB processes A and B. Each process A and B want to print something. At first process *A* requests and gets the printer, then starts to print. But, before it has finished the computation, it exceeds its time quantum. Process *B* now runs and tries, unsuccessfully as it turns out, to acquire the printer. Now this is a deadlock situation, because *A* has the printer and *B* has the memory, and neither one can proceed without the resource held by the other. But we can get rid of this deadlock situation because it is possible to preempt (take away) the memory from *B* by swapping it out and swapping *A* in. Now *A* can run, do its printing, and then release the printer. No deadlock occurs.

Space for learners:

A *non-preemptable resource*, in contrast, is one that cannot be taken away from its current owner without causing any effect. If a process has begun to burn a Blu-ray, suddenly taking the Blu-ray recorder away from it and giving it to another process will result in a garbled Blu-ray. Blu-ray recorders are not preemptable at an arbitrary moment.

3.5 DIFFERENT TYPES OF DEADLOCK

3.5.1 Resource Deadlock

There are different types of deadlocks. The resource deadlock is one of the common deadlock. As mentioned earlier if each member of the set of deadlocked processes is waiting for a resource (non-preemtable) that is owned by a deadlocked process then none of the processes can run, none of them can release any resources and none of them can be awakened. This kind of deadlock is called a **resource deadlock**. Here the number of processes, the number of resources possessed and requested, hardware or software resources these things are unimportant.

3.5.2 Communication Deadlocks

Another kind of deadlock can occur in communication systems (e.g. networks), in which two or more processes communicate by sending messages. For example, consider a situation where process *A* sends a request message to process *B* and then blocks until *B* sends back a reply message. Suppose the request message gets lost. Then *A* is blocked waiting for the reply and *B* is blocked waiting for a request asking it to do something. This is a deadlock situation. But as we see that there are no resources involve in the above situation, so this is not classical resource deadlock. This situation is called a communication deadlock. Since there are no resources communication deadlocks cannot be prevented by ordering the resources. Again since there are no moments when a request could be postponed communication deadlocks cannot be avoided by careful scheduling. *Timeouts* is a technique to break communication deadlock. In most of the network communication

Space for learners:

systems, whenever a sender sends a message, it also starts a timer for a specific time duration. If an acknowledgment is not received from the receiving end before the timer timeouts, then the sender has to retransmit the message again. In this way, the deadlock is broken.

Resource deadlocks can also occur in communication network. As we know that in a network when a packet comes into a router from one of its hosts, it is put into a buffer for forwarding to another router and then to another until it gets to the destination. These buffers are resources and there are a finite number of them. Now consider four routers A, B, C and D. Each one has equal numbers of buffers. Suppose that all the packets at router *A* need to go to *B* and all the packets at *B* need to go to *C* and all the packets at *C* need to go to *D* and all the packets at *D* need to go to *A*. No packet can move because there is no extra buffer at the other end and we have a classical resource deadlock.

3.6 CONDITIONS FOR RESOURCE DEADLOCKS

Coffman et al. (1971) showed that four conditions must hold to occur resource deadlock. If one of them is absent, no resource deadlock is possible.

1. The first condition is mutual exclusion condition according to which each resource is either currently assigned to exactly one process or is available.

2. The second condition is hold-and-wait condition. According to this condition processes currently holding resources that were granted earlier can request new resources.

3. The third condition is no-preemption condition. This condition holds when resources are non-preemptable i.e. resources previously granted cannot be forcibly taken away from a process. They must be explicitly released by the process holding them.

4. The fourth condition is circular wait condition. According to this condition there must be a circular list of two or more processes, each of which is waiting for a resource held by the next member of the chain.

3.7 DEADLOCK MODELLING

Holt (1972) showed how the four conditions of resource deadlock can be modelled using a directed graph as follows-

- A *circle* represents a process.
- A *square* represents a resource.
- A *directed arc from a resource (square) to a process (circle)* represents that the resource is currently held by that process. In Figure 3.1(a) resource *R* is currently assigned to process *A*.
- A *directed arc from a process to a resource* means that the process is currently blocked waiting for that resource. In Figure 3.1(b) process *B* is waiting for resource *S*.
- A *cycle* in the graph means that there is a deadlock involving the processes and resources in the cycle. In Figure 3.1(c), process *C* is waiting for resource *T*, which is currently held by process *D*. Process *D* is not about to release resource *T* because it is waiting for resource *U*, held by *C*. Both processes will wait forever.

This directed graph can be mentioned as resource allocation graph. Resource allocation graphs are a tool using which we will be able to see if a given request/release sequence leads to deadlock. The requests and releases works are performed step by step and after every step the resource allocation graph is checked to see if it contains any cycles. If so, deadlock occur; if not, there is no deadlock. Here we consider the resource allocation graphs for the case of a single resource of each type. These resource allocation graphs can also be generalized to handle multiple resources of the same type (Holt, 1972).

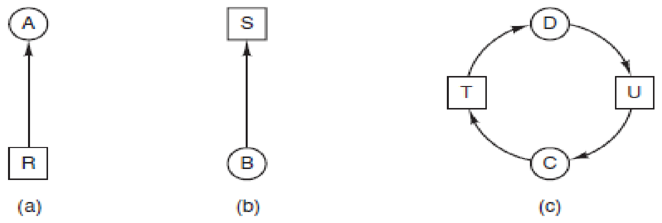


Figure 3.1: Resource allocation graphs. (a) Holding a resource (b) Requesting a resource (c) Deadlock

(Reference: "Modern Operating Systems" by Andrew S Tanenbaum)

Space for learners:

Now, how these resource allocation graphs can be used? For example, suppose there are three processes, *A*, *B*, *C* and three resources *R*, *S*, *T*. All these processes do both I/O and computing. The operating system is free to run any unblocked process at any instant. Thus, it could decide to run *A, B, C* sequentially without any pre-emption. Since all three processes are running sequentially, so there is no competition for the resources. Hence deadlock will not occur. When the processes are run sequentially, there is no possibility that while one process is waiting for I/O, another can use the CPU. Thus, running the processes strictly sequentially may not be optimal in this situation.

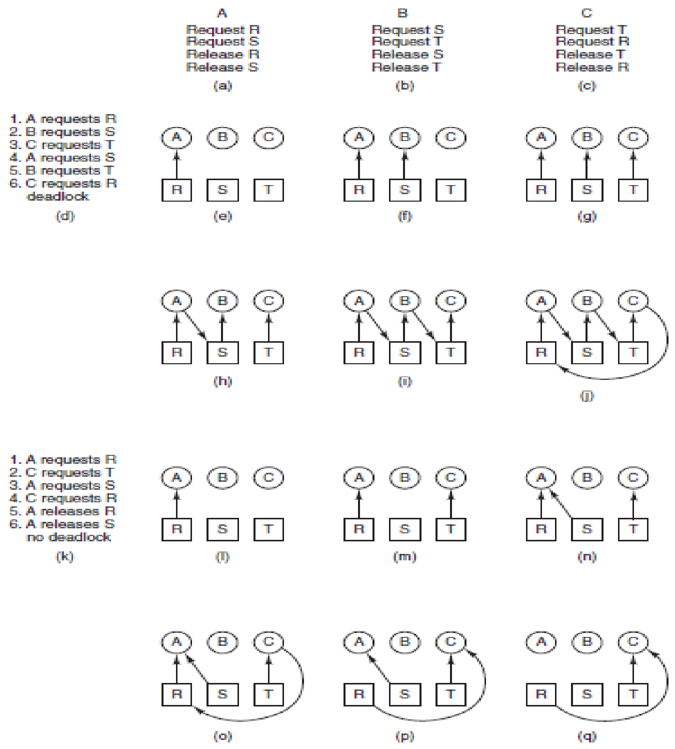


Figure 3.2: An example of how deadlock occurs and how it can be avoided.
(Reference: "Modern Operating Systems" by Andrew S Tanenbaum)

Suppose the resource requests occur in the orders as shown in Figure 3.2(d). For this order the six resulting resource allocation graphs are as shown in Figure 3.2(e)–(j). From the Figure 3.2(j) it can be conclude that this order leads to deadlock as there is a cycle A→S→B→T→C→R→A present in the graph.

Space for learners:

However, if operating system knew that granting a particular request might lead to deadlock then it can simply suspend the process without granting the request until it is safe. In this example, it could suspend *B* instead of granting it *S* as the orders shown in Figure 2.2(k). This order sequence leads to the resource allocation graphs of Figure 3.2(l)–(q), which do not lead to deadlock. After step (q), process *B* can be granted *S* because *A* is finished and *C* has everything it needs. Even if *B* blocks when requesting *T*, no deadlock can occur. *B* will just wait until *C* is finished.

3.8 STRATEGIES TO DEAL WITH DEADLOCKS

In general, there are four strategies to deal with deadlocks –

- Just ignore the problem. Maybe if you ignore it, it will ignore you.
- Detection and recovery. Let them occur, detect them, and take action.
- Dynamic avoidance by careful resource allocation.
- Prevention, by structurally negating one of the four conditions.

3.9 STARVATION

A problem closely related to deadlock is starvation. Starvation occurs if a process is indefinitely delayed. This may happen if the process wants a resource for execution which is never provided to the process or if the process is never provided the processor for some reason.

Some of the common causes of starvation are as follows –

- If a process is never allotted the resources it wants for execution.
- If high priority processes keep executing and low priority processes get blocked for indefinite time
- If there are not enough resources to provide to every process as required.

- If processes are selected randomly for execution, then a process may wait for a long time because of non-selection.

Some ways to handle starvation are as follows –

- An independent manager can be used for allocation of resources. This resource manager distributes resources fairly and tries to avoid starvation.
- Random selection of processes for resource allocation or processor allocation should be avoided as they encourage starvation.
- The priority scheme of resource allocation should include concepts such as aging, where the priority of a process is increased the longer it waits. This avoids starvation.

CHECK YOUR PROGRESS

1. What is deadlock?
2. Mention the name of different resources.
3. What is the purpose of resource allocation graph?
4. Mention the technique used to break communication deadlock.
5. What is starvation?

State TRUE or FALSE:

6. Deadlock occur when resources are pre-emptable.
7. In resource allocation graph a circle represents a process.
8. Resource allocation graphs are undirected.
9. Resource deadlock does not occur in network.
10. Starvation and deadlock are closely related.
11. A resource can be a hardware device or a piece of information.

3.10 SUMMING UP

- A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause.

Space for learners:

Space for learners:

- A resource can be a hardware device (e.g. a Blu-ray drive) or a piece of information (e.g., a record in a database).
- A resource is anything that must be request, used, and released over the course of time.
- If the resource is not available when it is requested, the requesting process has to wait.
- Usually a request or open system calls are provided to allow processes to explicitly ask for resources.
- The resources mainly classified into two types- Preemptable and nonpreemptable.
- A *preemptable resource* is the resource that can be taken away from its current owner (and given back later) without causing any effect.
- A *nonpreemptable resource*, in contrast, is one that cannot be taken away from its current owner without causing any effect.
- If each member of the set of deadlocked processes is waiting for a resource (non-preemtable) that is owned by a deadlocked process then none of the processes can run, none of them can release any resources and none of them can be awakened. This kind of deadlock is called a **resource deadlock**.
- Communication deadlock can occur in communication systems (e.g. networks), in which two or more processes communicate by sending messages.
- *Timeouts* is a technique to break communication deadlock.
- Four conditions must hold to occur resource deadlock- mutual exclusion condition, hold-and-wait condition, no-preemption condition, circular wait condition.
- Resource deadlock can be modelled using a directed graph called resource graph.
- A *cycle* in the resource allocation graph means that there is a deadlock involving the processes and resources in the cycle.
- Starvation occurs if a process is indefinitely delayed.

3.11 ANSWERS TO CHECK YOUR PROGRESS

Space for learners:

State TRUE or FALSE:

6. False.
7. True.
8. False.
9. False.
10. True.
11. True

3.12 POSSIBLE QUESTIONS

Short answer type questions:

1. Briefly explain about preemtable and nonpreemtable resources.
2. Briefly explain about different types of deadlock.
3. Mention about the four conditions which must be satisfied to occur resource deadlock.
4. From a resource allocation graph how can we conclude whether there occurs deadlock or not?
5. Mention four strategies to deal with deadlock.

Long answer type questions:

1. Briefly explain about the resource allocation graph used to modelling deadlock for single resource type each.
2. Briefly explain with the help of an example how can we use the resource graph?

3.13 REFERNCES AND 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

Space for learners:

UNIT 4: DEADLOCK PREVENTION, DETECTION AND AVOIDANCE

Unit Structure:

4.1 INTRODUCTION

In this unit, you will learn about different strategies to deal with deadlock in detail. The strategies are mainly divided into four categories- simply ignore the deadlock, early detection and recovery of deadlock, avoid deadlock and prevent deadlock. Both the detection and avoidance methods consider the facts of single resource type and multiple resource types.

4.2 UNIT OBJECTIVES

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

- Understand about different strategies to deal with deadlock
- Know about different methods to detect deadlock for both single resource type and multiple resource types
- Learn about different ways to recover from deadlock
- Learn about different ways to avoid deadlock for both single resource type and multiple resource types
- Learn about different methods to prevent deadlock

4.3 STRATEGIES TO DEAL WITH DEADLOCK

Space for learners:

4.3.1 Ignore the Problem all Together

The simplest approach is the ostrich algorithm: stick your head in the sand and pretend there is no problem. If deadlocks only occur once a year or so, it may be better to simply let them happen and reboot as necessary than to suffer the constant overhead and system performance penalties associated with deadlock prevention or detection. This is the approach that both Windows and UNIX take.

4.3.2 Deadlock Detection and Recovery

The second technique is detection and recovery. In the technique the system tries to detect the deadlock only when it is happening. After detection it will take some action to recover.

4.3.2.1 Deadlock Detection with One Resource Of Each Type

Suppose there is only one resource of each type. For example, one scanner, one Blu-ray recorder, one plotter, and one tape drive, but no more than one of each class of resource. As discussed in Unit 2, a resource allocation graph can be construct to detect deadlock in such a system.

Consider a system with seven processes *A, B, C, D, E, F, G* and six resources *R, S, U, T, V, W*. The state of the system is as follows-

- Process *A* holds *R* and wants *S*.
- Process *B* holds nothing but wants *T*.
- Process *C* holds nothing but wants *S*.
- Process *D* holds *U* and wants *S* and *T*.
- Process *E* holds *T* and wants *V*.
- Process *F* holds *W* and wants *S*.
- Process *G* holds *V* and wants *U*.

The question is: ‘‘Is this system deadlocked, and if so, which processes are involved?’’

The resource allocation graph for this system will be as follows-

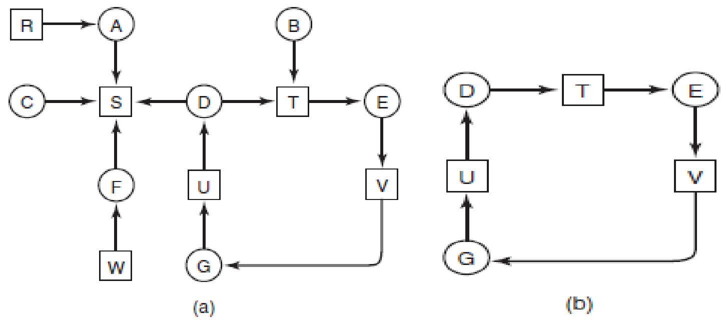


Figure 4.1 (a) Resource allocation graph (b) Cycle from (a) (Reference: "Modern Operating Systems" by Andrew S Tanenbaum)

We can see that the graph in Figure 4.1 (a) contains one cycle and processes *D*, *E* and *G* are all deadlocked. But Processes *A*, *C* and *F* are not deadlocked because *S* can be allocated to any one of them, which then finishes and returns it. Then the other two can take it in turn and also complete.

Now to detect deadlock using this resource allocation graph, it is needed to detect cycle in a directed graph. The following algorithm can detect cycles in a directed graph. The algorithm uses one dynamic data structure *L,* a list of nodes, as well as a list of arcs. The arcs are either marked or unmarked. Marked arc means it is already visited and unmarked arc means it is not visited yet. The steps are as follows-

<u>Algorithm 4.1</u>

```
Step 1:  For each node N, in the graph, perform the following steps
         with N as the starting node.
Step 2:  Initialize L to the empty list and all arcs remain unmarked.
Step 3:  Add the current node to the end of L and check to see if
         the node now appears in L two times. If it does, the graph
         contains a cycle and the algorithm terminates.
Step 4:  From the given node, see if there are any unmarked
         outgoing arcs. If so, go to Step 5; if not, go to Step 6.
Step 5:  Pick an unmarked outgoing arc at random and mark it.
         Then follow it to the new current node and go to Step 3.
```

<u>Space for learners:</u>

Step 6: If this node is the initial node, the graph does not contain any cycles and the algorithm terminates. Otherwise, we have now reached a dead end. Remove it and go back to the previous node, that is, the one that was current just before this one, make that one the current node and go to Step 3.

Working of Algorithm 4.1 on the resource allocation graph of Figure 4.1(a)

Iteration 1:

Let us first start the algorithm from a randomly selected node *R* (Step 1). After that successively consider A, B, C, S, D, T, E, F as the starting node.

Initialize *L* as empty list (Step 2).

Add *R* to the *L* and move through the only unmarked outgoing arc to node *A*. Add *A* to *L*. Thus *L* becomes *L* = [*R*, *A*] (Step 3, Step 4, Step 5).

From *A* go to node *S*. *L* becomes *L* = [*R*, *A*, *S*] (Repeat Step 3, Step 4, Step 5).

S has no outgoing arcs, so it is a dead end. Thus backtrack to node *A* (Repeat Step 3, Step 4, Step 6).

A has no unmarked outgoing arcs, so backtrack to *R*. Since *R* is the starting node so inspection of *R* has been completed (Repeat Step 3, Step 4, Step 6).

Iteration 2:

In second iteration start the algorithm from the randomly selected node *A* (Step 1).
Initialize *L* as empty list (Step 2).

Add *A* to *L*. From *A* move through the only unmarked outgoing arc to node *S*. *L* becomes *L* = [*A*, *S*] (Step 3, Step 4, Step 5).

Space for learners:

S has no outgoing arcs, so it is a dead end. Thus backtrack to node *A*. *A* is the starting node so inspection of *A* has been completed (Repeat Step 3, Step 4, Step 6).

Iteration 3:

In third iteration start the algorithm from the randomly selected node *B* (Step 1).

Initialize *L* as empty list (Step 2).

Add *B* to *L*. From *B* move through the only unmarked outgoing arc to node *T*. *L* becomes *L* = [*B*, *T*] (Step 3, Step 4, Step 5).

From *T* go to node *E*. *L* becomes *L* = [*B*, *T, E*] (Repeat Step 3, Step 4, Step 5).

From *E* go to node *V*. *L* becomes *L* = [*B*, *T, E, V*] (Repeat Step 3, Step 4, Step 5).

From *V* go to node *G*. *L* becomes *L* = [*B*, *T, E, V, G*] (Repeat Step 3, Step 4, Step 5).

From *G* go to node *U*. *L* becomes *L* = [*B*, *T, E, V, G, U*] (Repeat Step 3, Step 4, Step 5).

From *U* go to node *D*. *L* becomes *L* = [*B*, *T, E, V, G, U, D*] (Repeat Step 3, Step 4, Step 5).

From *D* go to node *T*. *L* becomes *L* = [*B*, *T, E, V, G, U, D, T*] (Repeat Step 3, Step 4, Step 5).

Now *T* is the current node and it appears two times in *L*. Thus, there is a cycle in the graph. So the algorithm terminates here (Step 3).

4.3.2.2 Deadlock Detection with Multiple Resources of each Type

Suppose there are multiple copies of some of the resources. For such case the deadlock detection algorithm has been discussed below-

Space for learners:

Let, n is the numbers of processes and m is the number of resource Classes. The ith process is denoted as P_i, the ith class resource is denoted as E_i.

The processes are either marked or unmarked. Initially all processes are unmarked. At the end of the algorithm if all processes are marked, then it indicates that they are able to complete and are thus not deadlocked. Other it indicates deadlock occurs.

At any instant of the algorithm some of the resources are assigned and are not available. A_i denotes the number of instances of ith class resource that are currently available (i.e. unassigned).

The ***existing resource vector*** $E= (E_1, E_2, \ldots, E_m)$ gives the total number of instances of each resource in existence. For example, if class 1 is tape drives, then $E_1=2$ means the system has two tape drives.

The ***available resource vector*** $A=(A_1, A_2, \ldots, A_m)$ gives the number of instances of each resource class that are currently available. If both of the two tape drives are assigned, A_1 will be 0.

Let, C be the ***current allocation matrix*** and R be the ***request matrix***.

C_{ij} is the number of instances of resource E_j that are held by process P_i. Similarly, R_{ij} is the number of instances of resource E_j that P_i wants.

Again, every resource is either allocated or is available.

$$\sum_{i=1}^{n} Cij + A_j = E_j$$

Suppose A and B are two vectors. Then the relation $A \leq B$ means that each element of A is less than or equal to the corresponding element of B i.e. $A \leq B$ holds if and only if $A_i \leq B_i$ for $1 \leq i \leq m$.

The steps of deadlock detection algorithm are as given below. Note that initially all processes are unmarked.

Algorithm 3.2

Step 1. Find an unmarked process P_i such that the i^{th} row of R is less than or equal to A.

Step 1.1. Add the i^{th} row of C to A.

Step 1.2 Mark the process and go back to step 1.

Step 2. If no such process exists, the algorithm terminates.

Space for learners:

Working of Algorithm 3.2

Suppose we have 3 processes (P_1, P_2, P_3) and 4 resource classes (tape drives, plotters, scanners, and Blu-ray drives). Process P_1 has one scanner, Process P_2 has two tape drives and a Blu-ray drive, Process P_3 has a plotter and two scanners. Each process needs additional resources, as shown in the matrix R of Figure 4.2.

$$E = \begin{array}{cccc} \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{Blu-rays} \\ (4 & 2 & 3 & 1) \end{array} \qquad A = \begin{array}{cccc} \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{Blu-rays} \\ (2 & 1 & 0 & 0) \end{array}$$

Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

Figure 4.2: An example of deadlock detection algorithm

(Reference: “Modern Operating Systems” by Andrew S Tanenbaum)

Iteration 1:

Suppose start the algorithm by picking the unmarked process P_1. Now, is ($R_1 \leq A$) true? No. (Step 1)

Pick another unmarked process P_2 and check whether ($R_2 \leq A$) is true. No. (Step 1)

Pick another unmarked process P_3 and check whether ($R_3 \leq A$) is true. Yes. (Step 1)

Add C_3 to A and mark P_3 and go to Step 1(Step 1.1).

Thus A becomes A= (2 2 2 0)

Iteration 2:

Space for learners:

Pick the unmarked process P_2 and check whether ($R_2 \leq A$) is true. Yes. (Step 1)

Add C_2 to A and mark P_2 and go to Step 1(Step 1.1).

Thus A becomes A= (4 2 2 1)

Iteration 3:

Pick the unmarked process P_1 and check whether ($R_1 \leq A$) is true. Yes. (Step 1)

Add C_1 to A and mark P_1 and go to Step 1(Step 1.1).

Thus A becomes A= (4 2 3 1)

Iteration 4:

No unmarked process found (Step 1).

So go to Step 2 and terminate.

At the end of the algorithm no unmarked processes remain. So there is no deadlock in the system.

4.3.2.3 Recovery from Deadlock

After detect a deadlock the next work will be recover from deadlock. Some ways to recover from deadlock are discussed below-

- ***Recovery through pre-emption***

One way to recover from deadlock is pre-emption i.e. temporarily take a resource away from its current owner and give it to another process. Recovering this way is frequently difficult or impossible.

- ***Recovery through rollback***

If the system designers and machine operators know that deadlocks are likely, they can arrange to have processes checkpointed periodically. Checkpointing a process means that its state is written to a file so that it can be restarted later. The checkpoint contains not only the memory image, but also the resource state, in other words, which resources are currently assigned to the process. To be most effective, new checkpoints should not overwrite old ones but should be written to new files, so as the process executes, a whole sequence

Space for learners:

accumulates. When a deadlock is detected, it is easy to see which resources are needed. To do the recovery, a process that owns a needed resource is rolled back to a point in time before it acquired that resource by starting at one of its earlier checkpoints. In effect, the process is reset to an earlier moment when it did not have the resource, which is now assigned to one of the deadlocked processes. If the restarted process tries to acquire the resource again, it will have to wait until it becomes available.

- ***Recovery through killing processes***

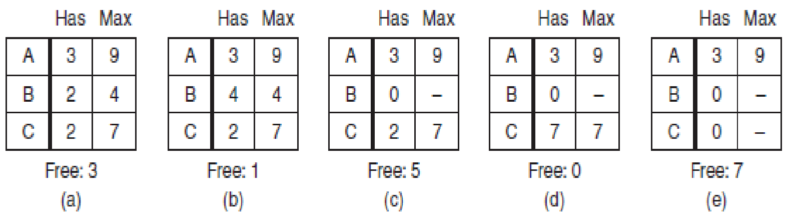
The simplest way to break a deadlock is to kill one or more processes. In this approach, the process to be killed is carefully chosen because it is holding resources that some process in the cycle needs.

4.3.3 Deadlock Avoidance

The deadlock detection methods assume that a process asks for all the necessary resources at once. But in most of the system resources are requested one at a time. In this situation the system should decide whether granting a resource is safe or not. The deadlock avoidance algorithms are designed based on the concept of safe states.

A state is said to be ***safe*** if there is some scheduling order in which every process can run to completion even if all of them suddenly request their maximum number of resources immediately. Otherwise the state is ***unsafe***. A safe state can guarantee that all processes will finish. In unsafe state, there is no sequence that guarantees completion. An unsafe state is not a deadlocked state.

For example- Suppose we have 10 resources of same type. These 10 resources are used by three processes A, B, C as shown in Figure 4.3 (a). Process *A* has 3 resources and it may need as many as 9 resources to complete it. Similarly, B has 2 resources and may need as many as 4 resources, C has 2 resources and may need as many as 7 resources to complete it. Now, the question is- “Is the state shown in Figure 4.3 (a) safe?”



(a)	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

(b)	Has	Max
A	3	9
B	4	4
C	2	7

Free: 1

(c)	Has	Max
A	3	9
B	0	–
C	2	7

Free: 5

(d)	Has	Max
A	3	9
B	0	–
C	7	7

Free: 0

(e)	Has	Max
A	3	9
B	0	–
C	0	–

Free: 7

Figure 4.3: Demonstration that the state in (a) is safe (Reference: "Modern Operating Systems" by Andrew S Tanenbaum)

Space for learners:

From Figure 4.3 (a) it is seen that all three processes already have 7 resources. So, number of available resources is 3. The maximum number of resources needed by B is 4. Thus, scheduler can run B as B needs 2 more resources to complete it (Figure 4.3 (b)). After completion B will release the resources (Figure 4.3 (c)). At this point number of available resources is 5 and A need 6 more resources to complete, C need 5 more resources to complete. In this scenario scheduler can run C (Figure 4.3 (d)). After Completion of C number of available resources will be 7 (Figure 4.3 (e)). At this point scheduler can run A. Hence the state shown in Figure 4.3 (a) is a safe state as there is an execution order B, C, A for the processes.

4.3.3.1 The Banker's Algorithm for a Single Resource Type

The ***banker's algorithm*** is a well-known deadlock avoidance algorithm. Suppose a banker will grant loans to a group of customers. The algorithm will check if granting the request leads to an unsafe state. If so, the request is denied. Otherwise the request is carried out.

For example- Suppose there are four customers *A*, *B*, *C, D* which needs total 22 credit units (Figure 4.4 (a)). The bankers assume that not all customers will need their maximum credit immediately, so he reserves only 10 credit units out of 22 credit units. Now from these 10 credit units he granted each customer a certain number of credit units as shown in Figure 4.4 (b). Thus a total of 8 units are granted and 2 units remain free. This state is safe because there is an ordered sequence like *C, D, B, A* to grant the credit units if all the customers suddenly asked for their maximum credit units.

	Has	Max
A	0	6
B	0	5
C	0	4
D	0	7

Free: 10

(a)

	Has	Max
A	1	6
B	1	5
C	2	4
D	4	7

Free: 2

(b)

	Has	Max
A	1	6
B	2	5
C	2	4
D	4	7

Free: 1

(c)

Figure 4.4 Three resource allocation states: (a) Safe. (b) Safe. (c) Unsafe
(Reference: "Modern Operating Systems" by Andrew S Tanenbaum)

Suppose at this point customer B request for one more unit and it is granted (Figure 4.4 (c)). Thus a total of 9 units are granted and 1 unit remain free. This state is an unsafe state as there is no sequence available to grant the credit units if all the customers suddenly asked for their maximum credit units.

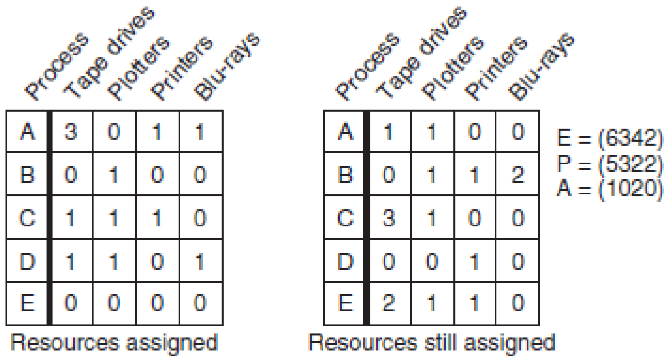
The banker's algorithm considers each request as it occurs, seeing whether granting it leads to a safe state. If it does, the request is granted; otherwise, it is postponed until later.

4.3.3.2 The Banker's Algorithm for Multiple Resource Types

The banker's algorithm can be generalized to handle multiple resources.

For example- Suppose there are 5 processes and 4 numbers of resources (Tape drives, Plotters, Printers, Blu-rays) The first matrix of Figure 3.5 is the current allocation matrix (*C*). It shows how many of each resource are currently assigned to each of the five processes. The second matrix is the request matrix (*R*) and it shows how many resources each process still needs in order to complete. The three vectors of the Figure 3.5 show the existing resources *E*, the assigned resources *P*, and the available resources *A* respectively.

<u>Space for learners:</u>



Process	Tape drives	Plotters	Printers	Blu-rays
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Resources assigned

Process	Tape drives	Plotters	Printers	Blu-rays
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources still assigned

E = (6342)
P = (5322)
A = (1020)

Figure 4.5 The banker's algorithm with multiple resources (Reference: "Modern Operating Systems" by Andrew S Tanenbaum)

Steps of Banker's algorithm for multiple resources-

Step 1. Assume that a process keeps its resources until it exits. Now search for a row in *R* which is less than or equal to *A*.

Step 1.1 If no such row exists, the system will eventually deadlock since no process can run to completion.

Step 1.2. If such a row exists, then assume that the process of the chosen row requests all the resources it needs and finishes. Mark that process as terminated and add all of its resources to the *A* vector.

Step 2. Repeat Steps 1 until either all processes are marked terminated (in which case the initial state was safe) or no process is left whose resource needs can be met (in which case the system was not safe).

Now, in Figure 4.5 the current state is safe. Suppose that process *B* makes a request for the printer. This request can be granted because the resulting state is still safe. After that process *D* can finish and then processes *A* or *E*, followed by the rest. Now imagine that after giving *B* one of the two remaining printers, *E* wants the last printer. Granting that request would reduce the vector of available resources to (1 0 0 0), which leads to deadlock, so *E*'s request must be deferred for a while.

4.3.4 Deadlock Prevention

Deadlock is essentially impossible, because it requires information about future requests, which is not known. Thus to avoid deadlock in

Space for learners:

real systems Coffman et al. (1971) provides four conditions. If we can ensure that at least one of these conditions is never satisfied, then deadlocks will be structurally impossible (Havender, 1968).

Space for learners:

- ***Attacking the Mutual-Exclusion Condition***

First let us attack the mutual exclusion condition. If no resource were ever assigned exclusively to a single process, we would never have deadlocks. Avoid assigning a resource unless absolutely necessary, and try to make sure that as few processes as possible may actually claim the resource.

- ***Attacking the Hold-and-Wait Condition***

The second of the condition stated is that- if we can prevent processes that hold resources from waiting for more resources, we can eliminate deadlocks. One way to achieve this goal is to require all processes to request all their resources before starting execution. If everything is available, the process will be allocated whatever it needs and can run to completion. If one or more resources are busy, nothing will be allocated and the process will just wait. An immediate problem with this approach is that many processes do not know how many resources they will need until they have started running. In fact, if they knew, the banker's algorithm could be used.

- ***Attacking the No-Preemption Condition***

If a process is holding some resources and requests another resource that cannot be immediately allocated to it, then all resources currently being held are pre-empted. The pre-empted resources are added to the list of resources for which the process is waiting. The process will be restarted only when the old resources are assigned to it and as well as the new resources that it is requesting.

- ***Attacking the Circular Wait Condition***

The circular wait is a scenario like there exists a set $\{P_0, P_1, ..., P_n\}$ of waiting processes such that: P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by P_2, …, P_{n-1} is waiting for a resource that is held by P_n and P_n is waiting for a resource that is held by P_0.

The circular wait can be avoided in several ways. One way is to a process is permitted only to a single resource at any moment. If it needs a second one, it must release the first one.

Another way to avoid the circular wait is to provide a global numbering of all the resources, as shown in Figure 4.6. Now the rule is- processes can request resources whenever they want to, but all requests must be made in numerical order. A process may request first a printer and then a tape drive, but it may not request first a plotter and then a printer.

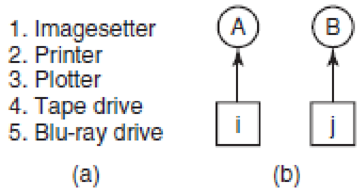


Figure 4.6 (a) Numerically ordered resources. (b) A resource graph. (Reference: "Modern Operating Systems" by Andrew S Tanenbaum)

With this rule, the resource allocation graph can never have cycles. In figure 4.6 it is seen that a deadlock will occur if and only if A requests resource j and B requests resource i. Assuming i and j are distinct resources, they will have different numbers. If $i > j$, then A is not allowed to request j because that is lower than what it already has. If $i < j$, then B is not allowed to request i because that is lower than what it already has. Either way, deadlock is impossible.

CHECK YOUR PROGRESS

1. What is existing resource vector?
2. What is available resource vector?
3.What is resource allocation matrix?
4. What is request matrix?
5. Suppose A and B are two vectors. Then what does the relation $A \leq B$ mean?

State TRUE or FALSE:

6. In safe state there is an order sequence to complete the processes.
7. Unsafe state is a deadlock state.
8. Banker's algorithm is a well-known deadlock avoidance algorithm.
9. To avoid circular wait one technique uses global numbering.

Space for learners:

4.4 SUMMING UP

Space for learners:

- Using deadlock detection algorithm, the system tries to detect deadlock detect the deadlock only when it is happening.
- A resource allocation graph can be construct to detect deadlock in a system with single resource type each.
- To detect deadlock (single resource type each) using resource allocation graph, it is needed to detect cycle in a directed graph
- Suppose there are multiple copies of some of the resources. For such case the resource allocation graph cannot detect deadlock. So, a different algorithm is needed to deadlock detection in such situation.
- The deadlock detection algorithm (multiple copies of some of the resources) considered all processes are either marked or unmarked. Initially all processes are unmarked. At the end of the algorithm if all processes are marked, then it indicates that they are able to complete and are thus not deadlocked. Other it indicates deadlock occurs.
- The ***existing resource vector*** gives the total number of instances of each resource in existence.
- The ***available resource vector*** gives the number of instances of each resource class that are currently available.
- Suppose A and B are two vectors. Then the relation $A \leq B$ means that each element of A is less than or equal to the corresponding element of B i.e. $A \leq B$ holds if and only if $A_i \leq B_i$ for $1 \leq i \leq m$.
- One way to recover from deadlock is pre-emption.
- The second way to recover from deadlock is rollback.
- The simplest way to break a deadlock is to kill one or more processes.
- The deadlock detection methods assume that a process asks for all the necessary resources at once. But in most of the system resources are requested one at a time. In this situation the system should decide whether granting a resource is safe or not.

- A state is said to be ***safe*** if there is some scheduling order in which every process can run to completion even if all of them suddenly request their maximum number of resources immediately. Otherwise the state is ***unsafe***.
- A safe state can guarantee that all processes will finish.
- In unsafe state, there is no sequence that guarantees completion.
- An unsafe state is not a deadlocked state.
- The ***banker's algorithm*** is a well-known deadlock avoidance algorithm. Suppose a banker will grant loans to a group of customers. The algorithm will check if granting the request leads to an unsafe state. If so, the request is denied. Otherwise the request is carried out.
- To avoid deadlock in real systems Coffman et al. (1971) provides four conditions.
- The first condition statedprevent deadlock is attack the mutual exclusion condition. If no resource were ever assigned exclusively to a single process, we would never have deadlocks.
- The second of the condition stated is that- if we can prevent processes that hold resources from waiting for more resources, we can eliminate deadlocks.
- The third condition is if a process is holding some resources and requests another resource that cannot be immediately allocated to it, then all resources currently being held are pre-empted.
- The fourth condition to prevent deadlock is the circular wait. It can be avoided in several ways. One way is to a process is permitted only to a single resource at any moment. If it needs a second one, it must release the first one.
- Another way to avoid the circular wait is to provide a global numbering of all the resources.

<u>*Space for learners:*</u>

4.5 ANSWERS TO CHECK YOUR PROGRESS

State TRUE or FALSE:

6. True
7. False.

8. True.

9. True.

4.6 POSSIBLE QUESTIONS

Short answer type questions:

1. What is checkpointing a process mean?
2. What is safe and unsafe state?
3. What is circular wait of processes?
4. Briefly explain two techniques to avoid circular wait of processes?

Long answer type questions:

1. Briefly explain the ways to detect deadlock with one resource of each type.
2. Briefly explain the ways to detect deadlock with multiple resource of each type.
3. Briefly explain the techniques to recovery from deadlock.
4. Briefly explain the Banker's algorithm to avoid deadlock with one resource of each type.
5. Briefly explain the Banker's algorithm to avoid deadlock with multiple resource of each type.
6. Briefly explain different ways to prevent deadlock.
7. For example- Suppose we have 12 resources of same type. These 12 resources are used by three processes A, B, C, D as shown in Figure 3.3 (a). Process *A* has 1 resources and it may need as many as 8 resources to complete it. Similarly, B has 2 resources and may need as many as 4 resources, C has 3 resources and may need as many as 6 resources to complete it, D has 4 resources and may need as many as 5 resources to complete it Now, the question is- "Is the state safe?"

4.7 REFERENCES AND SUGGESTED READINGS

- "Operating System Concepts" by Avi Silberschatz and Peter Galvin

Space for learners:

- "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

Space for learners:

UNIT 5: MULTIPROGRAMMING SYSTEM

Space for learners:

Unit Structure:

5.1 INTRODUCTION

As the name suggest, multiple process are in the main memory. In this case, CPU utilization is increased. Two or more processes reside in main memory are executed concurrently. This is done by switching the CPU from one program to another program almost instantaneously. To manage the entire resources of the system is the main motive of multiprogramming. Command processor, file system, I/O control system, and transient area are the primary components of multiprogramming system.

5.2 UNIT OBJECTIVES

After going through this unit you will be able to

- Understand the concept of multiprogramming
- Learn different scheduling algorithm used in multiprogramming
- Know the structure and operations of I/O
- Learn about the data transfer using direct memory access
- Understand memory management in multiprogramming
- Understand the algorithm for memory allocation
- Explain about paging and segmentation.

- Understand the file system Structure
- Understand the operations on File system
- Learn the file access methods

Space for learners:

5.3 BASIC CONCEPTS OF MULTIPROGRAMMING

The concept of multiprogramming depends on the capability of a computer to store instructions (programs) for long-term use. Multiple programs are loaded in main memory. Operating system assigns CPU to the first program. If that particular program needs some I/O operations then CPU instead of waiting for that program, OS allocates the next program to CPU. Once the I/O operation of the first program is completed, CPU continues with that program. In this fashion, execution takes place in multiprogramming system. The objective is to reduce CPU idle time by allowing new jobs to take over the CPU whenever the currently running job needed to wait (e.g. for user I/O).

Before multiprogramming was introduced, operating system working was very simple- it executes only one program at a time via CPU.

With the introduction of multiprogramming, operating system now executes multiple programs using different mechanism and several options existed for allocating CPU time

For decision-making two types of scheduling were introduced - **Job scheduling** and **CPU scheduling**. The selection of jobs to load into memory is termed as Job Scheduling and the selection of a job existing in memory to execute via the CPU is known as CPU scheduling. Both these decisions of Job scheduling and selection of a job are made by the operating system.

5.3.1 Process Scheduling

A process is a program which is in execution. The activity which involve removal of running process from the CPU and selection of another process based on some strategy is known as Process Scheduling. The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization and the objective of time sharing is to switch the CPU among processes so frequently that users can interact with each program while it is running. To meet these objectives, the process scheduler selects an available process (possibly from set of several available processes) for program execution on the CPU.

In Multiprogramming operating systems, process scheduling is an essential function. Multiprogramming system allows a number of process to load in main memory and scheduler decides which program to remove and to execute next.

5.3.1.1 Process Scheduling Queues

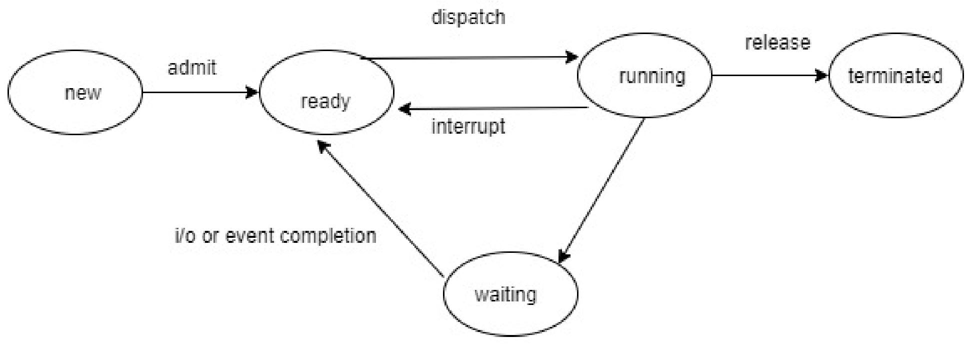
All PCBs are maintained by operating system in Process Scheduling Queues. PCB stands for Process Control Block. PCB is a data structure created by operating system. When a process is created, operating system maintains certain information in Process Control Block. A separate queue for each of the process states and PCBs are maintained by operating system. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

The Operating System maintains the following important process scheduling queues −

- **Job queue** − This queue keeps all the processes in the system.
- **Ready queue** − This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
- **Device queues** − The processes which are blocked due to unavailability of an I/O device constitute this queue.

5.3.1.2 Process State Transition Diagram

Space for learners:



PROCESS STATE TRANSITION DIAGRAM

Space for learners:

1. **Running:** The process in execution by the CPU.
2. **Waiting/Blocked:** When a process needs some I/O operation or waiting for some other event to happen, it goes into waiting/blocked state.
3. **Ready:** A process that is waiting to be executed is placed in the ready queue.
4. **New:** The process just being created is in new state. The process will be in new state until long-term scheduler brings it to the ready state.
5. **Terminated/Exit:** A process that is finished its execution or aborted due to some reason.

STOP TO CONSIDER

Process Scheduling is mainly use to Schedule the process. Many processes waiting in the ready queue are assigned to CPU one by one according to scheduling algorithm. A PCB is also called Process Descriptor. Whenever a new process is created, operating system creates a PCB which contains information like Pointer, Process state, Process state, list of open files, list of open devices, general purpose registers, priority.

5.3.2 Types of Scheduler

Following are the various scheduler used for scheduling process by the operating system.

1. Long term scheduler

Long term scheduler is responsible for creating processes and to bringing them into the system. It place processes from new state to ready queue. Since it creates processes that are why it is known as Job Scheduler also. It controls the degree of Multiprogramming. It chooses a perfect mix of IO bound and CPU bound processes among the jobs present in the pool. If maximum processes are I/O bound then CPU will remain ideal as state most of the time. This will decrease the degree of multiprogramming. So the job of long term scheduler is very critical and it may affect the system for long time.

2. Short term scheduler

This CPU scheduler is responsible for scheduling one of the processes from ready state to running state. In order to select which job is going to assigned CPU, scheduling algorithm is used.

If the selected job by Short term scheduler is CPU burst time, then for a long time processes have to wait in ready queue and a situation occur which is known as starvation.

3. Medium term scheduler

This medium term scheduler is also known as swapper as it swaps processes from main memory to secondary memory and vice versa. Processes which are IO bound are removed from the running state and placed in the waiting queue so that perfect mixes of processes are in the ready queue. Suspending and resuming of the processes is responsible of this scheduler. Various algorithms are used by operating system for this purpose.

STOP TO CONSIDER

Long term scheduler brings process from Job POOL to the main memory ready for execution. It has access to only job pool and ready queue. Whereas Short term scheduler selects job from ready queue and assigns CPU to that process. It has access to ready queue and CPU. Medium term scheduler is used to keep the flow smooth. Sometimes some process reserves memory in ready queue but may do nothing except reserving memory space. Medium term scheduler takes this process out of the memory.

Space for learners:

5.3.3 Scheduling Algorithm

Space for learners:

The Purpose of a Scheduling algorithm is:

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

The following algorithms can be used to schedule the jobs:

1. First Come First Serve

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

2. Round Robin

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

3. Shortest Job First

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

4. Shortest remaining time first

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

STOP TO CONSIDER

A Scheduling Algorithm tells us how much each processes will get CPU time. When a high priority process enters in the system, it preempts a low priority process in between and executes the high priority process first. These algorithms are either **non-preemptive or preemptive**. Non-preemptive cannot be preempted until it completes its allotted time, whereas preemptive process can be forcefully removed when a high priority process comes for execution.

Space for learners:

FCFS Scheduling

As the name implies, the processes which enters the ready queue will get the CPU first. If the arrival time of the process is lesser then sooner the job will get CPU. It may cause the problem of starvation if the burst time of the currently running process is more and thus the process waiting in the ready queue has to wait for longer time.

Advantages:

1. The logic of this algorithm is very simple; execution will be based on first cum first serve. Very easy to understand and easy to implement algorithm
2. Each and every process gets a chance to execute.

Disadvantages:

1. This algorithm is non-preemptive means the process once gets the CPU will continue till its completion.
2. The problem of starvation may occur, due to non-preemptive nature of this algorithm.
3. The average waiting time is higher as compare to other scheduling algorithms, thus poor performance.

Example

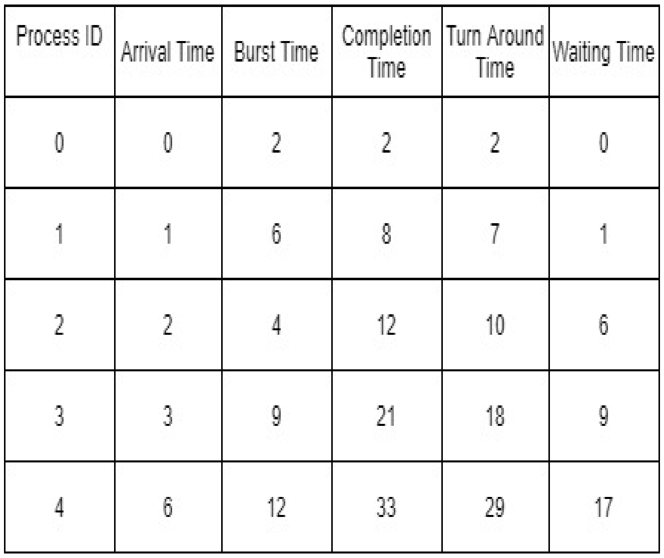
Let's consider 5 processes with process ID **P0, P1, P2, P3 and P4**. Arrival time of P0 is 0, P1 arrives at 1, P2 arrives at time 2, P3 arrives at time 3 and Process P4 arrival time is 4 in the ready queue. Arrival and Burst time of the processes are given in the following table respectively.

The Turnaround time and the waiting time are as follows-

Turn Around Time = Completion Time - Arrival Time

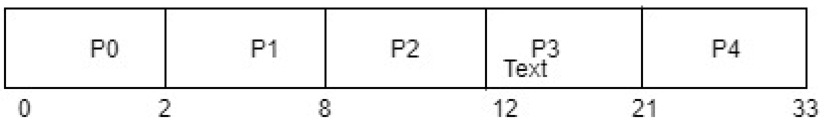
Waiting Time = Turnaround time - Burst Time

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



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

Average waiting time=31/5



GANTT CHART

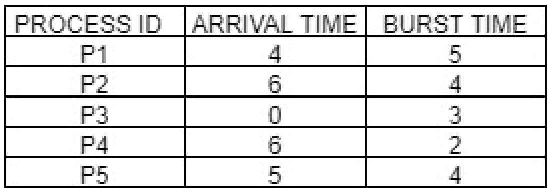
STOP TO CONSIDER

FCFS simply allocates the process according to the arrival time. Process which enters the ready queue first will be allocating the CPU. The problem of starvation may occur if the first process has largest CPU burst among all the process.

Space for learners:

CHECK YOUR PROGRESS

1. Consider five processes with arrival and burst time given. Calculate average waiting time and turnaround time



PROCESS ID	ARRIVAL TIME	BURST TIME
P1	4	5
P2	6	4
P3	0	3
P4	6	2
P5	5	4

Round Robin Algorithm

In Round Robin algorithm, the processes are dispatched in first in first out(FIFO) manner. Each processes are given limited amount of CPU time for execution in round robin fashion. This limited amount of CPU time is known as quantum time or time-slice or fixed time. If the process does not complete before CPU time expires, the CPU is preempted and given chance to next process waiting in queue.

ADVANTAGES:

1. All processes are given fair treatment.

2. Starvation does not takes place since each and every process given CPU time.

3.Simple and widely used algorithm.

DISADVANTAGES:

1. Determination of time quantum is too critical.If it is too short, it causes frequent context switching and lower CPU efficiency. If it is too big, it causes poor response time for short interactive process.
2. Process with long burst time may be starved.

Example of Round-robin Scheduling

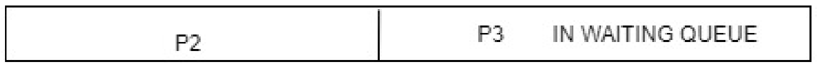
Consider this following three processes P1,P2,P3

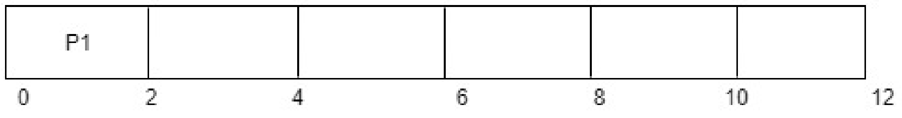
Space for learners:

PROCESS QUEUE	BURST TIME
P1	4
P2	3
P3	5

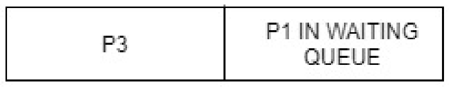
Time slice =2

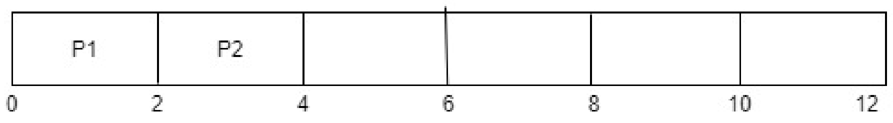
Step 1) The execution begins with process P1and it has burst time 4. As time slice given is 2 so every process executes for 2 seconds. P2 and P3 are still in the waiting queue.





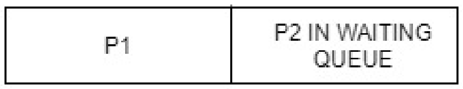
Step 2) At time =2, P2 starts executing and P1 is added to the end of the Queue

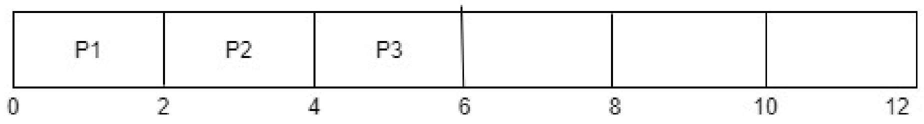




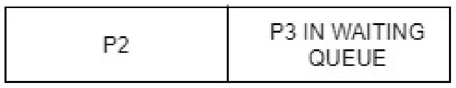
Step 3) At time=4 , P2 is preempted as its burst time is 3 and given time slice is 2 and added at the end of the queue. P3 starts executing.

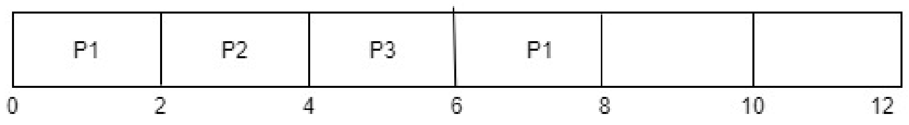
Space for learners:



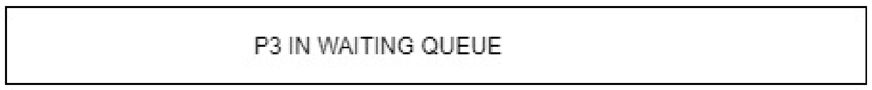


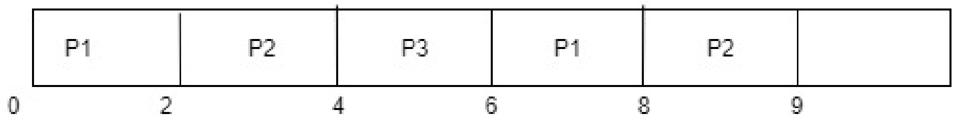
Step 4) At time=6 , P3 is preempted after time slice of 2 and add at the end of the queue. P1 starts executing.





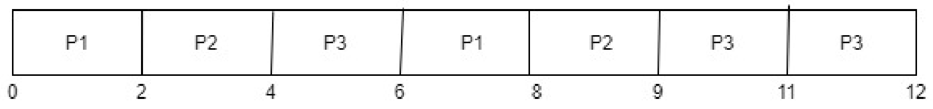
Step 5) At time=8 , P1 has a burst time of 4. It has completed execution. P2 starts execution





Step 6) P2 has a burst time of 3. P2 has already executed for 2 interval. P2 completes execution at time=9. Then, P3 starts execution till it burst time completed.

NO PROCESS IN WAITING QUEUE



Step 7) T**he** average waiting time for above example is calculated as

Space for learners:

Wait time

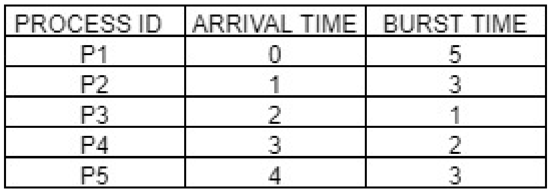
P1= 0+ 4= 4

P2= 2+4= 6

P3= 4+3= 7

CHECK YOUR PROGRESS

Q2. Consider five processes P1,P2,P3,P4,P5 with arrival and burst time. Given time quantum=2 units. Calculate Average turnaround time and average waiting time using Round Robin algorithm.



PROCESS ID	ARRIVAL TIME	BURST TIME
P1	0	5
P2	1	3
P3	2	1
P4	3	2
P5	4	3

Shortest Job First Algorithm

Out of all available (waiting) processes,it selects the process with the smallest burst time to execute next. If process with small waiting time occurs frequently then problem of starvation occurs. This can be solved with the concept of ageing. Two version of SJF exist- preemptive or non-preemptive.

Advantages:

- Minimum average waiting time and minimum turnaround time.
- Maximum throughput provided.
- It provides a standard for other algorithms incase of average waiting time.

Disadvantages:

- Processes having larger burst time may face starvation.
- It is difficult to know the length of the upcoming CPU request.
- Requires prior knowledge of how long a process or job will run.

Non-Preemptive SJF:

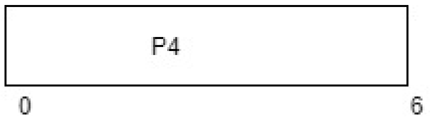
Space for learners:

Once the CPU is executing a process then that process can be released only after completing its execution. That is in the middle of execution the process cannot be released.

PROCESS QUEUE	ARRIVAL TIME	BURST TIME
P1	2	1
P2	1	5
P3	4	1
P4	0	6
P5	2	3

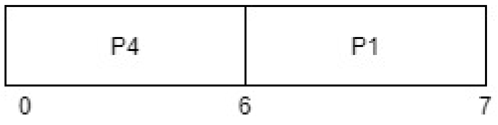
GANTT CHART

1)At T=0,P4 arrives in ready state,so CPU will be allocated to it. And it will continue till T=6 since forcefully cannot removed it.

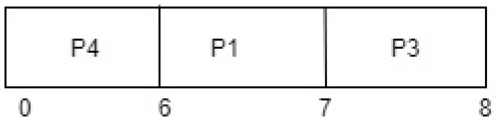


Note that in ready queue all the processes(P1,P2,P3,P5) has arrived within time period 6(which is burst time of P4).

2)At T=6, P1 arrives and it executes till T=7 since it has burst time of 1.

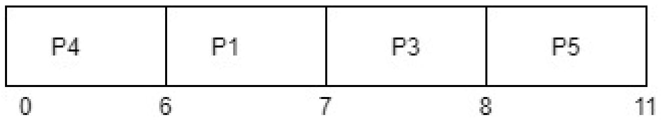


3)At T=7, P3 arrives and it executes till T=8,since it has burst time of T=8

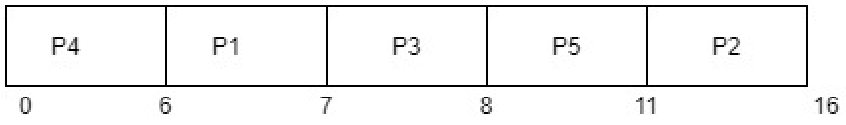


4)At T=8 ,P5 arrives and executes till T=11 since it has burst time T=3.

Space for learners:



5) At last T=11,P2 arrives and it executes till T=16 since it has burst time T=5.



At T=16, no more processes are left for execution.

Now lets see Completion time(CT),turn around time(TAT), waiting time(WT) and response time(RT).

PROCESSES	ARRIVAL TIME(AT)	BURST TIME	COMPLETION TIME	TAT=CT-AT	WT=TAT-BT	RESPONSE TIME
P1	2	1	7	5	4	4
P2	1	5	16	15	10	10
P3	4	1	8	4	3	3
P4	0	6	6	6	0	0
P5	2	3	11	9	6	6

Therefore from the above we can easily calculate average turnaround time which is 39/5=7.8 and average waiting time which is 23/5=4.6.

Note: Response time is the time at which CPU responded for first time minus arrival time. Eg for P2=11-1=10

STOP TO CONSIDER

Shortest Job First is a non preemptive algorithm. This algorithm associates with each process the length of the processes next CPU burst. The process having least next CPU burst will be one to get the CPU first.If two processes having same length arrives and waiting for CPU then FCFS scheduling takes place to break the tie.

Space for learners:

CHECK YOUR PROGRESS

Q3. Consider five processes with arrival and burst time. Calculate average waiting time. Apply Shortest job scheduling with non-preemption.

PROCESS ID	ARRIVAL TIME	BURST TIME
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

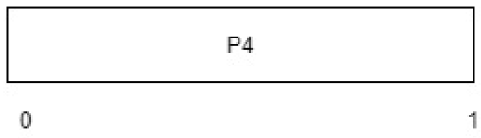
Space for learners:

PREEMPTIVE SHORTEST JOB FIRST(SJF WITH PREEMPTION)

This algorithm is also known as Shortest remaining time first(SRTF).Whenever new process arrives, there may be preemption of the running process. The process can be removed while it is executing before termination of that process. It happens if the newly arrived process has shorter burst time then the currently running process. This algorithm gives optimal solution compared to all other algorithm.

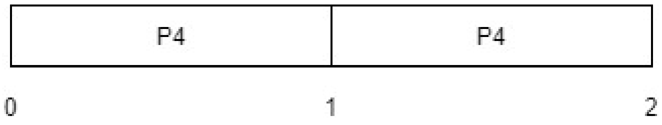
PROCESS QUEUE	ARRIVAL TIME	BURST TIME
P1	2	1
P2	1	5
P3	4	1
P4	0	6
P5	2	3

1)At time T=0,P4 arrived at ready state, so CPU is allocated to P4

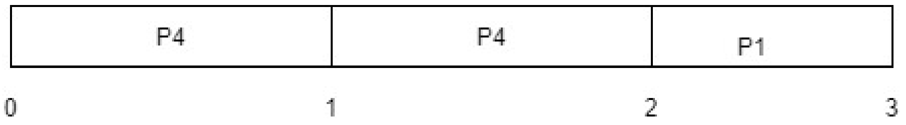


Space for learners:

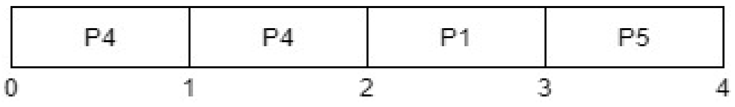
2) At time T=1,P2 arrived but since remaining burst time of P4=5 and burst time of P2=5 is same so no context switching takes place and P4 will continue till next process comes.



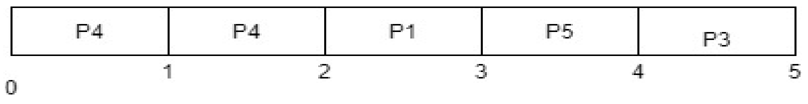
3) At T=2, P1 and P5 has arrived but since burst time of P1=1 so CPU is allocated to P1 .



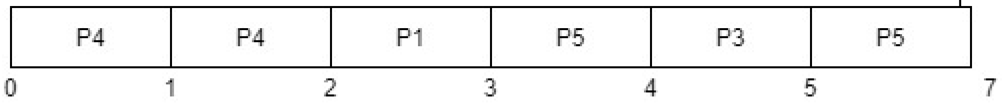
4)At T=3, P5 is allocated since it is in ready queue.



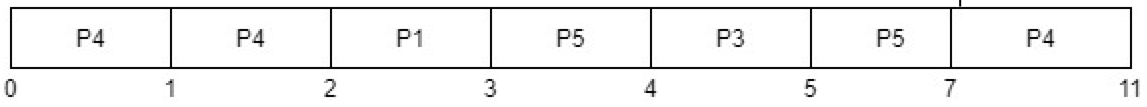
5)At T=4, P3 has arrived and is allocated the CPU.



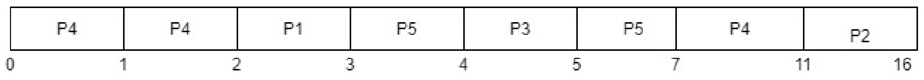
6) Till now all the processes are arrived in the ready queue, so from this time SJRN will work as same as SJF with non preemption. So at T=5, P5=2 (remaining burst time) is short of all remaing avaible process ,so P5 allocated to CPU.



7)At T=7, burst time of P4 =4 and P2=5, since P4 burst time is short so P4 is allocated to CPU.



8)At T=11 , only left process in the ready queue is P2 with burst time P2=5 and is allocated CPU.



Now lets see Completion time(CT),turn around time(TAT), waiting time(WT) and response time(RT).

PROCESSES	ARRIVAL TIME(AT)	BURST TIME	COMPLETION TIME	TAT=CT-AT	WT=TAT-BT	RESPONSE TIME
P1	2	1	3	1	0	0
P2	1	5	16	15	10	10
P3	4	1	5	1	0	0
P4	0	6	11	11	5	0
P5	2	3	7	5	2	1

Therefore from the above we can easily calculate average turnaround time which is 33/5=6.6 and average waiting time which is 17/5=3.4

STOP TO CONSIDER

In Pre-emptive Shortest Job First Scheduling., when a new process arrives with shorter burst time then currently running process is pre-empted or removed from CPU, and executed process with shorter burst time. Once completes the previous suspended process is executed.

CHECK YOUR PROGRESS

Q4. Consider five processes with arrival and burst time. Apply SJF with preemption in order to calculate average waiting time.

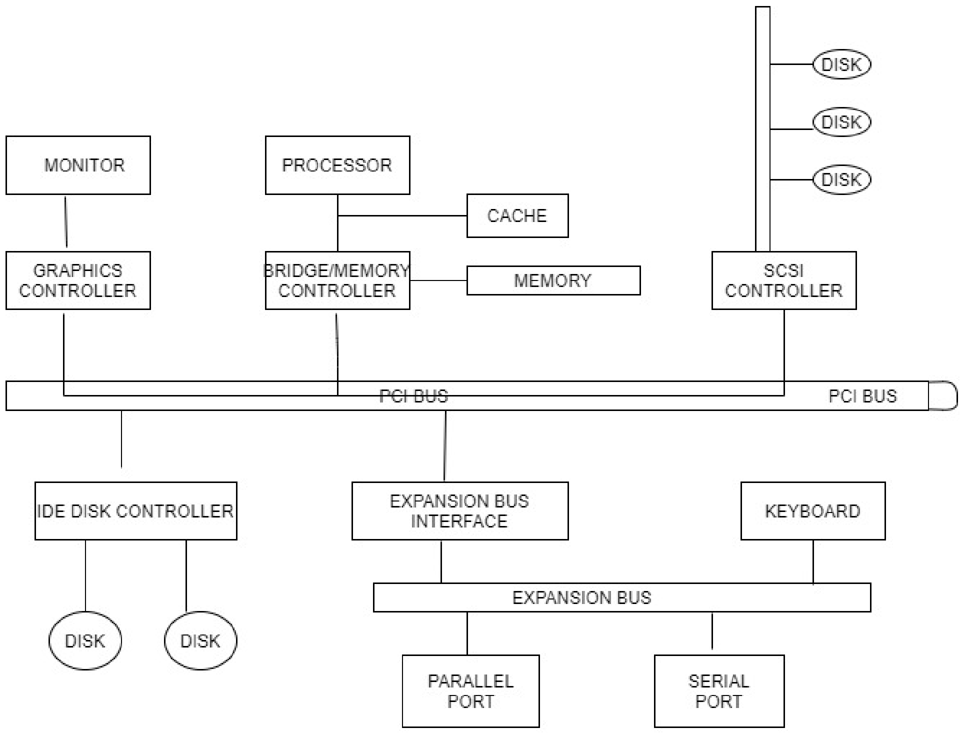
PROCESS ID	ARRIVAL TIME	BURST TIME
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

Space for learners:

5.4 I/O SUPERVISORS

5.4.1 I/O Structure

We know without I/O a computer system is useless. I/O is gateway for the outside world. There are thousands of devices, each slightly different from one another. How we will standardize the interfaces to those devices? Some devices provide single byte at a time (i.e. keyboard), other devices provide whole blocks (i.e. disks, networks etc.). Some devices must be accessed sequentially (i.e. tape), other can be accessed randomly (i.e. disk, CD etc.). Some devices require continual monitoring. Others generate the interrupts when they need service. A large portion of operating system code is devoted to managing I/O, both due to its importance to the reliability and performance of a system and since of the varying nature of the devices. A general purpose computer system consists of CPUs and multiple device controllers that are connected through a common bus. Typically, operating systems have a device driver for each device controller. The device driver understands the device controller and presents a uniform interface to the device to the rest of the operating system.



A TYPICAL PC BUS STRUCTURE

This figure depicts how different I/O devices are connected. Processor is connected with cache memory which is required to store frequently required data and instruction. Processor is connected with bridge/memory controller. Data comes from bridge to memory may be for some I/O devices. It is connected with PCI bus. PCI stands for peripheral component interconnect. This PCI bus is also known as I/O bus because it is dedicated to established communication between the system and I/O devices. Monitor is an output device and is connected via graphics controller. SCSI is a small computer system interface. With SCSI high speed hard disks are connected. SCSI provides SCSI bus and with SCSI, disks are connected. Integrated device electronics (IDE disk) where multiple disk are connected with local dedicated disk buses. Also expansion bus interfaces are available, which provides more port. Other peripheral devices are connected with the help of port. Ports are categorized into two-Serial port in which against each and every clock pulse, one

Space for learners:

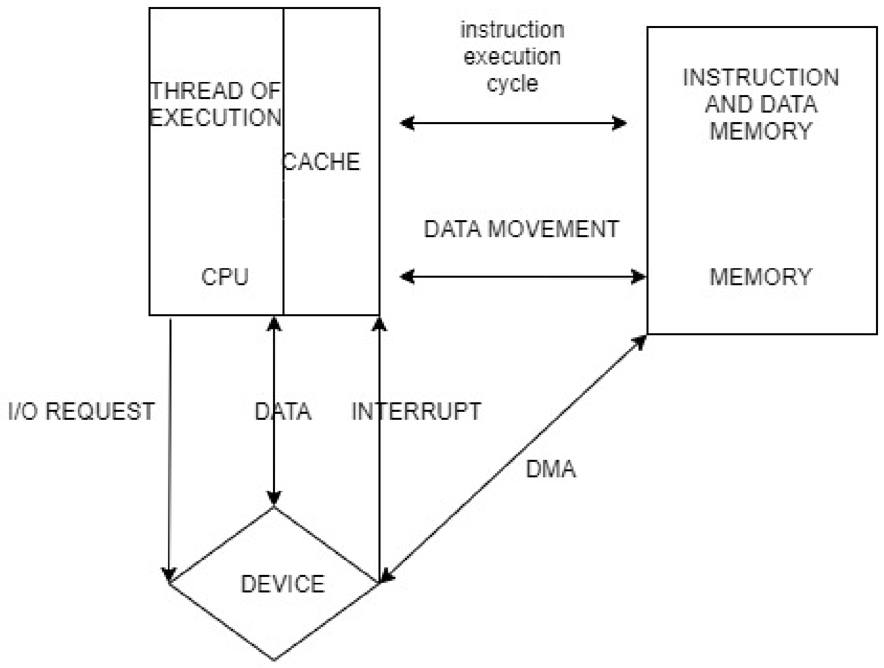
bit at a time is sent for communication. Parallel port where multiple number of bits can be transferred against each clock.

Space for learners:

STOP TO CONSIDER

In modern PC, three out of four bus types commonly found are 1.PCI bus 2.Expansion Bus. 3. SCSI Bus. High speed bandwidth devices connected to the memory subsystem (and the CPU) via **PCI Bus**, Slower low-bandwidth devices which transfers one character at a time are connected by **expansion** bus. SCSI devices are connected to a common SCSI controller via **SCSI bus**.

5.4.2 Working of an I/O Operation



WORKING OF AN I/O OPEARATION

- When any I/O operation has to be performing by any device, the respective device driver loads appropriate registers within the device controller.
- The device controller in turn examines the contents of these registers because in that registers the information or data about what is the exact I/O operation that has to perform is available.
- Whatever has to perform that data transferred to the local buffer of the device controller.
- Once the transfer of data is complete, the device controller informs the device driver that it's finished its operation via an interrupt.
- The device then returns control to the operating system.
- This form of interrupt-driven I/O is good for transferring small amount of data but its produce overhead when need to transfer large amount of data. In that case Direct Memory Access (DMA) is used. CPU is free to do other work while DMA is used.
- After setting up pointers, buffers and counters for the I/O device, the device controller transfer an entire block of data directly to or from its own buffer storage to memory with no intervention by the CPU.
- In DMA only one interrupt is generated per block to tell the device driver that the operation has completed. CPU perform other task while device controller is performing these operation.

5.4.3 Direct Memory Access

We know if we involve our CPU to perform read/write operation from the peripheral devices, then actually we are mis-utilising the potentiality of the CPU. CPU performs best if we involves CPU for the I/O device operation. But all the I/O devices are very slow; it cannot get synchronized with CPU. So CPU will have long waiting time. And CPU is not actually meant for this. So concept of DMA

Space for learners:

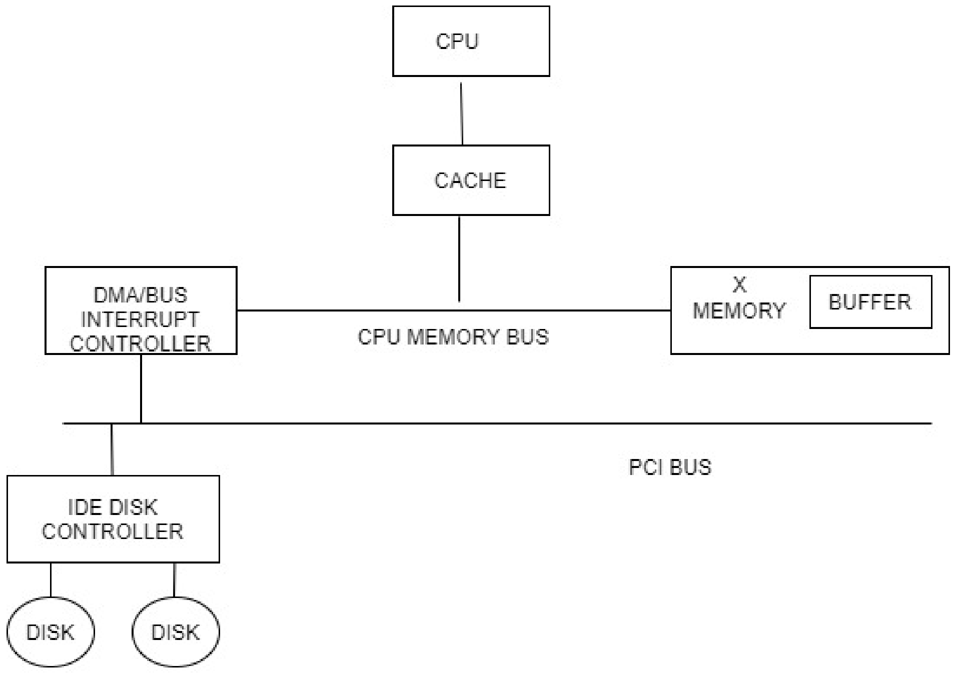
controller has come. DMA controller does the needful data transfer between I/O devices and the memory. DMA controller gets four parameters-

1. Source address from where data is to be read.
2. Target address here this data is to be transferred.
3. Bytes count that is how many data is to be transferred.
4. Whether it is read or writes operation so that it can decide that from which direction data is to be transferred.

So, after getting all this parameter the DMA controller will do the needful data transfer in between I/O device and the memory and this way CPU keeps busy in some other tasks. But it is suggested use of CPU instead of DMA if device speed is fast or if CPU has nothing to do as DMA cost is very high. There are six steps to perform data transfer-

1. Device driver is told to transfer disk data to buffer at address x.
2. Device driver tells disk controller to transfer C bytes from disk to buffer at address x
3. Disk controller initiates the DMA transfer.
4. Disk controller initiates the DMA transfer.
5. DMA controller transfers bytes to buffer x, increasing memory address and decreasing c until c=0.

Space for learners:



DATA TRANSFER USING DMA

6. When c=0, DMA interrupts CPU to signal transfer completion.

5.5 MEMORY MANAGEMENT

We know computer memory is a location of computer system used to store information. It is a very important function of operating system. In order to run any program, that program has to be load in computer memory. So computer memory is important resources to execute a program inside the computer. Therefore we should use this resource to our fullest usage, we should not keep any program in memory which is not executed or required in near future. So only those programs are to be loaded in memory which is demanded in near future. Therefore the concept of memory management comes here.

5.5.1 Binding

A) Address Binding:

1) **Compile time binding**: In this compilation, the absolute address will get embedded in executable code. That means the program knows from

Space for learners:

which address it is supposed to get loaded and where supposed to get executed.

Space for learners:

Advantage: It requires minimum set up time.

Disadvantage: If the memory space is occupied by another program, collision takes place. The current program will overwrite the previous existing program in memory.

2) **Load time address binding**: When the program gets compiled then all the addresses will be in re-locatable address. It is **the translation of the logical addresses to physical addresses at the time of loading**. The relocating loader contains the base address in the main memory from where the allocation would begin

3)**Execution time address binding**: The program has got loaded into memory might be executing but during period of time if operating system can move the program to one block of memory to another memory block then it is said to be execution time address binding. The new address will be allocated to the respective program.

B) Dynamic loading:

A program written is divided into number of modules. When the program is in execution, the program must be in main memory. All the modules at the same time are not required to bring in the main memory. Dynamic loading says load the main module at first and load the other module when they are required. That means all the module will not be jumbled up in the main memory, which may or may not be required in future.

C) Overlays:

Whenever assembly language program gets executed, assembler comes into play. Assembles does this in two phase-pass 1 and pass 2. Both the phase may not be required to be loaded in the main memory. In this overlay comes into play. It decides which pass to reside in the memory for execution.

5.5.2 Logical Address Versus Physical Address

The address generated by CPU is called logical address and the address generated by the memory unit is called physical address. Memory management unit converts logical address into physical address. Logical address space is a set of all logical address generated by CPU and the physical address space is the set of all physical address generated by the program. Logical address is Virtual and physical address is real.

5.5.3 Contiguous Memory Allocation

In contiguous memory allocation, each process is contained in single contiguous section of memory. All available memory resides at one place which means unused memory are not scattered here and there across the whole memory space.

Fixed Partitioning

Operating system uses various techniques to manage the memory. The degree of multiprogramming says that keep more and more program in main memory so that whenever CPU needs a process for execution, it is easily available. Main memory must accommodate both operating system and various user processes. The memory is divided into two partitions. One for resident operating system and one for user processes. We want several user processes to reside in memory at the same time.

. In contiguous memory allocation, one of the simplest methods is fixed size partitioning. In fixed size partitioning, memory is divided into several fixed size partition. Each partition may contain exactly one process. Degree of multiprogramming depends on the number of partition.

Variable Size Partitioning

In variable size partitioning, initially all memory is available for user processes and is considered one large block of available memory, a hole. Eventually memory contains a set of holes of various sizes. The operating system keeps a table initially which parts of memory are available and which are occupied. The memory blocks available comprise a set of holes of various sizes scattered throughout the main memory.

So whenever new process arrives, the system searches for a hole large enough for that process. If the hole is larger then what is required by the process; in that case the hole is divided into two parts. One part is allocated to arriving process and other is written to the set of holes. Now when a process terminates, it releases its block of memory and which is then placed back to set of holes. If the new hole is adjacent to other hole, then these adjacent holes are merged to form a larger hole.

Space for learners:

5.5.4 Partitioning Algorithm

Infixed partition, memory partition is fixed, once we allocate a process, the leftover space is unused. For example P0 is assigned to the first hole which is large enough for the process but the space leftover is left unused. Since every partition is allocate to one process only so the process left cannot be allocate to any other process. Thus causes internal fragmentation. So it does not make sense to use various algorithms in Fixed size partition.

The various algorithms for memory allocation for variable size partitioning are:

Here also it creates many holes but the spaces left over can be used for other process. That's why it makes sense to use various algorithm in variable size partition.

1. First fit
2. Next fit
3. Best fit
4. Worst fit

First fit: Allocate the first hole that is big enough. Scanning is done from the beginning in order to find the hole which is big enough to size of the process need to allocate.

Next fit: Here it allocates the first hole that is big enough. It is similar to first fit but scanning is done from the location at which previous search ended. Example say there are two processes. P0 is allocated first hole which is big enough, after that scanning is performed for P1 where last search ended.

Best fit: Allocate the smallest hole that is big enough thus it produces smallest leftover hole. While scanning many holes may be bigger than the size of process which we are going to load. So minimum of these holes which fits better will be taken.

Worst fit: Allocate the largest hole which is big enough. It produces largest leftover hole which may be more useful than a leftover hole from a best fit approach. After scanning one having the maximum size hole will be taken for loading the correct process.

First Fit:

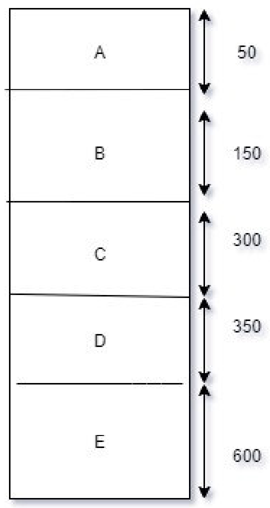


Fig 1

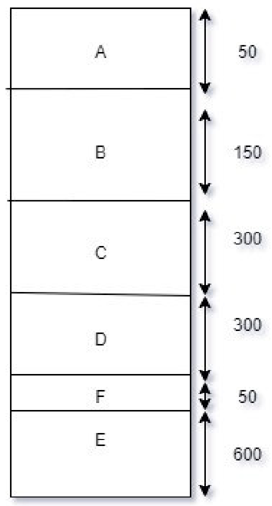


Fig 2

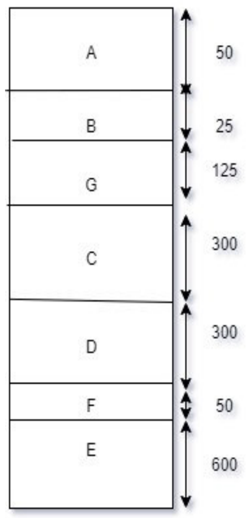


Fig 3

The figure shows memory layout. Consider A, C and E memory space is occupied by some process. B,D are empty and is consider as available memory hole. Now we have four process arriving. First P1 having memory requirement 300, second P2 having requirement 25, third P3 having 125 and fourth P4 having 50. Now we have to check whether the first fit able to satisfy the entire four requests or not.

So for first process with memory requirement 300, first fit checks the first hole empty from the beginning whether hole is greater than the arriving process size. Starting available space is 150 which is smaller then 300, so it searches for next available space i.e. 350(D) in fig 1. Since 350 is larger than 300, process P1 will be allocated to that memory and thus 350-300=50 (F)is new leftover hole(fig 2). Next process size is 25 and starting available hole is 150. Since it is satisfied so P2 is allocate to memory hole i.e. B and thus 150-25=125(G) hole is left in fig 3. Next third arriving process size is 125 and available space

Space for learners:

is also 125(G) in fig 3. Thus this memory is occupied. Last process size is 50 and also available hole is of size 50(F) in fig 3. Thus memory is occupied by P4. In this way, first fit works.

Space for learners:

Next Fit:

We have to check whether the next fit able to satisfy all the four request or not. First P1 having memory requirement 300, second P2 having requirement 25,third P3 having 125 and fourth P4 having 50.

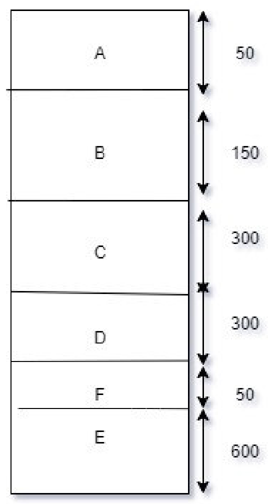


Fig 4

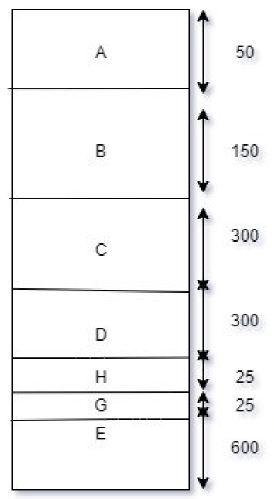


Fig 5

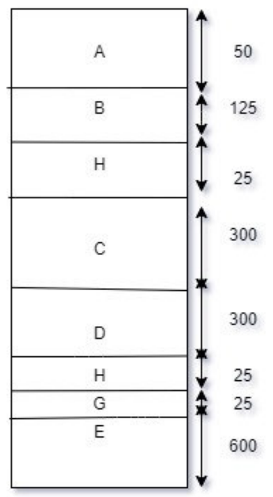


Fig 6

Consider figure 1 , Since hole D is satisfying the size of Process P1 i.e 300,so D will be occupied by Process P1,leaving 350-300=50(F) as leftover hole shown in fig 4. Now next Process is P2 with size 25. Next fit searches from where last searches took place i.e from D. Since F satisfies the process P2 memory size i.e 25.So F will be allocate to process P2 and thus 50-25=25(G) leftover hole available shown in fig 5. Next arriving process is P3 with 125 memory requirement. G is smaller so next fit searches next larger hole. B is satisfying thus P3 will be allocate to hole B and thus 150-125=25(H) leftover hole available as shown in fig 6. For the next process P4 with size 50, available hole is H (25) which is not sufficient and also G(25) which is not sufficient. Thus the last process P4 is not allocated memory by next fit algorithm.

BEST FIT:

Space for learners:

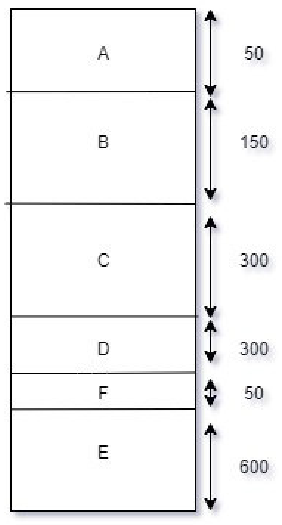


Fig 7

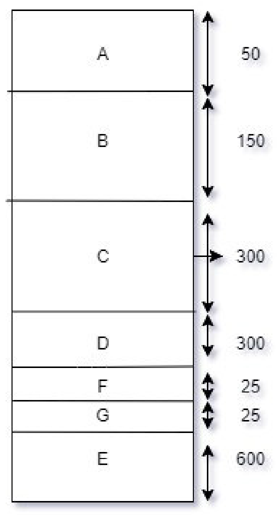


Fig 8

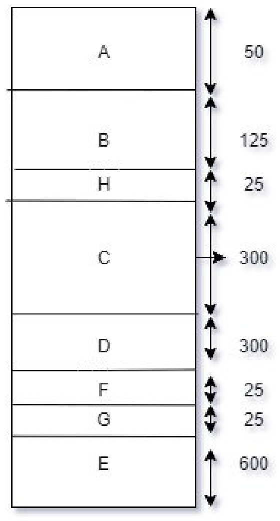
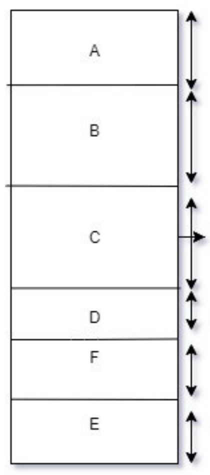


Fig 9

Initially we have two free memory hole B with 150 and D with 350 in fig 1. Consider four same process P1 with 300, P2 with 25,P3 with 125 and P4 with 150. Among this memory hole P1 occupies memory hole D leaving 350-300=50 say F hole as shown in Figure 7. Now the available memory holes are B with 150 and F with 50. Next process P2 has size 25 which both memory holes B and F satisfy. But since best fit searches smallest best hole that satisfies that process so P2 occupies space F leaving 50-25=25 say G as shown in figure 8. Now memory hole B of size 150 and G of size 25 available. Process P3 require 125 memory which is provided by B hole thus 150-125=25 left over hole say H in fig 9. Now, available memory hole are H and G of size 25 each. Process P4 with size 50 arrives but none of the holes satisfies P4 request. Two holes if we sum up it gives 50 but since they are not contiguous it is not satisfying P4 request. This is known as **External fragmentation**. So Process P4 is not allocated with memory space.

Worst Fit:



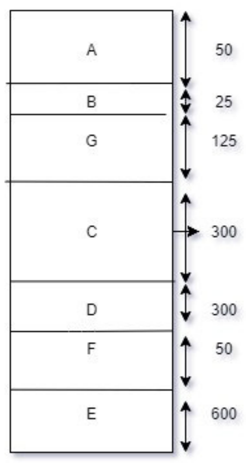


Fig 10 Fig 11

Initially holes of size 150 that is B and D of size 350 available as in fig 1. Process P1 size 300 is satisfied by hole D leaving 350-300=50 say F as shown in fig 10. Next process P2 of size 25 arrives, both hole B and F satisfies the requirement. But worst fit choose the largest hole that satisfies the process. Thus B is occupied by process P2, leaving 150-25=125 say G as shown in fig 11. Process P3 arrives with size 125 and is satisfied by G. Process P4 of size 50 is satisfied by F. Thus the entire requests are satisfied by this algorithm.

STOP TO CONSIDER

First fit scans from beginning and chooses first available block which is large enough. Best fit chooses among the free memory partition, the smallest sufficient partition among for a process. Worst fit is just the opposite of best fit, allocates the process in the largest sufficient partition among freely available partition. Next fit search for the first sufficient partition from the last allocation point.

Space for learners:

CHECK YOUR PROGRESS

Q5. 100 KB, 500 KB, 200 KB, 450 KB and 600 KB are five memory partitions in the same order. If requests for blocks of size 212 KB, 417 KB, 112 KB and 426 KB in same order comes sequentially; then which of the following algorithm makes the efficient use of memory?

A. Best fit algorithm

B. First fit algorithm

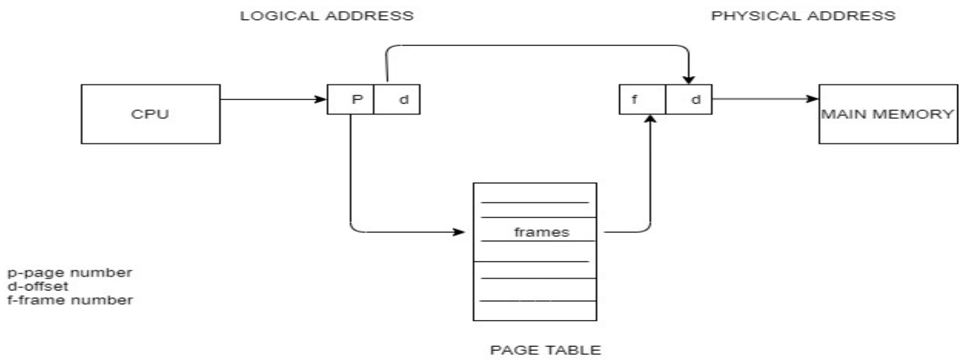
C. Next fit algorithm

D. Both next fit and best fit results in same

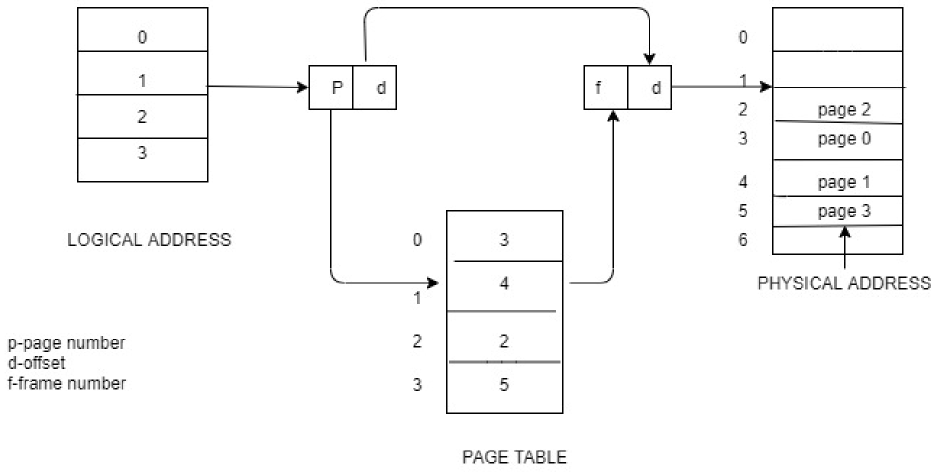
Space for learners:

5.5.5 Paging

Paging is a physical mapping. It avoids external fragmentation. The need of compaction is also reduced here. This is physical mapping in which physical memory is divided into fixed size called frames. The size of each frame is same. The logical memory is divided into fixed size block called pages. The size of each page is same in logical memory. Whenever a process to be executed, the pages are to be loaded into the frames. That means size of page is equal to size of frames. Size of pages and frames are same but number may be different. The logical address space is generated by CPU is divided into parts page number and page offset. By using page number we identify the corresponding frame in physical memory using Page table. The page table contains base address of page in the physical memory. That means in which location Page is loaded in physical memory. The page table contain frame number at which address location the base address of the page resides. Frame number is added to offset by adding simply location of physical memory.



For example, suppose we have logical memory with four pages say page 0,1,2,3. Page table contain page number 0,1,2,3. Again physical memory is supposed divided into 7 frames. In the page table page 0 contains 3,page 1 contain 4,page 2 contain 2 and page 3 contain 5 simultaneously. Page 0 contains 3 means it is the frame number where page 0 will be loaded in physical memory. Page 1 is loaded in frame 4 and so on. This is how paging works. See the diagram below for understanding of above explanation.



Advantages:

1.No external fragmentation takes place

2.Efficient use of memory

3.User’s view of memory and actual physical memory are separate. The user view memory as a simple contiguous memory that contains only one process.But the user process is non-contigous in physical memory.

Space for learners:

Disadvantages:

0 It suffers from internal fragmentation.

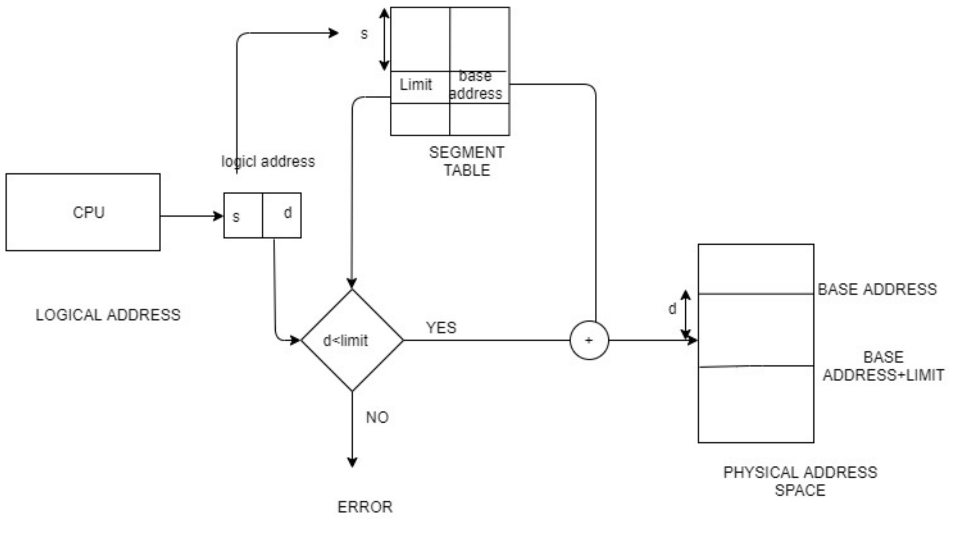
1 Page table requires extra memory. So it may not be good for a system having small RAM.

STOP TO CONSIDER

It is a fixed-size partitioning scheme**.** The secondary memory and main memory are divided into equal fixed-size partitions in Paging. It helps to avoid external fragmentation**.** The paging technique divides the main memory also called physical memory into fixed-size blocks that are known as Frames and divides the secondary memory also called logical memory into blocks of the same size that are known as Pages.

Space for learners:

5.5.6 Segmentation



Translation of 2-dimentional logical address to one dimention Physical Address

Here CPU generates logical address. Logical address is divided into two parts-segment number(s) and offset (d).

Segment number (s): It is the number of bits required to represent the segment.

Segment offset (d): It is the number of bits required to represent the size of the segment.

Each segment number and offset is stored in the form of segment table. Limit is the length of each segment and base is the starting physical address where segments reside. Each limits and base and each segment is checking whether it is less than equal to limit or not. If it is yes, the physical address will be base address added with the offset. If it is greater than segment table length register then addressing error will occur. This is how hardware implementation for the segmentation takes place

Advantages of Segmentation –

- No Internal fragmentation takes place.
- Segment Table consumes less space in comparison to Page table in paging.

Disadvantage of Segmentation –

- External fragmentation takes place when the processes are loaded and removed from the memory; the free memory space is broken into little pieces.

STOP TO CONSIDER

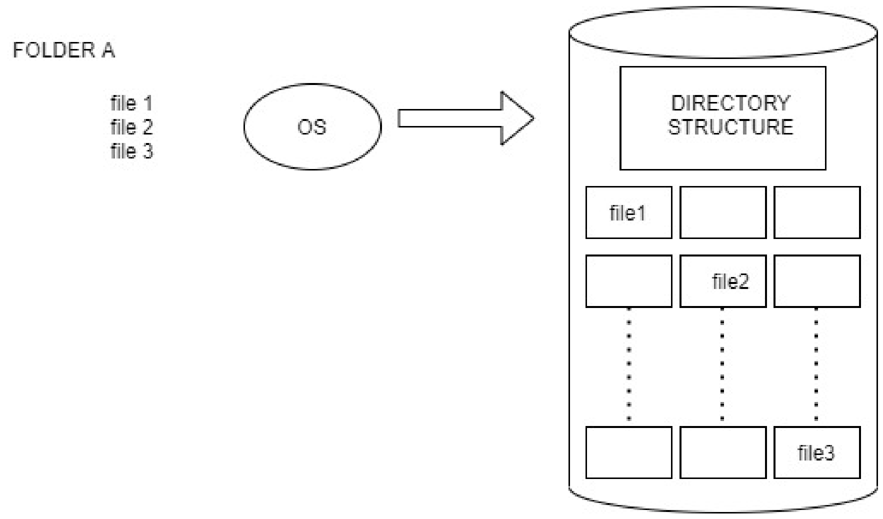
External fragmentation occurs when sufficient quantity of memory is available for a process but that available memory is not contiguous. Internal fragmentation occurs when memory block assigned to process is bigger than process. Some portion of memory is left unused, as it cannot be used by another process. Thus results in internal fragmentation.

5.6 FILE SYSTEM

A file is a named collection of related information recorded on a secondary storage device. File can be anything. It can be document file, audio, video etc. Secondary storage can be anything magnetic tape,

Space for learners:

optical device, hard disk etc. In the computer, file is the basic logical storage unit. According to user perspective everything is kept on a file and the file is stored at a single place on secondary storage device. But in secondary storage device, these files are stored as blocks. These files may occupy a block or may occupy more than a single block and these blocks may not be contagious. Consider an example, user kept file1, file2, file 3 in a folder named A. From user point of view, all the three files are in the same folder but actually these files may occupy different blocks as shown in figure. All the information is maintained in a directory structure which is stored at the beginning of secondary storage device. This directory structure keeps all the information about the files that is where the file exactly located, its address and so on. These collections of files on secondary storage device together with directory structure to manage, organize and to provide all the information together form the file system or file management system task of operating system.

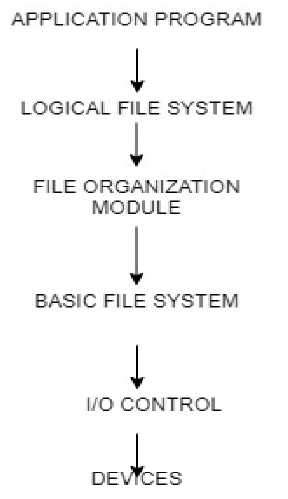


Organization of files in secondary memory

5.6.1 File System Structure

File system is stored generally in secondary storage device such as hard disk. To perform different operations file system is organized into different layers. So the layers are

Space for learners:



LAYERED FILE SYSTEM STRUCTURE

The top layer is application program and the bottom layer is devices. In between logical file system, file organization module, basic file system, I/O control.

Application Program-The program which is developed by user is called application program. That application program is given as input to the logical file system.

Logical file system-It accepts the file name as input and checks whether that file is available in directory structure. If the file is available in directory structure then it finds location of the file as well as logical block member of that file.

File organization module-That logical block member will be given as input to the file organization module and it performs a mapping in order to find physical block member in which location file is stored in hard disk. The task of this module is to find physical member block of the corresponding logical member.

Basic file system-Physical block member will be input to the basic file system. Basic file system issues a command to I/O control with the help of block member.

I/O control-I/O control accepts the command given by basic file system. Every I/O control contains device drivers. It is duty of device drivers to work with device. Device driver takes responsibility of interacting with

Space for learners:

devices so that corresponding operation takes place. This is how the layered structure of file system works.

Space for learners:

5.6.2 Operations on File

The various operations which can be implemented on a file such as read, write, open and close etc. are called file operations. Some common operations are as follows:

1. Creating a file-
    - Space for newly created file must be found in the file system.
    - After that directory must have entry for the newly created file.
2. Open operation-
    - Once file is created,it has to open in order to perform any operations.
3. Writing a file-
    - Make a system call specifying both the name and the info to be written to the file.
    - A write pointer must be maintained by the system to the location where next write should take place.
4. Reading a file-
    - Use a system call that specifies the name of the file and where in memory the next block of the file should put.
    - The read pointer is updated, once the read operation taken place.
5. Repositioning or Seek operation –
    - The seek system call reposition the file pointer to particular position in the file. Movement may be forward or backward depending on requirement of the user.
6. Delete operation-

- Delete operation not only remove contents of file but also remove the file from disk inorder to freed memory space occupied by it.

7. Truncate operation-

- This operation no doubt deleted the contents of a file but the file is not deleted completely.

8. Close operation-

- After performing all operation ,it is suggested to close the file so that the changes made are saved permanently and also resources used by the file releases.

9. Append operations-

- This operation add data to the end of the file.

10. Rename operation-

- This operation is used to rename the existing file.

5.6.3 Access Methods

There are three methods to access the files

1. Sequential access
2. Direct access or relative access or random access
3. Indexed sequential access

Sequential access:

It is the simplest method of all the three method. As the name implies information of the file are accessed one by one sequentially. For example let's say a file contains 100 records and the file pointer is in 50th record and we need 75th record to access. Here the file pointer moves one by one from 50th record sequentially in order to read 75th record. Thus it is not possible to access the 75th record directly. Magnetic tape is an example where files are accessed sequentially.

Operations are:

Read next// It reads the next instruction pointed by pointer

Write next// It writes the instruction in the next position

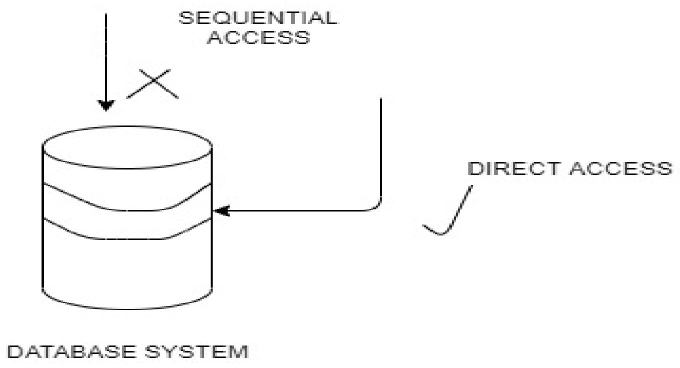
Reset (or) rewind//moves the pointer to the beginning

Advantage: Simplest of all the three method.

Disadvantage: Time consuming since need to access each record sequentially.

Direct access or relative access or random access:

By using this technique we can access the record directly. For the previous example, using this technique it is possible to access the 75th record directly without accessing one by one record sequentially. Example Hard disk, magnetic are random access device.



Let's see the operation of random access

Read n// read the instruction at position n pointed by pointer

Write n //write content in the location n pointed by pointer.

Position to n //file pointer can be move to any location say n here

Advantages: Pointer can be move directly to particular position in order to access.

Disadvantage: Implementation of direct-access systems is often difficult because of the complexity and the high level of programming (software) support that such systems need. In addition, the cost of developing and maintaining is greater than the expense of a sequential processing system.

3. Indexed sequential access:

It is used mainly in order to remove drawback of sequential access. Indexing is a data structure technique which is used to quickly locate

Space for learners:

and access the data in database. From the name itself it is saying that index are used in sequential order to access the data from the database. It is a static index structure in order for creating, maintain and manipulating files of data. Here records can be retrieved sequentially or randomly by one or more keys. It is an advanced sequential file organization.

Advantage: Searching a data in large database is very easy and quick using this technique.

Disadvantage: Extra space in the memory is required in order to store the index value.

STOP TO CONSIDER

Sequential access allows file to read/write sequentially up to the location where it is attempting to read or write. In direct access the records does not need to be in sequence because they are updated directly and rewritten back in the same location directly. In Index sequential index are used in sequential order to access the data from database.

5.7 SUMMING UP

- Scheduling algorithm is use in order to schedule the process.
- Four types of scheduling algorithm are mainly discussed here. They are round robin algorithm , First cum first serve algorithm , Shortest job first ,shortest remaining time first.
- How memory is allocated to different process are explained using the concept of paging and segmentation.
- Different memory partition algorithm like first fit, next fit, best fit and worst fit are explained using example .
- Concept of file along with file structure, file access methods are explained very clearly.

Space for learners:

5.8 ANSWER TO CHECK YOUR PROGRESS

Space for learners:

1. Average turnaround time=8 units

   Average waiting time=4.4 units

2. Average turnaround time=8.6 units

   Average waiting time=5.8 units

3. Average waiting time(non-preemption)=5.2 units

4. Average waiting time(preemption)=4.6 units

5. D

5.9 POSSIBLE QUESTIONS

1. Among all memory management techniques …………….. is simple to implement little operating system overhead.

2. Write difference between fixed and variable partition.

3. What is contiguous and non-contiguous memory allocation. Explain with example.

4. What is the function of DMA? Explain with diagram.

5. Why DMA data transfer is necessary?

6. What are job queues, ready queues and device queues?

8. What are the benefits of multiprogramming?

9. What is PCB? What are the information associate with it?

10. Explain FCFS, SJF with example.

11. Explain the concept of pages, frames. What is physical and logical memory?

12. Define swapping.

13. What do you mean by compaction?

14. Process Burst time P1 10, P2 29, P3 3, P4 7, and P5 12 given for five different processes in milliseconds. Consider the First come First serve (FCFS), Non Preemptive Shortest Job First (SJF), Round Robin (RR) (quantum=10ms) scheduling algorithms. Calculate average

waiting time and turnaround time. Illustrate the scheduling using Gantt chart.

15. Write about various scheduling algorithm.

5.10 REFERENCES AND SUGGESTED READINGS

- Operating System Concept by Abraham Silberschatz, Peter Baer Galvin, Greg Gagne
- Operating System Principles (Internal and design principles) by William Stallings
- Modern Operating Systems by Andrew S Tanenbaum/ Herbert BOS
- Operating System Concept by Willey

Space for learners:

UNIT 6: SECONDARY STORAGE MANAGEMENT

Space for learners:

Unit Structure:

6.1 INTRODUCTION

This unit gives an overview of secondary storage management. The secondary storage means non-volatile memory management. The unit explains the disk structure along with the different disk scheduling algorithms. The swap space management is also discussed in the unit. The Redundant Arrays of Independent Disks (RAID) architectures are also discussed in the unit. The stable-storage implementation is also highlighted in the unit along with the outcomes of the disk. The tertiary storage consists of high-capacity data achieves and it is discussed nicely in the unit.

6.2 UNIT OBJECTIVES

After going through this unit, you will be able to

- Understand about the mass storage structure.
- Learn about the disk Structure

- Learn about the different disk scheduling algorithm
- Understand about the swap space management
- Understand about the RAID structure
- Explain the stable storage and tertiary storage

Space for learners:

6.3 MASS STORAGE STRUCTURE

The secondary storage is those where the memory is non-volatile. It means that the data will be intact with the device even if the device or system turns off. The secondary storage structure is auxiliary storage and is less expensive but has less speed than primary storage. Non-frequent data are saved in the secondary storage. Examples of secondary steerages are magnetic disk, pen drive, HDD, etc.

Mass storage refers to the storage of large amounts of data. The data are stored in persisting and machine-readable form. The mass storage device includes tape, RAID system, HDD, magnetic tape, optical disk, memory cards, and SSD, etc. The mass storage doesn’t include the RAM. There are two types of the mass storage structure. The first one is local data storage which is a Smartphone or local computer. The other one is global data storage which includes servers, data centres, cloud, etc.

The mass storage device is characterized by:

i) Sustainable speed of the device
ii) Seek time of the device
iii) Cost of the device
iv) The capacity of the device.

6.4 DISK STRUCTURE

The modern disk structure contains tracks and each sector contains multiple sectors. The disks are arranged in a 1-D array of blocks. These blocks are the storage unit of the disk structure which is known as the sector. For each surface, a read-write desk is available in the disk structure. The tracks on the surfaces are known as a cylinder. The disk has the basic following structure.

i) The disks are in the form of platters that are covered with magnetic media. The hard disk platters are metal whereas the floppy disk platter is plastics.

ii) Every platter has two working surfaces and each working surface has some rings called tracks. The tracks which are in the same distance from the edge of the platter is known as a cylinder.

iii) Each track is further divided into sectors. The sector contains 512 bytes of data. Some sector uses larger sector size. Each sector contains a header and trailer.

iv) The data on a hard drive is read by read-write heads. In standard, one head is reserved per surface, each on a separate arm., controlled by arm assembly.

v) The storage capacity of a disk is equal to the number of heads or number of bytes per sector.

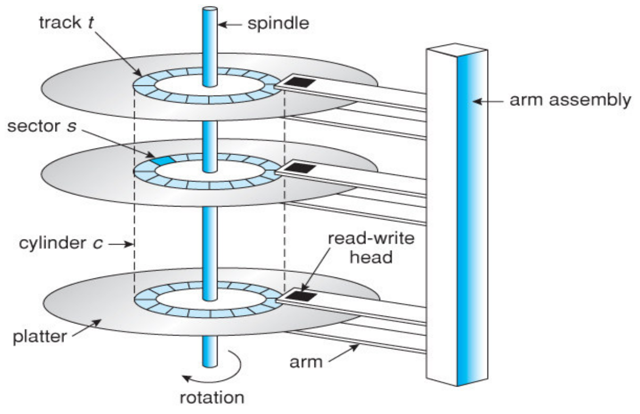


Fig.6.1 Disk Structure

CHECK YOUR PROGRESS

1. True or False
   i) Pen drive is a mass storage device
   ii) Track of disk surface is known as cylinder.
   iii) Hard disk platters are plastics.
2. What is a sector?
3. What is the data size of a sector?
4. What do you mean by the storage capacity of disk structure?

Space for learners:

6.5 DISK SCHEDULING ALGORITHM

Space for learners:

Disk scheduling is a process where the operating system schedules the I/O requests and it is also known as I/O scheduling. Disk scheduling is important because to manage the multiple I/O requests, this may arrive from a different process. But only one I/O request can be served at a time by the disk controller. In this situation, other I/O request needs to wait in the waiting queue.

The types of disk scheduling algorithms are:

i) First Come First Serve (FCFS) Algorithm

ii) Shortest Seek Time First Scheduling

iii) SCAN Scheduling Algorithm

iv) C-Scan Scheduling Algorithm

Before discussing disk scheduling, you should learn the following parameters.

i) **Seek Time**: It is time to locate the disk arm to read or write data.

ii) **Rotational Latency:** It is the time taken by the sector to rotate itself into a position to access the read and write heads.

iii) **Transfer Time**: It is the time to transfer the information depending on the speed of the disk.

iv) Disk Access Time: It is the combination of seek time + rotational latency + transfer time.

v) **Disk Response Time:** It is the average request waiting time to perform its I/O operation.

6.5.1 FCFS Disk Scheduling Algorithm

The First Come First Serve (FCFS) is the simplest disk scheduling among all. In this algorithm, the first request is always served first in the disk queue. Though there is no starvation in the algorithm it does not provide the fastest service. Let understand the FCFS disk scheduling algorithm with the following examples. Let's you have following order of request. Request = {70, 3, 2, 40,4,6,90} and the position of the read and write head is 5 and the total number of the track is 100.

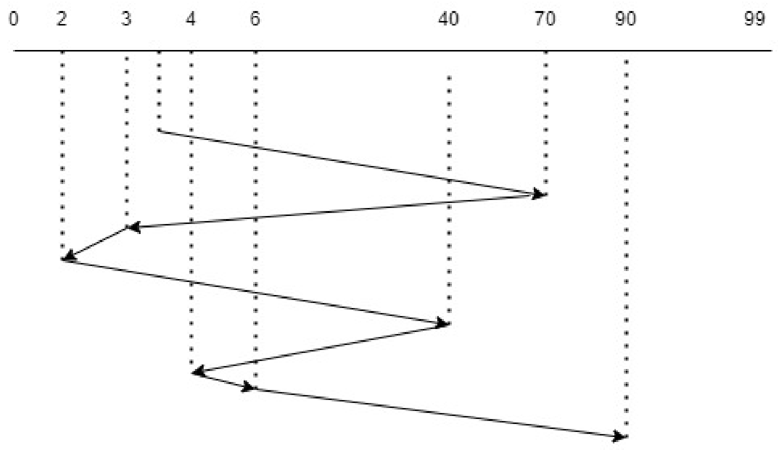


Fig. 6.2 FCFS Disk Scheduling

The current position of the read and write head is 5. So, we start at 5. As the algorithm is FCFS, the first move is towards 70. Then 3, 2, 40, 4, 6, and finally 90. So, the total cylinder move (seek time) is:

seek time = (70-5) + (70-3) + (3-2) + (40-2) + (40-4) + (6-4) + (90-6) = 303

In this algorithm, every request gets a reasonable chance. But it does not have any technique for optimization of seek time.

6.5.3 SSTF Disk Scheduling Algorithm

In Shortest Seek Time First (SSTF) algorithm, the request which has less seek time execute first. So, the average response time is decreased in SSTF. It increases the throughput of the scheduling. But there is a chance of starvation in SSTF. Let's understand the algorithm by taking the following examples. Let's you have a disk that contains 100 tracks (0-100). The requests are with the track numbers 70, 3, 2, 40, 4, 90, respectively. The current position of the read/write head is 5. As we need the seek time for this algorithm, so the seek times of the heads are as follows.

i) If you will from 5 to 4, then it will give you the shortest seek time among all the requests and that is (5-4) = 1.

ii) Then from 4, the head will move 3.

iii) Then it will move to 2. From 2, it will move to 40, 70, and finally 90.

Space for learners:

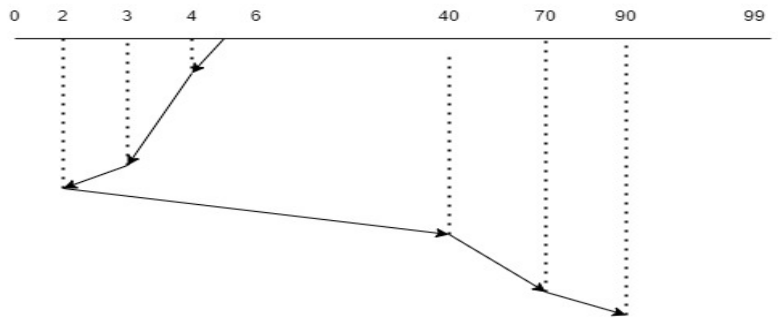


Fig. 6.3 SSTF Disk Scheduling

So, seek time = (5-2) + (90-2) = 91

Space for learners:

6.5.3 SCAN Disk Scheduling Algorithm

In the SCAN disk scheduling algorithm, we move the head either to the smaller value or to the larger value. In the moving path, each request is addressed. When the disk arm reaches to end, it will move towards reverse and all the requests are addressed. Let's you have a disk that contains 100 tracks (0-100). The requests are with the track numbers 70, 3, 2, 40, 4, 90, respectively. The current position of the read/write head is 5. Let's the head will move towards the larger end and it will execute as follows.

i) As head movement is towards the larger value, so it will move in the right direction from 5. From 5 it will reach 40, 70, and then 90.

ii) From 90, it will move to 4, 3, and finally 2.

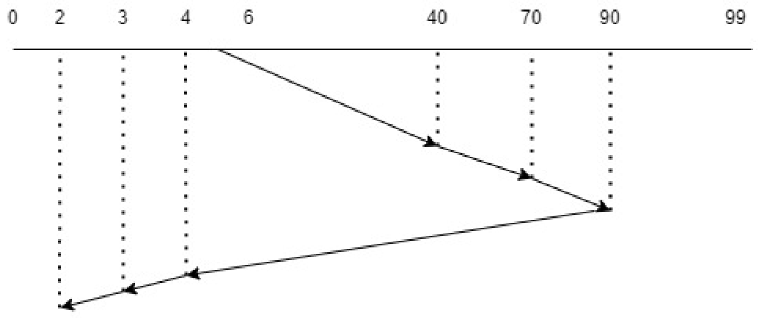


Fig. 6.4 SCAN Disk Scheduling

So, seek time = (90-5) + (90-2) = 173

6.5.4 C- SCAN Disk Scheduling Algorithm

Space for learners:

In the C-SCAN algorithm, the disk moves in a particular direction serving the requests to the end of the direction, and then comes back to the reverse direction until the end. From that end, only the arm starts moving with serving the remaining requests. Let's you have a disk that contains 100 tracks (0-100). The requests are with the track numbers 70, 3, 2, 40, 4, 90, respectively. The current position of the read/write head is 5. Let's the head will move towards the larger end and it will execute as follows.

i) Let's the arm move in the right direction. So its first move from 5 to 40, 70, the, 90 and finally it will go to 99.

ii) From 99, it will directly come to the reverse end, i.e. to the left side end 0.

iii) From 0 it will go to 2, 3, and 4.

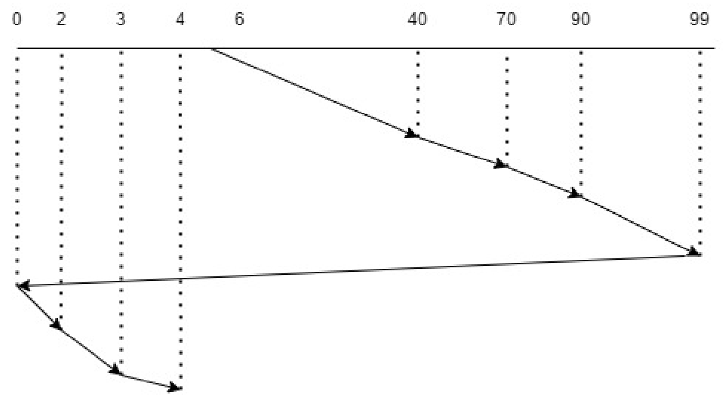


Fig. 6.5 C – SCAN Disk Scheduling

So, seek time = (99-5) + (99-0) + (4-0) = 19

CHECK YOUR PROGRESS

5. What is a FCFS disk scheduling?
6. Assuming that the disk head is located initially at 32, Find the number of disk moves required with FCFS if the disk queue of I/O block requests are 98, 37, 14, 124, 65, 67 .
7. Fill in the blanks
    i) The set of tracks that are at one arm position make up a ___________.
    ii) The time taken to move the disk arm to the desired cylinder is called the ____________

6.6 SWAP SPACE MANAGEMENT

Space for learners:

Swapping in memory management means swapping of the process so that the maximum number of processes sharing the CPU. It is used multiprogramming. It is a memory management technique used to remove a process from the main memory to secondary memory and then bring it back to the main memory. These are known as swap out and swap in. The area on the disk where the swapped-out processes are stored is called swap space.

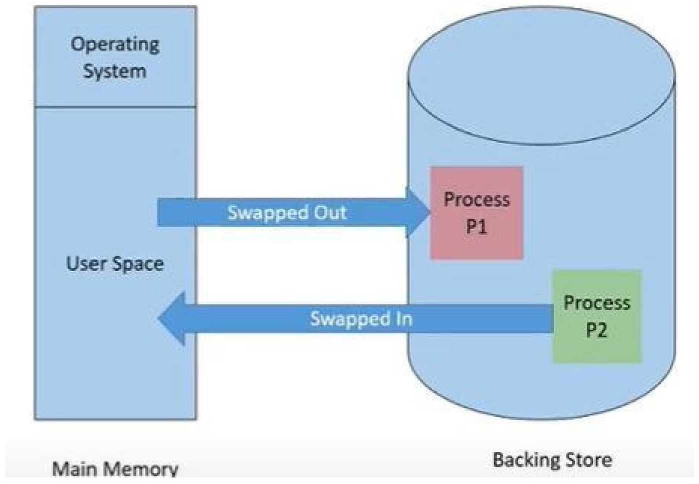


Fig. 6.6: Example of Swap space management

In Fig. 6.1, the process p1 is swapped out from the main memory to swap space, and process p2 is swap in the main memory. So this process is Swap in and out.

Swap space management is another low-level task of the operating system because it deals with disk space. As mentioned above, the process is out from main memory to secondary memory, i.e., disk space. So, disk space is required in swap space management. But the disk space is slower than the memory access. So it will reduce system performance. So, the goal of swap space management is to introduce virtual memory for better throughput.

Swap spaces are variously used by different operating systems. The swap space may contain the entire system or process images that are currently not in use or loaded in RAM. So it is using paging techniques to manage the space. The size of swap space may vary from megabytes to gigabytes. The amount of swap space needed on a system can vary depending on the amount of physical memory, the amount of virtual memory it is backing, and how the virtual memory is used.

Swap space can reside in two ways.

i) Normal File system

ii) Separate Disk Partition

In the Normal File system, swap space may create by using the normal file system routine. In this process, external fragmentation can increase the swapping times. Otherwise, swap space may create by using a raw partition. No presence of a file system is found here. Here, algorithms are used to increase the speed of the swapping rather than storage. So, it may increase the internal fragmentation of the system.

6.7 RAID STRUCTURE

The performance of a single disk is less as compared to the multiple disks. The Redundant Arrays of Independent Disks (RAID) is a technique where multiple disks are combined to form a single disk that increases the performance of the system along with the data redundancy. Though the data redundancy takes extra space it increases the reliability of the system. If a disk fails and the data is backed up in another disk, then at the time of disk failure, the information will not lose. You can perform the disk operation. A RAID system is evaluated using the following parameters.

i) Reliability: It denotes the number of disk failures, but the RAID performance will not reduce.

ii) Availability: It denotes the available time, the system for actual use.

iii) Performance: It denotes the response time and throughput of the RAID system.

iv) Capacity: It denotes the overall capacity of the RAID in terms of the number of disks per block.

A RAID structure is transparent. It appears as a big disk for the user. It has the following levels.

i) RAID 0: In this type, the blocks are “striped” across disks without any mirror and parity. Minimum two blocks are present in the RAID 0. As blocks are stripped, the performance of the

RAID in increasing as compared to the normal disk. As the system is very simple, it can not be used for a complex system.

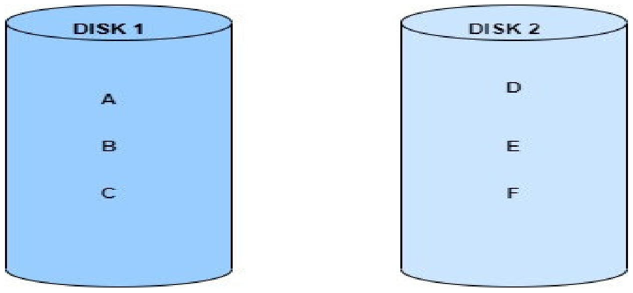


Fig. 6.7: RAID 0.

ii) RAID 1: In this level of RAID, the blocks are mirrors without stripe and parity. Here also, a minimum of two blocks are required for the implementation. Due to no striping and parity, the performance of the RAID 1 is more than the RAID 0. As blocks are mirrored, excellent redundancy is achieved in RAID 1.

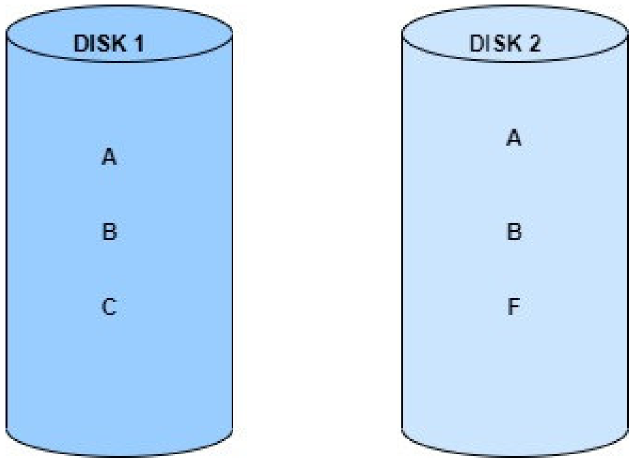


Fig. 6.8 RAID 1

iii) RAID 5: In this level of RAID, the blocks are striped and distributed parity is used in the RAID 5. Minimum 3 blocks are used in RAID 5. As the blocks are stripped, the performance of the RAID level 5 is increasing. The RAID 5 achieved good redundancy due to the distributed parity. In RAID 5, the best cost-effective option is provided using both performance and redundancy.

Space for learners:

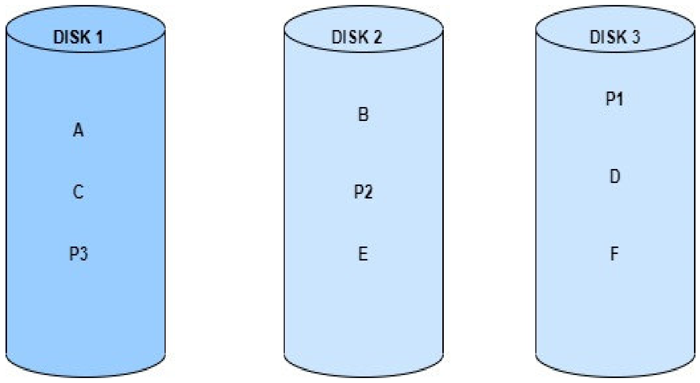


Fig. 6.9: RAID 5

iv) RAID 10: In this level of RAID, the blocks are striped and mirrored. This is also called a strip of the mirror. Minimum 4 blocks are used in RAID 5. Excellent performance and redundancy are observed in RAID 10 due to the stripped and mirror.

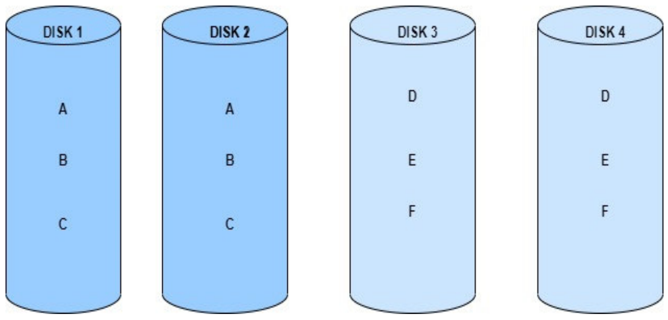


Fig. 6.10: RAID 10

CHECK YOUR PROGRESS

8. What are the level of RAID?
9. Can we use RAID 0 for complex system?
10. What is RAID 10?
11. Which RAID type doesn't use parity for data protection?
12. What is the minimum number of disks required for RAID1?

Space for learners:

6.8 STABLE STORAGE

Stable storage means no data loss even if the disk of the computer system fails. It is a computer data storage technology that guarantees atomicity for any read-write operation. To implement stable storage, replication of data on different devices is required. It will help to recover a copy of the data even if the data is removed from some devices.

The causes of the system or device failure are defined below.

i) System Crashes

ii) User Error

iii) Carelessness

iv) Sabotage (intentional corruption of data)

v) Statement Failure

vi) Application software errors

vii) Network Failure

viii) Media Failure

ix) Natural Physical Disasters

6.9 TERTIARY STORAGE STRUCTURE

Tertiary storage consists of high-capacity data archives using vast numbers of removable media, such as tapes or optical discs. Tertiary storage or tertiary memory is a level below secondary storage. It involves a robotic mechanism that will mount (insert) and dismount removable mass storage media into a storage device according to the system's demands; such data are often copied to secondary storage before use.

Figure 6.11: Tertiary storage platforms: (A) Quantum tape library, (B) BluRay optical jukebox [**Image Source**: https://www.sciencedirect.com/topics/computer-science/tertiary-storage]

The main objective of tertiary storage is to provide hug storage at a low cost in terms of magnetic tapes, optical disks, and optical tapes. They are consisting of fixed storage drives and removable media units. The storage drives are fixed to the computer system but the removable media can be removed to increase the storage capacity by increasing the media units. When data on a media are accessed, the media unit is accessed from its normal location and one storage drive is chosen from the local computer. If there is a media unit in the storage system, the old storage system is unloaded and ejected so that the new media unit can load in the drive. Each storage drive handles the driver and unit efficiently.

6.10 SUMMING UP

- The secondary storage is those where the memory is non-volatile. It means that the data will be intact with the device even if the device or system turns off.
- The secondary storage structure is auxiliary storage and is less expensive but has less speed than primary storage.
- The mass storage device is characterized by:
- Sustainable speed of the device
- Seek time of the device
- Cost of the device
- The capacity of the device.
- The disks are arranged in a 1-D array of blocks. These blocks are the storage unit of the disk structure which is known as the sector.
- The disks are in the form of platters that are covered with magnetic media.
- The hard disk platters are metal whereas the floppy disk platter is plastics.

- Every platter has two working surfaces and each working surface has some rings called tracks. The tracks which are in the same distance from the edge of the platter is known as a cylinder.
- Each track is further divided into sectors. The sector contains 512 bytes of data. Some sector uses larger sector size. Each sector contains a header and trailer.
- Disk scheduling is a process where the operating system schedules the I/O requests and it is also known as I/O scheduling.
- Disk scheduling is important because to manage the multipleI/O requests, this may arrive from a different process.
- The types of disk scheduling algorithms are:
    - First Come First Serve (FCFS) Algorithm
    - Shortest Seek Time First Scheduling
    - SCAN Scheduling Algorithm
    - C-Scan Scheduling Algorithm
- The First Come First Serve (FCFS) is the simplest disk scheduling among all. In this algorithm, the first request is always served first in the disk queue.
- In Shortest Seek Time First (SSTF) algorithm, the request which has less seek time execute first.
- In the SCAN disk scheduling algorithm, we move the head either to the smaller value or to the larger value. In the moving path, each request is addressed. When the disk arm reaches to end, it will move towards reverse and all the requests are addressed.
- In the C-SCAN algorithm, the disk moves in a particular direction serving the requests to the end of the direction, and then comes back to the reverse direction until the end. From that end, only the arm starts moving with serving the remaining requests.
- Swapping in memory management means swapping of the process so that the maximum number of processes sharing the CPU.

Space for learners:

Space for learners:

- Swap space can reside in two ways.
  - Normal File system
  - Separate Disk Partition
- The Redundant Arrays of Independent Disks (RAID) is a technique where multiple disks are combined to form a single disk that increases the performance of the system along with the data redundancy.
- The RAID has the following levels, RAID 0, RAID 1, RAID 5, and RAID 10.
- Stable storage means no data loss even if the disk of the computer system fails. It is a computer data storage technology that guarantees atomicity for any read-write operation.
- Tertiary storage consists of high-capacity data archives using vast numbers of removable media, such as tapes or optical discs.

6.11 ANSWERS TO CHECK YOUR PROGRESS

1. i) True ii) True iii) False
2. The modern disk structure contains tracks and each sector contains multiple sectors. The disks are arranged in a 1-D array of blocks. These blocks are the storage unit of the disk structure which is known as the sector.
3. The data size of a sector is 512 bytes.
4. The storage capacity of a disk is equal to the number of heads or number of bytes per sector.
5. The First Come First Serve (FCFS) is the simplest disk scheduling among all. In this algorithm, the first request is always served first in the disk queue. Though there is no starvation in the algorithm it does not provide the fastest service.
6. 321.
7. i) Cylinder ii) Seek Time
8. The levels of RAID are RAID 0, RAID 1, RAID 5, and RAID 10.
9. No

10. In RAID 10, the blocks are striped and mirrored. This is also called a strip of the mirror. Minimum 4 blocks are used in RAID 5. Excellent performance and redundancy are observed in RAID 10 due to the stripped and mirror.

11. RAID 1.

12. 2.

Space for learners:

6.12 POSSIBLE QUESTIONS

Short answer type questions:

1. What do you mean by mass storage?
2. What are the characteristics of mass storage?
3. Define the term cylinder and sector of disk structure.
4. Explain the term seek time and rotational latency.
5. Difference between FCFS and SSTF disk scheduling algorithm.
6. Consider a disk queue with request for input/output to block on cylinders 98, 183, 37, 122, 14, 124, 65, 67cin that order. Assume that the disk head is initially positioned at cylinder 53 and moving towards cylinder number 0. What is the total number of head movements using Shortest Seek Time First (SSTF) and SCAN algorithms?
7. What is swap space?
8. Why is the necessity of RAID structure?
9. What is stable storage?
10. What is a tertiary storage structure?

Long answer type questions

1. Explain the Disk scheduling algorithms with examples.
2. If the disk head is located initially at 32, find the number of disk moves required with FCFS, SSTF, SCAN, and C-SCAN if the disk queue of I/O blocks requests are 98, 37, 14, 124, 65, 67.
3. Explain about different RAID structure.

6.13 REFERENCES & FURTHER READINGS

- Schaum's Outline of Operating Systems.
- Operating system concept 9E by Silberschatz, Publisher: Wiley.

Space for learners:

UNIT 7: SECURITY

Space for learners:

Unit Structure:

7.1 INTRODUCTION

Security is the state of being free from threat. One of the major mechanisms of ensuring security is encryption. Basically security is concerned with the unauthorized access of information. Security ensures safe sharing of software and hardware resources of a system. Authentication is very important in this regard. Security attacks mainly focus on the illegal use of confidential resources such as data files.

7.2 UNIT OBJECTIVES

After going through this unit, you will be able to

- explain the basic concepts of security, threat, attack
- discuss the various security goals

- explain the concept of user authentication
- discuss about cryptography, computer security classification

Space for learners:

7.3 SECURITY

Security is a mechanism which provides protection to the system resources. System resources can be both software and hardware like CPU, disk, memory, data etc. stored in the system. Basically security is used to prevent unauthorized access of these resources. It deals with the threats that are external to the systems.

Some of the basic incidents which can be termed as security violation are given below:

i) **Theft of Data**: If an unauthorized user tries to steal information then this can be termed as theft of data.

ii) **Unauthorized Modification of Data**: If an unauthorized user tries to alter the data then it is termed as unauthorized modification of data.

iii) **Unauthorized Destruction of Data**: If an unauthorized user tries to delete the data then it is termed as unauthorized destruction of data.

7.3.1Security Goals

Security between intended sender and receiver can be achieved through following major security goals:

i) **Confidentiality**:

It is a service through which only the intended sender and the receiver will know the actual data.

ii) **Data Integrity**:

It is service through which only the authorized user can access or modify the actual data.

iii) **Nonrepudiation**:

It is a service through which no user can refuse the previous commitment after doing so.

iv) **Authentication**:

It is a service through which only the authorized user can access the system resources.

v) **Availability**:

The system resources need to be available for the authorized user when needed.

Space for learners:

7.3.2Security Issues and Measures

Security of a system can be violated by threats and attacks. If a system is hacked by unauthorized user then there will be loss of confidentiality, integrity of the system resources.

Some of the important security issues are:

i) Loss of data

ii) Modification of data

iii) Misuse of data

To protect the system from these security issues following measures need to be taken:

i) Protection mechanisms to prevent modification and loss of data.

ii) Control system resource sharing among the users.

iii) Authentication of the valid user needs to be done before accessing the system resources.

iv) Cryptographic techniques need to be used to ensure secure communication between sender and receiver.

v) Security policies need to be introduced among the users.

vi) The site containing computer system should be physically secured from attacker.

vii) The operating system must protect itself from malicious attacks.

viii) Secure network communication must be established among the systems.

ix) Anti-Malware programs need to be used to protect the system.

x) To protect the system from network threats firewall is used.

CHECK YOUR PROGRESS

1. What are the major security goals?
2. What are the different security issues a system can have?
3. How security can be achieved?

Space for learners:

7.4 THREATS

Threat can potentially harm the system resources. This means alteration or hiding or destroying the actual content of a message, occupying hard drive space and illegal use of passwords.

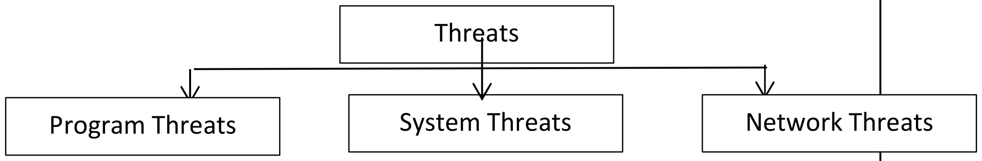


Fig. 7.1: Types of Threats

7.4.1 Program Threats

When a program is created by a user is used by another user then misuse of the program may occur. If misuse of the program is happened then this event is termed as program threat. Some of the examples of program threats are *Trojan horse, Trap door, Buffer overflow and Logic bomb*.

i) **Trojan horse**: This program sits ideally and transmits all the information to the attacker. Suppose if you login to a site using browser and if the Trojan horse is attached with the browser then the user id and password will be stored by the Trojan horse and it sends the user id and password to the attacker.

ii) **Trap door**: It is a program which is installed in a system and has some security hole in the code and due to this if the program performs illegal actions without the knowledge of the user then it is called to have a Trap Door.

iii) **Buffer overflow**: Suppose a program is created by a user and that program is installed in another system, and this program consumes all the resources of the system where it is being installed. In this situation Buffer Overflow may occur.

iv) **Logic Bomb**: This situation is very hard to detect. Here, an installed program misbehaves when certain conditions met otherwise it works as a genuine program.

7.4.2 System Threats

The misuse of Operating System (OS) and user files is termed as system threats. For example, mostly we install OS in C drive. There are many hidden folders in program files and few of the files we cannot even access. But these files can be accessed by the attackers by launching worms or viruses. Some of the examples of System Threats are Worms, Virus etc.

i) **Worms**: Worms create duplicate copies which contain malicious code that simply consume system resources and deny service of the user. This slows down the system.

ii) **Virus**: This can delete or alter the information available in a system. It contains small section of code embedded in a program. When this program is accessed by the user, the virus starts getting embedded in other files.

7.4.3 Network Threats

The misuse of user's confidential information while accessing the network without the knowledge of the user leads to network threats. Some of the examples of network threats are Port Scanning, Denial of Service etc.

i) **Port Scanning**: It is a mechanism through which unauthorized user can detect the system vulnerabilities to attack the system.

ii) **Denial of Service**: This prevents authorized access of a user. For example, if denial of service attacks in the browser's content setting then user may not be able to use the internet.

STOP TO CONSIDER

Program threat is concerned with the misuse of ones created program. System threat is concerned with OS of the system and network threat is concerned with the use of internet.

7.4.4 Attack

Attack is a kind of threat to a system from malicious users. It is of two types namely *Active Attack* and *Passive Attack*.

i) Active Attack: In this attack, the attacker tries to alter the content of the message. This type of attack can be easily detected so proper cure is needed. Here, attacker uses information to launch attack on the target. Example of active attack: *Masquerade, Replay, Modification of Messages, Denial of Service.*

ii) Passive Attack: In this attack, the attacker learns and uses information of the message and listens to the traffic to launch attack on the target. It is difficult to detect so prevention is better in case of passive attack. Example of passive attack: *Release of Message Content, Traffic Analysis.*

CHECK YOUR PROGRESS

4. State the examples of program threats, system threats and network threats.
5. Differentiate between active attack and passive attack.

7.5 CRYPTOGRAPHY

It is a technique to hide the actual content from unauthorized user. It is the study and practice of different mechanisms for secure communication in the presence of unauthorized user in between the intended sender and receiver. Here we have two basic terminologies, Encryption and Decryption. Secure communication is done between sender and receiver with the help of cipher test.

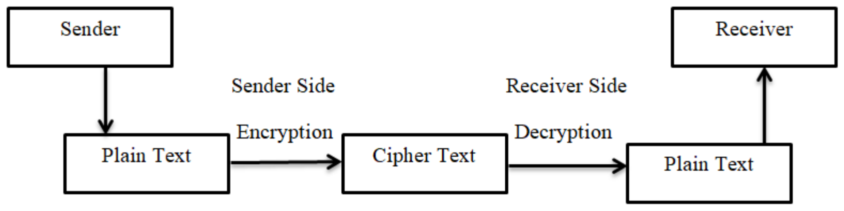


Fig. 7.2: Concept of Cryptography

Explanation of Fig. 7.2:

As we already know that cryptography ensures secure transmission of data between sender and receiver, so the sender will apply some encryption technique on the plain text that is the actual content and converts it to the cipher text. This cipher text will be sent to the receiver and at the receiver side, it will convert the cipher text to the plain text with the help of the encryption technique that has been used by the sender. The conversion of cipher text to plain text is termed as decryption.

Some of the important terminologies:

Cryptanalysis: The art of decoding the cipher text in order to hack without knowing the encryption technique is referred to as cryptanalysis. Cryptanalyst is the person who is always busy in cryptanalysis.

Cryptology: It is the combination of both cryptography and cryptanalysis.

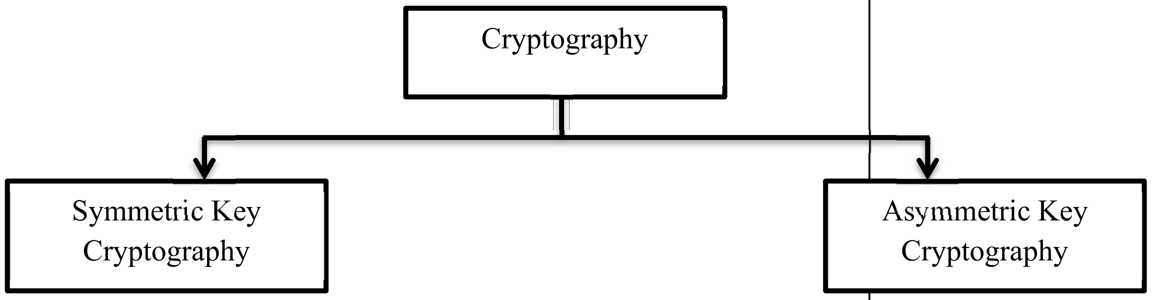


Fig. 7.3: Types of Cryptography

i) Symmetric Key Cryptography:

Here the sender and receiver of a message use the same key for both the encryption and decryption process respectively. Data

Encryption System (DES) is one of the most popular symmetric key cryptography techniques.

ii) Asymmetric Key Cryptography:

Here a pair of public and private key is used for both encryption and decryption process respectively. RSA algorithm is a good example of asymmetric key cryptography.

7.6 USER AUTHENTICATION

Authentication is a process of identification. User authentication means identifying whether the user is a valid user or not. Suppose, you login to e-commerce site using the user id and password. The user id and password are the essential credentials to authenticate your identification. Normally in our personal computer we use to provide PIN or Fingerprint or Password to protect our system from unauthorized access.

User of a system can be authenticated using the following mechanisms:

i) **Using password**

Users normally have their own user id and password to access the system resources. Password is a combination of special characters, numbers, and alphabets. Now a days, One Time Password (OTP) is also popularly used to access resources. Normally OTP is sent to the registered mobile number or email id.

ii) **Using physical object**

Users normally use to withdraw cash in Automated Teller Machine (ATM) with the help of a unique card which is registered with the user only with a secured PIN.

iii) **Using biometric**

User authentication using biometric method is the most safeguard mechanism to protect the resources. Here physical characteristics of user like fingerprint, voice, and retina are used for authentication purpose as these are very difficult to forge. Now a days, in most of the offices or organizations punching machine is used to record the employees attendance.

CHECK YOUR PROGRESS

6. Define the process of user authentication.
7. What are the different ways using which authentication of user is done?

Space for learners:

7.7 SECURITY DEFENSE MECHANISMS

To ensure security of the system different defense mechanisms need to be followed. Following are some of the efficient defense mechanism normally used to protect the system against malicious attacks:

i) **Encryption**: Plain text is converted to cipher text (encrypted text) to hide the actual content of the message to safeguard it from attacker.

ii) **Digital Signature**: User digitally or electronically signs the data and sends it to the intended receiver. And the receiver verifies the signature before accessing it for security reasons.

iii) **Data Integrity**: There is a check value embedded with the message and this check value is known by the receiver and sender. Whenever sender sends message to the receiver this check value is appended with the message before sending it and after that the receiver matches the check value after receiving the message with the check value that receiver already has. If the check value matches then receiver accepts the message and if the check value does not match then receiver assumes that modification has been made so simply discards it.

iv) **Access Control**: This ensures that the user has the right to access the system resources.

v) **Authentication**: Only the authorized users can establish secure communication and share resources.

vi) **Traffic Padding**: Here some extra bits are added with the actual content of the message for encryption.

7.8 PROTECT SYSTEMS AND NETWORKS WITH FIREWALLING

Space for learners:

Firewall is a network security mechanism to protect the system from network threat. Basically it monitors the incoming and outgoing packets that consists data/information in the network and based on the security rules it accepts or rejects the packets. It creates an interface between internal network and incoming traffic from the external network to protect the system from malicious threats. It can be both software and hardware. A software firewall is a program installed in the system. It regulates traffic through port numbers and applications. On the other hand, hardware firewall is a equipment installed between the network and gateway.

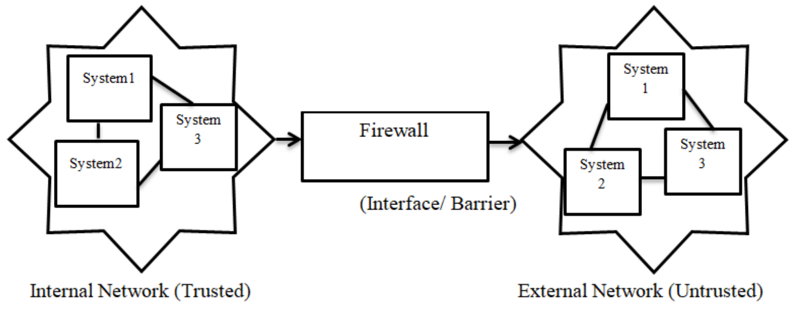


Fig. 7.4: Illustration diagram of Firewall

Some features of good firewall:

i) All the authorized incoming and outgoing packets must pass through the firewall.

ii) Firewall must be strong enough to reject unauthorized incoming packets.

Some limitations of firewall:

i) Inside a network if a malicious user tries to launch attack then firewall cannot protect the network.

ii) There are some applications where we need to disable the firewall. By doing so, there will be no control in the incoming and outgoing traffic.

iii) Firewall does not analyze the content of the packet so if a packet contains malicious content sent from an authorized user then it cannot discard the packet. This cause a threat to the system.

There are two types of firewall namely *Host based firewall* and *Network based firewall*.

i) **Host based firewalls**: These are installed on the network, which control incoming and outgoing packets. Host based firewall is a software application and it comes with the operating system. It provides protection to the internal network. Host based firewall protects systems from malicious attacks and unauthorized access.

ii) **Network based firewalls**: It operates on the network. These firewalls filter all the incoming and outgoing traffic across the network. It protects the internal network from the unauthorized access of the third party. A network based firewall might have two or more Network Interface Cards (NICs).

Working of Firewall:

Internet is untrusted network where different computers or systems are connected together to share resources/ information. Firewall maintains access list where authorized and unauthorized system's details are stored.

Example 1: Suppose we have 3 users A, B and C. In the access list of the firewall only user A and B have the access rights but user C does not have the access right. When A wants to access information through internet from B then B sends the information to A. Though user C hacks the connection and sends another reply to user A, then A simply discards it as C is not in the access list of the firewall. It method is named as ***Packet Filtering***.

Example 2: Suppose user A is currently visiting a specific site. Firewall of user A has the record of the user name and the visiting site in a conversation list. If an attacker hacks this connection and sends unwanted data to the user A, then firewall rejects these data as it already knows the visiting site from the conversation list. This method of protection is termed as ***Stateful Inspection***.

Example 3: Suppose user A is connected to the internet through a different user B. And user A requests some information from the internet via user B. User B passes the request to the internet. This way user B is hiding user A from the attackers available in the internet. This method of protection is termed as ***Proxy Firewall***.

Space for learners:

So, if you want to protect your system from malicious threat then never disable the firewall.

Space for learners:

CHECK YOUR PROGRESS

8. Define firewall.
9. How does firewall protect system from malicious user?

7.9 COMPUTER SECURITY CLASSIFICATION

Computer security means protection of system resources from unauthorized access. It prevents modification and deletion of data from malicious users. It restricts unauthorized users. Computer security is mainly concerned with three goals i.e. confidentiality, integrity and availability. These points are already discussed above. Computer security is important to safeguard the system resources from virus and worms. It protects crucial information of users.

According to the U.S. Department of Defense Trusted Computer System's Evaluation Criteria there are four security classifications available for computer: A, B, C, and D[4]. In the following table brief description of each classification is given.

Table 7.1: Security classes

Security Classes	Description
A (Highest Level)	This classification uses formal design specifications and verification techniques to grant access to user for secure communication or resource sharing.
B	This classification provides mandatory protection system. It is of three types. i) B1: This maintains the security label of objects in the system. Label is used to make decision to control the access. ii) B2: This extends the sensitivity labels to each system resource. iii) B3: This allows creating user groups for access control to grant access or restrict

	access to other object available in the system.
C	This classification provides protection and user accountability using audit capabilities. It is of two types. i) C1: This incorporates controls to protect the user's resources. Example: UNIX versions are mostly Cl class. ii) C2: This adds an individual-level access control to the capabilities of a Cl level system.
D (Lowest Level)	This classification is used for systems that have failed to meet the requirements of any of the other security classes. For example, MS-DOS and Windows 3.1 are in division D

Space for learners:

7.10 SUMMING UP

- Security is concerned with the unauthorized access of the system resources.
- Security mainly focuses on the confidentiality, integrity and availability of the system resources.
- Threats like virus and worms are used by the attacker to hack a system. To prevent this hacking we need to use some secure communication strategies like encryption, digital signature etc.
- Threats are of three types: program threat, system threat and network threat.
- There are two types of security attack namely active attack and passive attack.
- Cryptography is a mechanism of secret writing. It is of two types i.e. symmetric key cryptography and asymmetric key cryptography.
- Firewall is one of the major mechanisms to establish security.

7.11 SAMPLE QUESTIONS

Space for learners:

1. What do you mean by security?
2. What is packet filtering?
3. Compare stateful inspection and proxy firewall.
4. What do you mean be cryptography? How it is used to secure systems?
5. State the difference between virus and worms.
6. What are the classes of computer security? Explain.
7. Define attack. Explain the different types of attack.
8. Define threat.
9. Give some examples of program, system and network threats.
10. What are the major security goals?
11. Explain about firewall with a suitable diagram.
12. How firewall works?
13. Define access control.
14. How Trojan horse works?

7.12 REFERENCES AND SUGGESTED READINGS

- www.geeksforgeeks.org
- www.javatpoint.com
- www.tutorialspoint.com
- www.easyengineeringclasses.com

UNIT 8: DISTRIBUTED OPERATING SYSTEM

Space for learners:

Unit Structure:

8.1 INTRODUCTION

Space for learners:

Distributed Operating Systems are the systems where the processors are interdependent via some communication network in a loosely coupled environment. Processors in a distributed system have distinct names such as host, site, nodes, computers, system etc. Each of the nodes has their own resources such as memory, system clock, kernel etc. For a specific processor, the resources of all other processors appear to be remote. They communicate with each other with the help of different communication media such as telephone lines or high speed buses.

The Structure of a Distributed System is as shown below:

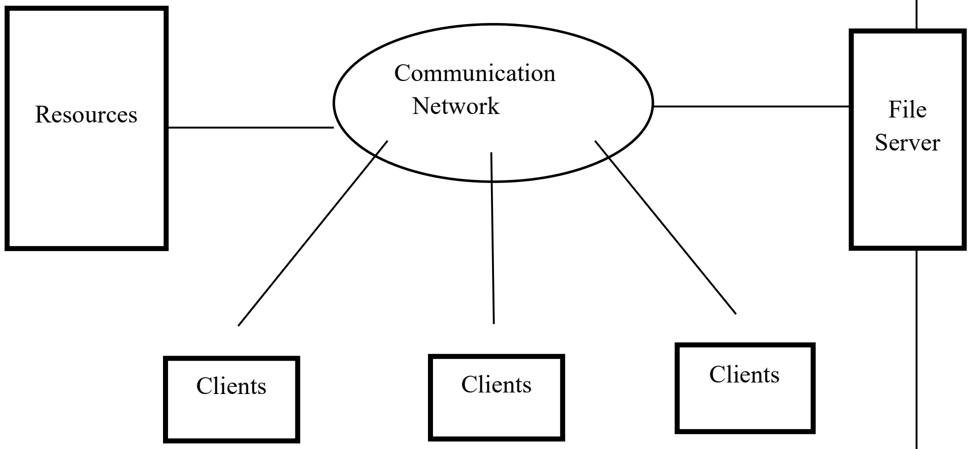


Figure 8.1: A Distributed System

8.2 UNIT OBJECTIVES

After going through this unit you will be able to:

- Learn the basics of Distributed Operating System
- Architecture of the system
- Technologies used for setting up the system
- Protocols used for communication
- Issues in designing a DS
- A brief description about distributed file system
- Some popular DFS

8.3 ADVANTAGES OF DISTRIBUTED OPERATING SYSTEM

Space for learners:

Different objectives can be there for using a distributed operating system: Sharing of Resources, Reliability, computation speed up and communication are some of them.

8.3.1 Sharing of Resources

The nodes those are connected to a distributed network can take the privilege of using resources from other nodes connected to the network. Resource may be either hardware (such as printer, scanner etc.) or software (such as text, audio or video files).If a node at site A needs a scanner, it can access it from some other site B. In the same way, a node at site C can access files remotely from some node in site D. Information processing can also be done remotely with the help of distributed database.

8.3.2 Scalability

Scalability of a system can be measured in different dimensions. It can be in terms of size that means extension of number of users or resources in the existing network. It can be administratively scalable, where you can expand the number of organizations of the system. Or the scalability can be in terms of geographical area where the organizations connected to the system are far apart from one another. A group of users can work on a project by geographical scalability. They can share files of the project they are working. They can make use of RPC (remote procedural call) and with the help of remote login; they can edit the code written by some other user.

8.3.3 Computation Speedup

If it is possible to split a computation into smaller parts then each of these parts can be executed parallelly in different nodes of the system and then recombine the sub-parts at the end of the execution. This can speed up the computation to a great extent. Furthermore, if a site is heavily loaded with processes, then some of the processes can be transferred to moderately loaded sites in the network. This technique is called load-sharing.

8.3.4 Reliability

Space for learners:

Reliability indicates the ability of the system being capable of functioning perfectly even after the failure of one or more components. For example, if a user has booked a railway ticket, and before the changes are permanently stored in the database, the system crashes, in such a scenario it is expected that changes made by the user should persist.

If the system is build up with multiple general-purpose computers, then working of the system will not be affected even when any of the computers fails. But in case, each of the machines of a system is responsible for performing some decisive task, then failure of one machine may lead to shut down of the whole system. In general, reliability depends upon redundancy. If one node shuts down suddenly, then there should be some other node which is carrying all the data and information of the previous one.

8.3.5 Communication

The sites that are connected in a distributed system can communicate with one another with the help of message passing. In a standalone system, message passing is done within the cooperating processes. This idea has been expanded in distributed system to send messages among users of different sites. Other functionalities such as email, remote procedural calls, file transfer etc. can also be incorporated in a distributed system. Communication can be of different types such as unicast, multicast and broadcast. In case of unicast communication one host will communicate with any other host in the system, in case of multicast, one host communicates with a number of hosts in the system and in broadcast communication, one host communicates with every single host present in the system at the same time.

Corporate sectors are also benefited from the distributed system. By replacing the mainframe with a distributed system, they can get ample number of resources from geographically dispersed areas, enhanced functionality in a minimal cost, and better user interface and less maintenance cost.

STOP TO CONSIDER

A distributed operating system is a set of loosely coupled systems connected via some communication network. Each of the systems has their own memory and processor. Resource sharing is possible in a distributed system even for geographically separated systems. A distributed system provides scalability in different aspects like geographical area, size or administrative scalability. Computation can be performed by dividing a problem into sub-problems. Corporate sectors can replace mainframe with a distributed system to reduce cost.

Space for learners:

8.4 TYPES OF NETWORK BASED OPERATING SYSTEM

A network based operating system is a group of computers having individual operating systems connected through a common network. The network works as a boss for the whole system. It groups the standalone computers and synchronizes their activities.

Network operating system can be broadly divided into two types:

Client server network and Peer to Peer network

CHECK YOUR PROGRESS

Q1. What can be the reasons for setting up a Distributed Operating System?

Q2. What are the attributes of the Processors of a DOS?

Q3. What is Load Sharing?

8.4.1 Client Server Network

These types of networks are considered to be the most common type of network operating system. The client-server concept changes slightly depending on the context it is using. One of them is the thin client computing, where the clients are light-weight computers having a small amount of memory. Only the graphical user interface is installed in the client machine and all other services are accessed from the server. The client machine acts as a terminal to have access on the operating system which is actually running on the Server. One example of thin client computing is the 80486 system having windows XP.

A wider sense of client server concept is used in Authentication Server Based Network. Each of the machines in the system has an account through which they authenticate themselves before using the resources from the server. A server is a powerful computer that is used for storing the information and resources to be accessed by the systems of the network. There are different security groups for different machines and depending on that, the user is given the access. The server provides security to the entire network and acts as resource manager. High cost software needs to be installed in the server in order to ensure smooth performance of the network. This type of network is termed as domain and the server is termed as domain controller in the Microsoft Network (MSN)

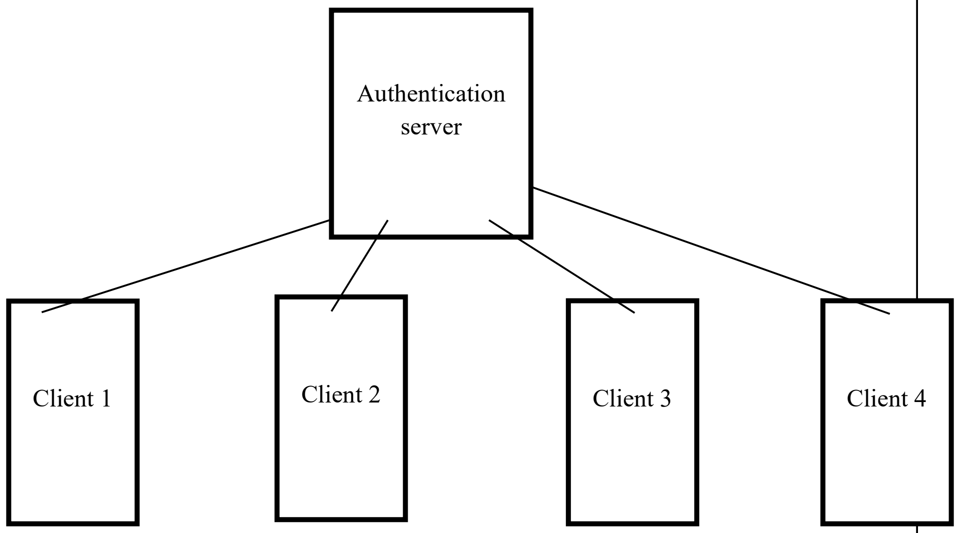


Figure 8.2: A client-server network

This architecture is mostly used in Organizations like Universities, colleges, banks and hospitals that need a dedicated server. It provides the facility of combining different parts of the network and gives concurrency transparency to its users, which means the same file can be accessed concurrently by different users leaving the database in a consistent state. These types of networks are useful for large organizations because of centralized security and management.

Space for learners:

8.4.2 Peer to Peer Network

Space for learners:

There is no authentication server present in peer-to-peer network but, the network may contain other type of servers like file server, fax server, remote access server and so on. Each of the computers in the network performs the services of both client and server. If any of the systems want to access data from other system, security is provided either by creating local account for the user or making the resources password protected. This type of architecture is suitable for small companies or in a home network. But if the network becomes bigger, managing of resources and creating local account becomes inconvenient. A user has to create hundreds of local accounts or he may have to remember all the passwords of different resources. On the other hand, in a client server network, user can access the network just by entering the network password and can avail the resources based on the permission given to him. The administrator will assign permissions for different resources making the task of the user much simpler.

Because of the absence of a dedicated server, compromised security has become the main pitfall of this architecture. On the other hand, absence of a server can reduce the cost of setting up a peer to peer network to a great extent.

The workgroups are less expensive as compared to client-server network for the following reasons:

- The operating system that runs on a server is costly as compared to general operating systems.
- More complex hardware are required for the server software
- System administrator needs to take extra load for maintenance of the tasks performed by the server.

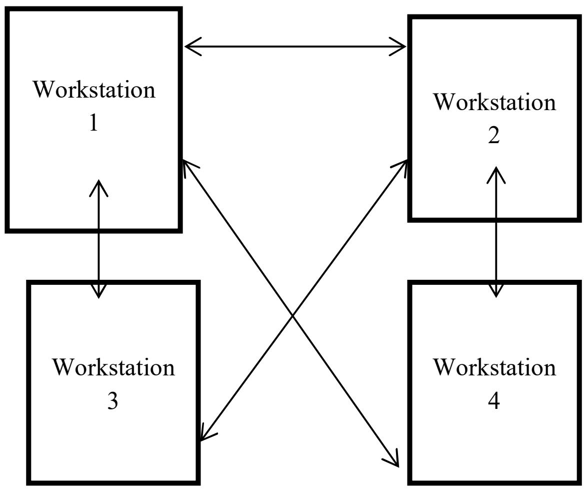


Figure 8.3: Peer to Peer Network

Space for learners:

8.5 NETWORK STRUCTURE

There are two types of networks LAN and WAN in a communication network. They differ mainly in the geographical area they cover. LAN (Local Area Network) as the name says, it covers a small geographical area such as a department, an institution etc. whereas, WAN(Wide Area Network) covers a large geographical area such as a country. It combines two or more LANs by using independent processors.

8.5.1 Local Area Networks

The Local Area Network has come into light in early 1970s. It has been developed as a substitute for mainframe computer. Most of the organizations prefer small computers each with its own application, instead of having a mainframe computer. Such an environment is more reasonable and user friendly as all the workstations have access to both software and hardware resources. And it is quite obvious that an organization needs to share a lot of information among the workstations, thus bringing LAN into focus.

As the LAN is meant for a small geographical area, such as in institutions, colleges, University Departments etc., the workstations of a LAN are adjacent to each other as compared to those of WAN. Because of this, the communication speed and rate of error are comparatively low. High-priced cables such as coaxial cable or fiber optic cables are used to achieve high speed and reliability. But if the LAN is covering a long distance, the cost of the network may increase with increasing cable length. Additionally, repeaters need to be added to boost the signal.

Most common way to construct a LAN is Ethernet cables, defined by IEEE standard 802.3. Speed of Ethernet cable ranges from 10mbps to 1gbps. The 802.3 standard defines the physical layer and MAC (Medium Access Control) sub-layers of OSI protocol. It is available in different versions.

802.3a (10Base2) uses thick coaxial cable with a bandwidth of 10mbps and maximum possible segment length 200m. Another variation of Ethernet cable IEEE 802.3i (10 Base T) uses Unshielded Twisted Pair Cable as a transmission media with a maximum speed of 10MBPS. Whereas the IEEE 802.3u (100BaseT) runs at a speed of 100mbps. Another variation called FDDI(Fiber Distributed Data Interface) is used by the LANs that extend up to 200kms. It uses fiber optics cable and provides a maximum speed of 100MBPS.

STOP TO CONSIDER

A communication network is built up with two types of networks: LAN and WAN. A LAN is preferred for small geographical areas. The Characteristics of the transmission Media used to set up a LAN are defined by IEEE standards. Variations in IEEE standard are available to provide different communication Speed. A LAN can be either wired or wireless. On the other hand, a WAN is preferred over a large geographical area. Technologies used for WAN connection are: leased line, dial-up connection, DSL, satellite communication etc.

Space for learners:

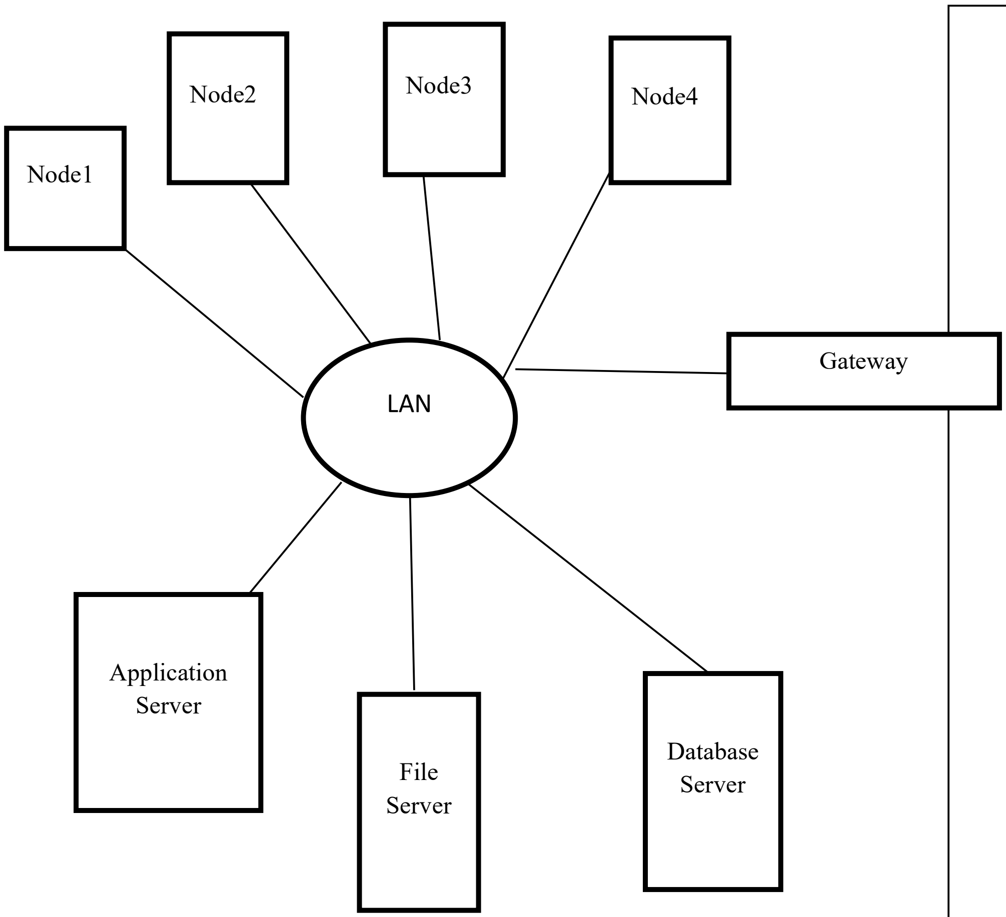


Figure 8.4: A LAN Network

As shown in Figure 8.4, nodes in a LAN are of different specifications from mainframe to personal digital assistant (PDA) which are connected to each other by different topologies. The network contain servers (File Server, Database Server, Application Server, Proxy server), peripheral devices (Printer, database Files), gateways to connect to different networks and Repeaters to amplify the signal.

In addition to cable connection, Local Area Network comes with wireless facility also. Wireless LANs use radio frequencies instead of cable. It is based on IEEE standard 802.11 released in the year 1997 with a speed of 2Mbps. Different variations have evolved over years, 802.11ax being the latest of all with a maximum possible data rate of 1.8 Gpbs.

Space for learners:

CHECK YOUR PROGRESS

Q4. What are the IEEE standards used in a wired LAN network?

Q5. What is the Job of an authentication server?

Q6. Point out the differences between LAN and WAN

Space for learners:

8.5.2 Wide Area Network (WAN)

The Wide Area Network was emerged in late 1960s. Idea was to connect workstations from a large geographical area, share resources, and perform confidential task in an economic way. LANs which are situated in different geographical area and are a part of same organization for example SBI(State Bank of India), can easily connect through WAN. Advanced Research Project Agency Network (ARPANET) was the first wide area network based on TCP/IP protocol and packet switching scheme. ARPANET initially connected five educational institutions including University of California and Los Angeles. With time, the researchers assembled the "network of networks" to give birth to modern day Internet. Internet is the largest WAN in today's world that establishes connections between LANs and MANs.

Some of the technologies used for setting up a WAN connection are:

Leased Line: These are dedicated lines that provide continuous data flow and can connect two or more LANs and MANs. Leased lines use fiber optic cable for high speed data and bandwidth.

Dial up Connection: Dial up connections use Public Switch Telephone Network (PSTN) for establishing an Internet connection. The telephone line is connected to a modem that converts the digital signals of the computer to analog signals used by the telephone lines. With times, this has become outdated because of its low data transfer rate and dependency on telephone lines to use the internet. That means the user is unable to use the telephone and the internet service at the same time.

Digital Subscriber Line: Digital Subscriber Line uses twisted pair cable for data transmission. These cables are generally used for telephone lines. By splitting the frequency, an uninterrupted service is provided to both phone calls and the internet. The technology behind it is as such: a variation of OFDM (Orthogonal Frequency

Division Multiplexing) called discrete multitone is used to divide the bandwidth in parallel paths so that at the same time, both phone calls and data transmission can be carried out. This is operated by the DSL modem connected to the telephone lines.

Satellite Communication: Satellite communication has become popular over the years for wireless internet accessibility. A satellite communication comes in a range of Low Earth Orbit (LEO) and Medium Earth Orbit (MEO). LEOs are mainly found in 1800 to 2100 miles above the earth and MEOs are found in 9000 to 10000 miles above the earth. Users can connect their laptops, cell phones and personal digital assistants to wireless network using the satellite communication.

Point to point link is used to connect the WAN with MANs and LANs. The WAN contains a networking device called packet switch that contains memory, Processor and input/output interfaces to connect with another packet switch. The message is stored in the memory, before being forwarded. Router is another networking device that takes the decision on which path; the arrived packet should be forwarded. There are two different routing mechanisms: static and dynamic. The static protocols take the decision based on the network topology, but it can't detect a link failure and modify the pre-defined route. On the other hand, dynamic protocols such as OSPF and RIP are capable of detecting link failure and accordingly update the route in the routing table dynamically.

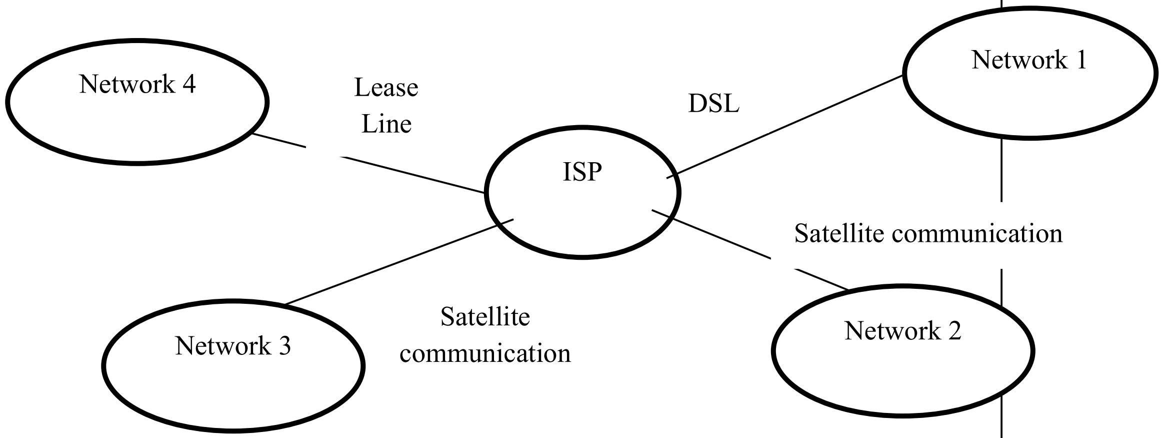


Figure 8.5: A WAN network

Space for learners:

8.6 COMMUNICATION STRUCTURE

Space for learners:

Communication Structure means the internal working of a distributed system. Here five issues need to be dealt with.

8.6.1 Name Resolution

In a distributed system, if a process in host A wants to communicate with a process in host B, they must know the address of each other. Therefore every process is recognized by two parameters <host name, id> where the host name is the unique name assigned to a particular host and id is the process within that host. Here comes the problem of Name resolution. Human beings communicate with the help of names, whereas computers find it convenient to use numbers for simplicity and better speed. Therefore, some mechanism needed to be introduced to convert the host names into numbers or ids so that the networking hardware can identify the receiver.

In the early days of Internet, each host used to maintain a data file which contains the host name and corresponding address of all the computers in the network. Whenever a system is added or removed from the network, every host needs to update their data file. With the growing number of networks, it has become cumbersome to update the file every time. Here comes the concept of Domain Name System (DNS). The protocols like TCP and IP convert the host names to IP addresses using the procedure called name resolution. The DNS is a distributed database system based on client/server architecture that is used for converting Host Name to IP addresses. A Domain Name (mostly called as Domain) is a name that is associated with an IP address. There are various kinds of Domain are available such as com for commercial sites, org for nonprofit organizations, country specific domain such as .in .uk .un etc. All these come under top level domain name where .org is a Generic Top Level Domain (GTLD) and .uk .un are Country code Top Level Domain (CcTLD).

When a host requests for an IP address, sayYahoo.com, the query is resolved in reverse order. The operating system will search the IP address in its local cache, in case it is not found, the query will be sent to resolver server. If the IP is not present in the cache of resolver server, then it will send it to the next level server i.e. the root server. Root server will send the resolver server to Top Level

Domain Server for .com domain. The resolver server will then direct the query to authoritative server for the second level domain i.e. yahoo.com. The authoritative server will provide the resolver with the IP address. The resolver will provide the host with the IP address of Yahoo.com. The resolver server keeps the address in its cache so that, next time when the request comes for yahoo.com, it can directly provide with the IP address.

Space for learners:

8.6.2 Routing Strategies

There may be different routing strategies for sending a message from host A to another host B. Routing tables are maintained to send packets through the most reliable path. Routing table contain the information like network id, subnet mask, next hop and minimum number of hops to reach the destination. Based on the network condition, the best possible route can be updated time to time. Most common routing strategies are: fix routing, virtual routing and dynamic routing.

- Fixed Routing: Here the optimal path is chosen between two hosts. The path is not updated later on, unless some hardware failure damages it permanently. Therefore, even if the path has a heavy traffic compared to the other paths, there is no option to change the path and adopt some lightly loaded path.
- Virtual Routing: In virtual routing a dedicated path is given for a particular session. Two hosts A and B can use different routes for different session.
- Dynamic Routing: The route is assigned dynamically before starting a communication between two sites. Messages may be sent through different paths. Dynamic routing algorithms such as RIP and OSPF permit the routers to share routing information with nearby routers in order to get the optimal path. Since the path is decided dynamically; out of order packets may arrive at the destination. Therefore a sequence number is added to each of the outgoing packets so that at the receiver side, they can be reassembled. Though dynamic routing is complex to set up, it is suitable for huge networks.

STOP TO CONSIDER

In order to reduce collision in the network, routing strategies are introduced. They can help sending a packet from source to destination through an optimal path. With advancement in technologies, dynamic routing algorithms are introduced that can update the path by collecting information from nearby routers.

Space for learners:

It is possible to use both fixed and dynamic routing in the same system. The sender may send a message to the gateway using a fixed routing scheme whereas the gateway uses dynamic routing to transfer the message to the destination network.

8.6.3 Packet Strategies

The data can be of variable length. To simplify the communication, a message is divided into fixed length unit named as packet, segment, frame etc. The transmission can be connectionless or connection-oriented. In case of connectionless transmission (such as in case of UDP), sender don't get any information whether the packet has reached the destination or not. If a message is divided into multiple packets, a connection is established to ensure reliability. In a connection oriented transmission, receiver sends an acknowledgement to the sender for the packets received. An acknowledgement can be either single or cumulative.

8.6.4 Connection Strategies

Three main strategies for establishing a connection are: packet switching, message switching and circuit switching.

- **Circuit Switching:** If two nodes want to communicate, a dedicated path is assigned to them. The Path cannot be used by other processes for the entire duration of the communication even if it is idle for some time. One of the examples of circuit switching is telephone network. Once user A calls another user B, others parties can't use that line until they hang up. Because of a dedicated communication channel, data rate is guaranteed in a circuit switching network but more bandwidth is needed to set up a circuit.

Space for learners:

- **Message Switching:** There is no direct connection between sender and receiver in a message switching network. Instead, the intermediate nodes or switches receives the message and transfers it to the next hop. Each message contains a header that carries the information like source and destination addresses, Error Checking code and expiry date. Each hop needs to have sufficient resource to retransmit the message to the next hop in the network. In case, enough resource is not available the message is stored for an indefinite period. This process is called store and forward. More than one message from various senders can be sent over the same link. Though message switching is better than packet switching strategy, it is not suitable for real-time data transfer since the processing takes place in each of the intermediate nodes, making the overall process slow. Also each of the intermediate nodes should have a large storage capacity to store the entire message, since the message can be of various lengths.
- **Packet Switching:** The message is divided into variable length data units called packets. Each of the packets contains control information and payload. The packets are free to follow different paths depending on the traffic of the network. On receiving side, the packets that belong to the same file are reassembled. Some of the advantages of this network are: it ensures reliability as the receiver can detect the missing packet. If a link is down, the packets can take another link, thus making it fault tolerant. Transmission latency is minimal. It makes best use of the network bandwidth therefore packet switching is the most commonly used connection strategy.

CHECK YOUR PROGRESS

Q7. What is the difference between fix routing and dynamic routing?
Q8. How the conversion of Domain name and IP address is performed?

8.6.5 Contention

Space for learners:

In a communication network, it is possible that more than two sites are transmitting messages simultaneously over the same link (for example in a mesh topology). If collision happens, there should be some mechanism to discard the message and inform the sender, so that the message can be retransmitted. If no mechanism is designed to avoid collision, it may be repeated resulting degradation of the system performance. Some of the techniques for collision detection are discussed below:

- CSMA/CD (Carrier Sense Multiple Access/Collision Detection): This is a media-access control (MAC) protocol used mainly in Ethernet LANs. Each of the stations in a network, sense the channel before starting the transmission. In case, the channel is busy, it will abstain from transmitting and continues to sense the channel. If the channel is free, the station will start sending. Suppose station A and station B starts transmitting at the same time, then both the signals will collide. As the stations receive the collision signal, they will stop transmitting. This is called collision detection. Each of the stations will again try after some random amount of time. If no collision is detected during the transmission, the sender will complete the transmission.

  As the number of nodes increases in a network, possibility of collision also increases, resulting in performance degradation. One solution to this problem is to limit the number of hosts in a network. Adding more nodes in a congested network may result in bad throughput.

- Token Passing: This is an access-control protocol implemented in a Ring topology. A token is a small message that contains a pre-defined bit pattern. The token is passed in the network in either clock-wise or anti-clock wise direction. When a node wants to transmit, it removes the token from the network and start transmitting. No other node is allowed to transmit without having the token. Once the node finishes its transmission, it releases the token allowing other nodes in the network to transmit data. If the token is lost, the system adopts an election algorithm to select a specific site for generating the token.

A token-passing protocol can give constant performance. As the number of hops increase in the network, average waiting time may increase but it will still perform better than that of an Ethernet network. However, for a small network, LAN is preferable.

Space for learners:

8.7 COMMUNICATION PROTOCOLS

The communication network must have some set of rules for establishing connections, transferring packets, error detection, selecting the shortest path and so on. To deal with all these issues, the whole process is divided into some layers. Each of the underlying layers performs their assigned duties and sends the message to the upper layer. A message sent by a host, passes through all of the layers and then enters to the recipients system. Each layer is bound to follow specific protocols while communicating. The International Standards Organization (ISO) has defined seven layers as described below:

- Physical Layer: This layer is responsible for transmitting the message in the form of bits. Bit rate (the number of bits transmitted per second) is also defined by physical layer. The physical and logical structure of the network (known as network topology) is defined in this layer. Along with that, physical layer also defines the mode of transmission i.e. duplex, half duplex and full duplex.
- Data-link Layer: Data link layer divides the bit streams sent by the physical layer into some data units called frames. It is responsible for detecting lost or damaged frames and adds mechanisms for retransmission of the frames. If two or more devices are connected to the same link, which device will get access of the link is determined by data link layer. Flow control is also performed in this layer.
- Network Layer: Network layer is responsible for source to destination delivery of the packets along with assigning logical address and selecting routes for outgoing packets.
- Transport Layer: Transport layer delivers the packet received from network layer to the correct process within that host. This is called process-to-process delivery. This layer

provides connections to the packets, performs end to end flow and error control.

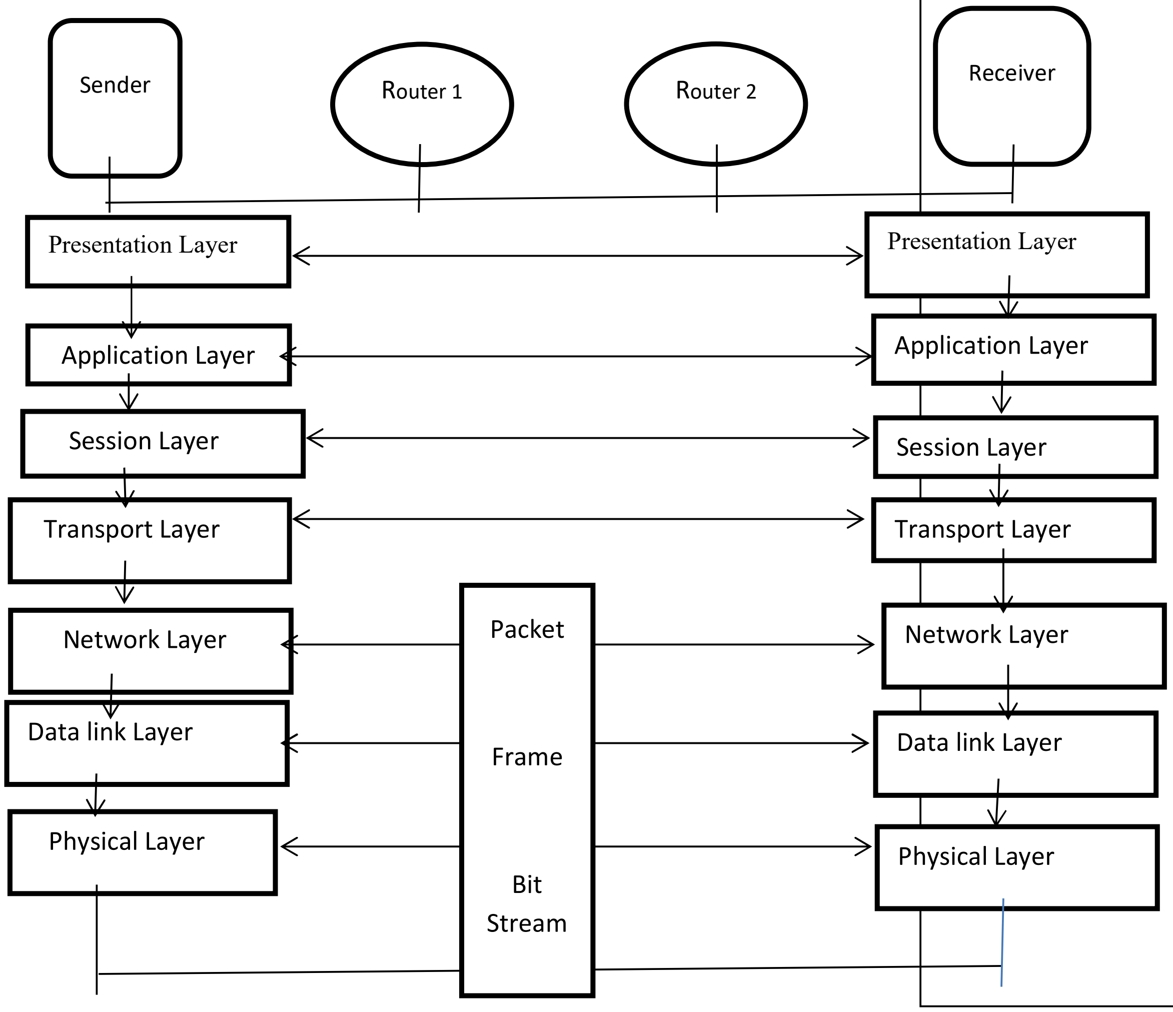
- Session Layer: This layer is responsible for identifying the mode of communication (half duplex, duplex, full duplex) in a particular session. This is called network dialog control. The session layer adds check points or synchronization point to a data stream so that in case of any failure, only the part of the message after the check point can be retransmitted.
- Presentation Layer: The presentation layer is responsible for translation of the messages from one format to another. As different computers use different encoding systems, message transferred in one format needs to be translated to some other format. This task is performed in the presentation layer. It also performs encryption and decryption of data. Another responsibility of this layer is the compression of data stream which is particularly important for multimedia data such as video, audio etc.
- Application Layer: This layer is responsible for direct interaction with the user. It allows a user for remote access to a system, and controlling files in the remote system. It deals with email and the organization of distributed databases.

Figure 8.6 explains the OSI protocol stack. In sender side, the message travels from presentation layer to the physical layer, each of the layers adding their own header with the message. Once the message is converted to a bit stream, it is transferred to the receiver; through some transmission media for example fiber optics or twisted pair cable in case of wired medium and radio wave or microwave in case of wireless media. On the receiving side, the message travels from physical layer to application layer, removing the corresponding headers in each layer. Logically each of the layers in sender side communicates with each corresponding layer in the receiver side as the protocols defined by a particular layer can be understood by that specific layer only.

A message needs to cross one or more router before reaching the intended destination. Each of the routers needs the IP address of that message in order to direct it into the correct route and thus unpacking the message upto the network layer. For this reason, the physical, data link and network layers are known as hardware layers.

Space for learners:

Space for learners:



Transmission Media

Figure 8.6: OSI protocol stack

The most widely used protocol stack is the TCP/IP protocol stack. It has fewer layers as that of OSI model. This model is more reliable as compared to OSI model. Application layer of TCP/IP model which is a combination of session layer, presentation layer and application layer, uses different protocols like HTTP, FTP, SMTP, SNMP etc. Transport layer has two protocols UDP (User Datagram Protocol) which is a connection less protocol and TCP (Transmission control protocol) which is a connection-oriented protocol. Next lower layer is the Internet Layer, the basis of which is the IP (Internet Protocol) which routes the IP packets. There is no dedicated physical link in TCP/IP model, allowing the data packets to travel in any physical path.

8.8 DESIGN ISSUES

The users of a distributed system should feel like they are working in a traditional centralized system. Transparency is one of the key design issues in a distributed system. It can be measured in different parameters. Location transparency hides the details of storage location of the resources. Replication transparency hides the number of copies present for the same data. Concurrent transparency allows multiple users to access the same file concurrently without their knowledge. Parallelism transparency allows parallel execution of the activities without user's knowledge. User mobility is another kind of transparency where a user is allowed to login to the system from any machine.

Fault tolerance is another important issue. A system should be able to tolerate different kinds of failure like machine failure, crash of storage devices, and link failure to some extent. However, the performance of the system will reduce because of the failures. A system is not fault tolerant if it stops working with the breakdown of some of its components.

Another important aspect is the scalability. Scalability means how well the system will work as it grows in terms of resources and number of systems. Scalability can be measured in three dimensions

- Size scalability

- Geographical scalability
- Administrative scalability.

Space for learners:

A size scalable system should function properly when the number of components increases. The system may experience high traffic in a specific day. The existing database may not be able to handle the traffic, which brings the need for adding more databases or more servers to the system. If the system is truly scalable, adding more resources should not reduce system's performance and it should not get slower. If the system is based on centralized data, centralized service and algorithms, size scalability may have to deal with different issues.

In case of centralized server, where a single server is responsible for the implementation of different services, congestion may result with growing number of users and applications. But it is unavoidable to use centralized server in some confidential situations like banking, medical, administration etc. where a single server is used to store all the sensitive information and separating it by special networking devices from the rest of the network. The problem persists in case of centralized data also. If the Domain Name Service (DNS) is implemented in a single database, each request over the internet would be forwarded to that particular database causing a severe congestion on the link. At last, the centralized algorithm is also a bad idea. A large distributed system has tremendous number of messages routed in different links. The idea here is to collect all the routing information and redirect it to a single machine and the algorithm computes the best suitable path. The information may create heavy traffic in a part of the network. To deal with this issue, decentralized idea has come up where no single machine stores all the routing information and they make decision only based on local information.

Stop to Consider

Design issues of a distributed system bring the issue of scalability, fault tolerance and transparency into focus. A scalable system should work perfectly even it grows in terms of size, geographical area or administrative organization. A fault tolerant system should not be able to tolerate faults to some extent. The term transparency means the underlying protocols should be hidden from the end-user.

Geographical scalability means whatever be the distance between the user and the resources, the user should be able to access them efficiently. Adding new nodes to the system should not slow down the transmission rate. The synchronous communication approach is suitable for small geographical area networks such as LANs. But in case of WAN, where two processes are geographically far apart, successful implementation of inter-process communication with synchronous communication is not an easy task.

Administrative scalability means even if an organization spans in many administratively independent domains, it should still be easily manageable. Achieving an administrative scalability is the toughest of all since it includes some non-technical issues such as policies of an organization and cooperation of humans. Security issues are also involved here. If a new domain is built up, all the other domains need to protect themselves from the new domain. The new domain may only have the read access to the files of other domains. Similarly, in order to protect itself from malicious attacks, the new domain may restrict its accessibility to the foreign code such as java applets in a web browser.

Check Your Progress

Q9. What do you mean by size scalability? Discuss the issues that have to be faced to ensure size scalability.

Q10. Why administrative scalability is hard to achieve?

8.9 DISTRIBUTED FILE SYSTEM

The nodes of a DFS (Distributed File System) can have remote access to the files of the system; i.e. the clients and servers are scattered over the network. Unlike the local file system, a distributed file system has multiple copies of storage over different servers. The DFS is implemented in a number of ways: in some systems servers run on particular machines, while some machines work as both client and server. The DFS may be implemented in the Operating System itself or as distinct software that connects the traditional operating systems with the file system.

Features of DFS are:

8.9.1 Naming and Transparency

Space for learners:

The users of a computer system deal with a file with the help of file name which is actually a logical representation. The operating system then locates the particular data blocks (for the file) stored in the disk. Once the file is referred by the user with a textual name, that name is converted to some numerical value which in turn is mapped to the blocks of disk. This mapping hides the storage details of the file from the user. This is called abstraction. The DFS provide file replication along with abstraction. Multiple copies of files are stored in different systems in case of a DFS. When a file name is referred, the mapping will come up with a set of the replicas of that file. But the user is unaware of the existence of multiple copies.

8.9.1.1 Naming Structure

Two important aspects related to naming are *location transparency* where the physical location of a file is hidden from the user and *location independence* where the physical location of a file can be changed without changing its name. Location independence is a dynamic mapping property as the same file name is mapped into more than one location at different times, whereas location transparency is a static property. For these systems location migration is not possible i.e the location of a file can't be changed automatically. However, manual changing of files within machines is possible.

8.9.1.2Naming Schemes

Three naming schemes are there for a DFS. In the first scheme, a file is identified by its host name and local name which differentiates it from other files in the system.This scheme does not provide location transparency or location independence. Local and remote files can be accessed with the help of same file operation. In a DFS each of the traditional file system is treated as one of its components. In the first approach, some provisions are there for remote access of these component units.

In the second approach; the remote directory can be attached with local directories giving the appearance of a coherent directory tree. An implementation of this approach is the Network File System (NFS).

In the third approach, a global name structure spans all the independent components. The file structure composed here is same as that of a traditional file structure. But this approach is difficult to implement because of some special machine specific files like device files and binary directories that exist in UNIX environment.

The NFS directory scheme is the most difficult scheme to implement. The reason being, any remote directory can be attached by any machine in any of the local directories. If the server is facing some issues, the directories added by some computer may not be available in the global structure. An accreditation scheme is used to decide which machine will add directories at what time. It may happen so that the same client will be able to access a directory in one machine while it will be denied access for that in some other machine.

8.9.1.3 Implementation Techniques

In order to manage the mapping easily, sets of files are aggregated to some component units and mapping is done on these components. The textual files are mapped to low level file identifies that helps in finding the component into which the file belongs. Structured names are used to implement low level identifiers. They are string of bits containing two parts: first part is for the component unit and second part is for the file within the unit. To maintain the uniqueness of a file, sufficient bits are used to ensure that the name used for the file is not being used by any other file. Another way to maintain uniqueness is to add a timestamp with the name.

8.9.2 Remote File Access

If a user wants remote access to a file, the server that is storing the file is identified by the naming scheme. The data transfer procedure for remote files is similar to that of traditional disk-access method. In case of traditional disk-access method, caching is used to reduce the input/output of disk, whereas in remote file access, caching helps both in reducing disk input/output and network traffic. When a request for a disk-access comes, it is first checked in the cache, if it's not present then it has to be brought from the server. A cache mapping technique (least recently used) is applied to store the files in the cache so that next time when an access request comes, there would not be any need to go to the server, thus reducing the network traffic. One master copy resides in the server and replicas of that copy exists in different caches. As a file in cache is modified, that

Space for learners:

modification should be reflected in the master copy to maintain the consistency. The concept of demand paging is also implemented exactly same as that of traditional file systems, except that the backing store is a remote server rather than a local disk.

Consistency is another issue in file access. A client machine may want to check whether the cached copy of data is same with the master copy. For verifying the data, two approaches are used: in client-initiated approach, the client will start a validity check by contacting the server and checking whether the replica of the file is consistent with the master copy. The validity check may be done on every access or on first access of the file. In server initiated approach, the server reacts to the inconsistencies. The server is notified when the same file is opened in conflicting mode (Read-write, write-write) by two different clients. It can disable caching for that particular file to avoid inconsistency.

Stop to Consider

Unlike the conventional file system, a distributed file system has files from different geographical areas. The storage details of the file are hidden from the end-user by a method called abstraction. The DFS also ensures the features like location transparency and location independence. For remote access of a file, caching is used to reduce the network traffic and latency. Consistency is maintained in a DFS with the help two approaches: client initiated approach and server initiated approach.

8.9.3 Stateful Versus Stateless Service

Stateful service is a connection-oriented service. Here a client first gives open ( ) command before accessing the file. The server stores the file in its memory and sends a unique connection-identifier to the file. The same identifier is used for all the file-access during a particular session. An example of stateful service is AFS(Andrew File System).In stateless services, each request can recognize the file in the memory along with the read/write access of the file. There is no need of establishing and terminating a connection here. There is no concept of session and each file request is considered as individual request. NFS (Network File System) is a stateless service.

Space for learners:

The performance is better in case of a stateful service as it can store the files in its cache memory thus reducing the disk access which can't be done by stateless service. Again a server in the stateful service knows where a file is open, thus it can directly read the next blocks of the file in case of sequential access. In case of a failure, the server of a stateful protocol losses its states and a recovery protocol is needed to restore the state. A stateless protocol does not face these problems since all the requests are self-contained.

Space for learners:

8.9.4 File Replication

Replication is the process of keeping multiple copies of the same data in different nodes of a distributed system. The reasons behind data replication are as follows: it provides better availability. The system can work even if one or more nodes fail. Replication reduces the latency of a file access, as the file is kept in a short distance from the user. Since the file read operation is a non-conflicting one, the read query request for multiple hops can be performed from different replicas thus increasing the throughput of the overall system.

One main problem associated with replication is consistency. Since file replication is transparent to the user, changes made in one copy should be reflected to all the other copies. Consistency models are broadly divided into client centric consistency model and data-centric consistency model. In client-centric consistency, data may not be updated in all the servers parallelly rather; they are propagated from one server to another. Therefore, some of the processes may have to work with the old data which results in compromising the consistency. But lower consistency may give server availability and increased throughput. Some of the models that come under client-centric consistency are: eventual consistency; where data is updated at the end, leading to inconsistent data in some servers. This model suffers from lost update problem. Next model is the monotonic read; here, if a process reads a data item x, the successive processes will get either the same value or the latest value of x. Another model is monotonic write, where a sequence is maintained for all the write operations of a process and value is updated based on that sequence. Next to this model is the read your write model, where the write operation performed by a process is reflected in the server where a read operation is being performed on the same file. Next comes the write follow read model where a process reads the value before performing the write operation.

CHECK YOUR PROGRESS

Q11. What are the advantages of DFS over traditional file system?
Q12. What are the reasons for replicating a file?
Q13. What do you mean by stateful and stateless service?

In data-centric consistency models an updated query is immediately reflected in all the other servers. Therefore many update operation are being performed at the same time. Different models are there in data-centric consistency model. The strongest of all is the external consistency. Any process that reads the value of a data-item x, will get the last updated value of the write operation. Google Spanner Distributed Database uses this consistency model. In situation-dependent consistency, the execution order of two processes is reflected in the same order into all the servers.

Some of the applications of distributed file systems are:

8.9.5 Andrew File System (AFS)

AFS was designed for the operating systems like BSD (Berkeley Software Distribution), UNIX and Mach. An AFS is made up of the structural elements called cells. Cells consist of servers and client machines. Servers and clients belong to a particular cell. The users can have accounts in more than one cell. The first cell, that a user logs in, is called home cell and all the other cells are named as foreign cells. Irrespective of the location of the user, the path of a file in the AFS tree will always be same. File access permissions such as read, write and update are set by the access control lists. AFS follows stateless protocol, therefore servers and clients don't store the file access information. AFS runs on TCP/IP. The Remote Procedural Call (RPC) that is designed for AFS performs the communication between client and server irrespective of their geographical area. Caching is applied to reduce the network load. For each file access request, the respective file is searched in the cache. If the file is not present then it will be accessed from the server. In case there is any modification in the cached copy of the file, the file is propagated back to the server. The frequently

__Space for learners:__

accessed files are stored in "working set" of the cache, thus can be accessed directly from the cache reducing the latency and network load. Generally the size of cache memory is 100MB.

Space for learners:

8.9.6 Google File System (GFS)

It was developed in the year 2003 to meet the growing demand of data processing systems. The system is scalable and can support a large number of clients without degradation of performance. It supports fault tolerance while implemented in inexpensive hardware. Like other Distributed File System, GFS ensures reliability transparency and availability. In addition to these, some specific design goals included in GFS are: a huge amount of data can be stored redundantly in inexpensive computers. It is capable of processing huge number of requests.

GFS is based on cluster based architecture. A cluster consists of one master node that manages Meta data, a number of chunk-servers that store the files in chunks and a number of clients. The Meta data includes the information about who can have access to the file, mapping the files to the memory chunks and determining the current location of the chunks. GFS is a stateful system therefore it manages the state information of all the clients. The client sends file access request to the master node. In response to the request, the master node sends Meta data to the clients. The client can then directly contact the chunk server for the file. Fault tolerance feature is also implemented in the system by keeping three replicas of the same files. If a chunk server is down, then the master server can redirect the client to one of the replicas. In case the master is down, any of the chunk servers can act as a master by keeping a Meta data list.

8.10 SUMMING UP

- A Distributed System is a collection of independent computer systems that have their own memory and processor.
- They may be of different specifications from single microprocessor to general purpose computers connected through some communication medium like twisted pair cables, fiber optic cables and satellite communication.

- They can be arranged either in client-server architecture or in a peer to peer network.
- The internal working of a distributed systems deals with different issues like name resolution, routing strategies, collision avoidance techniques, connection strategies and it should solve contention problem and ensure security.
- Each message travels through the layers of networking models before reaching the destination. Implementation details of a distributed system should be transparent to the user. S/he should feel like working in a conventional system having no difference between a remote file and a local file.
- The system should be scalable in different aspects like size, administrative organization and geographical area. If some part of the system fails, the performance of overall system should not degrade.
- A DFS is a file system consisting of geographically dispersed clients and servers. A user of a DFS is unaware of the location of a file it wants to access. A local and remote file appears same to the user.
- Different DFS have been designed based on technical needs. Each of which have their own tradeoffs and advantages. Some of them are useful in small networks while some are designed for large distributed systems.

8.11 POSSIBLE QUESTIONS

1. What do you mean by a Distributed System? Explain the advantages of a distributed system.
2. Describe the Architecture of Client-Server system and Peer-to-Peer system
3. Briefly discuss the technologies used in LAN and WAN network
4. Describe the concept of Domain Name System. How does it help in name resolution?
5. Describe the methods used for collision resolution
6. How does a Routing strategy help in transmitting a file in a distributed system?

Space for learners:

7. Discuss some of the fundamental differences between Andrew File System (AFS) and Google File System (GFS).

Space for learners:

8.12 REFERENCES AND SUGGESTED READINGS

- Distributed Systems: Principles and Paradigms Second Edition Andrew S Tanenbaum, Maarten Van Steen
- Operating System Concepts Seven Edition SILBERSCHATZ GALVIN GANGE