PG CA S3- 03 (DSC)

TWO YEAR MASTER'S DEGREE COURSE

SEMESTER-III

OPERATING SYSTEM

MSc IT

Hiranya Chandra Bhuyan School of Science and Technology

KRISHNA KANTA HANDIQUI STATE OPEN UNIVERSITY

Subject Experts

Prof. Anjana Kakati Mahanta, Gauhati University

Prof. Hiren Kumar Deva Sarma, Gauhati University

Dr. Lipi B Mahanta, IASST, Guwahati

Dr. Sanjib Kr. Kalita, Gauhati University

Course Co-ordinators :

Dr. Ridip Dev Choudhury, KKHSOU

Dr. Tapashi Kashyap Das, KKHSOU

Dr. Sruti Sruba Bharali, KKHSOU

SLM Preparation Team

UNITS	CONTRIBUTORS
1, 8, 9	**Mr. Dipankar Dutta,** NERIM
2, 3, 4	**Dr. Ridip Dev Choudhury,** KKHSOU
5, 7	**Dr. Sruti Sruba Bharali,** KKHSOU
6	**Dr. Khursid Alom Borbora,** IDOL
10	**Dr. Swapnanil Gogoi**, IDOL
11, 12	**Mr. Adarsh Pradhan,** GIMT
13, 14, 15	**Dr. Utpal Barman,** GIMT

Editorial Team

Content : **Dr. Pranab Das,** Assam Don Bosco University

Structue, Format & Graphics: **Dr. Sruti Sruba Bharali,** KKHSOU

July, 2024

This Self Learning Material (SLM) of the Krishna Kanta Handiqui State Open University is made available under a Creative Commons Attribution-Non Commercial-Share Alike 4.0 License (international): http://creativecommons.org/licenses/by-nc-sa/4.0/

Printed and published by Registrar on behalf of the Krishna Kanta Handiqui State Open University

Headquarters: Patgaon, Rani Gate, Guwahati-781017

City Campus: Resham Nagar, Khanapara, Guwahati-781022; Web:www.kkhsou.in

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY

Operating System

DETAILED SYLLABUS

		Pages
UNIT 1 :	**REVIEW OF COMPUTER ORGANIZATION** Major Subsystems, I/O Organization, Memory Organization, Bus Organization.	**7-27**
UNIT 2 :	**INTRODUCTION TO OPERAING SYSTEM** Definition of Operating System, Batch System, Multi-Programmed System, Time-Sharing System, Single User Operating System, Multi-User Operating System.	**28-36**
UNIT 3 :	**PROCESSES** Process: process models, process hierarchies, process states; Threads: use, design issues of thread; Types and Application.	**37-48**
UNIT 4 :	**INTERPROCESS COMMUNICATION** Interprocess communication, race conditions, critical-sections, mutual exclusion, solution to race condition, disabling interrupt, Peterson's solution, sleep & wake up (The Producer Consumer Problem), Semaphores	**49-59**
UNIT 5 :	**SCHEDULING** Basic concepts, primitive and non-primitive scheduling, types of scheduling - batch, interactive and real-time, scheduling algorithms, goals of scheduling algorithms, first come first serve, shortest job first and round robin scheduling	**60-75**

UNIT 6 : DEADLOCKS **76-89**

Definition of deadlock, principles of deadlock (deadlock conditions & modelling), deadlock detection, recovery & prevention, deadlock avoidance (banker's algorlithm)

UNIT 7 : MEMORY MANAGEMENT **90-110**

Multiprogramming: fixed partitions, relocation and protection; Swapping and its basic concepts; Virtual Memory: Basic concepts, Paging and Page tables, Page Replacement Algorithms (FIFO, LRU; Cache Memory.

UNIT 8 : MEMORY MAPPING **111-135**

Associative Mapping, Set-Associative Mapping, Block Set-Associative Mapping, DMA.

UNIT 9 : FILE SYSTEM **136-154**

Definition of File, File naming, File types (directory, regular), Sequential access and Random access files, File attributes, Operations on file, Hierarchical directory structure, Path name (relative and absolute), Operation on directories, File system implementation techniques.

UNIT 10: I/O MANAGEMENT **155-176**

Basic principles I/O Operations, I/O Devices, Device controllers, DMA, Principles of I/O Software, its goals, Interrupt Handlers, Device Drivers.

UNIT 11: PROTECTION **177-194**

Needs for protection, domain of protection, Example in UNIX, access matrix, Implementation of Access matrix.

UNIT 12: SECURITY **195-217**

The security problem, Authentication, Password, password vulnerabilities, Encrypted password, One time password(OTP), Biometrics and its types, Intrusion detection (definition only).

UNIT 13: MULTIPROCESSOR SYSTEMS **218-227**

Types of Multiprocessor Operating Systems, Multiprocessor OS Functions and Requirements, Multiprocessor synchronization.

UNIT 14: DISTRIBUTED OPERATING SYSTEMS **228-241**

Distributed Processing, Coping with Failures Models of Distributed systems, Remote procedure calls, distributed Shared Memory, Distributed File Systems.

UNIT 15: INTRODUCTION TO MULTIPROGRAMMING SYSTEM **242-257**

Queue management, I/O supervisors, memory management. File system, Disk and Drum Scheduling.

COURSE INTRODUCTION

Welcome to the “Operating Systems” course, a comprehensive exploration into the core mechanisms and functions of operating systems. This course is designed to provide you with a deep understanding of how operating systems work, focusing on their architecture, functionality, and critical roles in managing computer resources.

We will begin with a review of fundamental computer organization concepts to establish a solid foundation. From there, we will explore various types of operating systems, including batch, multi-programmed, time-sharing, single-user, and multi-user systems. Key topics such as process management, interprocess communication, and scheduling algorithms will be covered to understand how operating systems handle multiple tasks efficiently.

The course will delve into memory management techniques, including virtual memory and page replacement algorithms, ensuring you understand how operating systems optimize the use of RAM. File systems, I/O management, protection, and security will also be major areas of focus, highlighting how operating systems safeguard data and manage hardware interactions.

Advanced topics such as deadlocks, multiprocessor systems, and distributed operating systems will be addressed, providing insights into more complex and modern computing environments. By the end of the course, you will have a comprehensive understanding of operating systems, equipping you with the knowledge to tackle both theoretical and practical challenges in the field of computer science and software engineering.

Each unit of this course includes some along-side boxes to help you know some of the difficult, unfamiliar terms. Some “EXERCISES” have been included to help you apply your mind to some of the points in the units. You may find some boxes marked with: “LET US KNOW”. These boxes will provide you with some additional interesting and relevant information. Again, you will get “CHECK YOUR PROGRESS” questions. These have been designed to enable you to self-check your progress of study. It will be helpful for you if you solve the problems put in these boxes immediately after you go through the sections of the units and then match your answers with “ANSWERS TO CHECK YOUR PROGRESS” given at the end of each unit.

UNIT 1: REVIEW OF COMPUTER ORGANIZATION

UNIT STRUCTURE

1.1 Learning Objectives
1.2 Introduction
1.3 Computer Types
1.4 Functional Units
    1.4.1 Input Unit
    1.4.2 Memory Unit
    1.4.3 Arithmetic and Logic Unit
    1.4.4 Output Unit
    1.4.5 Control Unit
1.5 Basic Operational Concepts
1.6 I/O Organization
1.7 Bus Organization
1.8 Performance
    1.8.1 Processor Clock
    1.8.2 Basic Performance Equation
    1.8.3 Instruction Set
    1.8.4 Compiler
1.9 Let Us Sum Up
1.10 Further Reading
1.11 Answer To Check Your Progress
1.12 Model Questions

1.1 LEARNING OBJECTIVES

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

- describe the basic types of computer systems
- describe the functional units of a computer system
- explain I/O organization, bus organization, memory organization and memory hierarchy
- explain the steps required to execute an instruction and functioning of the control unit

- describe how to measure the performance of a processor.

1.2 INTRODUCTION

A computer system can be defined as an electronic device that is capable to accept input data, process that data and generate result/output. A computer can receive information or data from external sources, perform the required operations on the inputs, and can send the output to external devices. A system consists of a set of electronic devices and circuits, a display unit, storage media, and a set of communication media. This chapter focuses on the basic functional units of a computer system including I/O devices and their organization, memory unit, and different bus structures to carry the data, addresses, and control signals among the input/output peripherals, CPU, and memory unit. The unit also describes the role of an instruction cycle, the performance of a system, clock rate, etc. It introduces a set of common terminologies and a brief overview of the fundamental aspects of computer organization and required system software.The next unit gives an introduction to the different operating systems.

1.3 COMPUTER TYPES

The computer system can be classified into different categories based on size, performance, cost, processing power, etc. Some of them are digital computers, mainframe computers, desktop computers, notebooks, personal computers, workstations, enterprise systems, servers, supercomputers, etc.

A digital computer or simply a computer can be defined as a fast electronic machine that accepts input information in the digital form, stores them in memory, processes them according to a set of programming instructions known as a program, and produces the result to an output device. To store data, the computer system uses a memory unit. The most common and popular computer type is the *personal computer* used in school, colleges, offices, home, etc. It is a common form of a *desktop computer* containing a processing unit, storage media, a monitor to display, audio output, keyboard, a mouse, etc. This may be available on a home or

office desktop. Microcomputers are also a type of personal computer. It first appeared in the year 1970s. Today's world of the personal computer is mostly divided between Macintosh by APPLE and PCs. *Mainframe computers* can be operated by many people simultaneously as it supports multiple programs to execute at the same time. It is largely used in big organizations like banks, the telecom sector, etc. and the cost of this computer system is very high. It is successfully used in the fields of healthcare, defence, education, and retail sector. The fastest and biggest computers are known as *supercomputers*. It can compute up to 10 trillion identical calculations per second. In a supercomputer, thousands of individual processors are interconnected. In the year 1976, Roger Cray has developed the first supercomputer. PARAM is a series of supercomputers designed and developed by C-DAC, INDIA. The latest version of PARAM is PARAM Siddhi AI ranked 63 in the world, launched by C-DAC in the year 2020 with an RPEAK of 5.267 petaflops. One of the most expensive and high-end computer types is the *workstation computer*. It was designed and developed to solve complex problems and operations. It is a single-user computer designed for solving or performing scientific calculations. It generally consists of a large amount of RAM, a faster microprocessor, high-speed graphics adapters. It has a more powerful processor in comparison to a personal computer. The smallest and handy computer system is the *notebook* (Laptop) which can be easily carried anywhere. Minicomputer is a medium-size multiprocessing computer system. It can support multiple numbers of the user at a time. It lies between the mainframe and microcomputer. It can be used in process control, data management, communication portal, etc.

1.4 FUNCTIONAL UNITS

A computer system consists of five basic and prime functional units such as input, memory, arithmetic and logic unit (ALU), output, and control unit as shown in figure 1.1.

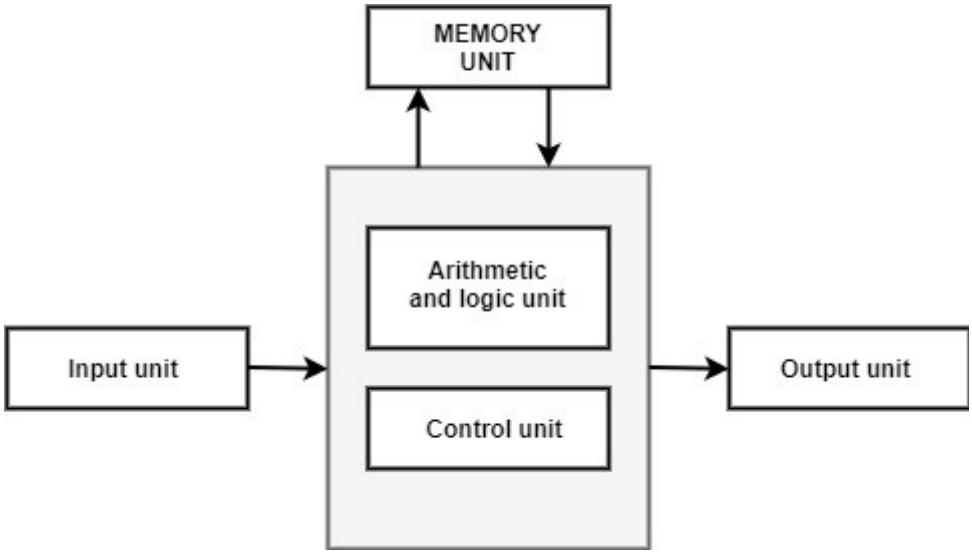


Figure 1.1: Functional units of a computer system

The input unit of a computer system accepts inputs from the outside world. The input devices may be a keyboard, mouse, joystick, sensor devices like camera, heat sensor, the sensor for pressure, etc. the inputs may be of different forms like characters, light, signals, etc. whatever the signals received by the input unit, converted into the digital format known as data or information and stored in the memory unit for future reference. The received data will be processed by the arithmetic and logic unit and produce output.The users or the outside world can access the output data through the output devices such as to monitor, printer, etc. All these functions are coordinated and controlled by the control unit. The functional units can communicate among them through a set of wires known as system buses.

The computer input can be categorized into two types. It may be either data or instruction that instructs the system about the operation to be performed. An instruction can govern the data transmission between the processor and I/O devices or between the processor and memory unit or in between the I/O devices and the memory unit. It is used to specify the arithmetic and logical operation to be performed. A set of instructions that are designed to perform a specific task is known as a program. The program is usually stored in memory and the ALU fetches the required instruction one by one from the memory to perform the desired operation. The entire

computer operations are controlled by the programs excluding the interrupt received from the I/O unit connected to the system. The instructions of a program use characters, symbols, and numerals as an operand, which can be defined as data or often known as digital information.

1.4.1 Input Unit

A computer system receives data from the user or the outside world through the input unit. Input devices are the key elements of this unit. The mouse and keyboards are the common input devices. Whatever the input characters entered through a keyboard, will be received by the input device interface and encoded into binary form, such that the machine can understand it.

1.4.2 Memory Unit

The basic function of the memory unit is either to read data from it or write data onto it. The memory management unit controls the operations related to memory. Memory can be classified into two prime categories such as primary memory or main memory and secondary memory. Main memory is volatile, i.e. the stored data will be lost due to power failure. The computer processor can receive data for execution from the main memory only. Memory consists of semiconductor storage cells that can store 1 bit of information. The processor is used to read or write memory contents in a fixed size block known as a word. The word length may be 1 byte or multiples of it. The memory is organized in such a way that the contents can be retrieved by the processor in a fixed word length containing n-bit to perform a basic operation. The memory locations have addresses for each cell. Memory locations are identified by using addresses. When a processor asks for a particular data, it passes the address of the memory location where the data is available. During a write operation also, the processor locates the memory address for the data to write or store into memory.

The number of bits in a word is fixed for a system known as the word length. Typically the word length lies in between 16 – 64 bits. It determines the size and capacity of the computer. Typically a medium or a large machine can have many tens or hundreds of millions of words. When the memory of a system is accessible, only one word is usually read/write. During the execution of a program, it must be loaded into the main memory. The processor can read/write data words into the memory location as instructed by the instruction which is being executed. The time required to access a word from the memory is known as the memory access time. It is fixed and does not depend on the location where it is stored.

The computer system has different types of memories such as main memory, cache memory, ROM, secondary memory. Depending on the size, cost, and speed of the memories, a memory hierarchy is shown in **figure 1.2**. At the top, it contains the smallest and fastest memory element – the registers. Next smallest, fastest, and costly memory - cache memory, followed by main memory, secondary memories – flash memory, magnetic disc, magnetic tape, optical disk, etc.

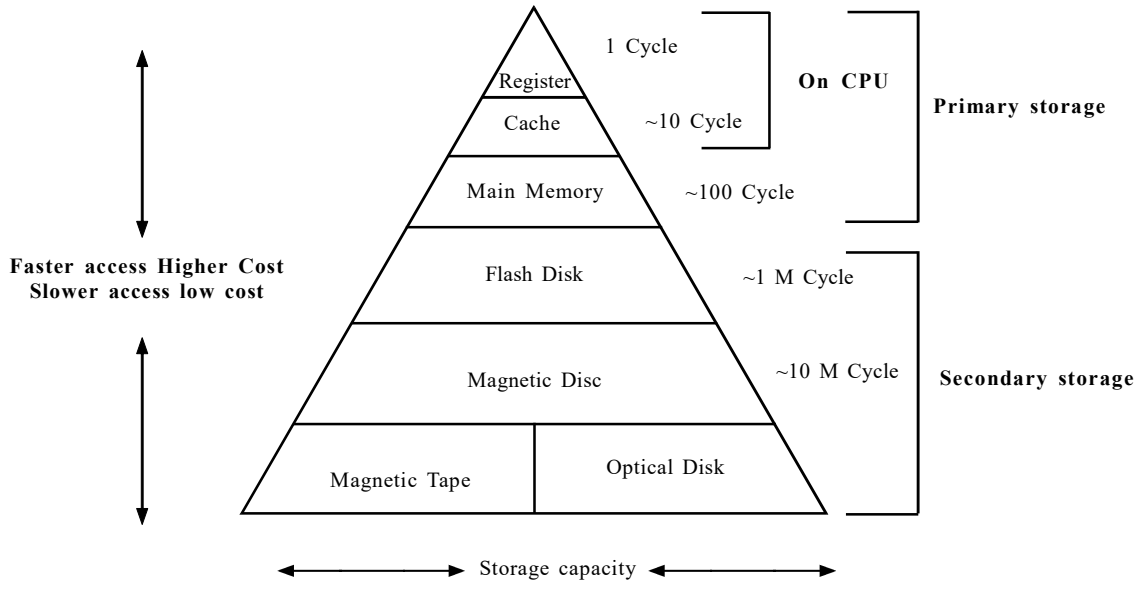


Figure 1.2: Memory hierarchy



CHECK YOUR PROGRESS

Q 1. The computer input can be categorized into two types _____ and _____ .

Q 2. PARAM is a series of supercomputers designed and developed by _____ in INDIA .

Q 3. The number of bits in a word is fixed for a system known as the _____.

1.4.3 Arithmetic and Logic Unit

The computer operations are executed or performed by the arithmetic and logic unit (ALU) of the processor. For example, the operations like addition, subtraction, multiplication, division, etc. among operands are done by the ALU only. It receives the input operands either from the instruction being executed or from the outside world by enabling an interrupt request line. The ALU stores the output in the memory locations as instructed by the instruction. When the operands are brought into the processor for execution, they are stored in the registers. Registers are faster than the cache memory as shown in figure 1.2.

1.4.4 Output Unit

The output unit plays a major role in a computer system. The computer systems work on electrical signals internally. As human beings, we can't understand those signals produced by the computer. The output unit decrypts the computer output into a human-understandable format and sends it to the output device interfaces. The output device may be a monitor, a printer, etc. the output format may be text, image, speech signals or voice, video, etc.

1.4.5 Control Unit

The operations performed by the I/O unit, memory unit and the ALU are coordinated by the control unit. A basic operation in computer systems is the I/O transfer. The instruction that initiates

the I/O transfer is also responsible for its control. But the control unit determines the timing signal and governs the complete transfer operation. The data transfer between the memory and the processor is also governed by the control unit through the time signal.

The basic operations of a computer system can be summarized as follows:

- The computer system accepts inputs in the form of either data or programs through the input unit and stores them in the memory.
- Data or the program instructions stored in the memory unit are fetched by the processor for execution purposes.
- After completion of the arithmetic and logic operation, the output is placed into the output unit to access by the outside world.
- All operations are controlled by the control unit.



EXERCISE

1. What are the different types of computer?
2. What is the difference between workstation and supercomputer?
3. What is memory access time and seek time?
4. Define memory hierarchy. What are the key factors affects the memory hierarchy?

1.5 BASIC OPERATIONAL CONCEPTS

All the ALU activities are controlled by the program instructions. An instruction determines what to do or what operation to be performed by the processor. A program can be defined as a set of instructions that are designed and developed to perform a particular task. During the execution of a program, the processor can accept at most one instruction at a time or one data word at a time from the memory. One instruction fetched from memory can perform only one task. For example, the instruction used in the 8085 microprocessor:

ADD B

STA 2500H

Instruct the computer to add the content of register B with the content of the accumulator and store the result into the accumulator. To execute this instruction at first, the processor will fetch it from the memory. Then it will fetch the contents of the accumulator and then the content of register B. Finally, the processor will perform the addition operation on these two operands. During execution, the STACK POINTER will be loaded with the address of the next instruction to be executed, i.e. STA 2500H in the example. After completion of the first instruction, the processor fetched the instruction from the memory as pointed by the stack pointer. The second instruction is used to store the contents of the accumulator into the memory location 2500H of the system.

Transfer between the processor and the memory is initiated by sending the address of the memory location issued by the programming instruction and appropriate control signals. The data are then can be transferred between the memory and the processor.

The processor used various registers to store different kinds of data during processing. The instruction register (IR) holds the address of the instruction that is currently being executed. The program counter (PC) is a register used to store the address of the next instruction to be executed by the processor. It helps to track the execution of a program. Apart from these two special registers, a set of general-purpose registers is used by the processor to hold data during execution. To assist the communication between the processor and the memory, two registers are used – memory address register (MAR) and memory data register (MDR). MAR stores the address of the memory location to be accessed and the MDR stores the data to be written into the memory or read from the specified memory location.

When the PC is updated with the address of the first instruction of a program, the execution of the program gets started. Then the content of the PC will be transferred to the MAR and a READ control signal is sent to the memory. After getting a time signal from the control unit, the content of

the memory address specified in MAR is loaded into MDR, i.e. the program instruction. Next, the instruction is transferred into the IR. At this point, the instruction is decoded and ready to process by the processor. If the executing instruction needs operands to perform the required operation by the ALU, it has to be fetched by sending the address to the MAR and initiate the READ cycle. The address of the operand is specified by the instruction itself. After initiating the READ cycle, the operand is fetched from the memory and transferred to the MDR. Next, it is transferred from the MDR to the ALU. After fetching all the desired operands, the ALU can do the operation successfully. Once it is completed the result is sent to the MDR and the address of the result is send to the MAR and the write cycle is initiated. During the write operation, the PC will be updated by the address of the next instruction such that the processor can start a new execution.



EXERCISE

5. List the steps required to execute the machine instructions:

   MVI B 05H

   ADD B

   STA 2500H

6. List the steps required to execute the HL programming language instruction:

   B = 5;

   C = 3;

   A = B + C:

It is all about a normal execution. If any device raises an interrupt request, current execution may be pre-empted to serve some urgent services. To deal with the urgently required services, the currently executed process has to be interrupted. An interrupt can be defined as a request signal rise by an I/O device for service by the processor. The processor responses to the interrupt request by executing an appropriate interrupt-service-routine. Before transferring the control to the interrupt-service-

routine, the CPU stores the address of the next instruction located by the PC, onto the stack. After executing the interrupt-service-routine the CPU resumes the previous execution from the address stored in the stack.

1.6 ORGANIZATION

We have already discussed the input and output unit in the functional unit of the computer system section. To perform the input/output operation efficiently, all the I/O peripherals have to be structured in an organized manner. The main elements of the I/O organization are the input and output devices. An input device can be defined as a hardware unit used to provide inputs into a system. The set of 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. 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 are of three different types– to transfer data, data bus is used; 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 depict the states of completeness, availability of a byte of information to be read from the **data-in** register and a device error. To start a control command or to change the mode of a device, the bits in the control register are used. The I/O device port registers are typically 1 – 4 bytes in size.

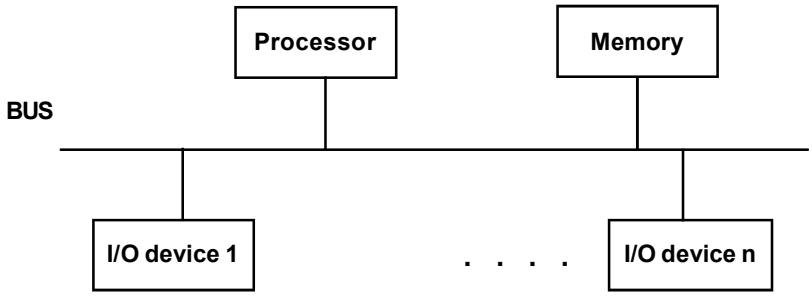


Figure 1.3 Bus structure of a computer system

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 1.3.** The bus system typically consists of three different sets of lines: to transfer addresses – address bus, data – data bus, and control signals – control bus. Each I/O device has a unique set of specified addresses. When the processor 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 transfers 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 varies 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 execute the input received from the keyboard, 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 1.4**. The interfaces consists of four different set of registers – data in, data out, status and control registers.

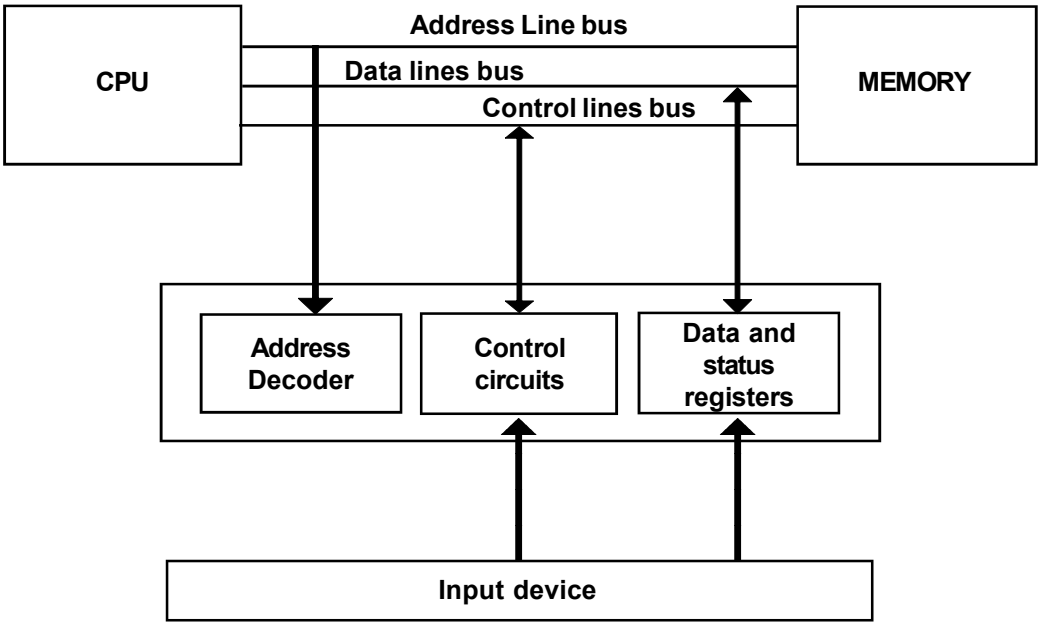


Figure 1.4 Communication of data between CPU and I/O devices

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 used 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.



CHECK YOUR PROGRESS

Q 4. The _____ register holds the address of the instruction that is currently being executed.

Q 5. What is a program?

Q 6. What is the function of an I/O interface?

1.7 BUS ORGANIZATION

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 timing 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 uses timing 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.



EXERCISE

7. What is port number? Write port number for atleast five computer peripherals.
8. What is bus organization?
9. What is memory access time and seek time?
10. Define memory hierarchy. What are the key factors affects the memory hierarchy?

In an asynchronous bus system, the common clock is replaced by two-time control lines such as ***Master–ready*** and ***Slave–ready***. 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. By activating the slave-ready line, the slave line informs the CPU and also performs the required operation. A full handshaking method can provide the highest degree of flexibility and reliability.

1.8 PERFORMANCE

The performance of a computer system can be defined as the throughput of the processor that is number of tasks completed per unit of time. The performance of a computer system depends on lots of factors. The I/O organization plays a vital role in the measurement of the performance of a system. Other factors are the operating system, the compiler used to compile the programming instruction into a machine-understandable format. To get a better performance the I/O unit, the compiler, and the instruction set have to be designed in a coordinated way.

During the processing of an instruction, the operating system algorithms tries to utilize all the available resources. The speed of the CPU, I/O devices, and disk are not the same and it affects the performance of the system. When datais transfered between I/O devices and the memory unit, the CPU may seat idle. During the printing of a large number of data, the involvement of the CPU is almost negligible. So to measure the performance of the processor, only that period will be considered in which the processor remains active. The summation of these times during the execution of a program is known as the processor-time. Apart from the processor time, the time required to complete the execution is known as the elapsed time. The elapsed time determines the performance of the entire system. So it can be said that the elapsed time for the execution of a program depends

on all the functional units except the ALU and the processor time depends on the hardware required to process the program instructions. This hardware consists of the processor and the memory unit which are interconnected through a bus system. When a program has to be executed by the processor, all the required program instructions and the required data are stored in the main memory. As the execution proceeds, the instructions are fetched one by one over the bus and placed into the cache memory. From the cache memory, the CPU can read the data directly. If the CPU requests an instruction that is already available in the cache, no need to copy it from the main memory again. The CPU can read it from the cache directly. The processing speed of the CPU is very high in comparison to the memory access time. To reduce this gap, the cache memory is placed in between the processor and the main memory. The cache is the smallest and the fastest memory used in a computer system. The use of cache memory helps to increase the performance of the processor by decreasing the elapsed time. To get a better performance cache memories are organized in a layered structure. The L1 cache is implemented within the CPU itself. The L2 cache may embed within the CPU or in between the L1 and the L2 cache. The L3 cache lies in between the processor and the main memory. The concepts of L1, L2, and L3 cache decrease a noticeable amount of the elapsed time which may help to get better performance.

1.8.1 Processor Clock

Processor circuits are controlled by a timing signal known as the *clock*. Clock cycles can be defined as the regular time interval. The processor can execute one instruction in a clock cycle. The length of a clock cycle “P” is an important role in the measurement of the performance of the system. The clock rate R can be defined as the number of cycles per second, i.e. ***R = 1 / P***. The clock rate can be measure in Hertz (Hz). The term million cycles per second mean 1 Mega Hz (MHz) and billions of clock cycles per second are denoted by the term Gigahertz (GHz). So, 1250 million cycles per second can be written as 1.25GHz.

1.8.2 Basic Performance Equation

Consider that **T** is the processor time required to execute a program. Let the total number of instructions to be executed by the processor is **N**. Assume that the average number of basic steps required to execute an instruction is **S**, where each instruction can be executed in one clock cycle. If the clock rate is **R** cycles per second, then the program execution time can be calculated as

$$\mathbf{T} = \frac{N * S}{R}$$

It is often known as the basic performance equation. The value of T is more important than the values of N, S, and R. Overall to get better performance; the design of the system organization is to be made in such a way that the value of **T** is reduced. i.e. for some lower values of **N** and **S**, and higher value of R can increase the processor time T or the performance of the system. We can conclude that by using a higher frequency clock the R can be increased. The N, S, and R are not independent parameters. Changing one can affect the other.



EXERCISE

11. A processor advertised as having a 900 MHz clock. Is it providing a better performance than a processor with 700 MHz clock?
12. Consider an implementation of MIPS ISA with 500 MHz clock and
    – each ALU instruction takes 3 clock cycles,
    – each branch/jump instruction takes 2 clock cycles,
    – eachsw(store word) instruction takes 4 clock cycles,
    – eachlw(load word) instruction takes 5 clock cycles.

    Also, consider a program that during its execution executes:
    – x=200 million ALU instructions
    – y=55 million branch/jump instructions
    – z=25 million sw instructions
    – w=20 million lw instructions

    Find CPU time

1.8.3 Instruction Set

A simple instruction requires a small number of basic steps (S) to execute. Whereas, a complex instruction is required more steps. A processor with simple instructions may need a large number of instructions (N) to perform a given task. In this case, **S** has a small value whereas **N** has a very high value. If individual instruction performs a complex operation, then a lesser number of instructions (N) are required but the number of basic steps (S) required performing a complex instruction will be increased. In this case, both the choices does not give a better solution. Based on the value of N and S, the computer architecture can be categorized as – Reduced Instruction Set Computers (RISC) and Complex Instruction Set Computers (CISC). The idea behind the structure of RISC is to make simple instruction set with a fewer number of basic steps for loading, evaluating, storing, etc. The idea behind the CISC is to design a single instruction to perform loading, evaluating, storing, etc. activities with a large number of basic steps. Both are used to increase the system performance.

1.8.4 Compiler

A compiler can be defined as system software that is used to translate source code written in a high-level programming language into some equivalent intermediate code. To get a lesser value of N, we need a compiler and a suitable machine instruction set. The compiler can optimize the source code by reducing some unnecessary code written by a user during compilation. The compiler may rearrange the source code to achieve better performance.



1.9 LET US SUM UP

- A computer system consists of five basic and prime functional units such as input, memory, arithmetic and logic unit (ALU), output, and control unit.
- An instruction can govern the data transmission between the processor and I/O devices or between the processor and memory unit or in between the I.O devices and the memory unit.
- The computer system has different types of memories such as main memory, cache memory, ROM, secondary memory.
- An instruction determines what to do or what operation to be performed by the processor.
- Transfer between the processor and the memory is initiated by sending the address of the memory location issued by the programming instruction and appropriate control signals.
- The program counter (PC) holds the address of the instruction which is to be executed next by the processor.
- To facilitate the communication between the memory and the processor, two registers are used – memory address register (MAR) and memory data register (MDR).
- An interrupt can be defined as a request signal rise by an I/O device for service by the processor.
- 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.
- 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.*
- In an asynchronous bus system, the common clock is replaced by two-time control lines such as ***Master–ready*** and ***Slave–ready***.
- The performance of a computer system can be defined as the number of throughputs of the processor per unit of time.

- The cache memory is placed in between the processor and the main memory. The cache is the smallest and the fastest memory used in a computer system.
- A simple instruction required a small number of basic steps (S) to execute. Whereas complex instruction required more steps.
- The idea behind the structure of RISC is to make simple instruction set with a fewer number of basic steps for loading, evaluating, storing, etc.
- The idea behind the CISC is to design a single instruction to perform loading, evaluating, storing, etc. activities with a large number of basic steps.
- A compiler can be defined as system software that is used to translate source code written in a high-level programming language into some equivalent intermediate code.



1.10 FURTHER READING

1. Silberschatz, A., Galvin, P. B., & Gagne, G. (2006). Operating system concepts. John Wiley & Sons.
2. Hamacher, V. C., Vranesic, Z. G., &Zaky, S. G. (1984). Computer organization. McGraw-Hill, Inc..



1.11 ANSWERS TO THE CHECK YOUR PROGRESS

Ans to Q No 1: Data, Instruction

Ans to Q No 2: C-DAC

Ans to Q No 3: Word-length

Ans to Q No 4: Instruction

Ans to Q.No 5: A program can be defined as a set of instructions that are designed and developed to perform a particular task.

Ans to Q No 6: 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.



1.12 MODEL QUESTIONS

Q1. What is memory hierarchy? Explain with a diagram.

Q2. What are the memory types? Explain different types of ROM.

Q3. What is computer organization? Why it is necessary?

Q4. Explain the functional units of a computer system.

Q5. What are the functions of the control unit in a computer system?

Q6. Explain the bus organization of a computer system.

Q7. What is interrupt? How a system can handle multiple devices explain.

Q8. What do you mean by performance of a computer? How it can be calculate?

Q9. Differentiate between elapsed time and processor time.

Q10. What is RISC and CISC? Explain.

*** ***** ***

UNIT 2 : INTRODUCTION TO OPERATING SYSTEM

UNIT STRUCTURE

2.1 Learning Objectives

2.2 Introduction

2.3 Operating System

2.4 OS as an Interface

2.5 OS as Resource Manager

2.6 Evolution and Types of Operating System

- 2.6.1 Serial Processing
- 2.6.2 Batch Systems
- 2.6.3 Multi-programmed Batch System
- 2.6.4 Time Sharing System
- 2.6.5 Real-Time Operating System
- 2.6.6 Distributed Operating System

2.7 Let Us Sum Up

2.8 Further Reading

2.9 Answers to Check Your Progress

2.10 Model Questions

2.1 LEARNING OBJECTIVES

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

- understand the basic functions of an operating system
- describe about serial processing, batch processing, multiprogramming systems
- explain time sharing system, real-time operating system and distributed operating system

2.2 INTRODUCTION

Operating system acts as an interface between users and computer system. User interacts with the computer through operating system in order to accomplish their task. There are various functions of an operating system

among which the primary two functions are to provide user interface and to act as a resource manager. The concept of operating system and its various functions are discussed in this unit. Operating systems have evolved from slow and expensive system to present day fast and inexpensive system with more features. The evolution of operating system along with various types of operating systems are discussed in this unit. In the next few units we will explore the concepts of process and threads.

2.3 OPERATING SYSTEM

Operating System is a system software. It is a set of programs that interacts and manages the computer hardware. The first software to be installed in a computer is the operating system. After the installation of the operating system, other software's can be installed in a computer. An operating system (OS) is a program that controls the execution of application programs and acts as an interface between application programs and the computer hardware. Any OS has three objectives:

- **Convenience:** An OS makes the computer more convenient to use.
- **Efficiency:** An OS allows the computer system resources to be used in an efficient manner.
- **Ability to evolve:** An OS should be constructed in such away as to permit the effective development, testing, and introduction of new system functions without interfering with service.

2.4 OPERATING SYSTEM AS AN INTERFACE

Operating system typically provides services in the following areas:

- **Program development:** Most of the OS provides a variety of facilities and services, such as editors (for example Notepad and WordPad in Windows OS) and debuggers.It assists the programmer in creating programs. Typically, these services are in the form of utility programs that are not strictly part of the core OS, are supplied with the OS and are referred to as application program development tools.

- **Program execution:** In program execution, a number of steps need to be performed such as instructions and data must be loaded into main memory, I/O devices and files must be initialized, and other resources must be made available. The OS handles these tasks.
- **Access to Input/Output (I/O) devices:** Each I/O device requires its own peculiar set of instructions or control signals for operation. The OS provides a uniform interface that hides these details so that programmers can access such devices using simple reads and writes.
- **Error detection and response:** While running a computer system, variety of errors may occur. These errors may be internal or external such as memory error, or a device failure; and various software errors, such as division by zero, attempt to access forbidden memory location. All these cases are handled by the OS.

2.5 OPERATING SYSTEM AS A RESOURCE MANAGER

Operating system works as a resource manager in a computer system. When a computer (or network) has multiple users, the need for managing and protecting the memory, I/O devices, and other resources is even greater, since the users might otherwise interfere with one another. In addition, users often need to share not only hardware, but information (files, databases, etc.) as well. In short, this view of the operating system's primary task is to keep track of who is using which resource, to grant resource requests, to account for usage, and to mediate conflicting requests from different programs and users.

2.6 THE EVOLUTION AND TYPES OF OPERATING SYSTEM

In attempting to understand the key requirements for an OS and the significance of the major features of a contemporary OS, it is useful to consider how operating systems have evolved over the years. Let us discuss how operating systems were evolved over the years with their pros and cons.

2.6.1 Serial Processing

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

- **Scheduling:** Most installations used a hard copy sign-up sheet to reserve computer time. Typically, a user could sign up for a block of time in multiples of half hour or so. A user might sign up for an hour and finish in 45 minutes; this would result in wasted computer processing time. On the other hand, the user might run into problems, not finish in the allotted time, and be forced to stop before resolving the problem.
- **Setup time:** A single program, called a job, could involve loading the compiler plus the high-level language program (source program) into memory, saving the compiled program (object program) and then loading and linking together the object program and common functions. Each of these steps could involve mounting or dismounting tapes or setting up card decks. If an error occurred, the user typically had to go back to the beginning of the setup sequence. Thus, a considerable amount of time was spent just in setting up the program to run.

  This mode of operation could be termed *serial processing*, reflecting the fact that users have access to the computer in series. Over time, various system software tools were developed in order to make serial processing more efficient. These include libraries of common functions, linkers, loaders, debuggers, and I/O driver routines that were available as common software for all users.

2.6.2 Batch System

To improve utilization, the concept of a batch operating system was developed. It appears that the first batch operating system (and the first OS of any kind) was developed in the mid-1950s by General Motors for use on an IBM 701. The concept was subsequently refined and implemented on the IBM 704 by a number of IBM customers. By the early 1960s, a number of vendors had developed batch operating systems for their computer systems. IBSYS, the IBM operating system for the 7090/7094 computers, is particularly notable because of its widespread influence on other systems.

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

The problems with the batch system are:

- Since the speed of the I/O devices are slower than the CPU, the CPU remains idle.
- Lack of interaction between the user and the job.

2.6.3 Multi-programmed Batch System

Even with the automatic job sequencing provided by a simple batch operating system, the processor is often idle. The problem is that I/O devices are slow when compared to the processor. This inefficiency can be handled.

We know that there must be enough memory to hold the OS (resident monitor) and one user program. Suppose that there is room for the OS and two user programs. When one job needs to wait for I/O, the processor can switch to the other job, which is likely not waiting for I/O. Furthermore, we might expand memory

to hold three, four, or more programs and switch among all of them. This approach is known as multiprogramming, or multitasking. It is the central theme of modern operating systems.

2.6.4 Time Sharing System

Time sharing system refers to multiple users to access a computer system from various terminals simultaneously. Thus it enables to share the CPU time among various users.Just as multiprogramming allows the processor to handle multiple batch jobs at a time, multiprogramming can also be used to handle multiple interactive jobs. In this latter case, the technique is referred to as time sharing, because processor time is shared among multiple users. In a time-sharing system, multiple users simultaneously access the system through terminals, with the OS interleaving the execution of each user program in a short burst or quantum of computation.Thus, if there are n users actively requesting service at one time, each user will only see on the average 1/n of the effective computer capacity, not counting OS overhead. The operating system takes the responsibility of CPU scheduling and multiprogramming.

2.6.5 Real Time Operating System

Another type of operating system is the real-time system. Here the time required to process and respond to user data is very less. The time required by the system to respond to an input and display the required information is termed as response time. Some examples of real time system are: medical imaging systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.

There are two types of real time system:

Hard real time system: It ensures that critical tasks complete on time. Here, secondary storage is limited or missing and the data is stored in ROM.

Soft real time system: Here,missing an occasional deadline is acceptable. Digital audio or multimedia systems fall in this category.

2.6.6 Distributed Operating System

Here, multiple central processors are used to serve multiple users with multiple real time applications. The tasks are handled automatically and efficiently by the operating system and the users are not aware of where their programs and files are being stored and running. Distributed operating system often allow programs to run on several processors at the same time, thus requiring more complex processor scheduling algorithms in order to achieve maximum utilization of CPU's time.

Few advantages of distributed systems are:

- If one computer system fails, the other system continue operating.
- Work load is distributed
- Exchange of data is faster
- With resource sharing facility a user from one end can use the resources available on another computer.



CHECK YOUR PROGRESS

Q 1. Fill in the blanks:

(a) An _____ is a program that controls the execution of application programs and acts as an interface between applications and the computer hardware.

(b) In a batch-processing scheme, the software used is known as the _____.

(c) In a _____ system, multiple users can simultaneously access the system through terminals.

(d) In _____ system, the user submits the job on cards or tapes.

(e) In _____, processor time is shared among multiple users.

(f) The _____ operating system perform even if certain part of the hardware is malfunctioning.



2.7 LET US SUM UP

- An operating system is a system software.
- The first software to be installed in a computer system is an operating system.
- An Operating System is a program that controls the execution of application programs and acts as an interface between applications and the computer hardware.
- A computer is a set of resources for the movement, storage, and processing of data and for the control of these functions the OS is responsible for managing these resources.
- In a Batch-processing scheme, a software used is known as the monitor.
- In Batch Processing system, the user no longer has direct access to the processor. Instead, the user submits the job on cards or tape to a computer operator, who batches the jobs together sequentially and places the entire batch on an input device, for use by the monitor.
- In a time-sharing system, multiple users can simultaneously access the system through terminals.
- In real-time system, the time required to process and respond to user data is very less.
- There are two types of real time system: (a) Hard real time system and (b) Soft real time system
- In Distributed system, multiple central processors are used to serve multiple users with multiple real time applications.



2.8 FURTHER READING

1. Tanenbaum, A. S., & Woodhull, A. S. (1997). *Operating systems: design and implementation* (Vol. 68). Englewood Cliffs: Prentice Hall.
2. Deitel, H. M., Deitel, P. J., &Choffnes, D. R. (2004). Operating systems. Pearson/Prentice Hall.



2.9 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q No 1:

(a) operating system

(b) monitor

(c) time sharing

(d) batch processing

(e) time sharing

(f) distributed



2.10 MODEL QUESTIONS

Q1. What is an operating system?

Q2. Write briefly about the functions of an operating system?

Q3. How does operating system act as an interface?

Q4. How does operating system act as a resource manager?

Q5. What is a batch system?

Q6. What is multiprogramming or multitasking?

Q7. What is time sharing system?

Q8. What is real time operating system?

Q9. What is distributed system?

*** ***** ***

UNIT 3: PROCESSES

UNIT STRUCTURE

3.1 Learning Objectives

3.2 Introduction

3.3 Process

3.3.1 Process Control Block

3.4 Process States

3.4.1 Creation of Processes

3.4.2 Termination of Processes

3.5 Let Us Sum Up

3.6 Further Reading

3.7 Answers to Check Your Progress

3.8 Model Questions

3.1 LEARNING OBJECTIVES

After going through this unit, you will be able to

- define process
- describe process control block
- illustrate the different states of processes
- learn about process model
- explain how processes can be created and how they can be terminated
- learn about the concept of thread and its usage

3.2 INTRODUCTION

In the earlier unit, we have learnt about the concept of operating systems. Different types of operating systems like batch, multiprogramming, time sharing, real time and distributed operating systems were also discussed in the previous unit. Initial computer could run only one program at a time. This program had complete control of the system and had access to all the system's resources. In contrast, the current-day computer systems allow multiple programs to be loaded into memory and executed

concurrently. Processor management is concerned with the management of processors (CPUs) i.e., the allocation of processes to a processor. A process is basically a program while it is being executed i.e., it is a running program with some specific tasks to do.

In this unit, we will discuss the concept of process in details which includes process control blocks, process model, creation and termination of processes, etc. In the next unit, we will explore the concepts of interprocess communications and its related topics.

3.3 PROCESS

A process is basically a program while it is being executed. A process can be defined as:

- a program in execution
- an instance of a program running on a computer
- the entity that can be assigned to and executed on a processor
- a unit of activity characterized by the execution of a sequence of instructions, a current state, and an associated set of system resources

We can also think of a process as an entity that consists of a number of elements. There are two essential elements of a process: (a) program code and (b) a set of data associated with that code. Let us assume that the processor begins to execute a program code, and we refer to this executing entity as a process.

3.3.1 Process Control Block

A **process control block** (PCB) can be defined as a data structure which groups all information that is needed for a particular process. Figure 3.1 depicts a PCB.

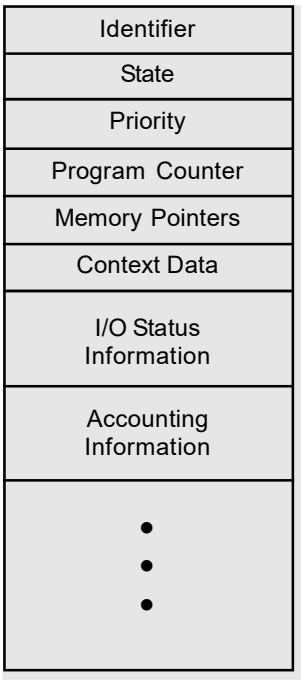


Figure 3.1: Process Control Block

Any process during execution can be uniquely identified by a number of elements such as:

- **Identifier:** A unique identifier associated with the process, to distinguish it from all other processes.
- **State:** A process goes through one of the several states during its course of execution. Some of the states are new, running, ready, waiting and terminated.
- **Priority:** Priority level relative to other processes.
- **Program counter:** The address of the next instruction in the program to be executed.
- **Memory pointers:** Includes pointers to the program code and data associated with this process, plus any memory blocks shared with other processes.
- **Context data:** These are data that are present in registers in the processor while the process is executing.
- **I/O status information:** Includes out standing I/O requests, I/O devices (e.g., tape drives) assigned to this process, a list of files in used by the process, and so on.

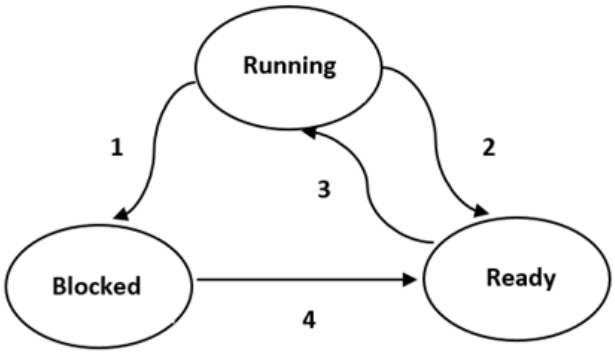
- **Accounting information:** May include the amount of process or time, clock time used, time limits and so on.

The significant point about the Process Control Block (PCB) is that it contains sufficient information so that it is possible to interrupt a running process and later resume its execution as if the interruption had not occurred. The process control block is the key tool that enables the OS to support multiple processes and to provide multiprocessing. When a process is interrupted, the current values of the program counter and the processor registers (context data) are saved in the appropriate fields of the corresponding process control block and the state of the process is changed, such as *blocked* or *ready*. The OS is now free to put some other process in the running state. The program counter and context data for this new process are loaded into the processor registers and this process now begins to execute.

Thus, we can say that a process consists of program code and associated data along with a process control block. At any given time, for a single-processor computer,only one process can execute and that process will be in the *running* state.

3.4 PROCESS STATES

Processes often need to interact with other processes. One process may generate some output that another process uses it as input.



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Figure 3.2: Process transition diagram

In Figure 3.2, we see a state transition diagram showing the three states of aprocess.

- ***Running*** (using the CPU at that instant).
- ***Ready*** (runnable; temporarily stopped to let another process run).
- ***Blocked*** (unable to run until some external event happens)

Logically, the first two states are similar. In both cases, the process is willing to run, but only in the second one, there is temporarily no CPU available for it. The third state is different from the first two in that the process cannot run, even if the CPU is free.

From figure 3.2, we can observe that among the three process states, four transitions are possible: (a) The first transition occurs when a process discovers that it cannot continue. The second and third transitions are caused by the process scheduler which is a part of the operating system. The second transition occurs when the scheduler interrupts the running process, and it is time to let another process to have some CPU time. The third transition occurs when all the other processes have had their fair share and it is time for the first process to get the CPU to run again. The fourth transition occurs when the external event for which a process was waiting (such as the arrival of some input) happens. If no other process is running at that instant, the third transition will be triggered and the process will start running. Otherwise, it may have to wait in ready state for a little while until the CPU is available and its turn.

Using the process model, it becomes much easier to think about what is going on inside the system. Some of the processes run programs that carry out commands typed in by a user. Other processes are part of the system and handle tasks such as carrying out requests for file services or managing the details of running a disk or a tape drive. When a disk interrupt occurs, the system makes a decision to stop running the current process and run the disk process, which was blocked waiting for that interrupt. Thus, instead of thinking about interrupts, we can think about user processes, disk processes, terminal processes, and so on, which block when they are waiting for something to happen. When the disk has been read or the character typed, the process waiting for it is unblocked and is eligible to run again.

3.4.1 Creation of Processes

When a new process is to be added, then the OS builds data structures, that are used to manage the process and allocates address space in main memory to the process. We have already described these data structures while describing PCB.

New batch job	The OS is provided with a batch job control stream, usually on tape or disk. When the OS is prepared to take on new work, it will read the next sequence of job control commands.
Interactive logon	A user at a terminal logs on to the system.
Created by OS to provide a service	The OS can create a process to perform a function on behalf of a user program, without the user having to wait (e.g., a process to control printing).
Spawned by existing process	For purposes of modularity or to exploit parallelism, a user program can dictate the creation of a number of processes.

Figure 3.3: Events for creation of a process

Four common events lead to the creation of a process, as indicated in Figure 3.3. In a batch environment, a process is created in response to the submission of a job. In an interactive environment, a process is created when a new user attempts to log on. In both cases, the OS is responsible for the creation of the new process. An OS may also create a process on behalf of an application. For example, if a user requests that a file to be printed, the OS can create a process that will manage the printing. The requesting process can thus proceed independently of the time required to complete the printing task.

Traditionally, the OS created all processes in a way that was transparent to the user or application program, and this is still commonly found with many contemporary operating systems. However, it can be useful to allow one process to cause the creation of another. For example, an application process may generate another process to receive data that the application is generating

and to organize those data into a form suitable for later analysis. The new process runs in parallel to the original process and is activated from time to time when new data are available. This arrangement can be very useful in structuring the application. As another example, a server process (e.g., print server, file server) may generate a new process for each request that it handles. When the OS creates a process at the explicit request of another process, the action is referred to as **process spawning**.

When one process spawns another, the former is referred to as the **parent process**, and the spawned process is referred to as the **child process**. Typically , the "related" processes need to communicate and cooperate with each other.

In Figure 3.3, we have discussed the events that lead to the creation of a new process. Having discussed the data structures associated with a process, we are now in a position to describe briefly the steps involved in actually creating the process. Once the OS decides to create a new process, it can proceed as follows:

1. **Assign a unique process identifier to the new process:** A new entry is added to the primary process table, which contains one entry per process.
2. **Allocate space for the process:** This includes all elements of the process image. Thus, the OS must know how much space is needed for the private user address space (programs and data) and the user stack. These values can be assigned by default based on the type of process, or they can be set based on user request at job creation time. If a process is spawned by another process, the parent process can pass the needed values to the OS as part of the process-creation request. If any existing address space is to be shared by this new process, the appropriate linkages must be setup. Finally, space for a process control block must be allocated.
3. **Initialize the process control block.** The process identification portion contains the ID of the process plus other appropriate IDs,

such as the ID of the parent process. The processor state information is initialized with most entries as zero, except for the program counter (set to the program entry point) and system stack pointers (set to define the process stack boundaries). The process control information is initialized based on standard default values plus attributes that have been requested for this process. For example, the process state would typically be initialized to Ready or Ready/Suspend. The priority maybe set by default to the lowest priority unless an explicit request is made for a higher priority. Initially, the process may owr more sources (I/O devices, files) unless there is an explicit request for these or unless they are inherited from the parent.

4. **Set the appropriate linkages:** For example, if the OS maintains each scheduling queue as a linked list, then the new process must be put in the Ready or Ready/Suspendlist.
5. **Create or expand other data structures:** For example, the OS may maintain an accounting file on each process to be used subsequently for billing or for performance assessment purposes.

3.4.2 Termination of Process

A running process ends its life when it has completed executing its last statement. The process may also terminate due to one of the following conditions:

1. Normal exit (voluntary).
2. Error exit (voluntary).
3. Fatal error (involuntary).
4. Killed by another process (involuntary).

Most processes terminate because they have finished their work.When a compiler has compiled the program given to it, the compiler executes a system call to tell the operating system that it has finished.This call is exit in UNIX and Exit Process in Windows. Screen-oriented programs also support voluntary termination. Word

processors, internet browsers and similar programs always have an icon or menu item that the user can click to tell the process to remove any temporary files it has opened and then terminated.

The second reason for termination is that the process discovers a fatal error. For example, if a user types the command **cc foo.c** to compile the program *foo.c,* and no such file exists, the compiler simply exits. Screen-oriented inter active processes generally do not exit when given bad parameters. Instead, they pop up a dialog box and ask the user to try again.

Normal completion	The process executes an OS service call to indicate that it has completed running.
Time limit exceeded	The process has run longer than the specified total time limit. There are a number of possibilities for the type of time that is measured. These include total elapsed time ("wall clock time"), amount of time spent executing, and, in the case of an interactive process, the amount of time since the user last provided any input.
Memory unavailable	The process requires more memory than the system can provide.
Bounds violation	The process tries to access a memory location that it is not allowed to access.
Protection error	The process attempts to use a resource such as a file that it is not allowed to use, or it tries to use it in an improper fashion, such as writing to a read-only file.
Arithmetic error	The process tries a prohibited computation, such as division by zero, or tries to store numbers larger than the hardware can accommodate.
Time overrun	The process has waited longer than a specified maximum for a certain event to occur.
I/O failure	An error occurs during input or output, such as inability to find a file, failure to read or write after a specified maximum number of tries (when, for example, a defective area is encountered on a tape), or invalid operation (such as reading from the line printer).
Invalid instruction	The process attempts to execute a nonexistent instruction (often a result of branching into a data area and attempting to execute the data).
Privileged instruction	The process attempts to use an instruction reserved for the operating system.
Data misuse	A piece of data is of the wrong type or is not initialized.
Operator or OS intervention	For some reason, the operator or the operating system has terminated the process (for example, if a deadlock exists).
Parent termination	When a parent terminates, the operating system may automatically terminate all of the offspring of that parent.
Parent request	A parent process typically has the authority to terminate any of its offspring.

Figure 3.4: Some possible reasons for process termination

The third reason for termination is an error caused by the process,often due to a program bug. Examples include executing an illegal instruction, referencing nonexistent memory, or dividing

by zero. In some systems (e.g. UNIX), a process can tell the operating system that it wishes to handle certain errors itself, in which case the process is signaled (interrupted) instead of being terminated when one of the errors occurs.

The fourth reason why a process might terminate is that a process executes a system call telling the operating system to kill some other process. In UNIX this call is called kill. The corresponding Windows function is *Terminate Process*. In both cases, the killer must have the necessary authorization. In some systems, when a process terminates, either voluntarily or otherwise, all processes it created are immediately killed as well. However, neither UNIX nor Windows works this way.



CHECK YOUR PROGRESS

Q 1. Fill in the blanks:

(a) A program in execution can be referred to as a _____.

(b) Two essential elements of a process are _____ and _____

(c) Processes can be distinguished from all other processes by an ___

(d) If the process is currently executing, it is in the _____ state.

(e) A process can have three states named _____, _____ and _____.

(f) When the operating system creates a process at the explicit request of another process, the action is referred to as _____

(g) When one process creates another process, the former is referred to as the _____, and the spawned process is referred to as the _____.



3.5 LET US SUM UP

- The most fundamental concept in a modern OS is the process. The principal function of the OS is to create, manage, and terminate processes.
- To perform process management functions, the OS maintains a description of each process, or process image, which includes the address space within which the process executes, and a process control block.
- During its lifetime, a process moves among a number of states. The most important of these are Ready, Running, and Blocked.
- A ready process is one that is not currently executing but that is ready to execute.
- The running process is that process that is currently being executed by the processor. In a multiple-processor system, more than one process can be in this state.
- A blocked process is waiting for the completion of some event, such as an I/O operation.
- Operating systems must provide a mechanism for parent processes to create new child processes. The parent may wait for its children to terminate before proceeding, or the parent and children may execute concurrently.



3.6 FURTHER READING

1. Tanenbaum, A. S., & Woodhull, A. S. (1997). *Operating systems: design and implementation* (Vol. 68). Englewood Cliffs: Prentice Hall.
2. Deitel, H. M., Deitel, P. J., &Choffnes, D. R. (2004). Operating systems. Pearson/Prentice Hall.



3.7 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q No 1:

(a) process

(b) program code, set of data associated with the code

(c) identifier

(d) running

(e) running, ready, blocked

(f) process spawning

(g) parent, child

(h) lightweight

(i) Multithreading



3.8 MODEL QUESTIONS

Q1. What is a process? How can a process can be uniquely identified?

Q2. What are the different states a process can have?

Q3. Explain briefly the process model with a diagram?

Q4. How can a process be created? What is parent and child process?

Q5. How can a process can be terminated?

*** ***** ***

UNIT 4: INTERPROCESS COMMUNICATION

UNIT STRUCTURE

4.1 Learning Objectives

4.2 Introduction

4.3 Interprocess Communication

4.4 Process Synchronization and Coordination

- 4.4.1 Race Condition
- 4.4.2 Critical Sections

4.5 Mutual Exclusion

- 4.5.1 Disabling Interrupts
- 4.5.2 Peterson's Solutions
- 4.5.3 Sleep and Wake Up
- 4.5.4 Semaphores

4.6 Let Us Sum Up

4.7 Further Reading

4.8 Answers to Check Your Progress

4.9 Model Questions

4.1 LEARNING OBJECTIVES

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

- describe how processes coordinate with each other
- define race condition
- define critical sections
- explain the concept of mutual exclusion
- solve the mutual exclusion problem with various techniques

4.2 INTRODUCTION

In the previous unit, we have learnt about the different types of operating systems. We have also learned about process creation, termination and life cycle of process. In this unit, we will learn about process synchronization and process coordination concepts. Race conditions, critical sections problems and their various solutions are explained in detail

in this unit. In the next unit, the concept of process scheduling is explained in detail.

4.3 INTERPROCESS COMMUNICATION

We have already discussed the concept of process in Unit 3. In this unit, we will discuss how communication takes place between different processes. A process is a program in execution. Processes frequently need to communicate with other processes, preferably in a well-structured way and not using interrupts. This is termed as Interprocess communication. Interprocess communication (IPC) can also be defined as a mechanism which allows processes to communicate with each other and synchronize their actions. The communication between these processes can be seen as a method of co- operation between them.

Now the question will arise: why do we need inter process Communication? Some of the reasons why we may need inter process communication between processes are listed below:

Single user, multiple processes: A single user may request some activities which need the operating system to run different processes. These processes will need a way to communicate so that the user requests are performed smoothly.

Information sharing: It may happen that many processes require the same set of data or information, so it will be easier to share that information instead of storing redundant information.

Modularity: It is easier for the operating system to divide most of the work requested by the user into sub tasks where they can be parallelly computed. The operating system uses multiple processes, threads, functions etc. to perform these sub tasks.

A process can be categorized into two types: independent process and cooperating process. A process is independent if it can not affect or be affected by other processes executing in the system. Any process that does not share data with any other process is independent. A process is cooperating if it can affect or be affected by the other processes executing in the system. Clearly, any process that shares data with other processes

is a cooperating process. All processes executing in multiprogrammed operating system environment compete with other processes for system resources such as memory and CPU. Therefore, they must be synchronized with each other while accessing.

Processes can communicate with each other using two methods:

Shared Memory Model: In this mechanism, one process writes a message in the shared memory location and another process reads the message from that location.

Message Passing Model: In this mechanism, one process sends a message to the other process and expects it to read the message.

Operating systems can choose any one of these two methods for communication between processes. Let us now look in to these methods of interprocess communication in detail.

4.4 PROCESS SYNCHRONISATION AND COORDINATION

There are many processes that operate concurrently in the operating system. Two or more processes may request the same resource or data. Process synchronization means sharing of system resources by processes in such away that, concurrent access to shared data is handled there by minimizing the chance of inconsistent data. To minimize the chance of inconsistent data, certain mechanisms need to be ensured so that data consistency is maintained during execution of processes.

Process coordination is required to maintain synchronization between processes in the operating system. Process synchronization and coordination need to handle problems that arise with multiple process executions. Some of these problems like race condition, critical section and mutual exclusion are discussed in detail in the next few sections.

4.4.1 Race Condition

A situation where several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place, is called a **race condition.**

Examples:

As a first example, suppose that two processes, P1 and P2, share the global variable *a*. At some point in its execution, P1 updates *a* to the value 1, and at some point in its execution, P2 updates a to the value 2. Thus, the two tasks are in a race to write variable *a*. In this example the "loser" of the race (the process that updates last) determines the final value of *a*.

For our second example, consider two process, P3 and P4, that share global variables b and c, with initial values b = 1 and c = 2. At some point in its execution, P3 executes the assignment b = b + c, and at some point in its execution, P4 executes the assignment c = b + c. Note that the two processes update different variables. However, the final values of the two variables depend on the order in which the two processes execute these two assignments. If P3 executes its assignment statement first, then the final values are b = 3 and c = 5. If P4 executes its assignment statement first, then the final values are b = 4 and c = 3.

4.4.2 Critical Sections

Sometimes processes have to access shared memory or files, or doing other critical things that can lead to races. That part of the program where the shared memory is accessed is called the critical region or critical section.

So, how do we avoid race conditions? We need four conditions to hold to have a good solution to the race condition. They are:

(a) No two processes may be simultaneously inside their critical regions.

(b) No assumptions may be made about speeds or the number of CPUs.

(c) No process running outside its critical region may block other processes.

(d) No process should have to wait forever to enter its critical region.

4.5 MUTUAL EXCLUSION

If one process is in the critical region, according to mutual exclusion, no other processes are allowed to enter the critical section at that time. To avoid race condition and to achieve mutual exclusion, let us examine some techniques.

4.5.1 Disabling Interrupts

After entering into the critical region, each process has to disable all interrupts and re-enable them just before leaving it. With interrupts disabled, no clock interrupts can occur. The CPU is only switched from process to process as a result of clock or other interrupts and with interrupts turned off the CPU will not be switched to another process. Thus, once a process has disabled interrupts, it can examine and update the shared memory without fear that any other process will intervene.

A process can then enforce mutual exclusion in the following way:

```
while(true){/*disableinterrupts*/;
/*criticalsection*/;
/*enableinterrupts*/;
/*remainder*/;}
```

4.5.2 Peterson's Solution

In 1981, G.L. Peterson discovered a much simpler way to achieve mutual exclusion. Peterson's algorithm consists of two procedures written in ANSI C, which means that function prototypes should be supplied for all the functions defined and used.

```
#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*/
voidenter_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;/* show that you are interested */
turn = process; /* set flag */
while(turn==process && interested[other]==TRUE);
/* null statement */}
voidleave_region (int process)
/* process, who is leaving */
{
interested[process] = FALSE;
/* indicate departure from critical region */
}
```

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 parameter. This call will cause it to wait, if needed, 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 desires.

Let us see how this solution works. Initially neither process is in its critical region. Now process 0 calls *enter_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*, an event that only happens when process 0 calls *leave_region* to exit the critical region.

Now consider the case that both processes call enter_region almost simultaneously. Both will store their process number in turn. Whichever store is done last is the one that counts; the first one is overwritten and 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 until process 0 exits its critical region.

4.5.3 Sleep and Wake Up

Sleep is a system call that causes the caller to block, that is, be suspended until another process wakes it up. The wakeup call has one parameter, the process to be awakened. Alternatively, both sleep and wakeup each have one parameter, a memory address used to match up sleeps with wakeups.

The Producer-Consumer Problem

Let us consider one example. Let there are two processes, one is producer and the other is consumer. Both the processes, producer and consumer share a common, fixed-size buffer. The producer, puts information into the buffer, and the other one, the consumer, takes it out.

Trouble arises when the producer wants to put a new item in the buffer, but it is already full. The solution is for the producer to go to sleep, to be awakened when the consumer has removed one or more items. Similarly, if the consumer wants to remove an item from the buffer and sees that the buffer is empty, it goes to sleep until the producer puts something in the buffer and wakes it up.

The code for both producer and consumer is shown below.

```
#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();
```

If one process is in the critical region, according to mutual exclusion, no other processes are allowed to enter the critical section at that time. To avoid race condition and to achieve mutual exclusion, let us examine some techniques.

```
/* 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 */
}
```

To keep track of the number of items in the buffer, we will need a variable, count. If the maximum number of items the buffer can hold is N, the producer's code will first test to see if count is N. If it is, the producer will go to sleep; if it is not, the producer will add an item and increment count.

The consumer's code is similar: first test counts to see if it is 0. If it is 0, go to sleep, if it is nonzero, remove an item and decrement the counter. Each of the processes also tests to see if the other should be awakened, and if so, wakes it up.

Now, let us get back to the race condition. It can occur because access to count is unconstrained. The following situation

could possibly occur. 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 inserts 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 *wake up* to wake the consumer up.

Unfortunately, the consumer is not yet logically asleep, so the wakeup signal is 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.

4.5.4 Semaphores

In 1965, E. W. Dijkstra (1965) suggested using an integer variable to count the number of wakeups saved for future use. In his proposal, a new variable type, called a semaphore, was introduced. A semaphore could have the value 0, indicating that no wakeups were saved, or some positive value if one or more wakeups were pending.

Dijkstra proposed having two operations, *down* and *up* (generalizations of sleep and wakeup, respectively). The *down* operation on a semaphore checks to see if the value is greater than 0. If so, it decrements the value (i.e., uses up one stored wakeup) and just continues. If the value is 0, the process is put to sleep without completing the *down* for the moment. Checking the value, changing it and possibly going to sleep, is all done as a single, indivisible atomic action. It is guaranteed that once a semaphore operation has started, no other process can access the semaphore until the operation has completed or blocked. This atomicity is absolutely essential to solving synchronization problems and avoiding race conditions.



CHECK YOUR PROGRESS

Q 1. Fill in the blanks:

(a) _____ refers to the communication among the processes without any interruption.

(b) A situation where several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place, is called _____.

(c) The part of the shared memory where race condition occurs is called _____

(d) If one process is in the critical region, according to _____, no processes are allowed to enter the critical section at that time.

(e) The producer-consumer problem is a technique to achieve _____.



4.6 LET US SUM UP

- Processes frequently need to communicate with other processes, preferably in a well-structured way not using interrupts. This is termed as Interprocess Communication.
- A situation where several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place, is called a race condition.
- Processes have to access shared memory or files, or doing data that can lead to races. The part of the program where the shared memory is accessed is called the critical region or critical section.
- According to the mutual exclusion, no two or more processes are allowed to enter the critical region simultaneously.
- Different types of mutual exclusion techniques are available such as Disabling Interrupt, Sleep and Wake Up, Peterson's solution, Semaphores etc.



4.7 FURTHER READING

1. Tanenbaum, A. S., & Woodhull, A. S. (1997). *Operating systems: design and implementation* (Vol. 68). Englewood Cliffs: Prentice Hall.
2. Galvin, P. B., Gagne, G., &Silberschatz, A. (2003). Operating system concepts (Vol. 10). John Wiley & Sons.



4.8 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q No 1:

(a) Interprocess Communication
(b) race condition
(c) critical region
(d) mutual exclusion
(e) mutual exclusion



4.9 MODEL QUESTIONS

Q1. What is process co-ordination?
Q2. What is race condition and mutual exclusion?
Q3. What are the different types of mutual exclusion techniques?
Q4. When more than one processes share the same memory location at the same time, what problems may arise?
Q5. What are the requirements to achieve mutual exclusion?
Q6. Explain the disabling interrupt method to achieve mutual exclusion?
Q7. Explain the Peterson's solution to achieve mutual exclusion.
Q8. What is sleep and wakeup? What is semaphore?
Q9. Explain the producer-consumer technique.

*** ***** ***

UNIT 5: SCHEDULING

UNIT STRUCTURE

5.1 Learning Objectives

5.2 Introduction

5.3 Process Scheduling

5.4 Preemptive and Non- Preemptive Scheduling

5.5 Types of Scheduling Algorithms

5.6 Scheduling Criteria

5.7 Scheduling Algorithms

- 5.7.1 First Come First Serve (FCFS) Scheduling
- 5.7.2 Shortest Job First (SJF) Scheduling
- 5.7.3 Priority Scheduling
- 5.7.4 Round Robin (RR) Scheduling

5.8 Let Us Sum Up

5.9 Further Reading

5.10 Answers To Check Your Progress

5.11 Model Questions

5.1 LEARNING OBJECTIVES

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

- define process scheduling
- describe preemptive and non-preemptive scheduling
- describe the types of scheduling
- discuss different scheduling criteria for selecting a scheduling algorithm
- describe scheduling algorithms like FCFS, SJF, RR and priority scheduling

5.2 INTRODUCTION

In the previous unit, we have learnt how inter-process communication takes place. We have also learnt about important concepts like critical sections, mutual exclusions, race conditions and semaphores.

In this unit we will learn how process scheduling is done for operating systems with multiprogramming. We will also learn about the different criteria's for selecting a scheduling algorithm. Different scheduling algorithms like first come first serve, shortest job first, priority scheduling and round robin are discussed in detail in this unit. We will explore the concept of deadlock in the next unit.

5.3 PROCESS SCHEDULING

Process scheduling is an important function of the operating system. CPU is an important resource and so the scheduling of CPU between processes is a vital function of the operating system.

All processes have bursts of CPU execution and I/O requests. Generally, a process runs for a while and then a system call is made to read or write from a file. Once the I/O job is done, the CPU runs again till more I/O requests are made and so on. The processes which spend most of their time in computing are called **compute-bound**. Whereas the processes which spend most of the time waiting for I/O are called **I/O bound**. Compute bound processes generally have long CPU bursts and infrequent I/O waits, while I/O bound processes generally have short CPU bursts and frequent I/O waits.

5.4 PREEMPTIVE AND NON- PREEMPTIVE SCHEDULING

The scheduling for the CPU may occur under the following four conditions:

1. When a process terminates.
2. When a process goes from the running state to the waiting state (for example, the process may make a I/O request)
3. When a process switches from running state to the ready state (for example, an interrupt occurs)
4. When a process switches from waiting state to the ready state (for example, the process completes the I/O)

For the situations in 1 and 2, a new process needs to be selected for allocation of the CPU in all cases. However, for situations in 3 and 4,

there is a choice. If the scheduling takes place only in cases of 1 and 2, then we say that the scheduling scheme is non-preemptive. But, if the scheduling happens in cases of 3 and 4 also, then the scheduling scheme is said to be preemptive. In case of non-preemptive scheduling, the CPU cannot be taken away from the process until the process terminates or goes to the waiting state like by making an I/O request. But, in the case of preemptive scheduling, the CPU may be taken away even when a new process arrives or a clock interrupt occurs.

5.5 TYPES OF SCHEDULING ALGORITHMS

Different scheduling algorithms are needed in different environments. There are different goals for the different application areas and also many types of operating systems. Broadly, three environments are identified as follows:

1. Batch
2. Interactive
3. Real time

In batch systems, there are no users waiting for fast responses at the terminals. So, non-preemptive algorithms along with preemptive algorithms with long time periods are accepted. This improves the overall system performance since switching is reduced.

In interactive systems, we have users waiting for quick responses. This requires the systems to have preemptive scheduling so that no process hogs the CPU for a long duration of time.

In real time systems, preemptive scheduling is sometimes not needed because the process do not generally run for long periods of time and do their work quickly. In real time systems, programs are not run unnecessarily but to further the work or application that is required. The difference with interactive systems is that interactive systems are general purpose and may not run application specific programs like in real time systems.

5.6 SCHEDULING CRITERIA

The following are some of the criteria on the basis of which scheduling algorithms may be compared. The criteria's have different characteristics which help in making a decision on which scheduling algorithm is best for a specific case.

CPU Utilization: CPU utilization is about keeping the CPU busy and utilizing it to the fullest. Conceptually, it should range from 0 to 100 percent, but, in the real world system, it ranges from 40 percent to 90 percent.

Throughput: Throughput is the measure of work done per time unit. In this case, throughput can be measured as the number of processes that have been completed per time unit. For long processes it may be one process per hour and for short transactions, it may be ten processes per second.

Turnaround time: Turnaround time is the time taken in total to completely finish executing a process. Turnaround time can be defined as the time interval starting from the time of submission of a process to the time of its completion. So, turnaround time includes the time of the process spent waiting to get memory, waiting in the ready queue, execution time and I/O time.

Waiting time: Waiting time is the sum total of periods of time the process spends waiting in the ready queue.

Response time: For interactive systems, instead of turnaround time,a different criterion called response time can be used for evaluation. Response time is the time from the submission of a request until the first response is given. It is the time taken to get a response and not the time taken to get the output.



CHECK YOUR PROGRESS

Q 1. What are compute bound and I/O bound processes?

Q 2. What are the different criteria's for choosing scheduling algorithms?

5.7 SCHEDULING ALGORITHMS

Scheduling algorithms decides on the process from the ready queue that needs to be allocated the CPU. The CPU needs to be fairly distributed among all the processes. The scheduling algorithm needs to satisfy some basic goals in order to be a good scheduling algorithm.

Goals of Scheduling Algorithms

Some of the desirable goals that a scheduling algorithm must satisfy are as follows:

1. Fairness
2. Balance

Fairness is an important desirable goal for any scheduling algorithm. The scheduling algorithm needs to be fair to all the process while allocating the CPU to them. Allocating more CPU time to one process is not fair. Similarly, allocating CPU to all the rest of the process and starving one process in particular is also not fair. Balance is another important desirable goal for any scheduling algorithm. The scheduling algorithm should maintain a balance the between the CPU bound processes and I/O bound processes. If all the CPU bound processes are chosen in the memory, then some of the processes will be busy fighting for the CPU and the I/O resources will be idle. Whereas, if all I/O bound processes are chosen, then they will fight for I/O resources and the CPU will be idle. There should be a good mixture of CPU bound and I/O bound processes. The system performance will be maximized only when both the CPU and the I/O resources are optimally utilized.

In addition to these, we would want any scheduling algorithm to maximize the CPU utilization and throughput of the system. We would also want the scheduling algorithm to decrease the waiting time, turnaround time and response time for the system. There are many scheduling algorithms. Let us look at some of the different CPU scheduling algorithms in detail in this section.

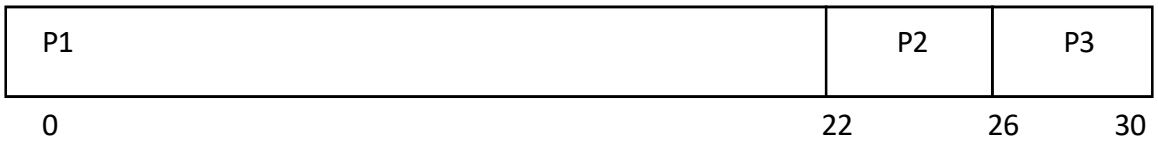
5.7.1 First Come First Serve (FCFS) Scheduling

First Come First Serve (FCFS) scheduling is one of the simplest scheduling algorithms. It is based on the principle that the process that requests the CPU first will be allocated the CPU first. This scheduling algorithm can be easily implemented by using a FIFO queue. When the CPU is free, the process that is at the head of the queue will be allocated the CPU.

Let us look at an example to understand the functioning of FCFS scheduling algorithm. We have the following set of processes that arrive at time 0, along with their CPU burst mentioned in milliseconds:

Process	Burst
P1	22
P2	4
P3	4

Considering that the process arrive in the order P1, P2, P3, then the **Gantt chart** depicting the FCFS order with their starting and ending times will be as follows:

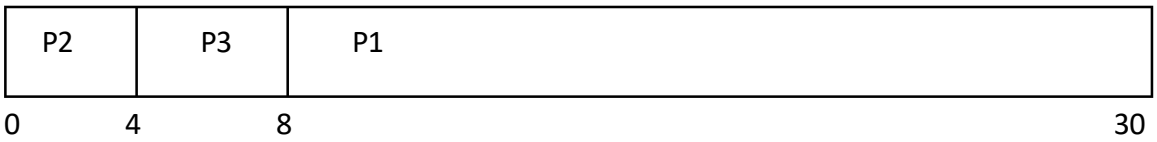


In the above Gantt chart, it can be clearly seen that the process P1 has been allocated the CPU first and run it for its complete CPU burst i.e. from 0 till 22 milliseconds. After process P1, the next process in the queue i.e. P2 is allocated the CPU and it runs its complete CPU burst of 4 milliseconds from 22 to 26 milliseconds. The last process P3 is then allocated the CPU and it runs its complete CPU burst of 4 milliseconds till 30 milliseconds.

The waiting time of the process P1 is 0 milliseconds since it had been allocated the CPU as soon as it arrived at 0 milliseconds. The waiting time for process P2 is 22 milliseconds since it had to wait in the ready queue for 22 milliseconds in order to get access to

the CPU. Similarly, the waiting time for process P3 is 26 milliseconds. So, the average waiting time is (0 + 22 + 26)/3 = 16 milliseconds.

Now, let us consider if the processes arrive in a different order, say P2, P3, P1, and then the process will be allocated the CPU in that order and is shown in the following Gantt chart:



Now, for the waiting time for the process P2 is 0 milliseconds, for process P3 is 4 milliseconds and the process P1 is 8 milliseconds. The average waiting time is (0 + 4 + 8 = 12)/ 3 = 4 milliseconds. This shows that the average waiting time under an FCFS scheduling algorithm varies greatly depending upon the order and the CPU burst times of the processes.

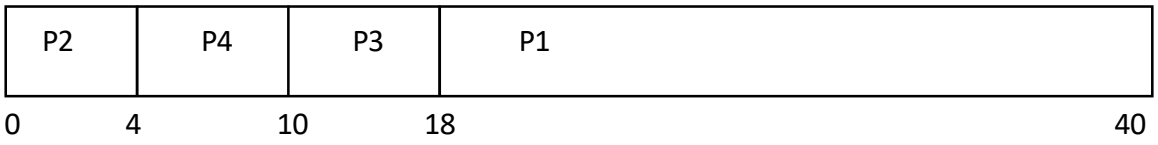
5.7.2 Shortest Job First (SJF) Scheduling

Shortest Job First (SJF) scheduling is another CPU scheduling algorithm. This algorithm requires the process to list the length of the next CPU burst.When the CPU is free, the CPU is allocated to the process with the next shortest CPU burst length. However, if two or more processes have the same next CPU burst then the FCFS scheduling algorithm is used to decide on which process will be allocate the CPU.

Let us look at an example to understand the functioning of SJF scheduling algorithm. We have the following set of processes along with the length of theirnext CPU burst mentioned in milliseconds:

Process	Burst
P1	22
P2	4
P3	8
P4	6

The **Gantt chart** depicting the SJF order with their starting and ending times will be as follows:



In the above Gantt chart, it can be clearly seen that the process P2 has been allocated the CPU first since it has the shortest next CPU burst time of 4 milliseconds. After process P2, the next process to be allocated the CPU is P4 since it has the next shortest length of CPU burst of 6 milliseconds. The next process to be allocated the CPU is process P3 as it has the next shortest length of CPU burst time. The last process P1 is then allocated the CPU.

The waiting time of the process P2 is 0 milliseconds since it had been allocated the CPU as soon as it arrived. The waiting time for process P4 is 4 milliseconds since it had to wait in the ready queue for 4 milliseconds in order to get access to the CPU. Similarly, the waiting time for process P3 is 10 milliseconds and the waiting time for process P1 is 18 milliseconds. So, the average waiting time is (0 + 4 + 10 + 18)/4 = 8 milliseconds.

The SJF scheduling algorithm is a scheduling algorithm that gives provably optimal in the sense that it gives minimum average waiting time for a given set of processes. But the difficulty with this algorithm is in knowing the lengths of the next CPU burst for all the given set of processes since there is no way to know the lengths of the next CPU burst for processes.

5.7.3 Priority Scheduling

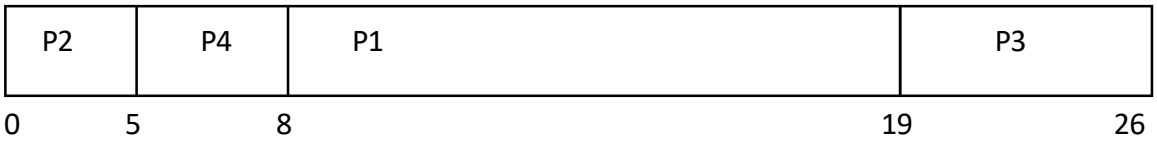
The Shortest Job First (SJF) scheduling algorithm can be considered as a special case of priority scheduling algorithm. In priority scheduling, a priority is associated with each process. The scheduling of the CPU is based on some defined order of priority where higher priority processes are given the CPU before lower

priority processes. Equal priority processes are scheduled in FCFS order. SJF algorithm can be considered as a priority algorithm where the priority is the inverse of the next CPU burst length of the process. Generally, priorities are indicated using some numbers within a fixed range. In some cases, lower numbers are considered as higher priority and in other cases, the higher numbers are considered as higher priority depending on the system.

Let us look at an example to understand the functioning of priority scheduling algorithm. In this example, we will consider lower numbers to be of higher priority. We have the following set of processes along with their associated priorities and length of the CPU burst mentioned in milliseconds:

Process	Burst	Priority
P1	11	3
P2	5	1
P3	7	4
P4	3	2

The **Gantt chart** depicting the order for the processes when using the priority scheduling algorithm along with the processes with their starting and ending times are as follows:



In the above Gantt chart, it can be clearly seen that the process P2 has been allocated the CPU first since it has the highest priority of 1. After process P2 is completed, the next process to be allocated the CPU is P4 since it has the highest priority in the ready queue now. Similarly, after completion of process P4, process P1 is allocated the CPU since it has the highest priority now in the ready queue. The last process P3 with the lowest priority is then allocated the CPU.

The waiting time of the process P2 is 0 milliseconds since it had been allocated the CPU as soon as it arrived. The waiting time for process P4 is 5 milliseconds since it had to wait in the ready queue for 5 milliseconds in order to get access to the CPU. Similarly, the waiting time for process P1 is 8 milliseconds and the waiting time for process P3 is 19 milliseconds. So, the average waiting time is (0 + 5 + 8 + 19)/4 = 8 milliseconds.

Priority scheduling can be either preemptive or non-preemptive. When a process arrives at the ready queue, its priority is compared with the process that is currently using the CPU. In preemptive scheduling, if the new process has higher priority than the currently running process, then pre-emption will occur and the CPU will be allocated to the newly arrived higher priority process at the ready queue. On the other hand, for non-preemptive scheduling, the new higher priority process will simply be placed at the head of the ready queue and CPU will be assigned to this process next.

Starvation is a major problem associated with priority scheduling. A low priority process may never get access to the CPU and will be waiting in the ready queue for a long time if there are a lot of higher order priority processes in the ready queue. The low priority process will be blocked and starve for the CPU for a long time in this scenario. A solution to this problem of blockage of low priority processes is aging. Aging is a technique where there is a gradual increase in the priority of processes that have waited a long time in the queue for the CPU. For example, there could be increase in the priority of a process by 1 for every 10 minutes that it has to wait in the queue. Thus, eventually a low priority process will turn into a high priority process and get its turn for the CPU.

5.7.4 Round-Robin (RR) Scheduling

Round Robin (RR) scheduling algorithm is another scheduling algorithm. Round Robin algorithm is similar to FCFS algorithm but uses preemption in its algorithm. A small unit of time,

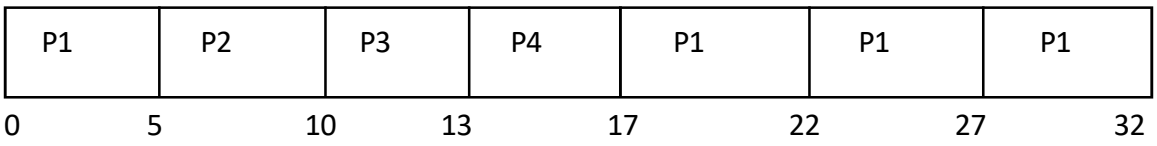
called **time quantum** or **time slice** is defined. This scheduling algorithm can be implemented by using a circular queue, where the CPU will be allocated to the processes one by one for a time quantum. The round robin scheduling algorithm is preemptive and so if a process's CPU burst length exceeds the time quantum then that process is pre-empted and put at the end of the ready queue.

Once the CPU is allocated to a process from the queue, two things may happen. If the CPU burst of the process is less than the time defined in the time quantum, then the process will release the CPU voluntarily and the next process in the queue will get the CPU. Otherwise, if the remaining CPU burst of the process is more than the time quantum defined, then the process will execute its full time quantum and then release the CPU and wait at the end of the queue for its next turn. The free CPU will now be allocated to the process at the top of the queue for the same time quantum.

Let us look at an example to understand the functioning of Round Robin scheduling algorithm where let the time quantum be 5 milliseconds. We have the following set of processes along with the length of their next CPU burst mentioned in milliseconds:

Process	Burst
P1	20
P2	5
P3	3
P4	4

The **Gantt chart** depicting the Round Robin order with their starting and ending times will be as follows:



In the above Gantt chart, it can be clearly seen that the process P1 has been allocated the CPU first since round robin

follows FCFS strategy with pre-emption. P1 gets the CPU for one time slice i.e. 5 milliseconds and then waits at the end of the queue to get its chance for the remaining 15 milliseconds of CPU burst. The CPU is then allocated to the next process in the queue which is P2. Since P2's CPU burst time is of 5 milliseconds same as the time slice, so it does not need any more allocation of the CPU. The CPU is then allocated to P3 which has a CPU burst of 3 milliseconds. Once P3 completes its execution, it releases the CPU voluntarily before the completion of the time quantum. The CPU is now allocated to the next process in the ready queue, i.e. P4, which has a CPU burst length of 4 milliseconds. Once P4 completes its execution, it releases the CPU voluntarily before the completion of the time quantum. The CPU is then allocated to the only process left in the queue i.e. P1 for one time quantum. After the second time quantum for P1 also, since process P1 still has CPU burst of length 10 left and is the only process in the queue, so it will be allocated the CPU again for another time quantum. After the third time quantum for P1, since process P1 still has CPU burst of length 5 left and is the only process in the queue, so it will be allocated the CPU again for another time quantum.

The waiting time of the process P1 is [{0 + (17-5)} = 12] milliseconds for the time P1 was waiting in the ready queue. The waiting time for process P2 is 5 milliseconds since it had to wait in the ready queue for 5 milliseconds in order to get access to the CPU. Similarly, the waiting time for process P3 is 10 milliseconds and the waiting time for process P4 is 13 milliseconds. So, the average waiting time is (12 + 5 + 10 + 13)/4 = 10 milliseconds.

The performance of the round robin scheduling algorithm depends on the length of the time quantum. If the length of the time quantum is too large, then the round robin algorithm acts like the FCFS scheduling algorithm. On the other hand, if the time quantum is too small, then there will be too much time wasted in process switching and this lowers the efficiency of the CPU.



CHECK YOUR PROGRESS

Q 3. What are the goals of scheduling algorithms?

Q 4. What is ageing?



5.8 LET US SUM UP

- All processes have bursts of CPU execution and I/O requests.
- The processes which spend most of their time in computing are called **compute-bound**.
- The processes which spend most of the time waiting for I/O are called **I/O bound**.
- In case of non-preemptive scheduling, the CPU cannot be taken away from the process until the process terminates or goes to the waiting state like by making an I/O request.
- CPU utilization is about keeping the CPU busy and utilizing it to the fullest.
- Throughput is the measure of work done per time unit.
- Turnaround time can be defined as the time interval starting from the time of submission of a process to the time of its completion.
- Waiting time is the sum total of periods of time the process spends waiting in the ready queue.
- Response time is the time from the submission of a request until the first response is given.
- The scheduling algorithm needs to be fair to all the process while allocating the CPU to them.
- The scheduling algorithm should maintain a balance the between the CPU bound processes and I/O bound processes.
- First Come First Serve (FCFS) scheduling is based on the principle that the process that requests the CPU first will be allocated the CPU first.

- Shortest Job First (SJF) scheduling is based on the principle that the CPU is allocated to the process with the next shortest CPU burst length.
- In priority scheduling, the scheduling of the CPU is based on some defined order of priority where higher priority processes are given the CPU before lower priority processes.
- Aging is a technique where there is a gradual increase in the priority of processes that have waited a long time in the queue for the CPU.
- Round Robin algorithm is similar to FCFS algorithm but uses preemption in its algorithm.



5.9 FURTHER READING

1. Silberschatz, A., Galvin, P. B., & Gagne, G. (2006). *Operating system principles*. John Wiley & Sons.
2. Tanenbaum, A. S., & Woodhull, A. S. (1997). *Operating systems: design and implementation* (Vol. 68). Englewood Cliffs: Prentice Hall.



5.10 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q No 1: The processes which spend most of their time in computing are called compute-bound. The processes which spend most of the time waiting for I/O are called I/O bound.

Ans to Q No 2: The different criteria's for choosing scheduling algorithms are: CPU utilization, throughput, turnaround time, waiting time and response time.

Ans to Q No 3: The goals of scheduling algorithms are fairness and balance. The scheduling algorithm needs to be fair to all the process while allocating the CPU to them. The scheduling algorithm should maintain a balance the between the CPU bound processes and I/O bound processes.

Ansto Q No 4: Aging is a technique where there is a gradual increase in the priority of processes that have waited a long time in the queue for the CPU.



5.11 MODEL QUESTIONS

Q1. What is process scheduling?

Q2. What are the differences between preemptive and non-preemptive scheduling?

Q3. Describe in detail the different scheduling criteria.

Q4. What are the different types of scheduling algorithms? What are the goals of scheduling algorithms?

Q5. Describe the working of FCFS scheduling algorithm using an example.

Q6. Describe the working of SJF scheduling algorithm using an example. Explain why it is considered as a special case of priority scheduling algorithm.

Q7. Describe the working of priority scheduling algorithm using an example.

Q8. Describe the working of Round Robin scheduling algorithm using an example.

Q9. Consider the following processes along with their CPU bursts.

Process	CPU Burst
P1	15
P2	6
P3	13
P4	4

Draw the Gantt charts to illustrate the execution of these processes using FCFS, SJF and RR scheduling algorithm. (Consider the time slice to be 4). Also, calculate the waiting time for each process and the overall average waiting time for each scheduling algorithm.

Q10. Consider the following processes along with their CPU bursts.

Process	CPU Burst	Priority
P1	5	4
P2	3	2
P3	8	1
P4	10	3

Draw the Gantt charts to illustrate the execution of these processes using FCFS, SJF and priority scheduling algorithm. Also, calculate the waiting time for each process. Find out which scheduling algorithm has the minimum average waiting time?

*** ***** ***

UNIT 6: DEADLOCKS

UNIT STRUCTURE

6.1 Learning Objectives

6.2 Introduction

6.3 Deadlock

6.4 Principles of Deadlock

6.4.1 Deadlock Conditions

6.4.2 Deadlock Modelling

6.5 Handling of Deadlock

6.6 Deadlock Prevention

6.7 Deadlock Detection and Recovery

6.7.1 Deadlock Detection

6.7.2 Deadlock Recovery

6.8 Deadlock Avoidance

6.8.1 Banker's Algorithm

6.9 Let Us Sum Up

6.10 Further Reading

6.11 Answers to Check Your Progress

6.12 Model Questions

6.1 LEARNING OBJECTIVES

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

- define deadlock
- explain the different mechanisms for deadlock prevention or avoidance
- describe the different deadlock detection and recovery techniques.

6.2 INTRODUCTION

The term deadlock itself is self-explanatory, which means that, in relation to O/S, the processes could not proceed as the required resources are not available to them. Now, let us understand the concept in real terms.

In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources and if the resources are not available at that time, the process enters a waiting state. Sometimes, a waiting process is never able to change the state again, because the resources it has requested are held by other waiting processes. This situation is called a deadlock. This unit describes the methods that an operating system can use to deal with deadlocks: how to prevent, detect and recover. The next unit explores the different memory management techniques.

6.3 DEADLOCK

Deadlock can be defined as the blocking of a set of processes that either compete for system resources or communicate with each other. Suppose two processes are running. Now, the deadlock will occur if one of the processes is entered in the “Waiting” state as the demanded resource is held by the other process and vice versa. Most of the time, deadlock is permanent because none of the processes that are under a deadlock situation will ever get out of it. Thus,in a deadlock situation, the processes could not finish their executions, and system resources are tied up and prevent other jobs from starting.

6.4 PRINCIPLES OF DEADLOCK

In a system, the resources can be categorized as Consumable and Reusable. A consumable resource can be created/produced or destroyed/consumed. In simple terms, the resource which gets produced for a process and then gets destroyed after its use is called a Consumable Resource. Examples of consumable resources are interrupts, signals, messages, data in I/O buffers, etc.

On the other hand, a Reusable Resource is a kind of resource which cannot be destroyed/deleted after its use by a process. And after the resource is released, can be used again by other processes. Processors, I/O channels, Main Memory, Secondary Memory, devices, etc., are examples of Reusable Resources.

6.4.1 Deadlock Conditions

Now, let us now discuss the conditions that must be held simultaneously for a deadlock to occur. The conditions are:

1. **Mutual Exclusion:** At least one of the resources should be held in a non-sharable manner.
2. **Hold and Wait Condition:** A process that is currently holding a resource also makes requests for resources(s) that are held by other processes.
3. **No Preemption Condition:** If a resource is granted to a process, then that resource can only be allocated to another process when that process itself releases the resource.
4. **Circular wait condition:** A circular chain of processes occurs when each process is waiting for the resource which is held by the next process in the chain.

6.4.2 Deadlock Modelling

In 1972, Holt showed how directed graphs can be used to model these four conditions. The graphs have two kinds of node: processes, shown as circles, and resources, shown as squares as in Fig. 6.1. Here it is assumed that there is one resource of each kind.

An arrow from a resource to a process signifies that resource is currently held by that particular process and is depicted in Fig. 6.1(b) where the resource **Ra** is currently held by the process **P1**. An arrow from a process to a resource signifies that the process is currently waiting for that resource. In Fig. 6.1(a), the process **P2** is waiting for the resource **Ra**.

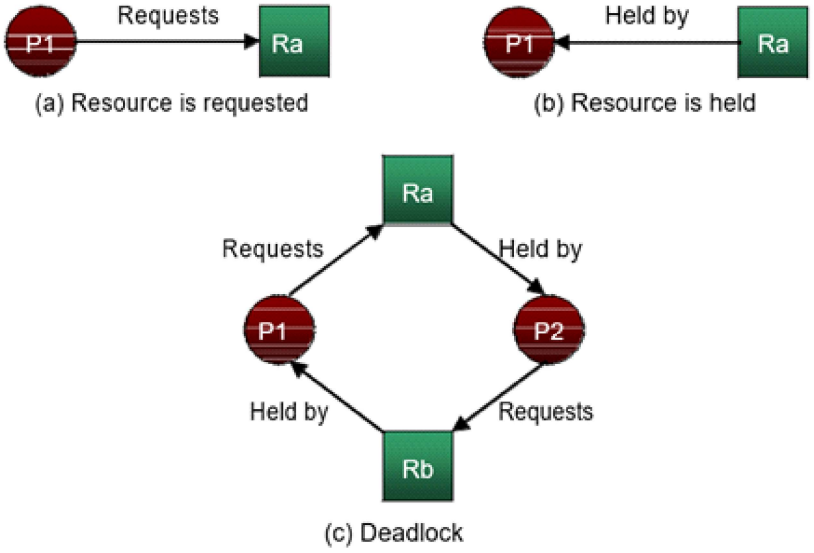


Figure 6.1: Resource allocation

A deadlock is represented in Fig. 6.1(c). Here,

- ✓ Process **P1** is currently holding the resource **Rb**,
- ✓ Process **P2** is currently holding resource **Ra**,
- ✓ Process **P1** has requested for resource **Ra**,
- ✓ Process **P2** has requested for resource **Rb**,

Both processes will wait forever. Thus, the cycle in the graph means that there is a deadlock involving the processes and resources in the cycle. In this example, the cycle is **P1–Ra–P2–Rb–P1**.

6.5 HANDLING OF DEADLOCK

In an O/S, there may be a possibility for a deadlock situation to occur although this is a very rare situation. For handling a deadlock, there are three possible ways and they are:

1) **Deadlock Prevention or Avoidance:**Deadlock prevention means not allowing deadlock to happen. This can be done by not allowing at least one of the necessary conditions for deadlock mentioned above.

   Deadlock avoidance means of making assumptions and based on those assumptions processes are allotted with resources.

2) **Deadlock Detection and Recovery:**Here in this case, without worrying about prevention or avoidance the O/S continues,and upon detection of a deadlock the recovery is done by preemption.
3) **Ignore the Problem Altogether**: If we consider that deadlock is very rare then do not bother about it and if happens reboot the system. This approach is followed by both Windows and UNIX.

6.6 DEADLOCK PREVENTION

In simple terms, the strategy of deadlock prevention is to design a system in such a way that the possibility of deadlock is excluded. The deadlock prevention methods fall into two classes. An indirect method of deadlock prevention is to prevent the occurrence of one of the three necessary conditions listed previously (conditions 1 through 3). A direct method of deadlock prevention is to prevent the occurrence of a circular wait (condition 4). Now let's examine techniques related to each of the four conditions.

Elimination of Mutual Exclusion:

If access to a resource requires mutual exclusion, then mutual exclusion must be supported by the operating system. Some resources, such as files, may allow multiple accesses for reads but only exclusive access for writes. Even in this case, deadlock can occur if more than one process requires writing permission.The resources, like printers and tape drives, need exclusive access by a single process. Thus, it is impossible to eliminate mutual exclusion completely.

Elimination of Hold and Wait:

This condition may be dis-satisfied by allocating all the required resources to a process before the start of the execution. But this approach is very much inefficient for two reasons. Firstly, a process may be held up for a long time since it may be waiting for all of its resource requests to be fulfilled. In fact,the process could have proceeded with a few of the requested resources that has been granted. Secondly,many of the resources allocated to that process may remain unused for a considerable period of time. And thus,it will definitely lead to low resource utilization. Also, in this approach,

every process should be aware of, in advance, all the resources required for its execution which may not possible for each and every process.

Elimination of No Pre-emption:

This could be done in several ways. These are:

- When a process, currently holding certain resources,is denied a resource request then the process should release the resources that are held by it. And after releasing resources, that process may again, if necessary, requests them all along with the resource which was denied earlier.
- When a resource, currently held by a process (e.g., process-1), is requested by another process (e.g., process-2), then O/S should be allowed to preempt the process-1, i.e., forcing the process-1 to release the resources currently held by it.

But, in the case of the 2nd scheme, the two processes must not have the same priority.

Elimination of Circular Wait:

In this approach, each of the resources will be assigned with a numerical value in a linear order. Thus, a process can request the resources in the increasing/decreasing order of numbering as set by the O/S.

To understand this approach, let's consider that each of the resources is numbered as "R" followed by a number starting from 1, e.g., "R1". Now consider that no process can request resources with a lower numerical value than that of the currently held resource. Thus, if a process A, currently holding a resource with value "R3", asks for resources with lower values (i.e. R1, R2) then such requests will never be granted. Process A can only ask for resources with values greater than "R5" only.

As with hold-and-wait prevention, circular-wait prevention may be inefficient, slowing down processes and denying resource access unnecessarily.

6.7 DEADLOCK DETECTION AND RECOVERY

6.7.1 Deadlock Detection

In the earlier section, we discussed the different deadlock prevention approaches which are actually very much conservative. These are implemented by limiting access to the resources, controlling the processes by imposing restrictions.But the deadlock detection approach does not limit the resource access or impose restrictions on process actions.

In case of deadlock detection, whenever possible, there are no restrictions on granting resources to processes. The O/S periodically checks for the circular wait condition, condition 4 of earlier section and illustrated in Fig. 6.1

The frequency of checking for deadlock may be as frequent as each request for resources. This has advantages as well disadvantages. The advantages of this strategy are: (i) detection of deadlock at early stages and (ii) requires a relatively simple algorithm because it is based on incremental changes to the state of the system. But the main disadvantage is the consumption of considerable time of the processor because of the frequent checks.

6.7.2 Deadlock Recovery

After the detection of a deadlock, recovery from it must be done. There are different possible strategies for deadlock recovery and they are:

1. The most common approach is to abort all the processes that are in deadlock.
2. Move back each of the processes, those are under deadlock situation, to a previously stable state with the help of checkpoint and then restart them. But here the risk is that the original deadlock may recur.

3. Successively abort deadlocked processes until deadlock no longer exists and the process selection order for abortion should be on the basis of some criterion of minimum cost. After each abortion, the deadlock detection must be invoked to see whether the deadlock still exists or not.
4. Successively preempt resources until deadlock no longer exists. As in (3), a cost-based selection should be used, and re-invocation of the deadlock detection algorithm is required after each pre-emption. Here, it should be noted that the process, from which a resource has been pre-empted, must be rolled back to a checkpoint prior to the acquisition of that particular resource.

The selection criteria of processes, as stated in 3 and 4, could be one of the following:

- The least amount of processor time consumed so far and least amount of output produced so far
- estimated time remaining (which is a difficult task)
- least total resources allocated so far.

6.8 DEADLOCK AVOIDANCE

As we all know that in case of deadlock prevention constraints are imposed while requesting/granting resources to processes to prevent at least one of the four conditions of deadlock. This leads to inefficient use of resources and inefficient execution of processes. But in Deadlock Avoidance, the three necessary conditions (Mutual Exclusion, Hold and Wait, Preemption) are allowed but with judicious choices so that deadlock situation is never reached. Thus, avoidance will allow more concurrency than prevention. Thus, it requires knowledge of future process' resource requests.Now let's discuss a deadlock avoidance algorithm.

6.8.1 Banker's Algorithm

This algorithm is a scheduling algorithm proposed by Dijkstra in 1965. The basis of this algorithm is how a banker (small-town)

may deal with a group of its customers while granting loans. The approach is basically to check whether the grant of the requested loan amount will lead to a safe or unsafe state. If unsafe, deny the loan request or grant a lesser amount for the time being;if safe grant the requested loan amount.

The system is in a safe state if there exists a sequence <P1, P2, …, Pn> of all the running processes such that for each process Pi, the resources that Pi can still request can be satisfied by currently available resources plus the resources held by all the Pj with j < i. Thus,

- If Pi does not require more resources immediately, then Pi has to wait until Pj has been finished completely.
- Whenever Pj finishes, then Pi can receive required resources, execute and return allocated resources to the resource pool and terminate.
- If Pi terminates, then Pi+1 can get its required resources, and so on.

In simple terms, a state is safe when resources are allocated to each process, maybe up to its maximum need, by avoiding a deadlock situation.But in an unsafe state, the system does not maintain a safe sequence of allocation of resources as discussed, which may lead to a deadlock situation.

Now, let's discuss an example to show how the Banker's Algorithm works. Suppose in a system there are 12 no. of resources of a particular type, suppose R and the no. of processes currently in running state is 3 (say P1, P2, P3). Table-1 describes the maximum requirements and current requirements of the resources of type R by each of the three processes.

Processes	**Maximum Requirements**	**Currently on Hold**
P1	10	5
P2	4	2
P3	9	2

Table 6.1: Process Requirements

As in table 6.1, the maximum no. of requirements by process P1 is 10, P2 is 4 and P3 is 9. Suppose at a particular time T0. processes P1, P2, and P3 are currently holding 5, 2, and 2 no. of resources of type “R”. Thus 3 no. of resources are left free. This state is safe according to the safety condition mentioned above.Now, the remaining 2 required resources may immediately be allocated to process P2 if requested. And if it is so then when P2 frees all the 4 resources then the system will have 5 available resources. After this, process P1 can be allocated with all its remaining required no. of resources (if requested) and when P1 frees all the resources, the no. of available resources becomes 10; and finally, process P3 can get all its remaining requires no. of resources and when P3 frees all the resources, the system will then have 12 no. of available resources.Thus, allocations to all three processes are done so that the safe state at all times was maintained.

The system will go to an unsafe state if, at time T1, P3 request for the remaining resources and the system grants the 3 available resources. This may lead to a deadlock.



CHECK YOUR PROGRESS

Q 1. State whether the following statements are true (T) or false(F):

a) Deadlock can be defined as the temporary blocking of a set of processes that either compete for system resources or communicate with each other Scheduling is done by switching the CPU among processes.

b) Examples of reusable resources include processors, I/O channels, main and secondary memory, devices, and data structures such as files, databases, etc.

c) A consumable resource is one that can be created and destroyed.

d) The circular-wait condition can be prevented by defining a linear ordering of resource types.

e) “Abort all deadlocked processes” is not a way to Recovery from Deadlock.

f) Deadlock may involve reusable resources or consumable resources.

Q 2. Answer the following by selecting the appropriate option:

a) Two general categories of resources
   i. Reusable and Consumable
   ii. Preempted and Nonpreempted.
   iii. Both (i) and (ii).
   iv. None of the above.

b) Examples of consumable resources are
   i. Interrupts
   ii. Signals
   iii. Messages
   iv. All of the above

c) The condition(s) that must be present for a deadlock to occur, are
   i. Mutual Exclusion Condition
   ii. Hold and Wait Condition
   iii. Circular Wait Condition
   iv. All of the above

d) Banker's Algorithm is a
   i. Deadlock Detection Algorithm.
   ii. Deadlock Avoidance Algorithm.
   iii. Both (i) and (ii).
   iv. None of the above.

e) Deadlock prevention guarantees that
   i. deadlock will not occur.
   ii. deadlock will occur.
   iii. Both (i) and (ii).
   iv. None of the above



6.9 LET US SUM UP

- Deadlock can be defined as the permanent blocking of a set of processes that either compete for system resources or communicate with each other. This will be permanent unless the O/S does not take some extraordinary actions to recover from it.
- Deadlock may involve resources of both types: Consumable and Reusable.
- Consumable resources are those that are destroyed/depleted after their acquisition by process(s). For example, messages and information in I/O buffers, etc.
- Reusable resources are those that are notdestroyed/depleted after their acquisition and use by processes. They are fully reusable. Examples of this kind are I/O channels, Main & Secondary Memory. Etc.
- Prevention, Detection & Recovery, Avoidance are the three approaches to deal with deadlock situations.
- Prevention assures that one of the necessary conditions for deadlock does not meet; thus, guarantees possible non-occurrence of deadlock.

- Deadlock detection is needed if the operating system is always willing to grant resource requests; periodically, the OS must check for deadlock and take action to break the deadlock.
- In avoidance, each resource request is analyzed and will be granted if and only if this would not lead to a deadlock situation.



6.10 FURTHER READING

1. Silberschatz, A., Galvin, P. B., Gagne, G., &Silberschatz, A. (1998). Operating system concepts (Vol. 4). Reading: Addison-wesley.
2. Tanenbaum, A. S., & Woodhull, A. S. (1987). Operating systems: design and implementation (Vol. 2). Englewood Cliffs, NJ: Prentice-Hall.



6.11 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q No 1.

(a) False
(b) True
(c) True
(d) True
(e) False
(f) True

Ans to Q No 2.

(a) (i)
(b) (iv)
(c) (iv)
(d) (ii)
(e) (i)



6.12 MODEL QUESTIONS

Q1. Define deadlock.

Q2. What are reusable resources and consumable resources? How are they related to deadlock?

Q3. What are the conditions that must be present for a deadlock to occur?Explain in brief.

Q4. Explain deadlock modelling.

Q5. What is the basic difference between deadlock detection and deadlock avoidance?

Q6. Explain the banker's algorithm for deadlock avoidance.

Q7. Write short notes on:

(i) Deadlock Modeling.

(ii) Deadlock Recovery.

(iii) Deadlock Prevention.

(iv) Deadlock Avoidance.

*** ***** ***

UNIT 7: MEMORY MANAGEMENT

UNIT STRUCTURE

7.1 Learning Objectives

7.2 Introduction

7.3 Memory Management

7.4 Memory Allocation

7.4.1 Fixed Size Partitioning

7.5 Relocation and Protection

7.6 Swapping

7.7 Virtual Memory

7.8 Paging

7.8.1 Hardware Support

7.9 Page Replacement Algorithms

7.9.1 Optimal Page Replacement Algorithm

7.9.2 First In First Out (FIFO) Page Replacement Algorithm

7.9.3 Least Recently Used (LRU) Page Replacement Algorithm

7.10 Let Us Sum Up

7.11 Further Reading

7.12 Answers to Check Your Progress

7.13 Model Questions

7.1 LEARNING OBJECTIVES

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

- define memory management
- describe memory allocation using fixed size partitioning
- discuss protection and relocation
- discuss swapping and virtual memory
- describe how paging memory management scheme works
- describe the page replacement algorithms like FIFO, Optimal and LRU

7.2 INTRODUCTION

In the previous unit, we have learnt the concept of deadlocks. We have learnt about the principles of deadlock and how deadlock can be detected. We have also learnt about how deadlock can be avoided and prevented, also how we can recover from a deadlock.

In this unit we will learn how about the different memory management techniques. We will also learn about fixed size partitioning and virtual memory. The concept of paging is also explained in detail in this unit. Different page replacement algorithms like FIFO, LRU and optimal page replacement are discussed in this unit. We will learn about the memory mapping in the next unit.

7.3 MEMORY MANAGEMENT

One of the most important resources for the computer system is the computer memory. Every user would like to have an indefinite amount of memory space if possible. But since it is not possible, most computers have memory hierarchy, ranging from very fast and expensive cache memory to large, slow and inexpensive disk storage. An important function of the operating system is to manage these different memory components efficiently.

Memory management deals with the management of the computer memory. It is an important function of the operating system. In memory management, the operating system needs to keep track of all parts of the memory that are in use and also the parts of the memory that are not in use. It needs to allocate memory to the processes when they need it and de-allocate the memory when the process completes its execution. It also needs to manage swapping of processes between main memory and disk and vice versa.

Memory management systems can have only one process running or multiple process running. One of the simplest memory management scheme is mono-programming where there is only one process running. In this scheme, the memory would be divided into two parts: one for the

operating system and the other part for the currently running process. But, most modern computer systems allow multiprogramming where multiple processes run at the same time. In multiprogramming, the computer memory has to be divided between the operating system and multiple other processes.

7.4 MEMORY ALLOCATION

The computer memory needs to be allocated to multiple processes. One of the simplest methods for allocating memory is to divide the memory into partitions. Each partition of the memory will hold one process. When a partition is free, then a process is selected and loaded into the partition. And once the process completes execution and terminates then the partition can be allocated to some other process in the queue.

7.4.1 Fixed Size Partitioning

In fixed size memory partitioning, the partitions of the memory are of fixed size. A process can be selected from the queue and allocated one of the free partitions.

Partition Sizes:

Fixed size partitioning can be of equal size or un-equal size. In equal size fixed partitioning, all the partitions would be of the same size. Whereas in case of unequal size fixed partitioning, the partitions are of different sizes. An example of both equal size fixed partitioning and unequal size fixed partitioning can be seen in figure 7.1.

As can be seen in figure 7.1 (a), in equal size partitioning a process can be placed in any one of the partitions and it would not make any difference provided the process size is less than or equal to the partition size. This is because all of the partitions in this scheme are of same sizes. But there are certain disadvantages in this memory management scheme; for example, if we have a process which requires a large space of memory (like 15MB) then it would not be possible to allocate this process in any of the partitions as all the partitions are of equal sizes (8MB). Another disadvantage

would be that if a process requires very less space (say 1 MB), then the remaining 7 MB space of the partition will be not used as all the partitions are of size 8MB. This problem where the internal space in a partition is wasted because of the process needing less space is known as internal fragmentation. These two problems makes the equal size fixed partitioning scheme very inefficient.

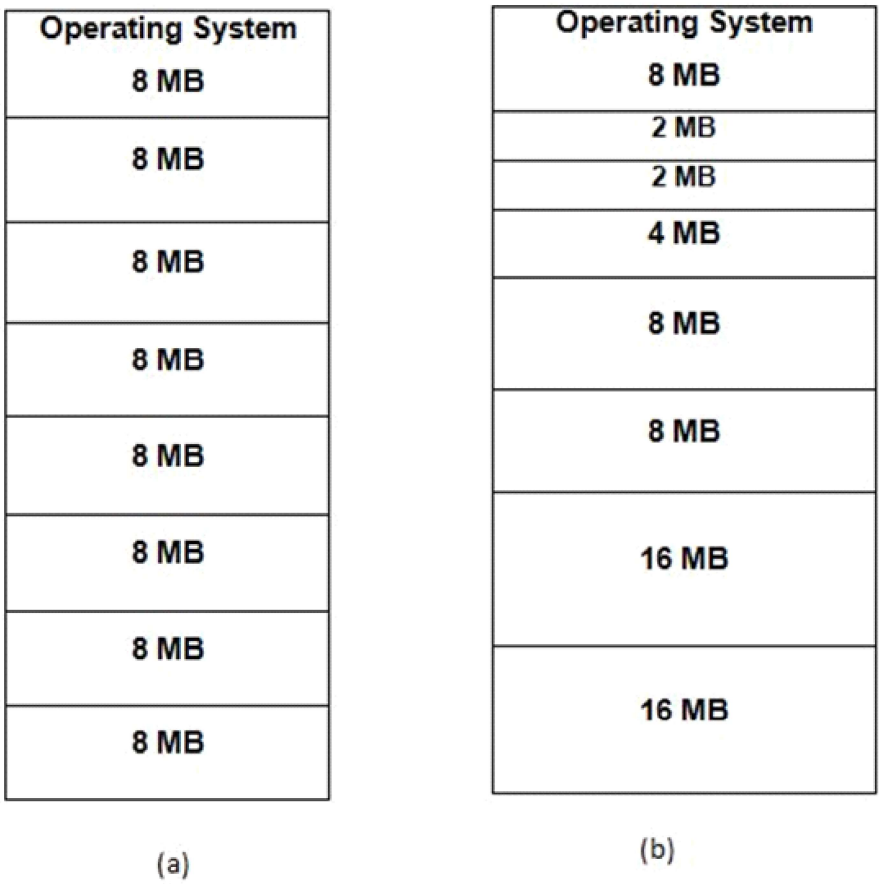


Figure 7.1: (a) Equal size fixed partitioning,(b) Unequal size fixed partitioning

Both of these problems can be lessened but not completely solved by using unequal size fixed partitioning. A process with large size (say 15MB) can be allocated memory space in the unequal size partitioning scheme as shown in figure 7.1 (b). Also, a process with small size (say 1 MB) can be allocated a portioning of size 2 MB, which would only leave 1 MB unused memory space. This will lessen the problem of internal fragmentation.

Placement Algorithm:

In the fixed size partitioning, the operating system maintains a table of memory partitions that are allocated and partitions that are free. The processes are kept in an input queue when they enter the system. The operating system keeps into account the memory requirements of all the processes and then allocates the process to an available partition that is large enough to hold the process.

In equal size fixed partitioning, the placements of the processes into these partitions do not make a difference since all the partitions are of equal sizes. In the case of unequal size fixed partitioning, there are two ways in which the processes can be placed into these partitions.

The simplest way would be to place each process in the smallest partition in which it will fit. For this, a scheduling queue is required for each partition which will hold the queue of processes that are intended for that partition as shown in figure 7.2 (a). Even though this technique looks optimum there are certain issues with it. For example, let us consider a case where there are no processes of higher size (say 32 MB), then the input queue for these large size partitions will be empty. Whereas, let us say we have a huge number of smaller size processes (say 8 MB) lined up for smaller size partitions. To have better efficiency and good utilization of the resources, it should have been possible to allocate some of the smaller size processes to the large size partitions since they are free and there is a queue of processes waiting to get memory space. Thus, another approach to solve this problem is to have one single input queue for all the processes as shown in figure 7.2 (b). In this approach, only one input queue is maintained and each process is then placed into the smallest available partition.

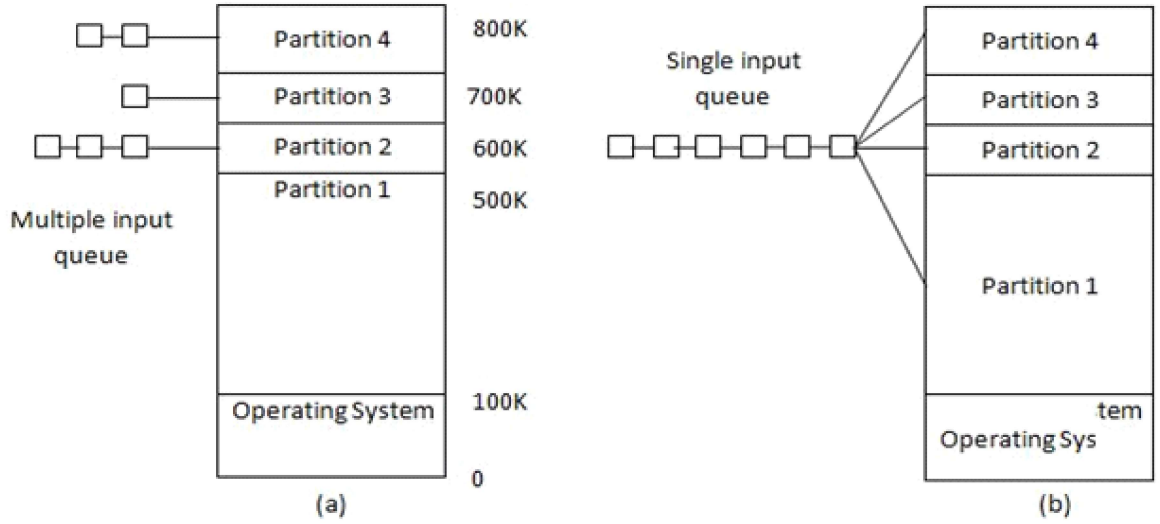


Figure 7.2 (a) Fixed memory partitions with separate input queues for each partition. (b) Fixed memory partitions with a single input queue

7.5 RELOCATION AND PROTECTION

Relocation and protection are two important issues that must be solved in multiprogramming memory management. In multiprogramming, we have different processes running at different addresses in the memory. A problem will have different parts (main program, user defined functions, inbuilt functions etc) that need to be linked and for that the linker will have to know the address of these various components of the program.

Let us try to understand the problem of relocation using an example. Suppose, the first instruction of a program is a call to a function at absolute address 200 within the binary file produced by the linker. If this program is loaded in a partition that starts at 200K, then that instruction will jump to absolute address 200. But, it should have jumped to (200 K + 200) position in the memory. If the program is loaded into a different location that starts at 400K, then the call must jump to (400K+200) location. This problem is called the relocation problem.

One way to solve this problem is to modify the instructions as the program is loaded into the memory. So, programs loaded at 200K, will have a 200K added to each address and a program loaded at 400K, will have a 400K added to each address. For this solution to work the linker must include

a list that will specify which program words are addresses to be relocated and which program words are other items that do not need to be relocated. But this technique does not solve the protection problem. A malicious program can always create a new instruction and jump to unauthorized locations. Such malicious programs can jump into another user's program and get access to other's data.

A solution to both the problems of relocation and protection would be to use two special registers called the base register and the limit register. For any process, the base register will keep the starting address and the limit register will keep the length of the partitions in which the components of the program are loaded in memory. Every memory address will have the address of the base register added to it before being sent to memory. So, if the base register holds the value 200K, then a call to a location at 200 will be finally a call to (200K + 200) location without making any changes to the instruction. The addresses are also checked with the value of the limit register so that any unauthorized location cannot be accessed outside the partition.

7.6 SWAPPING

Swapping is one of the simplest memory management schemes. The basic idea behind swapping is to bring an entire process into memory, run it for a while and then to put it back on the disk. Let us understand the concept of swapping using an example.

The swapping operation is shown in figure 7.3. In figure 7.3 (a), initially only process 1 was in the memory. Then process 2 and process 3 are created and swapped in from disk to the memory as shown in figure 7.3 (b, c). Then, process 2 is swapped out from the memory to the disk in figure 7.3 (d). Then process 4 is created and swapped in from the disc to the memory in figure 7.3 (e). Finally, process 2 is again swapped back in to the memory in figure 7.3 (f).

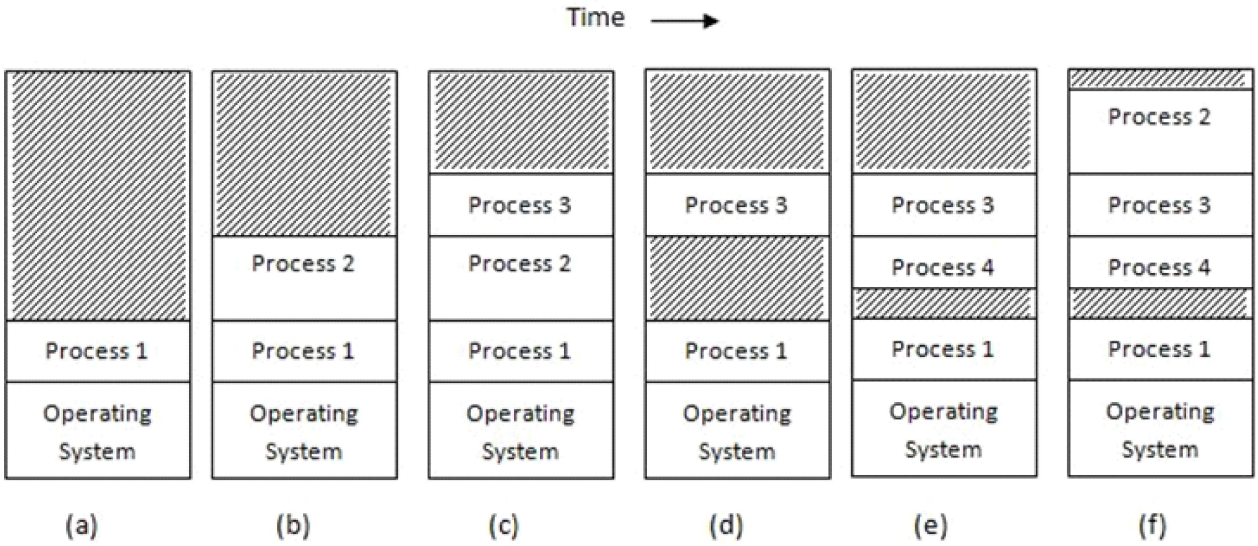


Figure 7.3: Memory allocation changes while process come and leave the memory. Shaded regions are available memory locations



CHECK YOUR PROGRESS

Q 1. Differentiate between equal size and unequal size fixed partitioning.

Q 2. What is swapping?

7.7 VIRTUAL MEMORY

Processes of large sizes sometimes cannot be loaded into the available memory space. In such cases, the concept of virtual memory can be used. The idea behind virtual memory is that the combined size of the program, data and stack may exceed the available physical memory space. So, the operating system keeps only the parts of the program, data and stack that are needed at that particular instant for execution in the memory. The rest of the program can be loaded from the disk into the memory as and when needed.

7.8 PAGING

Paging is a memory management scheme that does not need the physical address space in a contiguous memory location while allocating

space to a process. Paging involves dividing the physical memory into small fixed sized blocks called frames and dividing the logical memory into blocks of same size called pages. The basic idea behind this is that since all the pages are of same size, so any page can be allocated to any frame in the memory. The physical memory frames are numbered and the number of the frame that is allocated to each page of a process is stored in the page table.

In paging, every logical address generated by the CPU has two parts: a page number (p) and a page offset (d). The page number is used as an index to the page table. Each process is associated with a page table that stores the frame number at which each page of the process is stored. The page table contains the starting address of the frames of the process in the physical memory. By combining the page offset (d) the starting address of the frame (f) we get the address in the physical memory. This is demonstrated in the figure 7.4 below.

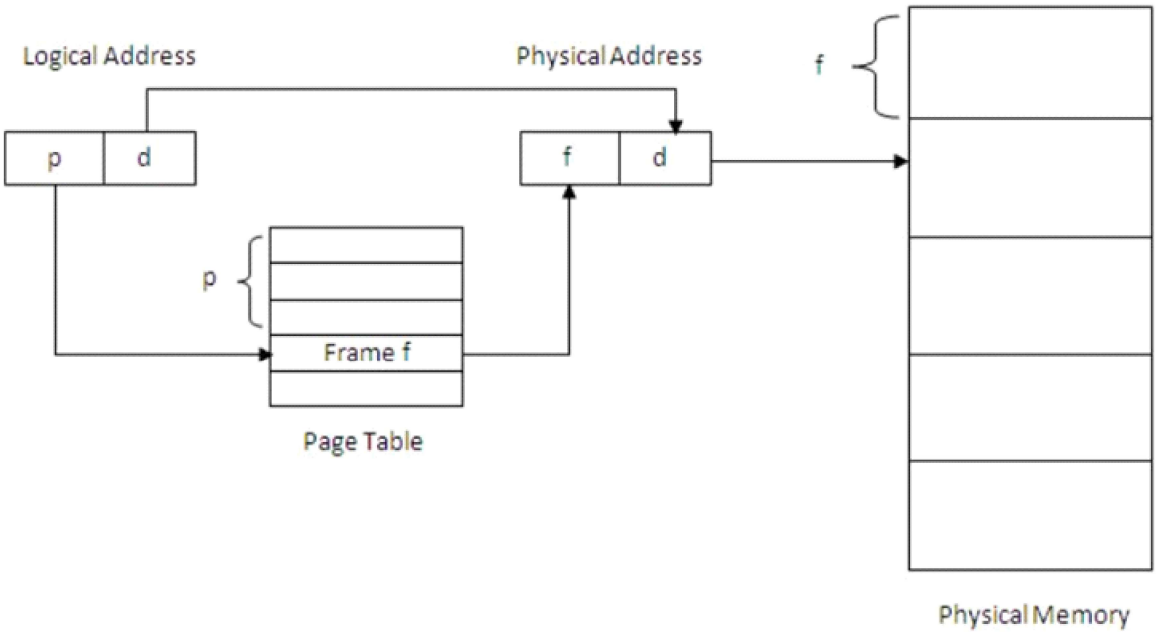


Figure 7.4: Paging architecture

The frame size and the page size in paging is generally a power of 2, like 512 bytes, 1 MB, 4 MB etc. The frame size is selected as power of 2 because it makes translation of logical to physical address easy. If the size of the logical address space is 2^m, and the page size is 2^n, then the high

order (m-n) bits of a logical address will represent the page number and the n lower order bits represent the page offset. So, the logical address can be represented as:

page number	page offset
P	d
m - n	n

where p is an index into the page table and d is the location within the page.

The size of a process that has arrived for execution is expressed in pages. So, if the size of the process is 'n' pages, then we will need 'n' frames to fit the whole process in memory. If 'n' frames are available, then these frames are allocated to the newly arrived process. Each of the pages of the process is loaded into the available frames one by one and the corresponding frame number is stored in the page table of the process. Let us look an example to understand how paging works. The logical address space of a process P is divided into four pages. All the four pages of the process have been allocated frames in the physical memory. The frames are not allocated in a sequential order and any free frame can be allocated to any page of the process. In this example we can see in figure 7.5 that, page 0 is allocated to frame 5, page 1 is allocated frame 2, page 2 is allocated frame 4 and page 3 is allocated frame 1. It can be seen that the physical memory holds the allocated pages of the process.

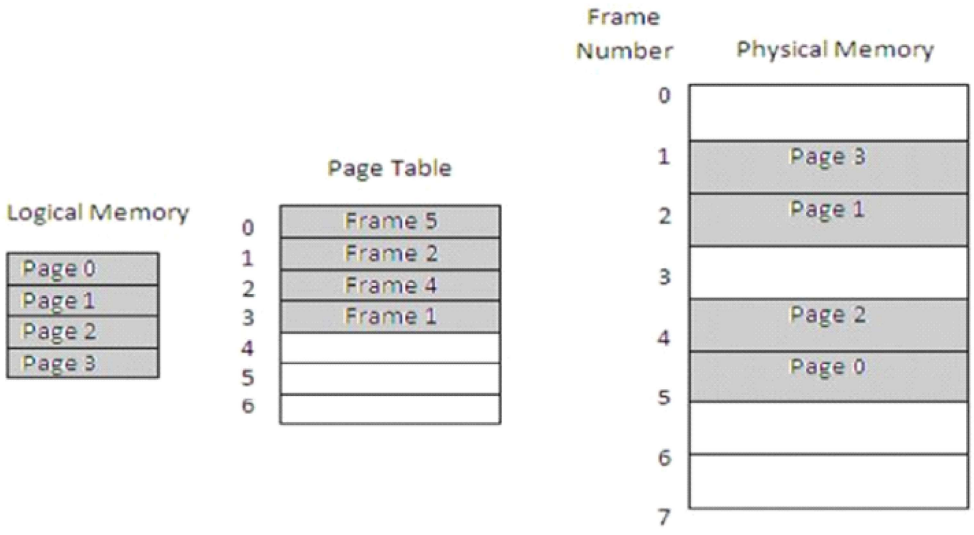


Figure 7.5: Paging example

External and Internal Fragmentation

External fragmentation is not present in paging memory management scheme since any frame from the physical memory can be allocated to any page. However, internal fragmentation can be present in some cases. For any process, the length of the last page of that process may be less than the frame size that is assigned to it. In this case, internal fragmentation may arise. For example, suppose a frame size and page size of 4 MB. Let us consider a process P3 of size 11 MB. The process will be divided into three pages and will need three frames in physical memory for storage. The first two pages cover the entire length of the frames, but the last page will only cover 3 MB size of the frame. So, in the frame allocated to the last page of process P3, a size of 1 MB will be left unused. This space cannot be allocated to any other process so internal fragmentation will occur.

7.8.1 Hardware Support

Every operating system has its own technique for storing page tables. In most systems, a page table is allocated for each process.The simplest solution is to implement the page table as a set of registers. But, the use of registers is only satisfactory if the page table is very small. But this is hardly the case as page tables are often very large. So, the page tables are kept in main memory and a page table base register (PTBR) is used which points to the page table. Now, for changing page tables only the page register needs to be changed which reduces the context-switch time. But the problem with this solution is that it requires two memory accesses. The first access is for the page table by using the value in PTBR register and the second access is for the exact location in memory by combining the frame number and the page offset. This causes a delay problem which needs to be solved.

The solution to this problem is to use a special, small and fast lookup hardware cache called a transition look-aside buffer (TLB). The TLB has two entries: a key (or tag) and a value. When the memory is presented with an item, it is compared with all the

keys in the TLB simultaneously. And if a match is found then the corresponding value for that key is returned. The search is very fast but the hardware is expensive.

Now, let us look at how the TLB scheme can be implemented with the page tables. The TLB contains a few of the page table entries. When a logical address is generated by the CPU, the page number is presented to the TLB. The paging hardware with TLB is shown in the figure 7.6. If the page number is found, its frame number is also found and it can be used to access the memory location. But, if the page number is not found in the TLB which is known as a TLB miss, then a memory reference to the page table must be made. When we get the frame number we can access the memory location. The page number and frame number is then added to the TLB register so that this page can be found in the next reference to this page. If the TLB is full, then the operating system must choose one of the entries for replacement. There are several replacement algorithms that can be used to select the page to be replaced.

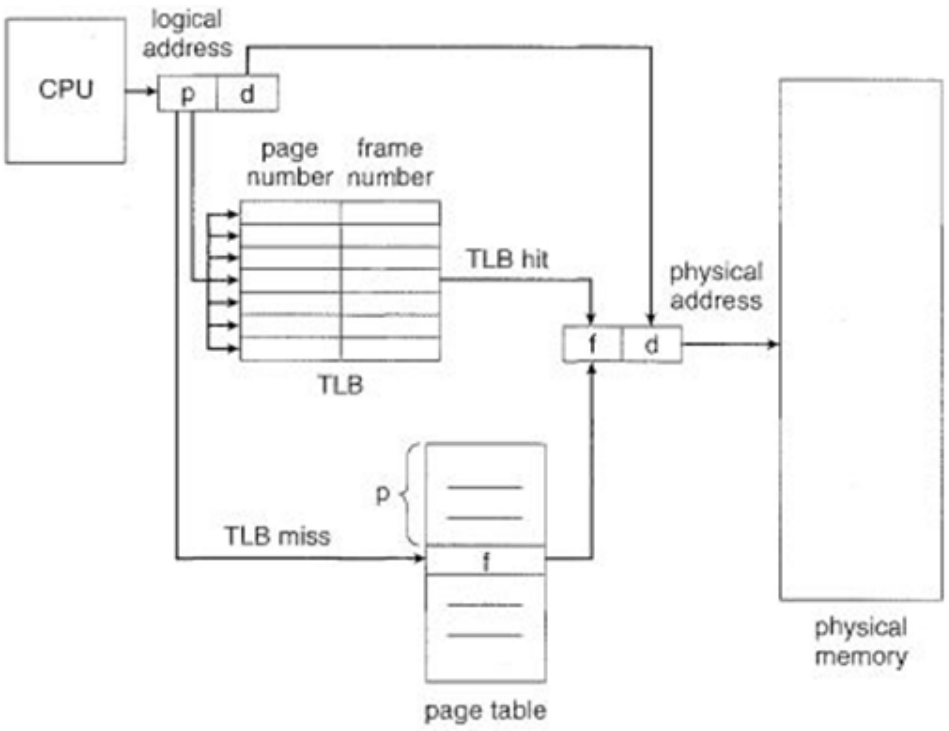


Figure 7.6: Paging hardware with TLB (Taken from Ref [1])

The percentage of times that a desired page number is found in the TLB is called the hit ratio. A 90 percent hit ratio means that the desired page can be found in the TLB 90 percent of the time. If it takes 25 nanoseconds to search the TLB and 150 nanosecondsto access the memory, then a mapped memory access takes 175 nanoseconds when the page number is in the TLB. In case the page number is not found in the TLB (25 nanoseconds), then the first access will be for the page number and frame number (150 nanoseconds) and the second access will be for the actual location in the memory (150 nanoseconds), so a total of 325 (25+150+150) nanoseconds will be needed. To get the effective access time, we weight each case with its probability:

effective access time = 0.90 * 175 + 0.10 * 325 = 190 nanoseconds

Protection

Memory protection in paging is achieved by keeping bits in the page table. One bit is used to define whether a page is read only or in read-write mode. When the page table is accessed to get the frame number, then at the same time the protection bits are checked to verify that nothing has been written in a read only page. Another additional bit 'valid-invalid' bit is generally attached to each entry in the page table. When this bit is set to valid, then the page is in the process's logical address space and is a legal page. When this bit is set to invalid, then the page is not in process's logical space. This memory access will be illegal and thus trapped by the operating system and not allowed to go through. The operating system sets this bit to allow or disallow access to a page.

7.9 PAGE REPLCAEMENT

In paging, a process is divided into fixed size pages and these pages are loaded into frames of same sizes in the physical memory. When a process arrives for execution, the process is allocated available free frames in the memory. But, sometimes there are not enough free frames available

to load all of the pages of a process. When such a page is requested to be executed but the page is not allocated any frame in the physical memory then a **page fault** occurs. In order for the page to be executed, it needs to be allocated a frame. To do this, an existing page needs to be replaced by the page that is desired to be executed. This is known as page replacement. There are many strategies available for choosing the page that is to be replaced. Some of the popular page replacement algorithms are:

- Optimal Page Replacement Algorithm
- First In First Out (FIFO) Page Replacement Algorithm
- Least Recently Used (LRU) Page Replacement Algorithm
- Not Recently Used (NRU) Page Replacement Algorithm
- Second Chance Page Replacement Algorithm
- Clock Page Replacement Algorithm

Let us look at how some of these page replacement algorithms in this section.

7.9.1 First-In First-Out (FIFO) Page Replacement Algorithm

First-In First-Out (FIFO) page replacement algorithm is the simplest page replacement algorithm. The FIFO algorithm uses the principle that the page that was first brought into the memory is the one that should be replaced first. This means that the oldest page is chosen to be replaced first. Let us look at an example to understand the working of the FIFO algorithm in detail.

Pages->	A	B	C	D	A	B	X	A	B	C	D	X
Frame 1	***A***	A	A	A			***X***	X	X	X	***D***	D
Frame 2		***B***	B	B			B	***A***	A	A	A	***X***
Frame 3			***C***	C			C	C	***B***	B	B	B
Frame 4				***D***			D	D	D	***C***	C	C

Figure 7.7: FIFO page replacement algorithm

In the example shown in figure 7.7, it can be seen that we have considered four frames and these frames are initially empty. The first four page reference's (A,B,C, D) all cause page faults and are brought into the empty frames. The next page reference (A) is

already loaded into a frame so there is no page fault. The next page reference (B) is also loaded into a frame so there is again no page fault. The next page reference (X) causes a page fault since it is not loaded into a frame, so the page that was brought first and is the oldest that is page A and is replaced by X. Similarly, the next page reference (A) again causes a page fault and this time, the oldest page (B) is replaced by A. Similarly, page C is replaced by B when a page fault occurs when trying to access the next page B. The next page reference (C) again causes a page fault and this time page D is replaced by C. The next page reference (D) again causes a page fault and this time page X is replaced by D. The last page reference (X) again causes a page fault and this time page A is replaced by X. In this figure, every time there is a page fault the pages in the four frames are shown and the newly referenced page is marked.

FIFO page replacement algorithm is easy to understand. But the performance of FIFO page replacement algorithm is not always satisfactory. In this example, a total of ten page faults occurred for the given reference pages. In some cases for page replacement algorithms, the number of page faults increases when there is an increase in the number of frames. We would expect that increasing the memory would increase the performance of the system, but this doesn't happen. This is known as Belady's anomaly. FIFO page replacement algorithm suffers from Belady's anomaly.

7.9.2 Optimal Page Replacement Algorithm

The optimal page replacement algorithm is the best algorithm for page replacement; hence the name is given as optimal. The optimal page replacement algorithm has lowest page faults and does not suffer from Belady's anomaly. In optimal page replacement algorithm, the page that will not be used for the longest period of time is the page to be replaced. Let us look at an example to understand the working of the optimal page replacement algorithm in detail.

Pages->	A	B	C	D	A	B	X	A	B	C	D	X
Frame 1	***A***	A	A	A			A				***D***	
Frame 2		***B***	B	B			B				B	
Frame 3			***C***	C			C				C	
Frame 4				***D***			***X***				X	

Figure 7.8: Optimal page replacement algorithm

In the example shown in figure 7.8, it can be seen that we have considered four frames and these frames are initially empty. In this figure, every time there is a page fault the pages in the four frames are shown and the newly referenced page is marked. The first four page reference's (A, B, C, D) all cause page faults and are brought into the empty frames. The next page reference (A) is already loaded into a frame so there is no page fault. The next page reference (B) is also loaded into a frame so there is again no page fault. The next page reference (X) causes a page fault since it is not already loaded into a frame, so the page that is going to be referenced last (D) is replaced by X. The page D is referenced last at 11, whereas page A is referenced at 8, page B at 9 and page C at 10. The next page reference (A) is already loaded into a frame so there is no page fault. The next page reference (B) is also already loaded into a frame so there is again no page fault. The next page reference (C) is also already loaded into a frame so there is again no page fault. The next page reference (D) causes a page fault. This time if we look into the pages to be referenced in the future then only page X is to be referenced. So, any page of the other three pages (A, B, C) can be replaced by page D. The last page reference is to page X and since this page is already loaded into a frame so there is no page fault. There were only six page faults for optimal page replacement algorithm in this example. So, optimal algorithm gives a much better performance than FIFO page replacement algorithm. But there is a disadvantage of optimal page replacement algorithm.The optimal page replacement algorithm requires

knowledge of the future reference string, which is unrealistic and not possible to know. So, the optimal page replacement algorithm is generally used only for comparison with other page replacement algorithms to judge the performance of these page replacement algorithms.

7.9.3 Least Recently Used (LRU) Page Replacement Algorithm

Least recently used (LRU) page replacement algorithm is another popularly used page replacement algorithm. In optimal page replacement algorithm we look at the future to decide which page should be replaced. Whereas in LRU page replacement, algorithm we look at the past references to decide the page which has to be replaced. In LRU page replacement algorithm, the page which has not been used for the longest period of time is the page that is chosen to be replaced. In this strategy, it is considered that the pages that have not been recently used may be replaced as they may not be referenced again in the future. Let us look at an example to understand the working of the LRU page replacement algorithm in detail.

Pages->	A	B	C	D	A	B	X	A	B	C	D	X
Frame 1	***A***	A	A	A			A			A	A	***X***
Frame 2		***B***	B	B			B			B	B	B
Frame 3			***C***	C			***X***			X	***D***	D
Frame 4				***D***			D			***C***	C	C

Figure 7.9: LRU page replacement algorithm

In the example shown in figure 7.9, it can be seen that we have considered four frames and these frames are initially empty. In this figure, every time there is a page fault the pages in the four frames are shown and the newly referenced page is marked. The first four page reference's (A, B, C, D) all cause page faults and are brought into the empty frames. The next page reference (A) is

already loaded into a frame so there is no page fault. The next page reference (B) is also loaded into a frame so there is again no page fault. The next page reference (X) causes a page fault since it is not already loaded into a frame, so the page that has not been used for the longest time (C) is replaced by X. The next page reference (A) is already loaded into a frame so there is no page fault. The next page reference (B) is also already loaded into a frame so there is again no page fault. The next page reference (C) causes a page fault so the page (D) is replaced by (C) since it is the page that has not been used for the longest time. The next page reference (D) again causes a page fault so the page (X) is replaced by (D) since (X) is the page that has not been used for the longest time. The last page reference (X) again causes a page fault so the page (A) is replaced by (X) as it is the page that has not been used for the longest time. A total of eight page faults occurred for LRU page replacement algorithm using the same set of reference page examples. LRU gives a good performance. LRU page replacement algorithm also does not suffer from Belady's anomaly.



CHECK YOUR PROGRESS

Q 3. What is Belady's anomaly?

Q 4. What is the principle behind Optimal page replacement algorithm?

Q 5. What is page fault?



7.10 LET US SUM UP

- Memory management deals with the management of the computer memory.
- In fixed size memory partitioning, the partitions of the memory are of fixed size.
- Fixed size partitioning can be of equal size or un-equal size.

- In equal size fixed partitioning, all the partitions would be of the same size. Whereas in case of unequal size fixed partitioning, the partitions are of different sizes.
- Relocation and protection are two important issues that must be solved in multiprogramming memory management.
- The basic idea behind swapping is to bring an entire process into memory, run it for a while and then to put it back on the disk.
- The idea behind virtual memory is that the combined size of the program, data and stack may exceed the available physical memory space.
- Paging is a memory management scheme that does not need the physical address space in a contiguous memory location while allocating space to a process.
- Paging involves dividing the physical memory into small fixed sized blocks called frames and dividing the logical memory into blocks of same size called pages.
- The basic idea behind this is that since all the pages are of same size, so any page can be allocated to any frame location in the memory.
- The frame size and the page size in paging is generally a power of 2, like 512 bytes, 1 MB, 4 MB etc.
- The TLB has two entries: a key (or tag) and a value. When the memory is presented with an item, it is compared with all the keys in the TLB simultaneously. And if a match is found then the corresponding value for that key is returned.
- The percentage of times that a desired page number is found in the TLB is called the hit ratio.
- When a page is requested to be executed but the page is not allocated any frame in the physical memory then a page fault occurs.
- The FIFO algorithm uses the principle that the page that was first brought into the memory is the one that should be replaced first.
- In optimal page replacement algorithm, the page that will not be used for the longest period of time is the page to be replaced.

- In LRU page replacement algorithm, the page which has not been used for the longest period of time is the page that is chosen to be replaced.



7.11 FURTHER READING

1. Silberschatz, A., Galvin, P. B., & Gagne, G. (2006). *Operating system principles*. John Wiley & Sons.
2. Tanenbaum, A. S., & Woodhull, A. S. (1997). *Operating systems: design and implementation* (Vol. 68). Englewood Cliffs: Prentice Hall.



7.12 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q No 1: In equal size fixed partitioning, all the partitions would be of the same size. Whereas in case of unequal size fixed partitioning, the partitions are of different sizes.

Ans to Q No 2: Swapping is one of the simplest memory management schemes. The basic idea behind swapping is to bring an entire process into memory, run it for a while and then to put it back on the disc.

Ans to Q No 3: In some cases for page replacement algorithms, the number of page faults increases when there is an increase in the number of frames. This anomaly is known as Belady's anomaly.

Ans to Q No 4: The principle behind the optimal page replacement algorithm is that the page that will not be used for the longest period of time is the page to be replaced.

Ans to Q No 5: When a page is requested to be executed but the page is not allocated any frame in the physical memory then a page fault occurs.



7.13 MODEL QUESTIONS

Q1. What is memory partitioning? Describe a memory partitioning technique.

Q2. Describe the process of swapping.

Q3. What is virtual memory?

Q4. Describe the process of paging.

Q5. Why do we need page replacement? Which page replacement algorithm is best among FIFO, Optimal and LRU?

Q6. Describe the working of FIFO page replacement algorithm using an example.

Q7. Describe the working of optimal page replacement algorithm using an example.

Q8. Describe the working of LRU page replacement algorithm using an example.

Q9. Consider there to be three frames in the physical memory. Find the number of page faults for FIFO, LRU and Optimal page replacement algorithms for the following page references: [1,2,3,4,2,3,5,1,2,4]

Q10. Consider there to be four frames in the physical memory. Find the number of page faults for FIFO, LRU and Optimal page replacement algorithms for the following page references: [X,Y,Z,P, R, X, Y, R, Z, T]

*** ***** ***

UNIT 8: MEMORY MAPPING

UNIT STRUCTURE

8.1 Learning Objectives

8.2 Introduction

8.3 Memory Mapping

8.3.1 Logical Versus Physical Address Space

8.4 Dynamic Loading

8.5 Basic Memory Operations

8.6 Memory Locations and Addresses

8.7 Byte Addressability

8.8 Fragmentation

8.9 Paging

8.10 Segmentation

8.11 Cache Memory and Memory Mapping

8.11.1 Basic Operations of Cache Memory

8.11.2 Cache Mapping Process

8.11.3 Cache Replacement Policies

8.12 Virtual Memory

8.12.1 Demand Paging

8.12.2 Performance of Demand Paging

8.12.3 Page Replacement

8.13 Let Us Sum Up

8.14 Further Reading

8.15 Answer To Check Your Progress

8.16 Model Questions

8.1 LEARNING OBJECTIVES

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

- describe the memory mapping operations
- list basic memory operations
- define internal and external fragmentation, segmentation
- define cache memory and its use

- describe themapping process of cache memory
- describe cache replacement policies
- explain the concept of virtual memory
- describe cache replacement techniques

8.2 INTRODUCTION

In the previous unit, we have learnt about fixed size partitioning and virtual memory. The concept of paging along with different page replacement algorithms like FIFO, LRU and optimal page replacement were also discussed in the previous unit. In this unit, we will learn about memory mapping, segmentation and virtual memory. Memory mapping techniques like direct mapping, associative mapping and set-associative mapping are explained in detail in this unit. In the next unit, we will discuss about file systems.

8.3 MEMORY MAPPING

Memory mapping is a set of techniques that are used to translate logical address to physical address. The memories used in a computer system are –secondary memory, main memory or primary memory, and cache memory. To compensate the speed mismatch between main memory and CPU, a faster memory called cache memory is used. Cache memory decreases the access time of the CPU by storing the required data in it. Thus the cache memory has a great role in increasing the throughput of a system. It is placed in between the processor and main memory. The memory access time of cache memory is very low in comparison to main memory and compatible with the speed of the processor. The cache is used to store the program segment currently executed by the CPU and the data used by the CPU frequently. Since the memory space of the cache is much smaller than the main memory, mapping is required to identify the location in the main memory as specified by the CPU. Using cache replacements algorithms, the contents of the cache can be changed by new program segments as required by the processor. During

programming, the user can use a huge amount of memory addresses that do not exist physically. In reality, the virtual memory concept is used to make availability of addresses beyond the existing one. Memory mapping is required to map between the logical addresses used by the user with the physical memory addresses. The memory mapping is depicted in figure 8.1.

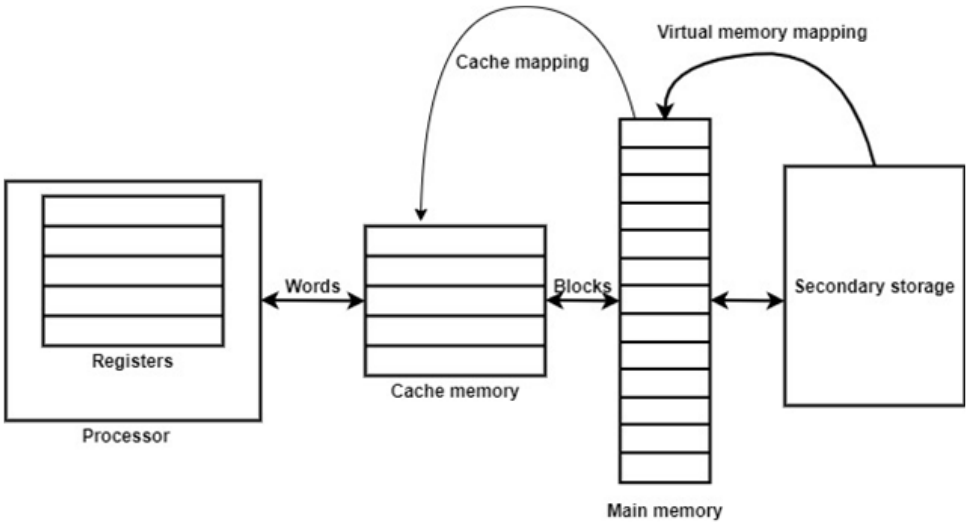


Figure 8.1: Memory mapping

8.3.1 Logical versus Physical Address Space

The address generated by the processing unit is known as the logical address. The address read by the memory unit or the address stored in the memory-address-register (MDR) is called the physical address. The logical address is also referred to as the virtual address. The set of addresses generated by a program is known as the *logical-address-space*, and the corresponding physical address set is defined as the *physical-address-space*. During the execution of a program or in runtime the memory mapping from logical-address-space to physical-address-space is done by a hardware unit called the *memory-management* unit (MMU). The user programs always generate logical addresses. Users never concern about the physical addresses. However, before accessing the memory contents, the logical address has to be mapped with the corresponding physical address.

8.4 DYNAMIC LOADING

An entire program has to be loaded in the main memory before its execution. Thus the size of a program will be restricted to the size of the physical memory. To obtain a better utilization of computer memory and the power of the CPU, a dynamic loading scheme can be followed. In a dynamic loading scheme, the main program and the required routine of a program have to be loaded into the main memory. The physical memory will be loaded with the subroutine referred by the main program. In dynamic loading, no sub-routine will be loaded without calling it by the main program. When a call is raised for a sub-routine, at first, the main memory is checked by the MMU. If the called subroutine is available in the main memory, no need to load it again. If it is not, then a loader is used to load the desired program segment from the secondary memory to the main memory. An unused subroutine is never being loaded into the main memory in a dynamic loading scheme.

8.5 BASIC MEMORY OPERATIONS

Computers memory is used to store both program instructions and data. To execute an instruction the processor should load or transfer the instruction into the processor from the primary memory. During the processing of instructions, the operands or the results are also transferred between the memory and the CPU. Thus the basic operations involving in the memory can be classified into two categories such as Load or Read / Fetch and store or Write.

The READ operation transfers a copy of the contents from the memory location specified by the CPU. During the transfer, the memory contents remain unchanged. The READ operation is initiated by the CPU sending a request to the memory with a specific memory location in the address bus. The memory unit will read the contents from the address specified by the CPU and send them to the processor by loading the data into the data bus.

The WRITE operation transfers data from the processor to a specific memory location. This operation will overwrite the contents in that memory location. The processor sends the data along with the memory location where it has to be stored. In a single operation, one word or 1 byte of data is transferred between the memory and the processor.

Memory allocation: The main memory must accommodate the operating system along with the user processes. To allocate memory locations among different processes there is a need for an efficient way to use the utmost capacity of the memory and reduce the loss of data or information due to overwriting. One of the simplest memory allocation methods is to divide memory into several fixed-sized partitions. One partition can hold information of exactly one process. Thus in this method, the number of processes in multiprogramming is restricted by the number of fixed-sized partitions. When a partition is free, the CPU loads a new process from the process queue. After completion, the partition is made available to another process.Another method of memory allocation is called a variable-partition scheme. Here the operating system maintains a table to keep the record of available and occupied memory partitions. Initially, all the partitions are free and available for the user processes. The block of available memory is known as a hole. In general, the available memory blocks can be defined as the set of scattered holes of different sizes throughout the memory. When a new process arrives, the operating system searches for a hole that is large enough to assign the process. There are three basic procedures to satisfy a request of a hole of size m from a list of n numbers of processes in the queue. The strategies used to select a suitable hole for a new process are first to fit, best fit, and worst fit. In the first fit, choose the first hole which is big enough to store the process. As soon as the hole is found, the search will stop. In the best fit procedure, choose the smallest hole which is large enough to hold the process. If the hole-size list is not in an ordered manner, it has to search the entire list to get the hole. In the worst fit, it chooses the largest hole from the free list of holes. Here also it has to search the entire list unless the list is ordered by size. It has been found that the first fit and best-fit strategies are better than the worst fit. The first fit procedure is comparatively faster than the others schemes.

8.6 MEMORY LOCATIONS AND ADDRESSES

Program instructions, operands, and results of arithmetic and logical operations are stored in computer memory. The computer memory is composed of millions of storage cells. Each storage cell can store one bit of information in the form of 0 or 1. To perform basic memory operations the cells are grouped into a fixed number of cells. Each group with n-bit is referred to as a "word" of information and the "n" will be known as "word length". It is shown in figure 8.2. Now a day's modern computers are typically range between 16 – 64 bits of word length.

A computer with 64 bit means that the address bus can carry an address of 64 bit. The address of memory locations is represented by 64-bit numbers. A computer with 32 bit can represent 2^{32} addresses.

Thus the addressing scheme of a system determined the maximum size of computer memory or address space. For example, a system with a 16-bit computer i.e. with an addressing scheme of 16-bit addresses can address up to 2^{16} = 64 K number of memory locations. Similarly, a machine with 32-bit addresses can generate 2^{32} = 4GB memory locations. The memory locations of a system determine the address space. Thus the address of each memory location is represented with k bits and using k address bit, 2^k nos. of locations or addresses can be represented. The address bit and number of locations can depict in the following table 8.1. Most computer systems are byte-addressable and memory is usually designed to store or access data in word-length. For a computer, the word length can be defined as the number of bits that store or are retrieved in one access. The processor reads the memory data by loading the address of the required memory location into the Memory Address Register (MAR). Similarly, during a write operation, the processor writes data into a memory location by loading the address of that location into MAR. To perform the read/write operation on a set of consecutive memory locations in the main memory, then a block transfer operation may be performed by sending the first address of the memory locations.

.

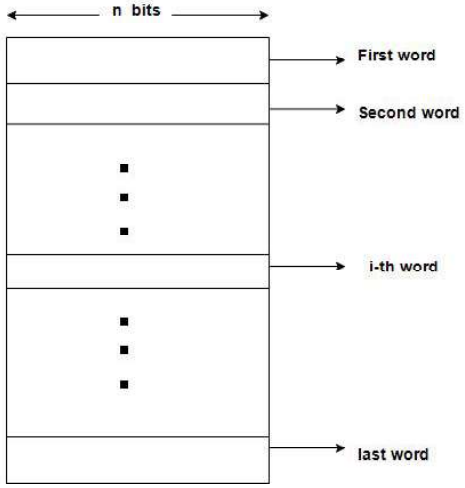


Figure 8.2 Memory words

Table 8.1: Number of Locations

K	Number of Locations
10	2^{10} = 1024 = 1 K
16	2^{16} = 65,536 = 64 K
20	2^{20} = 1,048,576 = 1 M
24	2^{24} = 16,777,216 = 16 M

8.7 BYTE ADDRESSABILITY

A nibble is 4 bits and a byte is 8 bits. The word length of a computer system can range between 16 – 64 bits. In modern practices, each successive address refers to successive byte locations in memory. For a computer system with 32 bits, successive words will be located at addresses 0000, 0004, 0008……. with each word of four bytes.

8.8 FRAGMENTATION

The first fit and best fit memory allocation strategies suffers from external fragmentation. External fragmentation can be defined as a memory status, where enough total memory space is available to satisfy a request, but not in a contiguous manner. All free memory blocks/holes are scattered. If all these scattered holes can be combined together to form a large free

block, it might be able to run a larger process. The first fit and best fit method can affect the amount of fragmentation. The external fragmentation becomes a major or minor problem depending on the total memory and the process size. In some cases, the allotted memory block to a process is a little bit bigger than required. The remaining memory space is too small to be assigned to other processes. If the allocated memory size is a little bit bigger than required, it can be defined as internal fragmentation. The difference between the allocated memory and required memory size is known as internal fragmentation.

8.9 PAGING

Paging is a method to permit physical address space to be non-contiguous. It avoids fragmentation and the need for compaction. The basic idea behind the paging is breaking the physical memory into some fixed size memory block called frame and divide the logical memory into some same size block known as page as shown in the following figure 8.3. When a process has to be executed, the memory pages are loaded into any available memory frame.

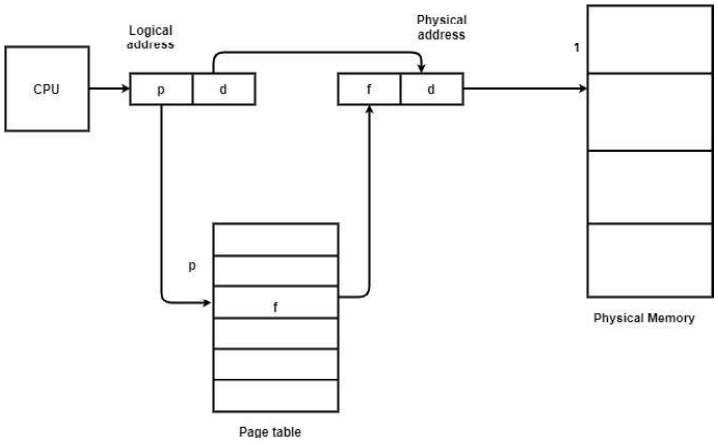


Figure 8.3: Paging

The address generated by the CPU is divided into two parts namely page number (p) and page offset (d) as shown in figure 8.4. The page number refers to the index of the page table. The page table contains the base address of each page in the physical memory (f).

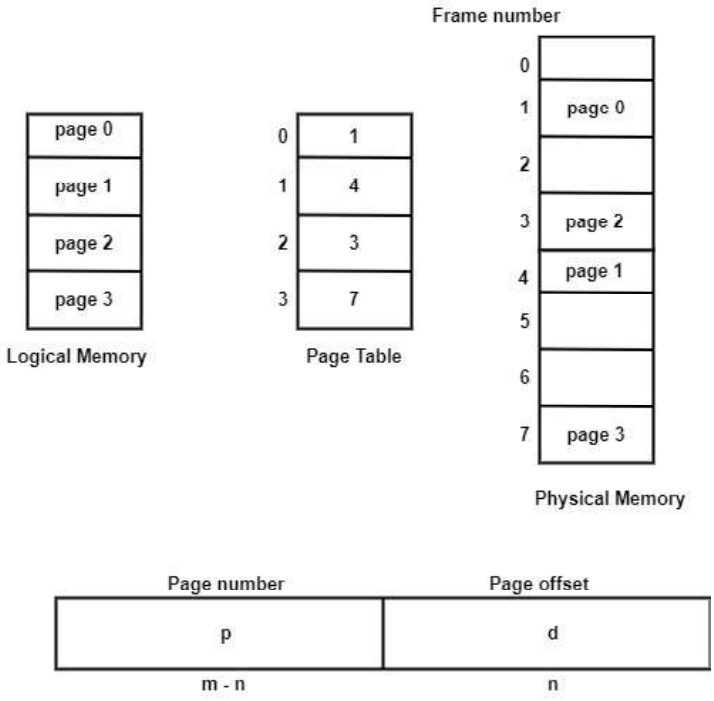


Figure 8.4: Implementation of page table with page number and offset

Combining the base address and the page offset (f + d) the address of the physical memory can be defined. The typical page size is always the power of 2 and ranges from 512 bytes to 16 MB per page. If the logical address space size is 2^m and page size is 2^n, the higher-order m – n bits represent the page number, and the n – low order bits represent page – offset.

Physical address space = m words
Logical address space = n words
Page size = p words
Physical address = $\log_2^m$ = m bits
Logical address = $\log_2^n$ = n bits
Page offset = $\log_2^p$ = p bits

For example, consider a physical memory of address space 32(m=5) bytes and page size is 4 (n = 2) bytes. The number of pages that can exist in the physical memory will be 8 pages. Using the paging scheme external fragmentation can be eliminated. But internal fragmentation may still exist. Suppose the page size is 2048 bytes and a process size is 72766 bytes. In

this case, the required number of pages will be 35 and additional 1086 bytes. So 1086 bytes of memory cannot be allocated separately. To hold the 72766 bytes, 36 frames will be needed. Then from the last frame 2048 – 1086 = 962 bytes will be unutilized. As a result, an internal fragmentation will exist. Each page of a process requires a frame. If the process has m number of pages, then atleast m number of frames should be free in the physical memory. The free frames may be scattered throughout the memory. After allocating a page into a free frame, the page table is updated by inserting the frame number. In reality, the user program is scattered throughout the physical memory. The mapping between logical to physical memory is controlled by the operating system. The user should not be concerned about it. The operating system maintains a data structure known as a frame table to keep the track of frame details such as – total no of frames, frames that are in use, available frames, etc.

8.10 SEGMENTATION

Generally, the programmers want to view the memory as a collection of segments of different sizes holding different information such as the main program,subroutine, symbol table, stack or other data structures etc. The segments created by a C compiler are shown in figure 8.5. When a programmer writes a code it consists of methods, functions, procedures, modules, data structures – stack, queue, array, and so on. Each of these elements is referred to by their names. Users should not use the address of these elements during coding, rather these can access through their names only. Segmentation can be defined as a memory management scheme that supports this type of user view of memory. So we can say that the logical address space is a collection of segments associated with its name and length. Within a segment, the elements are identified by their offset from the beginning of the segment. For implementation purposes, segments are assigned with some numbers and referred to by a segment number. So a logical address consists of two tuples:

< Segment-number, offset >

A c-compiler may create modules like – the code, global variables, stack used by each thread, standard library, heap as shown in the following figure 8.5.

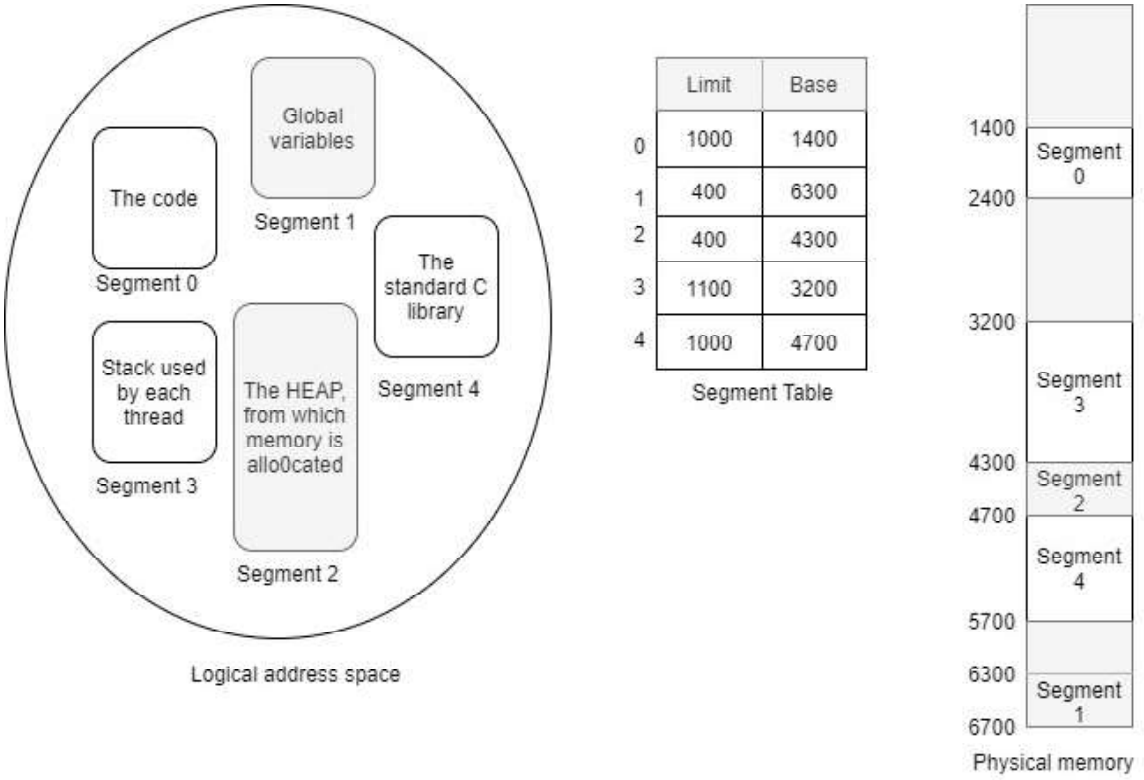


Figure 8.5: Segmentation

8.11 CACHE MEMORY AND MEMORY MAPPING

The processing speed of the CPU in comparison to the access time of primary memory is very high. Due to this bottleneck CPU cannot be utilized at its utmost level. To remove this barrier a smaller memory is being used in the system such that the average access time got decreases. This chip-based smaller and faster memory is known as cache memory. It is a temporary storage area from which the CPU can retrieve data easily during processing. Sometimes it is called the CPU Memory as it is typically integrated directly with the CPU chip or placed on a separate chip that has a direct connection with the CPU through a separate bus. As the cache memory is smaller in size and faster access time in comparison to primary memory, it decreases the average access time and efficiency of the

processor. The access speed of cache memory is 10 to 100 times faster than the primary memory. Cache consumes only a few nanoseconds to respond to a CPU request. Cache memory built with high-speed SRAM. It can be categorized into three different levels such as L1, L2, and L3 cache. L1 cache is extremely fast and usually embedded in the processor chip. L2 or secondary cache can be implanted on the CPU with a system bus. L3 cache is used to increase the performance of L1 and L2. The speed of L3 is comparatively slower than L1 and L2 but two times faster than DRAM.

8.11.1 Basic Operations of Cache Memory

When the processor needs a particular data during its execution, at first it searches the data in cache memory. If it is found, then it reads from the memory location of cache as specified by the processor. If the word addressed by the CPU is not available in the cache memory, the main memory is accessed to read the word. The program segment or a block of the word containing the desired one will be transferred from the main memory to the cache memory.

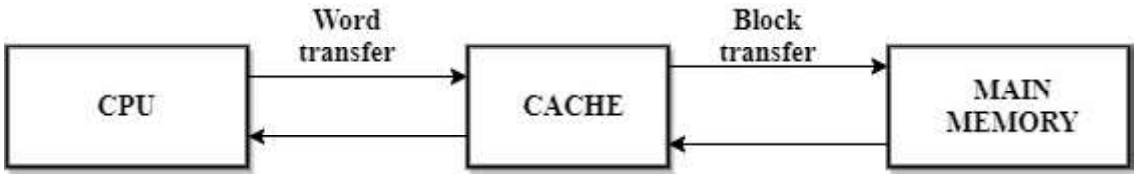


Figure 8.6 Cache memory

8.11.2 Cache Mapping Process

There are three different types of mapping techniques in cache organization such as: Associative mapping, Direct mapping and Set – associative mapping.

- **Associative Mapping**

In the case of the associative mapping procedure, each of the cache memory contents is associated with an address. But during the execution of a program statement, the data is not read or fetch by referring to or specifying any memory address. Instead of it, the data are searched by matching with the contents. In

associative mapping, the cache memory contains the data along with the main memory address of the corresponding data as shown in figure 8.7. The address bit sent by the CPU for searching the required data in the cache memory is matched with the stored addresses in the cache. If any address from the cache is matched with the main memory address referred by the CPU, the corresponding word from that memory location will be fetched by the CPU. For a miss or if no match is found for the required word, then it will be searched in the main memory. Then the word from the main memory along with the address will be transferred into the cache memory. If the cache is full, using any replacement technique must make room for the new word.

Associative mapping is a very fast access method. But the manufacturing difficulties and cost are more in comparison to other mapping methods.

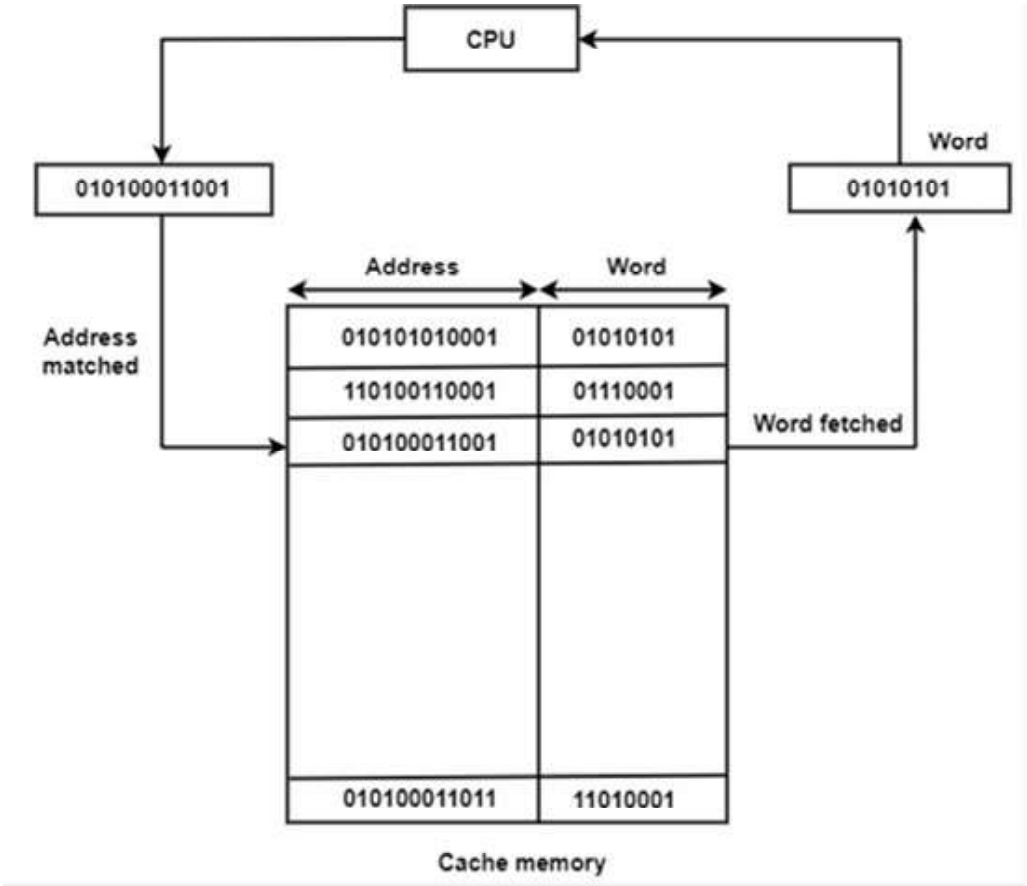


Figure 8.7: Associative mapping in cache memory

- **Direct mapping**

Consider a computer system with the main memory storage capacity is 4K, i.e. 4 x 1024 = 2^{12} bytes. Then the required number of bits to address the main memory location will be 12. Consider a cache memory of 1K = 2^{10} bytes, i.e. 10 bits are required to address a cache memory location. Thus the main memory required a 12-bit address line whereas the cache memory required only 10 bits of the address. In the direct mapping method, the address sent by the CPU is divided into two parts namely tag field and index field. The index field contains an equal number of bits that are required to address a word in cache memory. The remaining bits are used in the tag field. If a system contains the main memory of capacity 2^m and cache of capacity 2^n, then the bits in the index field will be n bits and in the tag field is an (m-n) bit. In the example cited above, the index field and the tag field are consisting of 10 bits and 2 bits respectively.

In direct mapping, the cache memory stores the data as well as the tag field as shown in figure 8.8. In the cache, the words are stored in a memory location as the index field defined. When an address is requested by the CPU, the index part of the address is used to get the location in the cache memory. If the tag of the cache is matched with the tag of the requested address, the word will be fetched by the CPU. Else there will be a miss and the data will be searched in the main memory. For a miss, the block of data from the main memory has to be transferred into the cache memory by dividing the main memory address into index and tag fields.

The main disadvantage of direct mapping is that, if the index field is the same for more than one word in cache memory with a different tag value, the hit ratio may drop considerably.

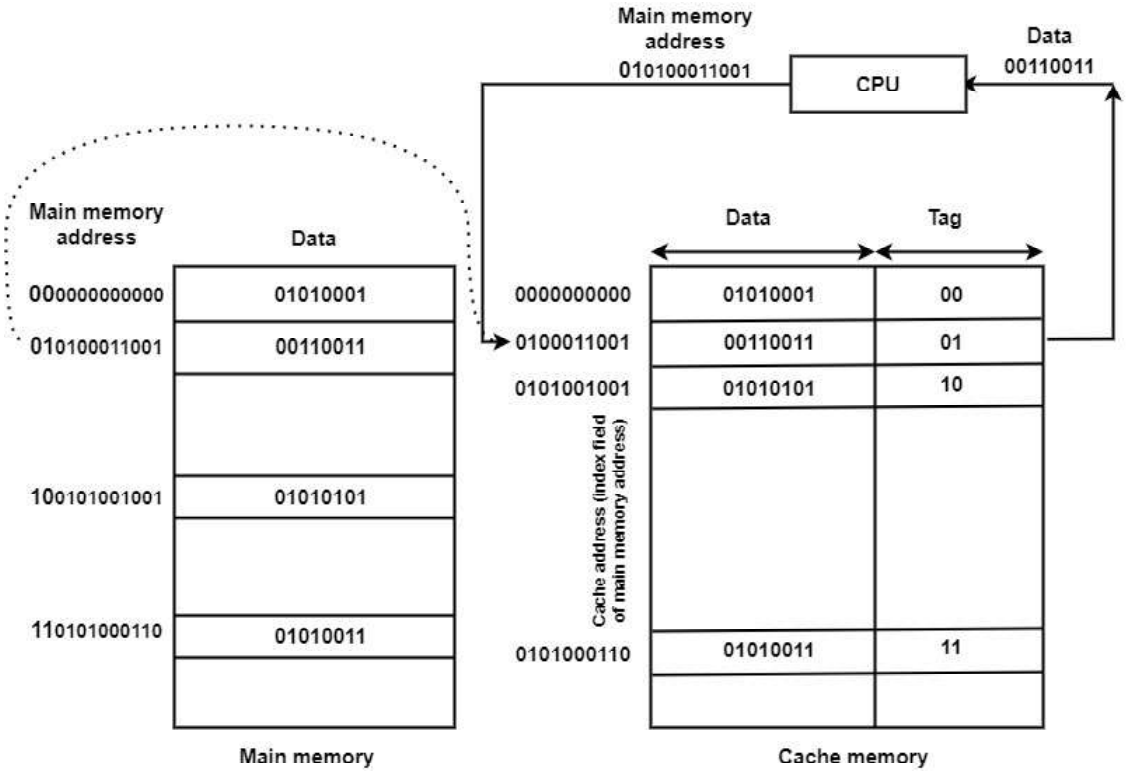


Figure 8.8 Block diagram showing direct mapping



CHECK YOUR PROGRESS

Q 1. Direct Mapping Question: Assume a computer has 32-bit addresses. Each block stores 16 words. A direct-mapped cache has 256 blocks. In which block (line) of the cache would we look for each of the following addresses? Addresses are given in hexadecimal.

a. 1A2BC012

b. FFFF00FF

c. 12345678

d. C109D532

- **Set Associative Mapping**

In direct mapping, two words or data of a similar index field cannot be stored at the same time. To overcome this drawback of direct mapping, the third method of mapping is used which is known as set-associative mapping. In this method, cache memories are

allowed to store more than one word with a similar index in the same word location along with a different tag. The number of tags–data pair in the one-word location of the cache is said to be as a set. An example of set-associative mapping has been depicted in figure 8.9. As shown in the figure, the word stored in the memory addresses 001010011001 and 011010011001 of main memory is stored in cache memory at index address 1010011001. Similarly, the word stored at address 101010000111 and 111010000111 of main memory is stored in cache memory index address 1010000111.

INDEX	TAG	DATA	TAG	DATA
1010011001	00	10000111	01	00111010
1010000111	10	10100110	11	11100001

Figure 8.9: Block diagram of set associative mapping



EXERCISE

1. A digital computer has a memory unit of 64k x 16 and a cache memory of 1k words. The cache uses direct mapping with a block size of four words.
   A. how many tags are there in the tag, index, block and words fields of the address format.
   B. how many bits are there in each word of cache, and how are they divided into functions? Include a valid bit.
   C. How many blocks can the cache accommodate?
2. A block set associative cache consists of a total of 64 blocks divided into 4 blocks sets. The main memory contains 4096 blocks each consisting of 128 words.

a) How many bits are there in main memory address?
b) How many bits are there in each of the TAG, SET and WORD fields?

8.11.3 Cache Replacement Policies

When the cache memory of a system is full, then cache replacement policies are used to make a decision about which page or data has to be replaced from the cache to make room for new data. The main problem in cache is that how to identify the page or data to be removed from cache memory. Lots of algorithms for cache replacement are being developed. The efficiency of those algorithms depends on the factors such as time, number of misses and balancing cost, etc. An efficient algorithm takes less time, lower number of miss rate and balancing cost. Some of the cache replacement algorithms are discussed as follows.

- **Least Frequently Used (LFU) algorithm**
  The LFU counts the number of uses of a particular data item or it counts how frequently the data item has been used. The data which is used very few times will be identified and removed from the cache first. If all the data items in the cache have the same count, then randomly any one of the items has been chosen and deleted. The min-heap data structure is a suitable one to implement this algorithm.
- **Random Replacement (RR) algorithm**
  The RR algorithm randomly selects any of the page from the cache memory and replaces it with the required one. It never keeps track of the history of removable pages and does not follow any data structure.
- **Pseudo – Least Recently Used (PLRU) algorithm**
  It is one of the most popular and common block replacement policies for the current generation's cache memory. It is widely used by the industry and a common policy for AMD and INTEL products. It uses the data structure binary tree for saving the

status of cache memory and hence it is also known as Tree – LRU. The tree structure is used to identify the block position which is to be replaced in case of a miss.

- **Lowest Latency First (LLF)**

  LLF algorithm keeps the average and minimum latency by removing the objects with the lowest latency. It shows the best performance during the execution of database queries in the relational database.

  All the algorithms discussed above can be classified into several classes such as –

  a. Recency-based algorithms
  b. Frequency-based algorithms
  c. Function-based algorithms
  d. Randomized algorithm

8.12 VIRTUAL MEMORY

A computer has a limited amount of memory space in primary memory or RAM. During programming, a user or developer has to concern about the limited memory address in RAM which increases the complexity of programming. To overcome this difficulty a technique called virtual memory is used. Virtual memory allows using more addresses than that the amount of physical memory available in the system. The main advantage of this memory is that the program may be larger than the size of the primary memory that physically exists. Using the concept of virtual memory, the logical and the physical memory can be separated. This separation allows using of a large virtual memory for the developers over the actual physical memory in the system. It gives an illusion to the programmer that large memory is available at their end even though the system has a smaller main memory.

The address generated by the user program is called a virtual address. A set of virtual addresses makes the *virtual address space*. The set of main memory addresses or locations are called *memory space* or *physical address space*. Usually, the virtual address space is larger than

the physical address space. To map between virtual addresses with physical addresses, memory mapping techniques are used by the memory controller of the system.

Consider a computer system with RAM of a storage capacity of 32K words. To specify a location in RAM with 32K physical address space (32K = 2^{15}) 15 bit is required. Consider that the system has 2^{20} = 1024K storage capacity auxiliary memory. Thus the address of the instruction code will have 20 bits whereas the memory address must be specified by 15 bit only. CPU will ask for reference instructions with the address of 20 bit, but at this point, the reference address must be taken from the primary memory of the address with 15 bit rather than auxiliary memory. Thus there is a requirement of mapping of virtual addresses of 20 bit to physical 15 bit.

8.12.1 Demand Paging

During the execution of a program by the CPU, it is not necessary to load the entire program into the main memory. If the program consists of several modules, and the system loads the entire program into the memory regardless of whether a module is selected by the user or not, can decrease the utmost utilization of the main memory and the capacity of the CPU. An alternative approach is to load the pages which are being requested by the CPU. This approach is known as *Demand paging* and is commonly used in the concepts of virtual memory. The demand paging scheme allows loading the demanded pages during execution. The pages which are never requested by the CPU will be never loaded into the physical memory. Demand paging is similar to the concept of paging. Here all the instruction sets reside in the secondary memory permanently. During execution, the required pages have to be swapped in between secondary memory and primary memory. Only the demanded pages will be replaced in the physical memory through swapping.

To separate the pages loaded into the secondary memory and main memory, the concept of a valid and invalid bit scheme is

used. When the bit is set to be valid, it represents that the page is legal and currently resides in the memory. The invalid bit says that the page either is not valid or valid but does not exist in the main memory, i.e. currently exists in the disk. For an invalid page, the processor never attempts to access that page. If the required pages reside in the memory during execution, the process can proceed and terminate normally. But if the process tries to access a page that is not in the memory, the page becomes invalid. An invalid page causes a *page fault*. The procedure to handle page fault is as follows:

Check the internal table to determine the valid/invalid status of the page.

If the page access reference is invalid, terminate the process. If the page is valid and not in the main memory, then it has to be brought into the memory.

Search for a free frame to locate the valid page in the memory.

Schedule a disk read operation to read the page into the newly allocated frame.

After completion of the reading process, the internal table and the page table will be updated. The updated info makes the page status valid.

Resume the execution from the interrupt point by accessing the missing page from the memory.

In some cases, the memory may not have any page during the execution of a process. At this point, if the CPU asks for a page that is not in the memory may cause a page fault. After bringing the page into the memory the process will continue its execution. Thus page fault will occur until all the required pages are loaded into the memory. After that, no page fault will occur. If no other page is brought into the memory except the required one, it is known as the *pure-demand paging*. The required hardware support for demand paging is as follows:

Page table – it is used to check the validity of a page using the valid-invalid bit.

Secondary memory – Secondary memory holds the pages which are not present in the memory.

8.12.2 Performance of Demand Paging

Consider that the memory access time of a system is denoted by *ma,* which may range between 10 to 200 nanoseconds. If no page fault occurs, the memory access time will be the *effective access time* for the system with a demand paging scheme. But, if any page fault occurs during the execution of a program, the effective access time will differ from the memory access time. During a page fault, the required page has to be brought into the main memory and it consumes extra time. Let *p* is the probability of occurring a page fault (where 0 d" p d" 1). It is expected that the value of *p* is closer to zero, i.e. only a few page faults. The effective-access-time can be expressed by the equation:

Effective-access-time=(1 – *p*) * ma + *p* * page fault time

The time to service a page fault needs to know for calculation of the effective-access-time.

8.12.3 Page Replacement

In the case of multiprogramming, if the number of processes is increased, the memory allocation will be overloaded and not all the processes can be in the memory at the same time. The most common solution to this problem is page replacement. In page replacement, the basic idea is that, if no frame is free, then search for a frame that is not being used and free it. In this situation atleast two-page transfers occur, one comes in and the other goes out. In the page replacement process, each page or frame is associated with a modify-bit in the hardware. When a page is written, immediately this modify bit is also set to indicate that the page has been modified. Before starting the replacement of a page, the modify bit is checked. If it is found set, indicates that it has been modified after reading from the disk, and the modified page has to write into

the disk.If the page is not modified, no need to update the disk. Page replacement is a basic method of demand paging. With the help of the page replacement technique, the implementation of the virtual memory concept is possible. There are many different page replacement techniques available. Each operating system follows its page replacement scheme. In general, the replacement algorithms are running on a particular string of memory references and computing the number of page faults. The string of memory references is known as the *reference string*.

The page replacements algorithms are: LIFO, FIFO, LRU, optimal page replacement, etc. these algorithms are similar to the replacement algorithms discussed in section 8.11.3 above. The term "miss" used in the cache memory can relate to the term page fault and the "block" can relate to the page in virtual memory.



EXERCISE

3. Let us consider a page referencing string 1, 3, 0, 3, 5, 6, 3 with 4 page frames. Find the number of page fault using FIFO page replacement algorithm.
4. Consider a set of page references string is as 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4 slots frame. Find the number of page fault using LRU page replacement algorithm.



8.13 LET US SUM UP

- The memory unit is a key part of a computer system.
- The set of addresses generated by a program is known as the *logical-address-space*, and the corresponding physical address set is defined as the *physical-address-space*.
- A set of physical addresses is known as memory space.
- The address space can be divided into groups of equal size known as a page.

- The memory space is broken into groups of equal size called blocks.
- A computer processing speed can represent using the address bit.
- External fragmentation can be defined as a memory status, where total memory space is available to satisfy a request, but not in a contiguous manner.
- The concept of virtual memory allows users to use memory space during programming which does not existsphysically.
- The speed of the CPU is very high than the average access rate of primary memory. The cache is used to remove this bottleneck between CPU and RAM.
- The basic idea behind the paging is dividing the physical memory into some fixed size memory block called frame and divide the logical memory into same size block known as page.
- Page table – it is used to check the validity of a page using the valid-invalid bit.
- Demand paging is similar to the concept of paging. Here all the instruction sets reside in the secondary memory.
- In the case of multiprogramming, if the number of processes is increased the memory allocation will be overloaded.



8.14 FURTHER READING

1. Galvin, P. B., Gagne, G., &Silberschatz, A. (2003). Operating system concepts (Vol. 10). John Wiley & Sons.
2. Hamacher, V. C., Vranesic, Z. G., &Zaky, S. G. (1984). Computer organization. McGraw-Hill, Inc..



8.15 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q No 1: Of the 32-bit address, the last four bits denote the word on the line. Since four bits are used for one hex digit, the last digit of the address is the word on the line. With 256 blocks in the cache, we

need 8 bits to denote the block number. This would be the third to last and second to last hex digit.

a. this would be blocking 01, which is block 1
b. this would be 0F which is block 15
c. this would be 67 which is block 103 (remember, 67 is a hex value)
d. this would be 53 which is block 83.



8.16 MODEL QUESTIONS

Q1. Choose the correct options from the following: (Multiple choice questions)

i. In a virtual memory system, the address space specified by the address lines of the CPU must be _____ than the physical memory size and _____ then the secondary storage size.
   A. Smaller, smaller
   B. Smaller, larger
   C. Larger, smaller
   D. Larger, larger

ii. In a virtual memory system, the addresses used by the programmer belongs to
   A. Memory space
   B. Physical address
   C. Address space
   D. Main memory address

Q2. What is virtual memory?

Q3. What is memory access time?

Q4. What is cache memory?

Q5. How a programmer can use more memory spaces than what exists?

Q6. Explain the cache replacement policy.

Q7. What are internal and external fragmentation? How it is related to paging?

Q8. Define the terms – word, block, page, frame, address space.

Q9. How to calculate the effective memory access for virtual memory?

Q10. Explain any three page replacement algorithms with examples.

Q11. Define the terms – miss, hit, hit ratio, page fault, locality of reference.

Q12. What is the function of modify bit?

Q13. Explain the virtual memory mapping procedure with an example.

Q14. Explain the concept of demand paging.

Q15. Explain the mapping techniques used in cache memory.

*** ***** ***