STUDENT'S FRIENDLY Edition

Always Buy a New Book for Revised & Updated Edition

Operating Systems

B.E. V-Sem (Computer Science and Engineering)

OSMANIA UNIVERSITY (Hyderabad)

[AS PER **AICTE MODEL CURRICULUM** FOR THE ACADEMIC YEAR **2020-2021**]

PROFESSIONAL PUBLICATIONS™

The Destination Towards Knowledge & Success

Head Office :

#12-2-825/826, Safa Arcade Building, 4th Floor, Door No. 562, Pillar No. 15, Above AXIS Bank, Mehdipatnam, Hyderabad - 500028, Telangana, India.

Phone : 040 - 6663 1899. **Mobile :** 7893 48 9991

e-mail : professionalpublications99@gmail.com

CONTENTS

Syllabus as per 2020-21 Curriculum

UNIT-WISE SHORT & ESSAY TYPE QUESTIONS WITH SOLUTIONS

INTRODUCTION

PART-A
SHORT QUESTIONS WITH ANSWERS

Q1. What is operating system? Discuss briefly the objectives of operating systems.

Answer : Model Paper-I, Q1

Operating System

An operating system is a program or a collection of programs that controls the computer hardware and acts as an intermediate between the user and hardware. It provides platform for application programs to run on it.

Objectives

OS has the following objectives,

(i) Efficiency

All the system resources present in the system should be utilized and managed efficiently.

(ii) Convenience

The operating system should provide an environment that is simple and easy to use.

(iii) Ability to Evolve

An operating system should be developed in such a way that it provides flexibility and maintainability. Hence, the changes can be done easily.

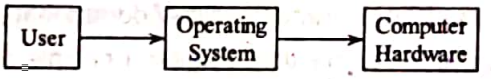


Figure: Operating System as an Interface

Q2. List various types of operating systems.

Answer :

The following are different types of Operating Systems (OS),

1. Batch Processing

A batch processing operating system reads a set of separate jobs, each with its own control card. This control card contains information about the task to be performed. Once the job is completed its output is printed.

Example

MS DOS

2. Multiprogramming

In multiprogramming, the OS picks one of the job from job pool and sends the job to CPU. When an I/O operation is encountered in that job, OS allocates I/O devices for that jobs and allocate CPU to next job in the job pool.

Example

Windows

3. Time Sharing

In timesharing systems, each program is given a certain time slot i.e., CPU is allocated to the program for certain period of time called "time quantum" (or) "time slice".

Example

Unix

4. **Real Time System**

Real time systems are time bounded systems wherein the system must responds to perform a specific task within predefine boundary.

Example

Airport traffic control space lights

5. **Distributed System**

The primary focus of distributed system is to provide transparency while accessing the shared resource i.e., a user should not worry about the location of the data.

Example

Novell network

Q3. Write the services of operating system.

Model Paper-II, Q1

Answer :

The following are the services provided by an operating system.

(i) Program creation and execution

(ii) User interface

(iii) I/O device support

(iv) File system management

(v) Interprocess communication

(vi) Resource allocation

(vii) Error detection

(viii) Accounting

(ix) Protection and security.

Q4. Define system call.

Answer :

The operating system provides a wide range of system services and functionalities. These services can be accessed by making use of system calls. The system calls acts as the interface between user applications and operating system (services). They are available as built-in functions or routines in almost all high-level languages such as C, C++, etc.

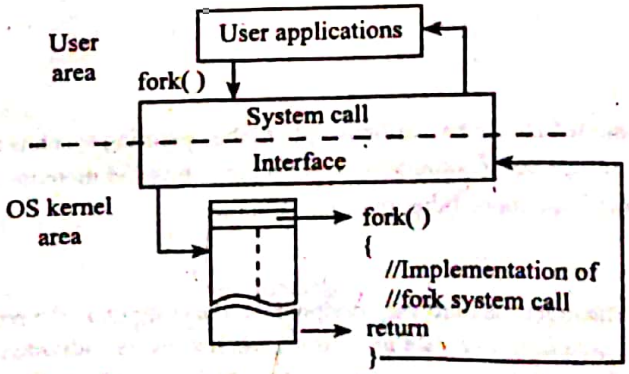


Figure: User Application Invoking System Call

Q5. What are the types of system calls?

Answer :

System calls are categorized into five groups depending upon the functionality offered by them. They are,

(i) Process control system calls

(ii) File management system calls

(iii) System information management system calls

(iv) Device management system calls

(v) Communication system calls.

Q6. What are the features of system call?

Answer : Model Paper-III, Q1

The following are the features of system calls,

1. It offers a process to create, load, execute and terminate them.
2. It offers a file to perform operations such as open, close, read, write, get file attributes and set file attributes.
3. It offers managing of system information like system date and time, operating system version etc.
4. It offers accessing of the system resources like main memory, disk drives etc.
5. It offers a process to exchange information by message passing or shared memory.

Q7. Write about design goals of operating system.

Answer : Model Paper-III, Q2

Designing a system requires in depth understanding of its goals and specification. However, it is very difficult to understand overall requirements because of the existence of different types of systems and hardware platforms. For this reason, the requirements are divided interms of user goals and systems goals.

From the user's point of view, the system must be,

- Fast and secure
- Easy to learn and use
- It must posses reliability and much more.

However, there is no particular way to define what design should be adopted to fulfill these goals.

From the system's (system designers) point of view, the design and implementation of the system must be carried out in an easy manner along with the features like flexibility, reliability etc. As there exist number of options for designing such a system, these design goals are also considered as vague.

Therefore, there is a no particular way of defining the overall requirements for designing the system. Rather, there exist some general principles such as that of Software Engineering that can be considered.

Q8. Write a brief note on privileged instructions and the concept of virtual machine.

Ans: Model Paper-II, Q2

Privileged Instructions

The instructions in a kernel mode that can be available only to the operating systems are known as privileged instructions. These instructions include halting a processes manipulating special mode bits and memory management timer management etc. These instructions can not be executed in a mode other than kernel mode.

Virtual Machines

Virtual machines are used to abstract the hardware peripherals of a computer into multiple execution environments. This abstraction enables the host operating system to create an illusion wherein every individual process is being executed on independent computer. Generally, virtual machine provides an interface which is similar to the interface provided by the underlying hardware.

Q9. List the advantages of virtual machines.

Ans: Model Paper-I, Q2

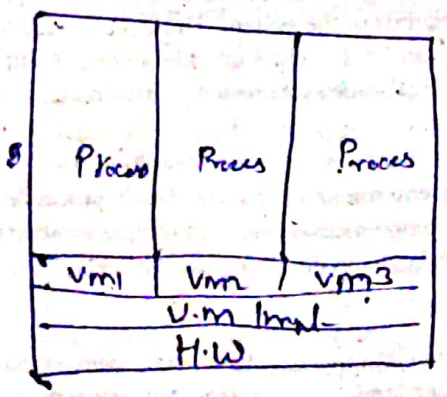
(i) Virtual machines protects the host system from any sort of virus attacks entering via a guest processes.

(ii) Virtual machines are preferable while performing research and development of operating system. This is because, modifying an operating system is a tedious task and basically consumes much of the system development time. Due to which, the system becomes unavailable to the user. Therefore, to avoid this issue, virtual machines are used.

(iii) Virtual machines enables the developers to execute multiple operating systems concurrently.

(iv) Virtual machines enables optimal usage of resources as it provides the provision of performing system consolidation (i.e., atleast two different systems are executed on separate virtual machines within a single system).

Benefits:



PART-B

ESSAY QUESTIONS WITH ANSWERS

1.1 CONCEPT OF OPERATING SYSTEMS

Q10. Define OS. Discuss in brief about the various objectives of an operating system. Also discuss its history.

Ans: Model Paper-III, Q11(a)

Operating System

An operating system is a program or a collection of programs that controls the computer hardware and acts as an intermediate between the user and hardware. It provides platform for application programs to run on it.

Objectives of an operating system

An operating system has three objectives. That is,

(i) Efficiency

An operating system must be capable of managing all the resources present in the system.

(ii) Convenience

An operating system should provide an environment that is simple and easy to use.

(iii) Ability to Evolve

An operating system should be developed in such a way that it provides flexibility and maintainability. Hence, the changes can be done easily.

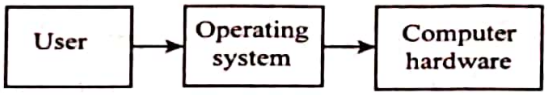


Figure: Operating System as an Interface

History of Operating System

The history or the evolution of the operating system is as follows,

1940's and 1950's

The digital computers invented in 1940's were capable of inserting the program instructions bitwise through mechanical switches and they does not had any operating system. This was then replaced by punched cards and later by assembly languages. And IBM 701 was first successful operating system used in early 1950s. These operating systems were capable of utilizing the computer effectively while executing single process at a time. These systems were typically referred to as single steam batch processing system.

1960's (BOS)

The batch processing systems were invented in 1960's were capable of processing multiple jobs concurrently. These systems allowed one process to use a processor and other processes to use other peripherals of the system. This concept resulted in the development of multiprogramming environment using which the processor can be switched quickly among multiple processes. Later on in late 1960's this multiprogramming evolved in timesharing and real-time systems which are time-based.

1970's (MPOS), (TSOS

Systems invented in 1970's were capable of providing various computing environments such as batch processing, timesharing and realtime in the form of various applications. They can also provide communication among multiple computers connected over a Local Area Network (LAN). Also personal computing environments began to develop with Apple's Apple-II.

1980's (RTOS)

Operating systems invented in 1980's started involving Graphical User Interface(GUI) that uses large set of icons, menus, windows etc. This made the use of computer interactive, interesting and easy to understand. These systems not only acts as a personal individual computer, but are also capable of performing single operation with multiple computers. This concept is known as distributed computing that work on the basis of client/server models.

1990's

Computers invented in 1990's were capable of processing thousands of MIPS (Million Instructions per Second) that save the data in gigabytes. Users of these systems were able to store and retrieve data from different remote devices with use of Internet (or) World Wide Web (WWW). Operating systems of these computers provide a wide range of networking features. However, they were prone to network and security threats. Some of the most popular operating systems of 1990's include Microsoft's DOS (Disk Operating System), Windows NT, 95, 98.

2000 and Above

Modern computers used these days are offering extra ordinary features including web services, middleware, parallelism, portability, virtual machines and many more.

1.2 GENERATIONS OF OPERATING SYSTEMS

Q11. Discuss about various generations of operating systems

Ans:

For answer refer Unit-I, Q10, Topic: History of Operating Systems.

Q12. Write a note on operating system generation.

Ans:

Operating System Generation

Operating System generation refers to the process of specifying configuration of specific machine or site where operating system has to be deployed and run. This process is often called as "SYSGEN". The operating system is usually stored in floppies or CD-ROMs or DVD-ROMs and distributed. The SYSGEN program is used to generate the operating system. It prompts the system operator to enter information about its hardware configuration or it may read the same from a file and automatically verifies that con-figuration is available or not.

The following information has to be verified,

- The type of CPU or processor used, the various components it has including instruction set, floating point arithmetics it supports. If there are multiple CPUs then each of them has to be defined.
- The memory capacity it allows.
- The various devices it supports and how it addresses them. Some systems maintain device number, interrupt number, type, model and special characteristic information for each device.
- Finally, the various options that user want from operating system like size of buffers, CPU-scheduling algorithms to be used, the number of processes supported at a time, etc. Usually, modern day operating systems decide themselves the type of scheduling algorithm to be used and buffer sizes, etc. These are dynamically changed according to system per-formance.

After getting the above information, operating system is compiled and data declarations, constants initializations are done to produce a version of operating system as desired by the user or particular system or machine.

Based upon the system descriptions several modules may be selected from a precompiled library and are linked together to generate the operating system as a whole. The selection may be choosing device drivers for I/O devices, etc. Such a system is faster because the modules are precompiled and generation does not need recompilation.

However, if the system is table driven, then selection of modules is done at execution time rather than compile or link time and generation is slower because recompilation is needed.

1.3 TYPES OF OPERATING SYSTEMS

Q13. What is an operating system? State and explain various types of operating system. And also discuss about operating system services.

Answer : Model Paper-II, Q11(a)

Operating System

For answer refer Unit-I, Q10, Topic: Operating System.

Types of Operating Systems

The following different types of operating systems (OS) are,

1. Batch processing
2. Multiprogramming
3. Time sharing
4. Real time
5. Distributed.

1. Batch Processing System (1960s)

A batch processing operating system reads a set of separate jobs, each with its own control card. This control card contains information about the task to be performed. Once the job is completed its output is printed. The processing in a batch system does not involve interaction of user and the job during its execution. However, in there systems the CPU was not utilized efficiently due to mismatch in processing speed of mechanical card reader and electronic computer.

2. Multiprogramming (1970s)

In mono-programming, memory contains only one program at any point of time. Whereas in multiprogramming, memory contains more than one user program.

In case of mono-programming, when CPU is executing the program and I/O operation is encountered then the program goes to I/O devices, during that time CPU sits idle. Thus, in mono-programming CPU is not effectively used i.e., CPU utilization is poor.

However, in multiprogramming, when one user program contains I/O operations, CPU switches to the next user program. Thus, CPU is made busy at all the times.

A single user cannot keep CPU busy at all times. Hence, multiprogramming increases CPU utilization by organizing jobs (programs), so that CPU is busy at all times by executing some user program or the other.

The idea in multiprogramming is as follows,

The OS picks one of the jobs from job pool and sends the jobs to CPU. When an I/O operation is encountered in that job, OS allocates I/O devices for that jobs and allocate CPU to next job in the job pool.

However, in mono-programming, CPU sits idle while I/O operation is being performed.

In multiprogramming most of the time CPU is busy. Advantages of multiprogramming are,

1. CPU utilization is high and
2. Higher job throughput.

Throughput is the amount of work done in a given time interval.

$$\text{Throughput} = \frac{\text{Amount of time CPU utilized}}{\text{Total time for executing the program}}$$

3. Time Sharing System (1970s)

For answer refer Unit-I, Q2, Topic: Time Sharing System.

4. Real Time System (1980s)

Real time operating systems are time bounded systems, wherein the system must respond to perform a specific task within predefined boundary. There are two types of real time system.

(a) Hard Real Time System

In this real time system, actions must be performed on specified time which could otherwise lead to huge losses. It is widely used in factories and production lines.

Example

In automobile, assembly line welding must be performed on time. This is because a weld before or after the specific instance can damage the product.

(b) Soft Real Time System

In this real-time a specified deadline can be missed. This because the level of loss is low compared to hard real time system.

Example

A video game can has voice not synchronized to the movie. This is still undesirable but does not causer huge loss.

5. Distributed System

It is a collection of independent, heterogeneous computer systems which are physically separated but are connected together via a network to share resources like files, data, devices, etc. The primary focus of distributed system is to provide transparency while accessing the shared resource i.e., a user should not worry about the location of the data. There are various advantages of distributed systems like they help in increasing computation speed, functionality, data availability and reliability.

Some operating systems provide generic functions or methods for accessing files over a network. They manipulate the networking details present in the device driver of network interface whereas, some other operating systems have separate functions for network access and other local access. There are various types of networks available for connecting computers together. These networks are distinguished by the protocols they use. Various protocols like ATM, TCP/IP, etc., are available but TCP/IP is the most popular network protocol and nearly all operating systems including Unix, Linux and Windows supports TCP/IP protocol.

A network operating system is one which has the features of accessing files over a network, communicating with other computers by sending messages. The network operating system provides an autonomous environment for each computer whereas, a distributed operating system provides a less autonomous environment, in addition to this, it gives a generic view to remote resources. Its primary focus is on transparency which hides the location details of a resource.

Operating System Services

For answer refer Unit-I, Q14.

1.4 OS SERVICES

Q14. State and explain operating system services that provide functions that are helpful to the user.

Answer : Model Paper-I, Q11(b)

The following are the services provided by an operating system,

(i) Program Creation and Execution

The operating system should support various utilities such as editors, compilers and debuggers etc., in order to give programmers the facility to write and execute their programs.

(ii) User Interface

The operating system should provide an interface through which a user can interact. Most of the earlier operating systems provide Command Line Interface (CLI), which uses text commands. All the users are supposed to type their commands through keyboard. Some systems support *batch interface* which accepts a file containing a set of commands and executes them. Now-a-days Graphical User Interface (GUI) is used where a window displays a list of text commands to be chosen by a user through an input or some pointing device.

(iii) I/O Device Support

There are numerous I/O devices. Each of them has its own set of instructions and control signals which are used during its operation. The operating system should take care of all these internal devices details and should provide users with simple *read*( ) and *write*( )functions for utilizing those devices.

(iv) File System Management

The user data is usually stored in files. An operating system should manage all these files and should provide functions to perform various operations on them, such as create, open, read, write, close, search (according to its name), delete etc. Additionally, it should protect files pertaining to different users from any unauthorized access.

(v) Interprocess Communication

There are several instances, when a process may require to communicate with other processes often by exchanging data among themselves. This interprocess communication is employed by operating system using techniques like message passing and shared memory.

(vi) Resource Allocation

In a system, multiple programs may be executing concurrently. It is the responsibility of the operating system to allocate resources (such as CPU time, main memory, files, etc.) to them. For example, various scheduling algorithms are used for allocating CPU time and resources to processes.

(vii) Error Detection

The operating system is responsible for keeping track of various errors that may occur in CPU, memory, I/O devices, user program, etc. Whenever errors occur, the operating system takes appropriate steps and may provide debugging facilities.

(viii) Accounting

It is a process of monitoring user activities, to keep track, which user has accessed which resources and how many times? This recorded statistical information can be used to improve the system performance by tracing out which resources are in demand and by increasing the instance of those resources.

(ix) Protection and Security

Modern computer systems allow multiple users to execute their multiple processes concurrently in the system. These multiple processes may access data simultaneously which has to be regulated so that only valid users are given access to the data. It is the job of operating system to apply protection and security mechanism to the system.

1.5 SYSTEM CALLS

Q15. Explain with a neat example how system calls are used.

Answer : Model Paper-II, Q11(b)

(System Calls)

The operating system provides a wide range of system services and functionalities. These services can be accessed by using system calls. The system calls acts as the interface between user applications and operating system (services). They are available as built-in functions or routines in almost all high-level languages such as C, C++, etc.

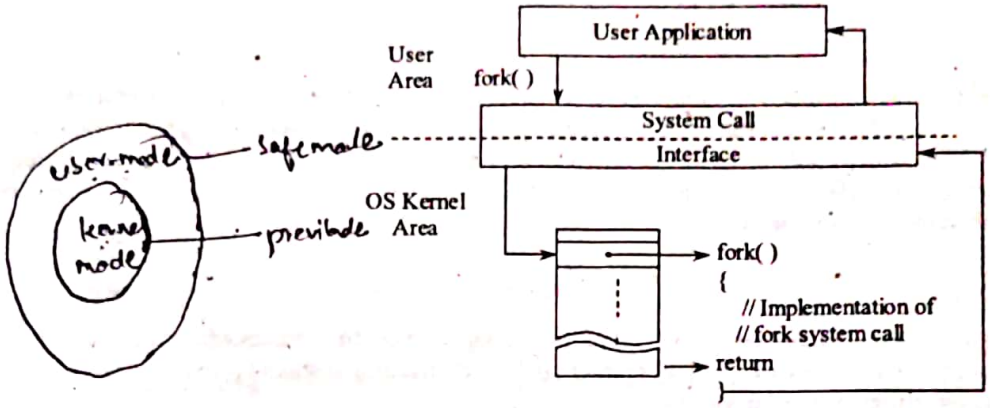


Figure: User Application Invoking System Call

Consider an example of copying contents of one file to another. To handle this task, several system calls are used from the time of prompting users to enter file names of source and destination files for copying the contents and closing these files.

- In GUI systems, users have the facility of selecting source and destination files with a mouse. This can be done using I/O system calls.
- The next step is to open the specified files. It is done using an *open*( ) system call. The system may have the possibility of containing errors if the specified files are not present or failed to open them due to access restrictions (security). Hence, a system call is aborted by the system.
- When both the files are opened, a *read*( ) system call is used to read the contents of source file and a *write*( ) system call is used to write those contents in the destination file. This is repeated till the end of file (eof) is reached.
- Finally, both the files are closed using another system call *close*( ) and probably user will be notified by displaying a message on screen using another system call.

However, developers of application programs need not concern about all these system calls. They use Application Program Interface (API), which gives them a set of built-in functions along with their parameters and return values. Examples of such APIs are Win32 API for windows based system, POSIX API for Linux, Unix, Mac OS based system and Java API for JVM (Java Virtual Machine) compatible programs. Different APIs though performing similar operations will have different function names.

The programming languages provide system call interface which acts as a link between application programs and system calls (of operating system). This system call interface maps the function calls of API to the necessary system call(s) of operating system. It uses a table for this mapping which maintains the numbers and addresses associated with system calls. After mapping, it invokes its respective system call. Hence, the API programmers are unaware of the complexity of OS system calls.

Parameters are passed to operating system using the following three methods,

1. **Registers**

The processor registers can be used to store the parameters, then system calls will receive those parameters by reading register values.

2. **Block of Memory**

If the number of parameters are more than processor registers, then they are stored in a block of memory and address of that block is stored in processor register.

3. **Stack**

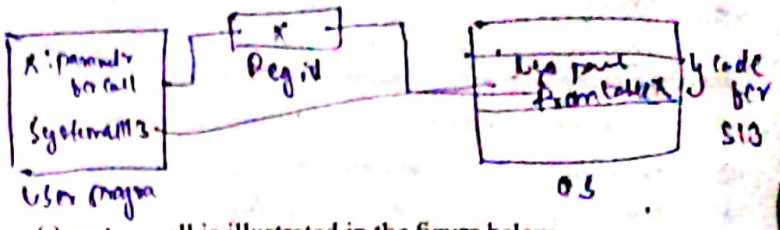
Parameters can also be pushed on a stack and system calls can receive them by popping them from stack. The advantage of this approach is that it doesn't limit the length of parameters being passed.

Q16. What is a system call? Explain how a user application invoking the open( ) system call is handled.

Answer :

System Call

For answer refer Unit-I, Q15, Topic; System Calls.



Handling open( ) System Call

The handling of a user application that invokes the open( ) system call is illustrated in the figure below.

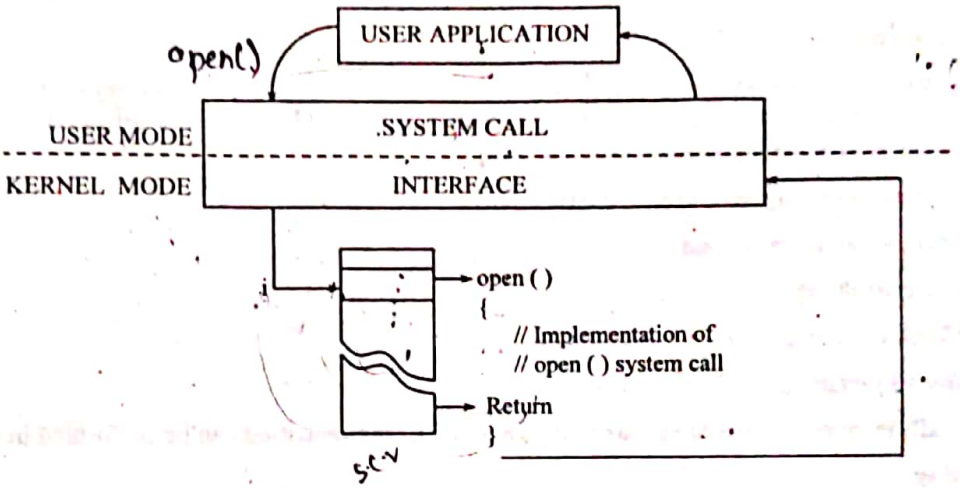


Figure: A User Application Invoking the open( ) System Call

When the user application invokes the open( ) system call, it puts the kernel into kernel mode. The kernel looks into the system call vector and then calls the corresponding open( ) kernel function. After completing the system call service routine, the kernel restores the original status of the user application and returns to user space, user application will continues its execution.

Q17. State and explain the various types of system calls in detail.

Answer : **Model Paper-I, Q11(a)**

Types of System Calls

System calls are categorized into five groups depending upon the functionality they provide. They are,

1. **Process Control System Calls**

A process refers to program under execution. Any operation in the computer system is executed in the form of a process. The system calls in this group are used to create processes, load, execute and terminate them. Also end or abort processes if runtime error occurs. The following are few process control system calls,

- Create process, terminate process
- Wait for event, wait for time, signal
- Load process, execute process
- Allocate memory and free memory
- Get attributes and set attributes
- Abort, end.

2. **File Management System Calls**

This category contains system calls to create, manage, read and write files. Some of the system calls in this group are,

- Create and Delete : Before the file is used for read or write we must be able to create the file. This system call usually takes the name of the file as parameter. The 'delete' system removes the file from the memory.
- Open file, close file
- Read file, write file
- Get file attributes and set file attributes.

3. **System Information Management System Calls**

This category contain system calls used to manage system information like system date, time, number of current users, operating system version, amount of free memory available etc. Some of these calls are,

- Get system information and set system information.
- Get current date/time and set current date/time.
- Get device attributes and set device attributes.
- Get process attributes and set process attributes.

4. **Device Management System Calls**

A computer system comprises several devices like main memory, disk drives etc. These are called resources of the system and are under control by operating system. A process that needs resources should uses system calls to access these resources from the operating system. Few device management system calls are,

- Request device from operating system and release device.
- Read from device, write to device and
- Get and set device attributes
- Attach device logically and detach device logically.

5. **Communications System Calls**

These system calls enables process to exchange information. Communication can be performed in two ways,

(i) **Message Passing**

In this method messages are exchanged between processes either through a mailbox or directly between processes. But before sending messages we need to create a connection.

(ii) **Shared Memory**

The communicating processes uses a memory area called shared memory process that wants to send a message, writes into shared memory. And process that wants to receive a message reads from the share memory. Shared memory is created using "shared memory create" and "shared memory attach" system calls.

Some of the communication system calls are,

- Create connection and delete connection.
- Shared memory create and shared memory attach.
- Send message, receive message.
- Send status information.

1.6 STRUCTURE OF AN OS-LAYERED, MONOLITHIC, MICROKERNEL OPERATING SYSTEMS

Q18. Discuss various approaches of designing an operating system.

Answer :

Model Paper-III, Q11(b)

Operating System Structure

For the better performance and proper functionality of operating system, it must be designed and organized carefully in such a way that it can be modified easily in the future. Usually, it is preferable to have several small components of a system instead of having a single or monolithic system. Each component should have a well-defined job and all the components are interconnected to form a single operating system. The following are the various approaches of operating system design,

1. Simple structure
2. Layered approach
3. Microkernels
4. Modules-based approach.

1. Simple Structure

There are several commercial operating systems which have simple but not well-defined structures. Usually, these systems were developed as small, simple and having limited functionalities, but their popularity grew beyond their original scope. One such operating system is MS-DOS which was developed keeping in mind that it should give more functionality within the limited space. Its structure does not carry carefull division of its modules.

MS-DOS has always experienced vulnerability towards threats and malicious programs which can cause damage to the entire system because of the improper separation between the interfaces and their functionalities. Due to this lack of security and protection, any application can easily gain access to the I/O operations and hardware of the system without any restrictions. In fact the hardware (i.e., 8088 processor) of that period also provides no hardware protection.

The earlier versions of Unix also falls in this category. It divides the system into two parts, the *kernel* (which is also called as heart of Unix) and the *system programs*. The kernel contains several device drivers which interacts with the system directly. Later the problem occurs in developing kernel because it became larger to implement as it has much more functionality.

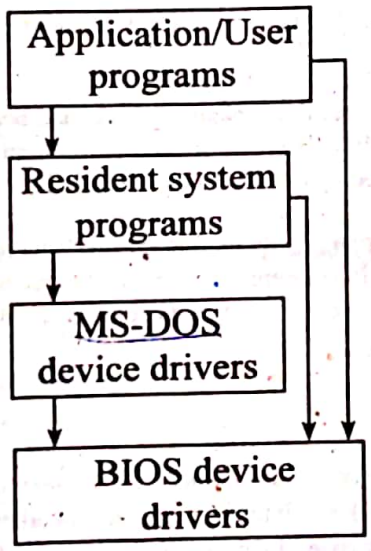


Figure (1): Structure of MS-DOS

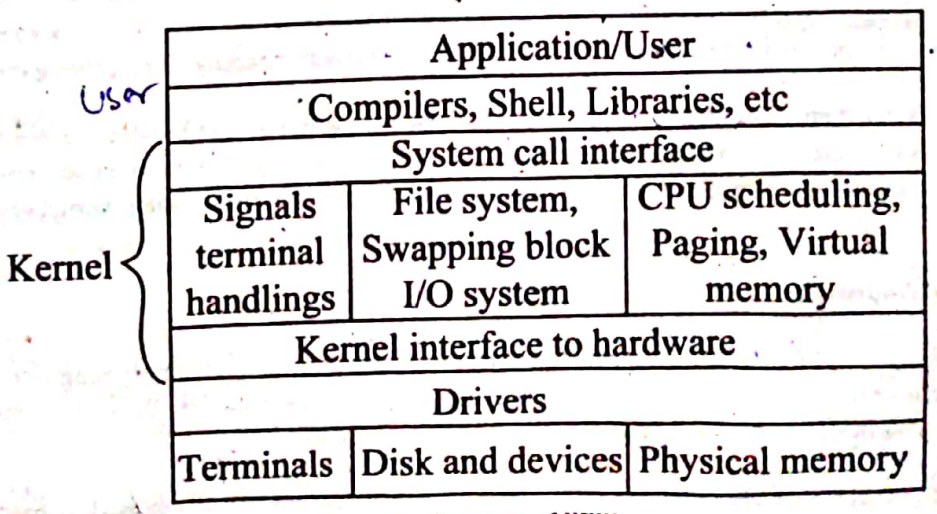


Figure (2): Structure of UNIX

2. Layered Approach

In layered approach an operating system is divided into multiple layers or levels. The highest layer (layer N) corresponds to users or application programs and the lowest or bottom layer (i.e., layer 0) corresponds to hardware.

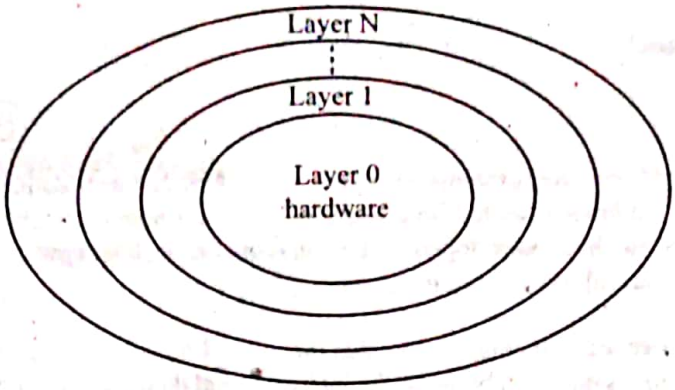


Figure (3): A Layered Operating System

Each layer consists of data structures and operations which are invoked by its upper layers. A lower layer provides some services to the upper layer. The advantage of this approach is that construction and debugging becomes simple. As we know the first layer (i.e., layer 0) is nothing but hardware and if we assume that hardware is running correctly, then its services can be used by layer 1. Now, the layer 1 is debugged and if any bug is found it is fixed. The advantage is that the errors can be fixed easily as they lie in that particular layer. Each higher layer simply uses the services of its lower layer without worrying about how these services are implemented (by the lower layer).

The limitation of this approach is that careful pre-planning is needed because a particular layer can use only lower-level layers. For example, the device drivers of hard disk should be at lower layers than the memory management layer, because memory management has to utilize the services provided by device driver of hard disk.

Another major problem in layered approach is its inefficiency as it increases the overall burden of an operating system. Consider an example, where a user executes an I/O system call, initially it is catched in I/O layer, which calls memory management layer and ultimately calls the CPU scheduler layer. In each of these calls an additional increase is observed in the system calls and processing time.

3. Microkernels

Microkernel approach is used to overcome the limitations of traditional unix kernel. The kernel in Unix was so large that it itself became a monolithic structure which is difficult to maintain. Microkernel approach removes all the unnecessary and non-essential components from kernel thereby decreasing its size. These non-essential components can be implemented outside the kernel as application level or system programs. The microkernel requires only minimal details about the process and memory management. Its primary function is to provide communication between user programs and various services running inside the user space. For example, a client program and a file server can interact indirectly by sending and receiving messages via microkernel.

The advantage is that it provides flexibility and extensibility. Any new service can be added to user space without modifying kernel. It also increases portability of operating system from one machine to another. It provides security and protection because most of the programs are running at user-level instead of kernel process. The examples of microkernel operating system are Tru64 UNIX, QNX, etc.

4. Module-based Approach

It is the best approach of operating system design which uses object-oriented programming techniques. In this approach various components of operating system are implemented as dynamically loadable modules. The kernel consists of core components and dynamic links to load those modules either at runtime or at compile time.

Operating systems such as newer version of Unix, Solaris, Linux and Mac OSX all use module-based approach. The Solaris operating system has seven loadable kernel modules as shown in figure (4).

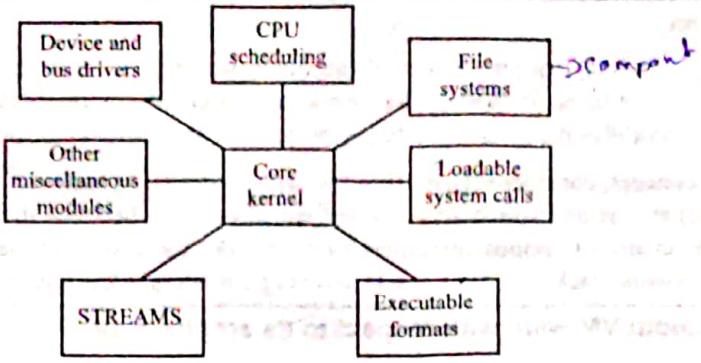


Figure (4): Loadable Modules of Solaris Operating System

The kernel consist of core and other services, like device drivers of certain hardware, various file system support etc. These can be added to the kernel dynamically when needed. The approach is similar to layered approach but it is more flexible when compared to it. The core services include the code for communication among them (modules) and loading modules.

1.7 CONCEPT OF VIRTUAL MACHINE

Q19. Write a brief note on history of virtual machines. Also discuss the benefits of virtual machines.

Ans:

History of Virtual Machines

Virtual Machines came into existence in 1972 which was developed by IBM to be used on its mainframe. It was called as VM operating systems which was later evolved to IBM VM 370. This version was capable of dividing the systems into various virtual machines each of which can act as a separate machine with its own operating systems. However, it failed in providing virtual disks because it is not possible to allot a separate disk to each of the virtual machine present in the OS. This features was included in the advanced versions of VM OS with a technique called minidisks. This techniques can divide the disk by using various tracks. When these features were included, it became possible to run multiple virtual machines, each with different operating system and running their own software packages.

Virtual machines are used to abstract the hardware peripherals of a computer into multiple execution environments. This abstraction enables the host operating system to create an illusion wherein every individual process is being executed on independent computer.

Generally, virtual machine provides an interface which is similar to the interface provided by the underlying hardware.

Advantages of Virtual Machines

1. Virtual machines protects the host system from any sort of virus attacks entering via a guest processes.
2. Virtual machines are preferable while performing research and development of operating system. This is because, modifying an operating system is a tedious task and basically consumes much of the system development time. Due to which, the system becomes unavailable to the user. Therefore, to avoid this issue, virtual machines are used.
3. Virtual machines enables the developers to execute multiple operating systems concurrently.
4. Virtual machines enables optimal usage of resources as it provides the provision of performing system consolidation (i.e., atleast two different systems are executed on separate virtual machines within a single system).

Q20. Write short notes on,

(i) Simulation

(ii) Para-virtualization.

Ans:

(i) Simulation

Simulation is a type of emulation methodology with which computer programs associated with one system architecture can be accessed over a other system with different architecture. For instance, consider an outdated system and an upgraded system. If the user wants to run certain programs compiled for the outdated system on an upgraded system, this type of emulator is used which usually translates that program instructions in such a way that it can be easily adopted by the new system. This type of environments eliminates the need of exploring old architecture. The upgraded machine needs to be much faster than the old machine because the emulation process runs with a slower magnitude. Therefore, performance is considered as a major challenge in emulation.

(ii) Para Virtualization

Another methodology which can run the virtual machines that are similar to each other but not identical with respect to there under lying hardware is referred to as para-virtualization. Unlike other virtualizations that treats the guest OS as its own system, para-virtualization modifies the guest operating systems inorder to use the resources more efficiently.

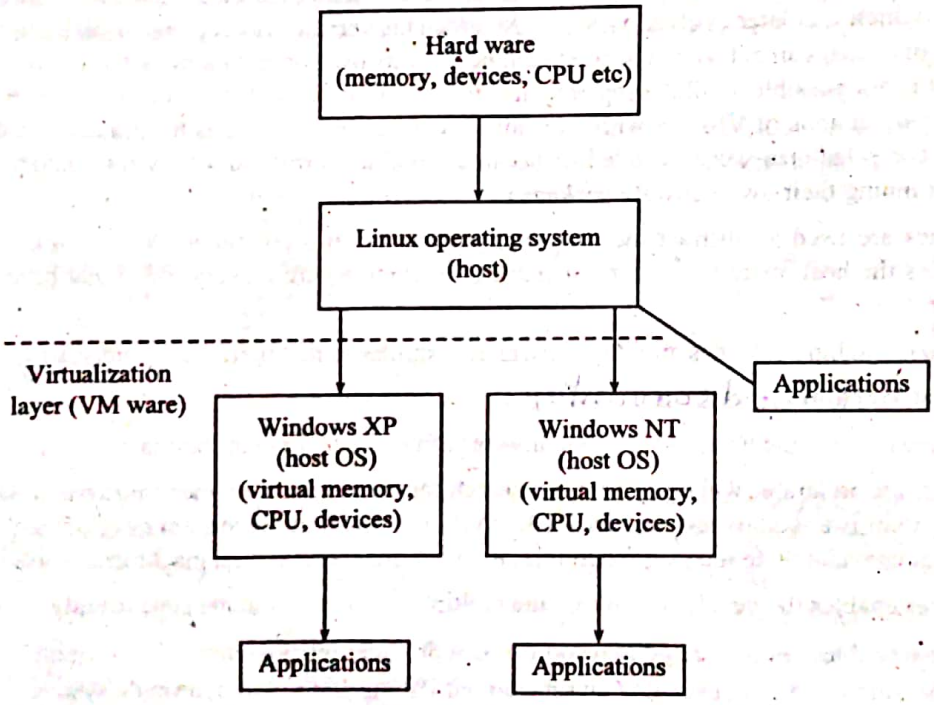
To understand the concept, consider solaris 10 which carries a single kernel with unvirtualized hardware. In such systems certains zones (containers) are created which acts as a virtualization layer between the programs and OS. Here, devices along with OS are virtualized by means of various containers which divides the system virtually. Each of these containers carry their own applications, parts, network stacks etc. Which acts as a single process in the whole system.

Q21. Explain briefly about VM ware with respect to its architecture.

Ans: VM ware is a virtualization platform which can be defined as a software/application mostly running an Intel X86 and other compatible hardware platforms. Multiple, OS platforms can be operated on the same host system using this application. It integrates flexible software layer into hardware and then divides into separate individual virtual machines. These virtual machines carry their own guest operating system and can run concurrently at any instance of time. They also carry their own memory, diskdrives, networks interfaces etc, which are just a file in the host operating system in reality.

This system provides increased efficiency because the moving of guest operating system requires moving just a file from one system to another. In addition to this, identical guest operating system platforms can be easy created by simply copying the associated file.

Consider an example of VM ware running on a Linux environment. The general architecture of VM ware would be as follows,



From the above figure, it can be observed that Linux OS(host) is divided into two guest as XP and Windows NT each carrying its own virtual CPU, memory and devices. This is done at the virtualization layer with which integrated hardware is virtually divided and allotted to the guest operating system.

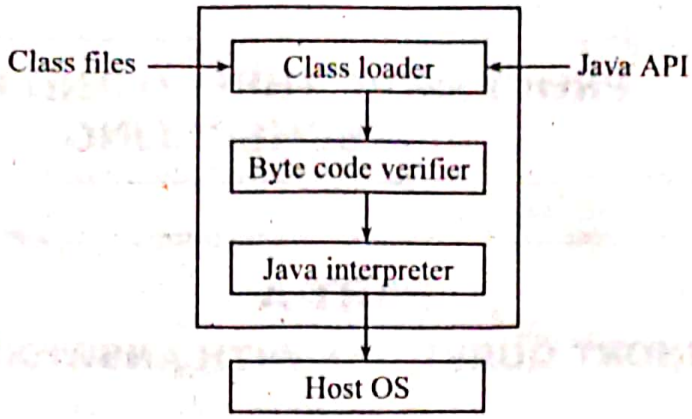
Q22. Explain in detail about java virtual machine.

Ans:

Java Virtual Machine

Java Virtual Machine(JVM) is the most important part of java technology. The JVM and Java API together, forms a platform for all java programs to run. JVM and Java API is also known as java runtime system.

Basically, JVM acts like an interpreter for java byte code, which is an intermediate code. The JVM loads the java classes, verifies the byte codes, interpreters and executes it. Additionally, it provide functions like security management and garbage collection. The figure shows the internal architecture of JVM.



Class Loader

It is mechanism that perform loading of classes and interfaces into JVM. When a program attempts to invoke a method of certain classes, then JVM checks whether that classes is already loaded, if not the JVM uses class loader to load the binary representation of that class.

Byte Code Verifier

After loading the class, the byte code verifier, checks whether the loaded representation is well-formed, whether it follows the semantic requirements of java programming language and JVM. Also checks whether the code contains proper symbol table. If a problem occurs during verification then an errors is thrown.

Java Interpreter

As already discussed that it is responsible for executing the byte code after their verification.

Memory management in JVM is done using garbage collection which returns the memory acquired by unused objects to the system. Most of the systems uses this approach for better performance of java programs.

One drawback of software based JVM is that the interpreter consider not more than one byte code at a time. For this reason most of users uses JIT (Just in Time) compiler which executes the byte code in terms of a native code.

UNIT 2

PROCESSOR, THREAD AND PROCESS SCHEDULING

PART-A

SHORT QUESTIONS WITH ANSWERS

Q1. List the attributes of the process.

Answer : Model Paper-I, Q3

Each process in an operating system is characterized by a set of attributes as follows,

1. Identifier
2. Process State
3. Priority
4. Program Counter
5. Memory Pointer
6. Context Data
7. Input/Output Status Information
8. Accounting Information.

Q2. Explain the following transitions,

(a) Blocked → Blocked/Suspended

(b) Blocked/Suspended → Ready/Suspended

(c) Ready/Suspended → Ready.

Answer : Model Paper-III, Q3

(a) Blocked → Blocked/Suspended

The process in the blocked state requires at least a single blocked process to swap out from the main memory because a new or unblocked process is made to enter into the main memory for continuing the execution. This swapping is even performed for reading processes, which require more memory space in order to produce adequate results.

(b) Blocked/Suspended → Ready/Suspended

A process that has been waiting in the blocked/suspended state is switched on to the ready/suspended state, when an event for which it is waiting takes place. In such a situation, an operating system can access the state information associated with the suspended process.

(c) Ready/Suspended → Ready

The process in the ready/suspended state requires the operating system to swap in a new process into the main memory, when no ready processes are available for continuing the execution. A process in the ready/suspended state may have a higher priority than a process in the other state. In this situation, the operating system executes a higher priority process so as to reduce swapping.

Q3. List the advantages of threads.

Answer : Model Paper-II, Q3

The advantages of threads are given as follows,

1. Creating a thread is ten times faster than creating a process i.e., it takes lesser time to create a new thread within an already existing process rather than creating it in a new process.
2. Thread termination is faster than the process termination.
3. Switching between the threads of a single process is faster as no memory mapping has to be set up and the memory and address translation caches need not be invalidated.
4. Threads are more efficient than processes as they provide feature of shared. They can communicate through shared memory hence, they do not require any system calls for inter-thread communication. So, threads are more suitable for parallel activities that are tightly coupled and uses the same data structures.
5. The processes of thread creation and destruction are cheaper as no allocation and deallocation of new address spaces or other process resources are required.

Q4. Is busy waiting always less efficient (in terms of using processor time) than a blocking wait. Explain.

Answer :

In general busy-waiting is less efficient than a blocking wait in terms of utilizing processor time due to multiple instruction cycles.

However, in some circumstances it is more efficient than blocking wait. For instance, when a process requires certain events on condition to occur for further execution. If this event has occurred already then the busy waiting will have information about the event occurrence instantly and the processor time will be used for other operation.

On contrary, blocking wait in this circumstance will perform process switching which lead to resources being utilized inefficiently.

Q5. Describe the differences among short-term, medium-term and long-term scheduling.

Answer : Model Paper-I, Q4

Long-term (Job) Scheduling

In operating system the number of processes submitted for execution is more than the number of processes executed instantly. Therefore, some processor are spooled (stored) in a storing device and executed later. The processor from job pool are selected by long term scheduler and loaded into the memory.

This scheduler executes process less frequently (in minutes) as it can afford to take time for selecting processes from the job pool.

- ❖ It is invoked only when a process exits a system.
- ❖ It also controls the level of multiprogramming.

Medium-term Scheduling

Medium-term scheduling reduces the degree of multiprogramming. This is done by temporary removal of some processes from the memory. Which are reintroduced into the memory and their execution starts from the previous state where they were taken out from the memory. This process of temporary removal and reassignment of the memory is quite often seen in medium-term scheduler.

This scheduler utilizes the swapping mechanism is adopted in time sharing system.

Short-term(CPU) Scheduling

Short term scheduling chooser the processes which are ready to get executed and assigned to the CPU.

- ❖ This schedules selects and submits a new process more frequently (in milliseconds).

Q6. What are the advantages and disadvantages of kernel level thread.

Answer :

Advantages of Kernel Level Thread

1. Multiple threads from the same process can be simultaneously scheduled by the Kernel on multiple processors.
2. If one thread in a process gets blocked, another thread within the same process can be scheduled to run by the operating system Kernel.
3. Kernel routines can be multi-threaded.
4. KLTs are especially good for applications that block frequently.

Disadvantages of Kernel Level Thread

1. Transfer of control from one thread to another within the same process causes the switching of mode to kernel.
2. The KLTs are slow and inefficient i.e., the thread operations in KLTs are very much slower than the User-Level Threads (ULTs)
3. The overhead and complexity of the Kernel are increased as it has to maintain Thread Control Block (TCB) for each thread and to manage and schedule both threads and processes.

Q7. Consider an environment in which there is a one-to-one mapping between user-level threads and kernel-level threads that allows one or more threads with in a process to issue blocking system calls while other threads continue to run. Explain why this model can make multi-threaded programs run faster than their single-threaded counter parts on a uniprocessor machine.

Answer :

Multi-threaded programs are executed at very high speed (when compared to their single threaded counter parts) on a uniprocessor machine, wherein there exist 1-1 mapping between user-level threads and Kernel-level threads. This is because Kernel-level-thread in multi-threaded program enables atleast one thread to issue a block system call independently without influencing other threads and thereby allowing other threads to un-interruptly continue with their execution. However, in single threaded counter parts of multi-threaded program, machine generally spends a significant amount of time waiting for I/O operation to be complete. Therefore on uniprocessor machine, a process that was allowed to be blocked continues executing its other threads.

Q8. What is meant by process preemption? Explain with examples.

Answer : Model Paper-II, Q4

Process Preemption

An operating system sets different priority levels for different processes. For example, the process *A* is executing at some given priority level and if some other higher priority level process, say process *B*, gets blocked due to some reason such as waiting for an occurrence of some event. The operating system continuously checks for the occurrence of that event and if it finds that the event has occurred, it immediately moves process *B* to the ready-state thereby interrupting the execution of process *A*. Hence, we can say that the operating system has preempted process *A*.

Process preemption also occurs in case of shared resources. Generally, in order to prevent deadlocks, the process which is currently holding a shared resource is preempted with the other process which is badly in need of that particular resource.

Example

If a process that is holding some resources, requests another resource that cannot be immediately allocated to it, then all the resources currently being held by that process are preempted i.e., the resources are implicitly released. All the preempted resources are added to the list of resources for which the process is waiting. The process will be restarted only when it can regain all its old resources along with the newly-requested resources.

Q9. Define Process Control Block.

Answer : Model Paper-III, Q4

In a multiprogramming system, it is necessary to get the information about each process that is being executed. All this information is available in the process control block.

Three categories of process control block information are,

(i) Process identification

(ii) Process state information

(iii) Process control information.

Process Control Block (PCB)

Process Identification
Process State Information
Process Control Information

Figure: Structure of PCB

Q10. Describe the differences between preemptive scheduler and non-preemptive scheduler.

Answer :

	Preemptive Scheduling		Non-preemptive Scheduling
1.	A scheduling scheme is said to be preemptive if and only if a process switches from the running state or waiting state to the ready state.	1.	A scheduling scheme is said to be non-preemptive if a process switches from the running state to the waiting state or if a process terminates.
2.	In this scheduling the server before completing the current request will switch to a new request for processing.	2.	In this scheduling the server will switch to a new request only after completing the currently scheduled request.
3.	Before a request gets completed, it may have to be scheduled for many times.	3.	Scheduling is performed only after completing the previously scheduled request.
4.	Here, preempted request is placed back into the pending requests list.	4.	Here, preemptive of request will never occur.
5.	There are four preemptive scheduling policies. (a) Round robin scheduling with time slicing (RR) (b) Least Completed Next (LCN) scheduling. (c) Shortest Time to Go(STG) scheduling. (d) Highest Response-ratio Next (HRN) scheduling.	5.	There are three non-preemptive scheduling policies (a) FCFS scheduling. (b) Shortest Request Next (SRN) scheduling (c) Highest Response Ratio Next (HRN) scheduling
6.	The algorithm of preemptive scheduling picks a process and lets this process to run for a maximum of some fixed time.	6.	The algorithm of non-preemptive scheduling will pick a process to run and let it run until it gets blocked or terminates.
7.	Preemptive scheduling requires a clock to interrupt the CPU to switch to a new process.	7.	Non-preemptive scheduling does not require a clock.

PART-B

ESSAY QUESTIONS WITH ANSWERS

2.1 PROCESSES

2.1.1 Definition

Q11. Define the following,

(i) Process

(ii) Program

(iii) Process control block

Answer :

(i) Process

Process is the fundamental concept of operating systems structure. A program under execution is referred to as a process. It can also be defined as an active entity that can be assigned to a processor for execution. A process is a dynamic object that resides in main memory. A process includes the current values of the program counter and processor's registers. Each process possesses its own virtual CPU. A process contains the following two elements,

(a) Program code

(b) A set of data.

(ii) Program

'Program' refers to the collection of instructions given to the system in any programming language. A program is a static object residing in a file. The spanning time of a program is unlimited. A program can exist at a single place in space. A program in contrast to a process is a passive entity. A program can consist of different types of instructions such as arithmetic instructions, memory instructions and input/output instructions, etc.

(iii) Process Control Block (PCB)

For answer refer Unit-II, Q14.

2.1.2 Process Relationship

Q12. What is the relationship between processes?

Answer :

The relationship that exist among various processes is a typical parent child relationship. The parent creates child processes and these child processes can create their own child processes forming a tree. Typical structure of such relationships can be drawn as follows,

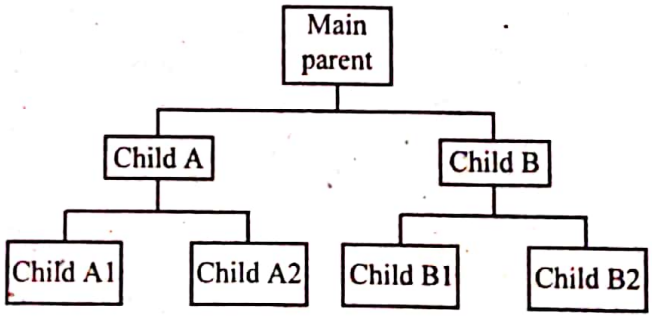


Figure: Parent-child Relationship of Processes

Each child has a single parent while a parent can have any number of children. Typically, all childs associated with parent receives interrputed if their parent gets an interrupt. Moreover, to terminate a parent process all its associated children need to be terminated.

However, there exist certain system calls with which, these properties can be modified. For example, a parent and its associated children usually execute the same process. This property can be changed using exec system call with which the parent and children can execute different programs.

Process state transition

2.1.3 Different States of a Process, Process State Transition

Q13. State and explain the various states that a process can be in.

Answer : Model Paper-I, Q12(a)

Process State

A process when being executed undergoes many states as per the demand. And the execution of the process is controlled by the operating system. The operating system is also responsible for allocating resources to the processes. In order to explain the behaviour of the process during their execution, process state transition models are used. The figure shown below depicts a five-state process model.

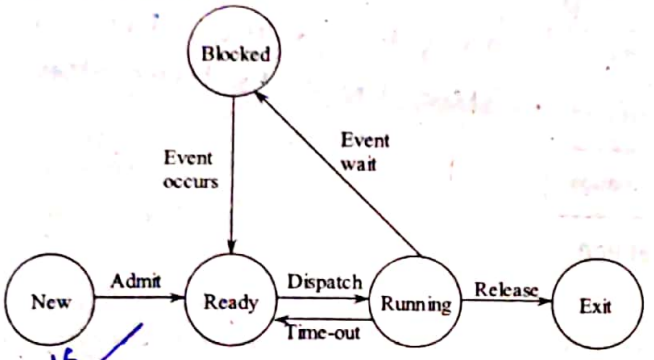


Figure: Five State Process Transition Model

When the number of processes to be executed are large, then those processes are moved to a queue and it would be operated in a Round Robin fashion. A process in the queue can be available in any one of the following states,

1. **Running**

   A process is said to be in running state, if it is being executed by the processor. In a uniprocessor system, only one process is executed by the CPU at a time. In case of multiprocessor systems, many processes can exist in a running state and the operating system has to keep track of all of them.

2. **Ready**

   A process in ready state is waiting for an opportunity to be executed. All the ready processes are placed in the ready queue.

3. **Blocked**

   Here, the process waits for the occurrence of an event, in order to be executed. Until that event is completed it cannot proceed further.

4. **New**

   A newly created process is one which has not even been loaded in the main memory, though its associated PCB has been created.

5. **Exit**

   A process is said to be in exit state, if it is aborted or halted due to some reason. An exit process must be freed from the pool of executable processes by the operating system.

The process can change their states according to the situations detailed below,

(i) **New-to-ready**

As soon as the operating system becomes capable of taking on additional process, it moves a process from new to ready state, then another process come to new state.

(ii) **Ready-to-running**

When a new process has to be selected for running, the operating system selects one of the processes in the ready queue then the state of the process which is in new state changes to ready state.

(iii) **Running-to-exit**

If the process which is currently running, either completes or aborts, that running process must be moved to an exit state by the operating system, then the process which is in ready state moves to running state.

(iv) **Running-to-ready**

If a process which is currently running goes to ready state when it is preempted either because a higher priority process becomes ready or its time elapses, then the higher priority process comes to running state.

(v) **Running-to-block**

If a running process needs some event to occur so that it can proceed with its execution in the running state, it waits for sometime. As the process is currently waiting it is pushed from the running state to the blocked state by the operating system. Then the process which is in ready state goes into running state. And as soon as the event occurs for which the process is waiting it is again moved back from block to ready state.

(vi) **Blocked-to-ready**

As soon as the event occurs for which the process is waiting, The process is pushed from blocked state to ready state, then the process which is running currently changes from running state to ready blocked state as per the preemption rules.

(vii) **Ready-to-exit**

A process would stay in the queue just because its parent process is not terminated once the parent process is terminated the child process can move from ready state to an exit state. Here the state of the child process is changed due to the change in its parent process state.

(viii) **Running-to-suspended**

If a process that is running completes using a resource, it goes from running state to suspended state, then the other process which is in suspended state goes into running state.

(ix) **Suspended-to-running**

As soon as the process which is running goes into suspended state, then the process which is in suspended state goes to running state.

2.1.4 Process Control Block (PCB)

Q14. Write about process control block.

Model Paper-II, Q12(a)

Answer :

Process Control Block (PCB)

In a multiprogramming system, it is necessary to get the information about each process that is being executed. All this information is available in the process control block.

Three categories of process control block information are,

(i) Process identification

(ii) Process state information

(iii) Process control information.

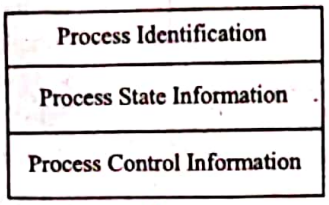


Figure: Structure of PCB

The typical elements of process control block are,

1. Identifiers — (Process identification)
2. User-visible registers
3. Control and status registers — (Process state information)
4. Stack pointers
5. Scheduling and state information
6. Data structuring
7. Interprocess communication
8. Process privileges — (Process control information)
9. Memory management
10. Resource ownership and utilization.

1. Identifiers

The following identifiers are stored in the process control block,

(i) Process identifier

(ii) Parent process identifier

(iii) User identifier.

2. User-visible Registers

These registers are used to manipulate mathematical operations.

3. Control and Status Registers

In order to control the operation of the processor, these registers are used.

(i) Program counter

(ii) Condition codes

(iii) States information.

4. Stack Pointers

These point to the top of the stack, where parameters and calling addresses for various procedures and system calls are stored.

5. **Scheduling and State Information**

Scheduling function needs the process state information. The operating system is responsible for gathering this information.

Following items comprise the state information,

(i) Process state

(ii) Priority

(iii) Scheduling related information

(iv) Event.

6. **Data Structuring**

It specifies the relationship among the processes.

7. **Interprocess Communication**

Two different processes can communicate with the help of flags, signals and messages. And this information is maintained in PCB.

8. **Process Privileges**

As system utilities and services make use of the privileges, memory and the type of instructions grants processes as privileges.

9. **Memory Management**

It refers to the pointers and page tables are assigned to the processes.

10. **Resource Ownership and Utilization**

This information specifies the resources which are being controlled and utilized by the processes. For example, opened files.

Q15. Describe the attributes of the process. Describe the typical elements of process control block.

Answer :

Process Attributes

Each process in an operating system is characterized by a set of attributes as follows.

1. **Identifier**

Each process is associated with a unique identifier which makes it distinguishable from all other processes.

2. **Process State**

The state of a process describes whether a process is ready, running, blocked or suspended.

3. **Priority**

Each process has some priority relative to other processes within the system.

4. **Program Counter**

The address of the instruction to be executed next is stored in the program counter.

5. **Memory Pointer**

It refers to the pointers associated with the program code and data along with the memory blocks shared with other processes.

6. **Context Data**

This data is usually stored in the processor registers, while the process is executing.

7. **Input/Output Status Information**

It contains all the information related to I/O processing, I/O requests, I/O devices etc.

8. **Accounting Information**

It contains the information about the processor time, clock time used, time limits, account numbers, etc.

Elements of Process Control Block

For answer refer Unit-II, Q14.

2.1.5 Context Switching

Q16. Write short notes on context switching.

Answer :

Context Switching

Context switching refers to the process of switching the CPU to some other process thereby saving the state of the old process and loading the saved state for the new process.

Context switching is actually an overhead which means that apart from switching no other tasks will be performed. Each machine carry different switching speed based on factors such as memory speed, number of registers that are needed to be copied and the presence of special instructions (for example, a single instruction that can be used to load or store all registers). The speed usually ranges from 1 to 1000 µsec.

The time required to do context switching typically depends on hardware support. An example of such type is a processor that can provide more than one set of registers. In context switch the pointer needs to be changed to active set of registers. The amount of work that is to be done during the process of context switching is more in case of complex operating systems.

A context switch may occur without changing the state of the process being executed. Hence, involves lesser overhead than the situation in which changes in the process states occurs from running to ready or blocked states. In case of changes in the process state, the operating system has to make certain changes in its environment, which are described below,

1. The context associated with the processor along with program counter and other registers are saved

2. Updates the PCB associated with the process being executed. This involves changing the state of the process to one of the available process states. Updation of other fields is also required.

3. The PCB of this process is moved to some appropriate queue.
4. Execution of the active process is transferred by selecting some other process
5. Updates the PCB of the chosen process as it includes the changes in its state (to running)
6. Updates the data structures associated with the memory management which may require the management of the address translation process.
7. Restores the context of the suspended process by loading the previous values of the PC and other CPU registers.

Thus, the process switch which involves a state change, requires considerably more effort than the context switch.

2.2 THREAD

2.2.1 Definition, Various States, Benefits of Threads

Q17. Define thread. What are the advantages of threads?

Answer :

Model Paper-III, Q12(a)

Thread

A thread can be thought as a basic unit of CPU utilization. It is also called as a light-weight process. Multiple threads can be created by a particular process. Each thread shares code, data segments and open files of that process with other threads. However, each of them has their separate register-set values and stacks.

Consider an example of a pagemaker or any word processing application. It has two important threads executing in it, one to read the keystrokes from the keyboard and other a spelling and grammar checking thread running in background. The following figure shows, a process having two threads in it.

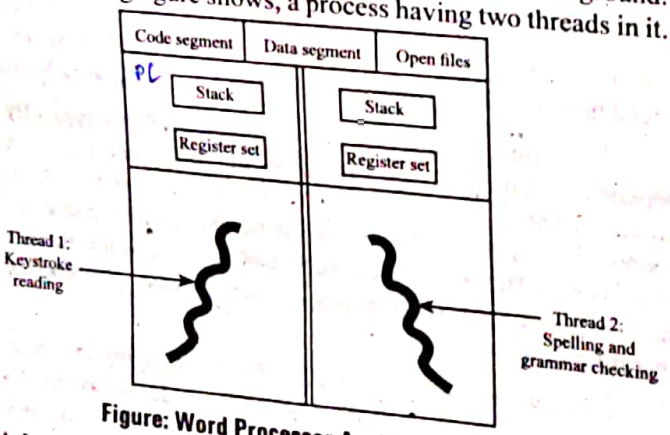


Figure: Word Processor Application's Process

Advantages of Threads

1. Creating a thread is ten times faster than creating a process i.e., it takes lesser time to create a new thread within an already existing process rather than creating it in a new process.
2. Thread termination is faster than the process termination.
3. Switching between the threads of a single process is faster as no memory mapping has to be set up and the memory and address translation caches need not be invalidated.
4. Threads are more efficient than processes as they provide feature of shared memory thereby requiring any system calls for inter-thread communication. So, threads are more suitable for parallel activities that are highly coupled and uses the same data structures.
5. The processes of thread creation and destruction are cheaper as no allocation and deallocation of new address spaces of other process resources are required.

Q18. Describe the relationship between threads and processes with an example.

Answer :

Relationship between Processes and Threads

A process is a program under execution. It is an abstraction of a stand-alone computer. They do not share memory but can communicate by sending messages to each other. This inter-process communication is based on the way in which hosts on a network can communicate with one another.

A thread is a software entity that can share memory and run concurrently. Some characteristics are common to both threads and processes but same memory space can be shared by multiple threads.

It is possible to execute a thread at any place within the code space of a process. Exactly similar code may or may not be executed at the same time, but if they execute the same code, it does not indicate the same execution but specifies fetching and executing the same instructions within different contexts.

Example

Execution of a thread is within the process environment and so a thread uses the resources of a process. The figure below shows the relationship between processes and threads. The process P_i has four threads.

Process environment for P_i

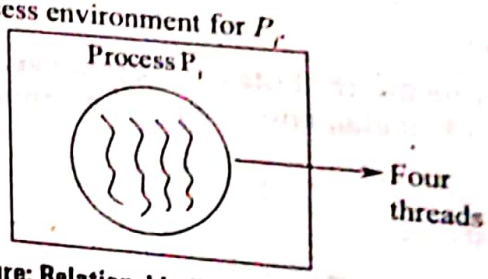


Figure: Relationship Between Process and Threads

Each thread is allocated with a stack and a Thread Control Block (TCB) which can be shown in above figure. All threads of the same process share code, data and resources. Thus the operating system needs to save only CPU state and stack pointer while switching between the threads of the same process.

In most of the system CPU register controls stack pointer so only the CPU state required to be saved before switching.

Usage of threads divides the process state into two states.

(a) A resources state that remains with the process

(b) An execution state that is associated with a thread.

The advantage of threads is that only the execution state need to be replicated for each thread to achieve concurrency within the environment of a process. There is no need to replicate the resource state.

Q19. What is the difference between a thread and a process?

Answer :

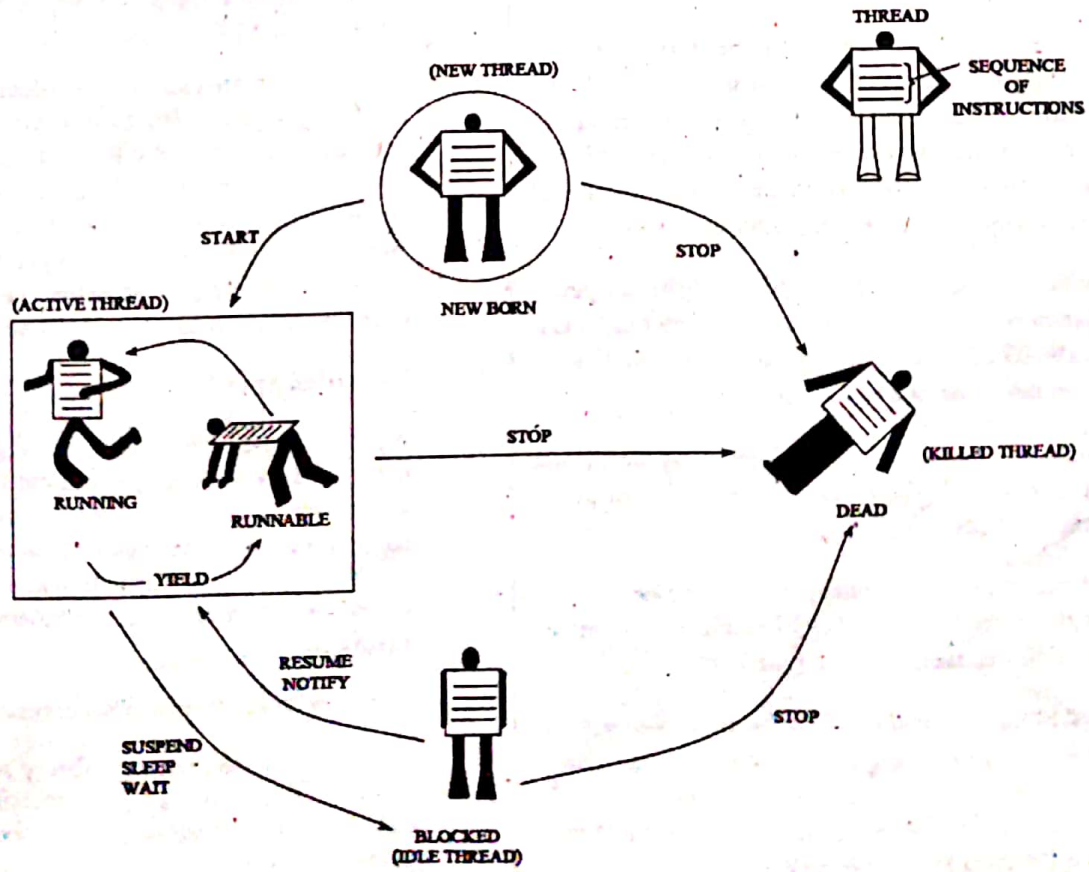
	Thread		Process
1.	Threads are software entities that can share memory and execute concurrently.	1.	Process is a program under execution.
2.	A thread is the 'lightest' unit of Kernel scheduling.	2.	A process is the 'heaviest' unit of Kernel scheduling.
3.	Threads share the address space of their parent processes.	3.	Process have their own address spaces.
4.	A thread can directly access the data segments of its parent process.	4.	A process can have its own copy of the data segment of its parent process.
5.	Threads can communicate directly with other threads of its parent process.	5.	Interprocess communication technique is usually followed for communicating between the process.
6.	Very little overhead, almost none, is incurred in case of threads.	6.	Considerable overhead is incurred in case of processes.
7.	Creating new threads is an easier process.	7.	Creating new processes involves duplicating the parent process.
8.	Threads can have considerable control over other threads of the same process.	8.	Process can have control only over their child processes.
9.	Any changes made to the main thread affects the behavior of other threads within the same process.	9.	Any changes made to the parent process does not affect its child processes.

Q20. Identify the different states of a thread.

Answer :

Life Cycle of Thread

The states of a thread can be clearly depicted by its life cycle.



In its life cycle the thread has five states they are,

1. Born state
2. Runnable state
3. Running state
4. Blocked state
5. Dead state.

1. **Born State:** A thread is said to be born when a new thread object is created. In this state the thread doesn't perform any work. Now it has two options as shown below.

   (a) It can be scheduled for running thread using start( ) method, then the thread is said to be in runnable state.

   (b) It can be killed using stop( ) method, then the thread goes to dead state.

2. **Runnable State:** If a thread is ready for execution and if the thread is waiting for the availability of processor then the thread is said to be in runnable state. In this state, the threads which are waiting for the availability of processor form a queue and whenever a thread gets processor it becomes active and moves to the running state. All the threads waiting in the queue have their own priorities. The thread with highest priority will go to the running state first. If the priorities of all threads are equal then the threads are activated according to first come first served fashion.

3. **Running State:** A thread is said to be in running state if it is being executed i.e., when a processor is being allotted to the thread. A running thread may relinquish from control on its own or it relinquishes when it is being preempted by a high priority thread. A running thread may relinquishes from control for three reasons.

   (a) When a thread is being suspended using suspend( ) method, it moves from running state to blocked state. The blocked thread can be renewed using resume( ) method.

   (b) When a thread is made to sleep using sleep(long milliseconds). The thread gets blocked for given milliseconds.

   (c) When a thread is asleep to wait using wait( ) method, the thread gets blocked and the thread can be renewed using notify( ) method.

4. **Blocked State:** A thread is said to be in blocked state, if it is avoided from entering the runnable to running state using suspend( ), sleep ( ) and wait( ) method. A thread which is active can be made inactive by using stop( ) and then the thread is said to be dead.

5. **Dead State:** This is the final state in the life cycle of a thread. A thread may be dead through different ways.

   (a) Natural death i.e., after completion of execution the thread generally dies.

   (b) Premature death i.e., a thread can be killed in new born state or running state or blocked state.

Q21. What are the advantages and uses of threads?

Answer :

Advantages of Threads

For answer refer Unit-II, Q17, Topic: Advantages of Thread

Uses of Threads

In a single-user multi-processing system, threads can be used for the following purposes.

1. **Foreground and Background Work**

   Consider a spread sheet program in which one thread performs the task of displaying menus and reading user input, while another thread is responsible for executing user commands and updating the spread sheet. Such an arrangement speeds up the application as it permits the program to fetch the next command before completing the previous command.

2. **Asynchronous Processing**

   Asynchronous program elements can be executed as threads.

   Consider for an example, protection against power failure, in such a situation a word processor, which periodically writes its RAM buffer to the disk for every minute must be designed. Hence, a thread is created which is responsible for the periodical backup and schedules itself with the operating system and no separate code is required in the main program for time checking and coordinating the input and outputs.

3. **Execution Speed**

   A multi-threaded process is one that can compute one batch of data reading the next batch from a device. In a multi-processor system, many threads belonging to a single process can be executed simultaneously. Thus, if one thread in a group gets blocked for an input/output operation, other threads in that process may still be executing.

4. **Modularity in Program Structure**

   Programs that uses large number of sources and destinations for input and output and which can perform multiple activities can be easily designed and implemented using threads.

2.2.2 Types of Threads

Q22. What is thread? List its types.

Answer :

The following are the types of threads,

(I) User Level Threads

In user level thread, application is responsible for the thread management task even though the kernel is unaware of the thread existence. Thread library consists of code for creation and termination of threads, in order to pass message and data among the threads. A sinel thread is used by the application to start the process and run inside that particular thread.

Advantages

1. No modification is required to the operation systems.
2. The representation of user-level threads is simple i.e., each thread is designated with the help of a stack, set of registers and a small control block, which are stored in the process address space of user.
3. User threads are simple to be managed i.e., there is no need of the kernel for the creation of a thread, synchronization between threads and switching between threads.
4. They are created very quickly and are efficient to use because switching between threadsisless expensive when compared to a procedure call.

Disadvantages

1. User-level threads are not supported by the operating systems. The entire process has only one time slice even if contains one thread or thousand threads. Thus, it depends upon each thread to give control to the other threads.
2. When a user-level thread gets blocked, it requires a multithreaded kernel called the non-blocking system so as to prevent the entire process from being blocked even though if the other threads are ready to run. For instance, if a page fault occurs in a thread then the process gets blocked

(II) Kernel Level Threads

In Kernel level thread, Kernel itself is responsible for the creation and management of threads. These threads are maintained by the operating system and works as a single process in an application. These threads are slower than user level thread.

Advantages

1. Multiple threads from the same process can be simultaneously scheduled by the kernel on multiple processors.
2. If one thread in a process gets blocked, another thread within the same process can be scheduled to run by the operating system kernel.
3. Kernel routines can be multi-threaded.
4. KLTs are especially good for applications that block frequently.

Disadvantages

1. Transfer of control from one thread to another within the same process causes the switching of mode to kernel.
2. The KLTs are slow and inefficient i.e., the thread operations in KLTs are very much slower than the ULTs (User-Level Threads).
3. The overhead and complexity of the kernel are increased as it has to maintain Thread Control Block (TCB) for each thread and to manage and schedule both threads and processes.

Q23. Bring out the differences between user-level threads and kernel level threads.

Answer :

User Level Threads (ULTs)	Kernel Level Threads (KLTs)
1. User level threads are managed by the application itself.	1. Kernel level threads are managed by the kernel. Hence, no thread management code is written in an application area.
2. They are scheduled by the thread library in the user address space.	2. They are scheduled by the kernel.
3. The thread library contains the context information.	3. Kernel maintains the context information.
4. No mode switching is required here, as all the thread management is done in the users address space.	4. Transfer of control from one thread to another within the same process requires mode switching to kernel.
5. On executing a system call if one thread gets blocked, all the other threads within that process are blocked.	5. If one thread in a process is blocked then the kernel can schedule some other thread within that process.

6.	Every user thread belongs to a process.	6.	Kernel threads need not be associated with a process.
7.	Lack of coordination between scheduling and synchronization.	7.	Scheduling and synchronization coordination exists among kernel-level threads.
8.	They are flexible and are of low cost.	8.	Semantic inflexibility exists as the kernel needs to be modified to adapt to the application thread model.
9.	They are efficient in carrying out certain operations like context-switching, as no kernel intervention is involved.	9.	Efficiency of the kernel-level threads lies in the ability of a kernel to simultaneously schedule multiple threads from the same process on multiple processors.

2.2.3 Concept of Multithreads

Q24. Write about multithreading.

Model Paper-II, Q12(b)

Answer :

The most prominent use of the concept of thread is an arrangement in which multiple threads may exist within a single process. Something approximating this approach is taken in MVS. More explicitly, this approach is taken in Windows, NT, OS/2, the Sun version of Unix, is an important operating system known as Mach.

Mach is designed specifically to work in a multiprocessor environment, although it is also well suited to a single-processor system. In Mach, a task is defined as the unit of protection, or the unit of resource allocation. The following are associated with tasks,

- A virtual address space that holds the task image.
- Protected access to processors, other processes (for interprocess communication), files and I/O resources (devices and channels).
- A thread execution state (Running, Ready etc).
- A saved processor context when not running; one way to view a thread is an independent program counter operating within a task.
- An execution stack.
- Some per-thread static storage for local variables.
- Access to the memory and resources of its task, shared with all the other threads in that task.

The key benefits of threads derive from the performance implications. It tasks for less time to create a new thread in an existing process than to create a brand-new task, less time to terminate a thread, and less time to switch between two threads within the same process. Thus, if there is an application or function that can be implemented as a set of related units of execution, it is far more efficient to do so, as a collection of threads rather than as a collection of separate tasks.

The following are four examples of uses of threads in a single-user multitasking system.

(i) **Foreground and Background Work:** This is in the sense of directly interacting with the user, not as in foreground and background session. For example, in a spreadsheet program, one thread could display menus and read user input while another thread executes user commands and updates the spreadsheet.

(ii) **Asynchronous Processing:** Asynchronous elements in the program can be implemented as threads. For example, as a protection against power failure, we can design a word processor to write its RAM buffer to disk once every minute. A thread can be created whose sole job is periodic backup and which schedules itself directly with the operating system; there is no need for fancy code in the main program to provide for time checks or to coordinate input and output.

(iii) **Speed Execution:** A multithreaded process can compute one batch of data while reading the next batch from a device. On a multiprocessing system, multiple threads from the same process will actually be able to execute simultaneously.

(iv) **Organizing Programs:** Programs that involve a variety of activities or a variety of sources and destinations of input and output may be easier to design and implement by using threads.

Q25. What is multithreading. Write a short note on various multithreading models.

Answer :

Multithreading

For answer refer Unit-II, Q24.

Multithreading Models

Main purpose of multithreading model is to make associations or associate threads at the user side and threads at kernel side with one another.

There are three models that can make this association possible.

1. Many to one model
2. One to one model
3. Many to many model.

1. **Many to One Model:** In this model, associations are made between one thread at the kernel side and many threads at the user side.

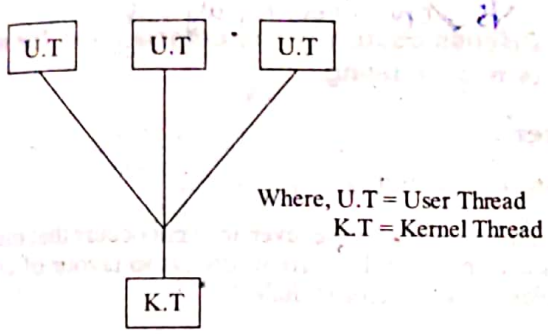


Figure (1): Many to One Model

2. **One to One Model:** In this model, associations are made between one thread at the kernel side and one thread at the user side i.e., suppose there are three kernel and user threads, then one kernel thread is associated with one user thread, then second kernel thread is associated with another user thread and so on.

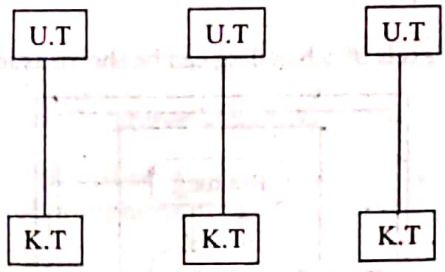


Figure (2): One to One Model

3. **Many to Many Model:** In this type of model, few threads at the kernel side are associated with more threads at the user side.

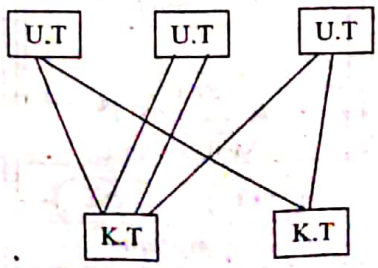


Figure (3): Many to Many Model

2.3 PROCESS SCHEDULING

2.3.1 Foundation and Scheduling Objectives, Types of Schedulers

Q26. What is process scheduling? Explain different types of process schedulers.

Answer :

Process Scheduling

In multiprogramming system, there are several processes running in the system, all are not simultaneously executed by CPU, but scheduling is done to choose a particular process for execution. It is done using a program called process scheduler.

Scheduling Queues

Process goes through several queues throughout their life cycle. Whenever, a process enters a system, it is put into a "job queue". From here if all the resources required by process are available it is put into "ready queue" where processes compete for getting CPU.

Whenever, the executing process is interrupted by an I/O request, then that processed is placed into "device queue". Each device like hard disk, floppy disk etc., has their own device queue. They take processes from their queue and serves them.

There are a number of reasons, when a process might stop executing and get placed in ready queue or in any other devices queue. Some events that interrupt process are,

- An I/O request occurred in the process, puts itself in I/O queue.
- If time slice of that process has expired, then it is put in ready queue.
- If a process creates its sub processes, then those sub process are first executed and after they terminate, the parent process is again placed in ready queue.
- If any I/O device interrupts CPU, then CPU puts the current process in ready queue and serves that interrupt by executing its respective Interrupt Service Routine (ISR).

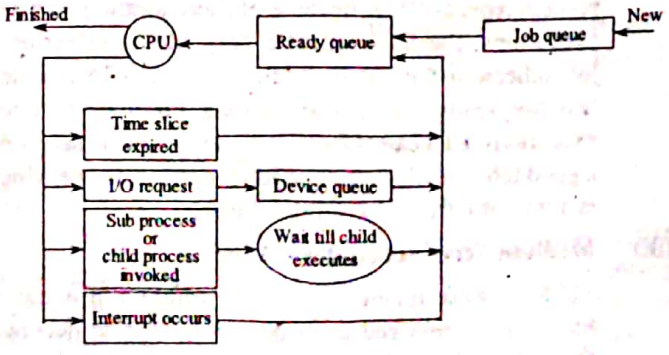


Figure: Various Scheduling Queues

Schedulers

Scheduling is defined as the activity of deciding about the resources to be send to the processes on their request.

Scheduler is defined as a program which selects a user program from disk and allocates CPU to that program. A process migrates between the various scheduling queues throughout its lifetime. The operating system must select (for scheduling purposes) processes from the queue in some fashion. The selection process is carried out by the appropriate scheduler.

There are three types of schedulers. They are as follows,

(i) Long Term Scheduler (LTS)

(ii) Short Term Scheduler (STS)

(iii) Medium Term Scheduler (MTS).

(i) Long Term Scheduler (LTS)

LTS is used to decide, which processes are to be selected for processing. Long term scheduler is defined as program (part of operating system) which selects a job from the disk and transfers it into main memory. If the number of ready processes in the ready queue becomes very high, the overhead on the operating system for maintaining long lists, context switching and dispatching overcrosses the limit. Therefore, it is useful to let in, only a limited number of processes in the ready queue to compete for the CPU. The long term scheduler manages this.

(ii) Short Term Scheduler (STS)

Short term scheduler is defined as a program (part of operating system) that selects among the processes that are ready to execute and allocate the CPU to one of them. It decides which of the ready processes are to be scheduled or dispatched next.

The difference between LTS and STS is that the LTS is called less frequently whereas, STS is called more frequently. LTS must select a program from disk into main memory only once i.e., when the program is executed. However, STS must select a job from ready queue quite often (for every 1 second in unix operating system) i.e., for every 1 second the STS is called, it will select one PCB from the ready queue and gives CPU that job. After 1 second is completed, again STS is called for selecting one more job from the ready queue. This process repeats. Thus, because of short duration between executions, the STS must be very fast in selecting a job, otherwise CPU will sit idle. However LTS is called less frequently, so because of long durations between executions, LTS can afford to take some time in selecting a good job from disk. A good job is defined as one which is a mix of CPU burst and I/O burst.

(iii) Medium Term Scheduler (MTS)

MTS is used during swapping, where a process is temporarily removed from memory, often to decrease the overhead of CPU and later resumed.

We know that, as the degree of multiprogramming increases, CPU utilization also increases. At one stage the CPU utilization is maximum for a specific number of user programs in memory. At this stage, if the degree of multiprogramming is further increased, CPU utilization drops. Immediately, operating system observes decrease in CPU utilization and calls MTS. The MTS will swap on excess programs from memory and puts on disk. With this, the CPU utilization increases. After some time, when some programs leave memory, MTS will swap in those programs which were swapped out back into memory and execution stops. This scheme which is known as swapping is performed by MTS. Thus, swap out and swap in should be done at appropriate times by MTS.

2.3.2 Scheduling Criteria

Q27. Discuss about various criteria used for short-term scheduling.

Answer :

Short-term Scheduling

This takes place whenever an event occurs that may lead to the interruption of the current process in favour of another. Examples of such events include,

(a) Clock interrupts

(b) I/O interrupts

(c) Operating system calls

(d) Signals.

This is the actual decision of ready process to execute next.

The levels of scheduling can be shown as follows,

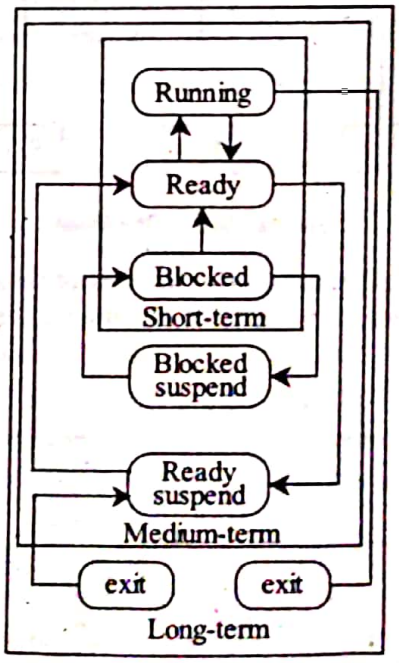


Figure: Levels of Scheduling

A set of criteria is established against which various scheduling policies may be evaluated and is categorized into user-oriented and system-oriented criteria. User-oriented criteria relate to the behaviour of the system as perceived by the individual user or process. E.g, Response time.

Other criteria is system-oriented i.e., the focus is on effective and efficient use of the processor. E.g, throughput.

Criteria for Short Term Scheduling

Many criteria have been suggested for comparing CPU scheduling algorithms. Criteria that are used to include are the following.

CPU utilization : This is the amount of time when the CPU is kept busy executing processes.

Throughput : This is the number of processes that are completed per unit time.

Turnaround time : The interval from the time of submission to the time of completion is the turnaround time.

Waiting time : This is the sum of periods spent waiting in the ready queue.

Response time : The time from the submission of a request until the first response is produced in an interactive system.

It is necessary to keep the CPU busy all the time which can be done by increasing the CPU utilization and throughput thereby decreasing response time and waiting time. CPU scheduling basically decides the alloting of the CPU to the processes in the ready queue the CPU is allocated next.

2.3.3 Scheduling Algorithms

Q28. Explain preemptive and non-preemptive scheduling algorithms.

Answer : **Model Paper-III, Q12(b)**

Preemptive and Non-preemptive Scheduling Algorithms

Scheduling is defined as the activity of deciding, when processes will receive the resources they request. There exist several scheduling algorithms among which some are discussed here.

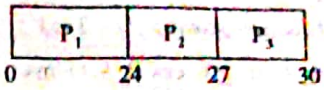
(i) First Come First Served (FCFS) Scheduling Algorithm

As the name suggests, it first allots the CPU to process that requests firstly from the ready queue. It is considered as the simplest algorithm as it works on FIFO(First in First Out) approach. In the ready queue when a new process requests CPU, it is attached to the fail of the queue and when the CPU is free, it is alloted to the process located at the head of the queue.

One of the difficulties associated with FCFS is that the average waiting time is quite long. For instance, consider a set of three processes P_1, P_2, P_3 whose CPU burst times are given below.

Process	Burst Time (ms)
P_1	24
P_2	3
P_3	3

If the sequence of arrival is P_1, P_2, P_3 then we get the following result.



So,

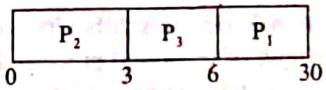
Waiting time for process P_1 = 0 ms

Waiting time for process P_2 = 24 ms

Waiting time for process P_3 = 27 ms

Average waiting time = $\frac{0+24+27}{3}$ = 17 ms

If the sequence of arrival is P_2, P_3, P_1, then we get the following Gantt chart.



Waiting times for P_1, P_2, P_3 are now 6 ms, 0 ms, 3 ms respectively and average waiting time is, $\frac{6+0+3}{3}$ = 3 ms. So, average waiting time varies with the variation in process CPU-burst times.

Another difficulty with FCFS is, it tends to favour CPU bound processes over I/O bound processes. Consider that there is a collection of processes one of which mostly uses CPU and a number of processes which uses I/O devices.

When a CPU-bound process is running, all the I/O bound processes must wait, which causes the I/O devices to be idle. After finishing its CPU operation, the CPU bound process moves to an I/O device. Now, all the I/O bound processes, very short CPU bursts execute quickly and move back to I/O queues and causes the CPU to sit idle. In this way FCFS may result in inefficient use of both processor and I/O devices.

Once the CPU has been allocated to a process, it will not release the CPU until it is terminated or switched to the waiting state. So, this algorithm is non-preemptive. It is difficult to implement for time-sharing systems in which each user gets the CPU on a time based sharing.

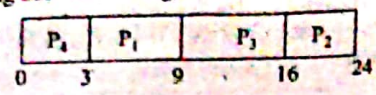
(ii) Shortest Job First (SJF) Scheduling

This algorithm schedules the processes by their CPU burst times which means the process with less CPU burst time will be processed first, before other processes. If two processes have same burst times then they will be scheduled by using FCFS scheduling. This is also called as "shortest next CPU burst".

Consider the following example,

Process	Burst Time (ms)
P_1	6
P_2	8
P_3	7
P_4	3

Using SJF scheduling, the following result is obtained.



Waiting time for process $P_1 = 3$ ms

Waiting time for process $P_2 = 16$ ms

Waiting time for process $P_3 = 9$ ms

Waiting time for process $P_4 = 0$ ms

So, average waiting time $= \frac{3+16+9+0}{4} = 7$ ms

This algorithm gives the minimum average waiting time by moving a short process before a long one which decreases the waiting time of the short process. CPU needs to know the length of requested process which is difficult to compute. This cannot be implemented at the level of short-term CPU scheduling and is used frequently in long-term scheduling.

One way to overcome this difficulty is to predict instead of knowing the length of requested process. This can be done by computing exponential average of length of previous CPU burst value associated with the process. It can be computed using the formula,

$$T_{n+1} = \alpha t_n + (1-\alpha)T_n$$

Where,

t_n = Most recent information related to CPU burst

T_n = Past history of CPU bursts

α = Parameter that is used to have control over weight of recent and past information.

The SJF algorithm can be considered as preemptive and non-preemptive. In a preemptive SJF, when a process is in running state and a new process arrives whose CPU burst time is shorter than the active process, then it preempt the active process. It is also called shortest - remaining time first scheduling.

In a non-preemptive SJF, the process is allocating with CPU till the completion of the process. It is also called as shortest Path Next (SPN) algorithm.

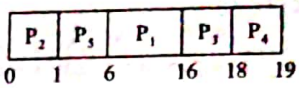
(iii) Priority Scheduling Algorithm

This algorithm associates each process with a priority, and the process with highest priority will get the CPU first. If there are two processes with same priority, FCFS scheduling is used to break the tie. Priorities are of generally some fixed range of numbers, such as 0 to 7 or 0 to 4064. Here 0 is alloted to the process with lowest CPU burst which is highest interms of priority.

Depending on the system we are using the high priority number which can be either lowest number or highest number. Considering the numbers represent high priority. For example, consider set of processes, arrived at time 0, in sequence $P_1, P_2, \ldots P_5$ and with the burst time and priority as follows,

Process	Burst time (ms)	Priority
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2

Using priority scheduling, the following Gantt chart is obtained.



Waiting time for process $P_1 = 6$

Waiting time for process $P_2 = 0$

Waiting time for process $P_3 = 16$

Waiting time for process $P_4 = 18$

Waiting time for process $P_5 = 1$

The average waiting time

$$= \frac{6+0+16+18+1}{5} = \frac{41}{5} = 8.2 \text{ms}$$

The allotment of priorities can be carried out internally as well as externally. In an internally defined priority priorities are alloted based on the computation of certain measurable quantities such as CPU burst time, time limits etc. In an externally defined priority, these are assigned based on certain external factor associated with the process. An example of such allotment is assigning the priorities based on the importance of the process.

Similar to SJF, priority scheduling can also be used to preemptive and non-preemptive. A major drawback associated wit this algorithm is indefinite blocking which is also called starvation. In this case a process with lowest priority will never get the CPU because it keeps on alloting the higher priority to other processes. To avoid this, a technique called 'Aging' is employed which increases the priority of processes that are present in the ready queue for a long time.

(iv) Round Robin Scheduling Algorithm

This algorithm is considered as a preemptive version of FCFS algorithm which is especially designed to be used in time sharing systems. Preemption i.e., switching between various processes is carried out by creating certain time intervals called time slices (or) time quantums whose typical value lies between 10 and 100 ms. Based on these time slices, CPU is alloted to the processes present in the ready queue making the ready queue act as a circular queue. These processes uses the CPU for 1 quantum of time and then the CPU is alloted to the next process.

In a ready queue of RR scheduling new processes are added based on FIFO queue. Starting from the head of the ready queue. Each process is alloted with certain time interval (time slice) and dispatched.

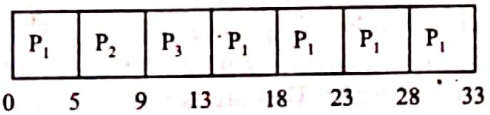
During the allocating of CPU to the process, either of the two situations can arise

1. The process completes within the time slice and the scheduler simply allocates the CPU to the next process present in a queue.
2. The process does not completes its execution within the time slice. In this case an interrupt in made and the process is jumped to the tail of the ready queue. Now, the CPU is allocated to the preceding process.

The average waiting time associated with each process of RR scheduling algorithm is long. To know why, consider a set of processes whose CPU burst times are as follows,

Process	CPU Burst Time (ms)
P_1	25
P_2	4
P_3	4

Let us assume the time quantum as 5 ms. In this case, P_1 is first allocated with CPU for 5 ms and then it is sent at the tail of the queue. P_1 requires another 20 seconds to complete its execution. Now, the CPU is allocated to P_2 which returns it in 4 ms because it needed only 4 ms to complete and hence it quits before the expiration of time slice. Now, the CPU is allocated to P_3 which also requires only 4 ms and hence it also quits before expiration. Now the Gantt chart will be as follows,



The process P_1 requires 4 time quantums to complete its execution. The waiting time associated with the above set of processes can be computed as, $\frac{8+5+9}{3} = \frac{22}{3} = 7.33$ mm

Where,

8 ms is the waiting time of P1(13 – 8 = 8)

5 sm is the waiting time of P_2

9 ms is the waiting time of P_3

None of the processes allowed to use the CPU for more than one quantum. For this reason, this algorithm is dependent on the size of time slice. It acts as a typical FCFS algorithm in case of large time slices whereas, in case of too small time slices, the algorithm is referred as processor sharing with the speed of $\left(\frac{1}{n}\right)^{th}$ value of real processors speed.

Q29. Consider the following four processes, with the length of CPU burst given in milliseconds.

Process	AT	BT
P_1	0	5
P_2	1	6
P_3	2	2
P_4	3	8

Calculate the average waiting time for,

(i) Non preemptive SJF scheduling

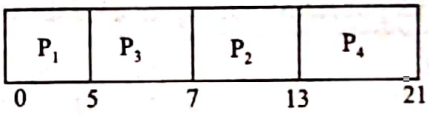
(ii) Average turnaround time.

Answer :

Given that,

Process	AT	BT
P_1	0	5
P_2	1	6
P_3	2	2
P_4	3	8

(i) Average Waiting Time for Non-Preemptive SJF Schedulinh



$$\text{Average waiting time} = \frac{\text{Total waiting time}}{\text{Number of Processes}}$$

$$= \frac{(0-0)+(7-1)+(5-2)+(13-3)}{4}$$

$$= \frac{0+6+3+10}{4} = \frac{19}{4}$$

$$= 4.75 \text{ milliseconds}$$

(ii) Average Turn Around Time

$$\text{Average turn around time} = \frac{\text{Total turn around time}}{\text{Number of Processes}}$$

$$= \frac{(5-0)+(13-1)+(7-2)+(21-3)}{4}$$

$$= \frac{5+12+5+18}{4} = \frac{40}{4}$$

$$= 10 \text{ milliseconds.}$$

Q30. Consider the following four processes, with the length of the CPU burst given in milliseconds.

Process	Arrival time	Burst time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

Calculate the average waiting time for,

(i) Preemptive SJF schedule

(ii) Non-preemptive SJF schedule.

Answer :

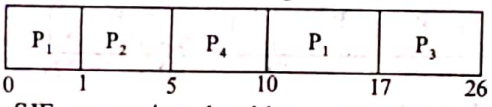
Given that,

Process	Arrival time	Burst time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

(i) Preemptive SJF Schedule

A preemptive SJF algorithm will preempt the currently executing process, if the next CPU burst of the newly arrived process is shorter than the currently executing process.

Grantt chart for preemptive SJF algorithm is shown below,



In SJF preemptive algorithm, process P_1 starts at time 0. As soon as P_2 arrives with a shorter burst time i.e., 4, P_1 is preempted and P_2 is executed. Because the remaining time for P_1 is 7 milliseconds where as for P_2 it is 4 millionseconds.

The average waiting time is

$P_1 = (10 - 1) = 9$ m sec

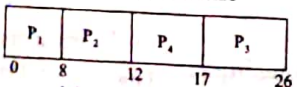
$P_2 = (1 - 1) = 0$ m sec

$P_3 = (17 - 2) = 15$ m sec

$P_4 = (5 - 3) = 2$ m sec

Average waiting time is $= \frac{26}{4} = 6.5$ millisecond

(ii) Non-preemptive SJF Schedule



The average waiting time is,

$P_1 = (0 - 0) = 0$ m sec

$P_2 = (8 - 1) = 7$ m sec

$P_3 = (17 - 2) = 15$ m sec

$P_4 = (12 - 3) = 9$ m sec

Average waiting time $= \frac{0 + 7 + 15 + 9}{4} = 7.75$ millisecond.

2.3.4 Multiprocessor Scheduling

Q31. Explain in detail about scheduling in a multiprocessor system.

Answer : Model Paper-I, Q12(b)

Multiprocessor Scheduling

Scheduling becomes more complex if there are multiple CPUs. The following are various approaches for scheduling processes on multiple homogeneous CPUs i.e., CPUs with similar or identical functionality.

1. Approaches

There are two approaches to implement multiplepr-ocessor scheduling,

❖ **Asymmetric Multiprocessing**

Here a master processor takes scheduling decisions and assigns activities to various (slave) processors.

❖ **Symmetric Multiprocessing (SMP)**

Here each processor is independent and takes scheduling decision by itself. In general, there is a common ready queue where all processes are present, each CPU takes processes from this queue and executes independent of each other. However, care must be taken to ensure that no two CPUs or processors should choose the same process for execution.

2. Processor Affinity

When a process is executed on a particular processor, then its recently accessed data is stored in its cache which it can refer in future. Suppose this process migrates to another processor, then its cache contents will get invalidated.

In this case we need to repopulate the cache in new processor which may be an expensive job. Hence, SMP systems avoid migration of processes and this is known as processor affinity. It has two forms. They are as follows,

❖ **Soft Affinity**

When operating system has a policy of avoiding migration of processes, but it does not give guarantee which means migration is possible here.

❖ **Hard Affinity**

It guarantees that processes do not migrate. Linux provides certain system calls which allows a user to specify that a process has processor affinity and hence should not be migrated.

3. Load Balancing

Load balancing is an approach to divide the workload of the system evenly among the various processors available, so as to utilize them to their maximum extent. It is not of much use when a common ready queue is used. It is useful when each processor maintains a separate ready queue. When the processor is idle the processes are migrated to it to balance the system workload. It is of two types,

- **Push Migration**

  Here a special method checks the load on each processor, if it is not balanced, some processes are taken from an overloaded processor and pushes to a less-overloaded processor to balance the workload.

- **Pull Migration**

  Here the idle or less-overloaded processors themself take processes from an overloaded processor to balance the workload.

The Linux operating system uses both the migration schemes. It executes push migration every 200 milliseconds and pull migration whenever ready queue of a particular processor is empty.

Multiprocessor

In SMP systems, multiple physical processors are used to execute threads concurrently whereas, multicore processor uses multiple logical processors to execute multiple threads concurrently.

Each logical processor on a physical processor consists of its own general purpose registers, other special purpose registers and is capable of handling interrupts on their own. However, each of them shares certain resources of their physical processor like buses, cache etc., The following figure below shows two logical CPUs in a single physical CPU.

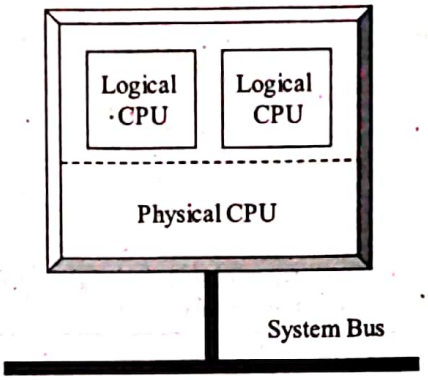


Figure: Multi-threading Processor

In this way, each logical processor acts as an individual physical processor to the operating systems. The major advantage of using multicore processors in SMP system is it consumeless power with faster processing speed. A modern approach to multicore processors is the use of multithreaded multicore processor which is used to overcome the situation of memory stall. It is nothing but the waiting time of a processor while accessing memory for the data to become available.

The scheduling of such processors is carried out in two levels.

Level 1

At this level, decisions are made related to the allotment of software threads to various logical processors.

Level 2

At this level, decisions are made related to the sequence of execution of these logical processors.

PROCESS SYNCHRONIZATION DEADLOCKS

PART-A

SHORT QUESTIONS WITH ANSWERS

Q1. Define pipes.

Model Paper-I, Q5

Answer :

Pipes

Unix system uses pipes, to establish inter process communication. A pipe provides an unidirectional flow of data. A pipe is created using the **pipe( )** function.

Syntax

```
#include<unistd.h>
int pipe(int fd);
```

A *pipe* ( )function returns two file descriptors, fd[0] and fd[1]. The fd[0] is open for reading from the pipe and fd[1] is open for writing to the pipe. The data flows from one end of the pipe to the other end.

Q2. What is critical resource?

Answer :

Model Paper-III, Q6

A resource that cannot be shared between two or more processes at the same time is called a critical resource. There may be a situation where more than one process requires to access the critical resource. Then, during the execution of these processes they can send data to the critical resource, receive data from the critical resource or they can just get the information about the status of the critical resource by sending related commands to it. An example of a critical or a non-sharable resource is 'printer'. A critical resource can be accessed only from the critical section of a program.

Q3. Define semaphore.

Answer :

Model Paper-II, Q5

Signals provide simple means of cooperation between two or more processes in such a way that a process can be forcefully stopped at some specified point till it receives the signal. For signalling between the processes a special variable called semaphore (or counting semaphore) is used. For a semaphore 'S', a process can execute two primitives as follows,

(i) semSignal(S)

This semaphore primitive is used to transmit a signal through semaphore 'S'.

(ii) semWait(S)

This semaphore or counting semaphore primitive is used to receive a signal using semaphore 'S'. If the corresponding transmit signal has not yet been sent then the process is suspended till a signal is received.

Q4. Define monitor.

Answer :

Model Paper-III, Q5

A monitor is a construct in a programming language which consists of procedures, variables and data structures. Any process can call the monitor procedures but access to the internal data structures is restricted. At any time, a monitor contains only one active procedure.

The condition variables can be used for blocking and non-blocking.

or

A monitor refers to the software module which consists of one or more procedures, an initialization sequence and local data.

Q5. Compare semaphore and monitor.

Answer : Model Paper-I, Q6

	Semaphore		Monitor
1.	Semaphores can be used anywhere within the program but can't be used inside a monitor.	1.	Monitor makes use of condition-variables anywhere inside it.
2.	The caller is not always be in a blocked state when *wait*( ) condition is executed.	2.	The *wait*( ) function always blocks the caller.
3.	*signal*( ) releases the blocked thread, if it exists, or increases the semaphore count value.	3.	*signal*( ) releases the blocked thread, if it exists, or is lost as if it never occurs.
4.	Upon releasing a blocked thread by the *signal*( ) function, the caller and the released thread can continue with their executions.	4.	Upon releasing a blocked thread by the *signal*( ) function, only one among the caller or the released thread can continue, but not both.

Q6. Discuss the prevention of no preemption with an example.

Answer :

Prevention of No Preemption

If a system follows no preemption, then the resources which are once allocated are not taken back from a process involuntarily. Any system with above behaviour can lead to hold and wait condition and thereby to deadlock. Hence, no preemption condition should not be applied in order to prevent deadlock.

If a process P_1 request any resource R_1 which is currently held by some other process P_2 then, P_2 is preempted and R_1 is given to P_1.

Example

Consider a car occupying some part of the street which cannot be taken by other car until and unless the first car has been moved. This situation is known as No preemption. An occurrence of this condition can be prevented only if the first car has been moved forcibly giving the other car a chance to place itself and finish its task.

Q7. What are the two protocols that are used to prevent hold and wait?

Answer :

The hold and wait condition precludes a process from holding some resources while requesting others.

There are two protocols to prevent hold and wait.

(a) Request all the resources before starting an execution. Any program must request all the resources before starting the execution, acquire them, use them and release them once execution is completed. In this way, no process will wait for the resources during execution. Thus, hold and wait is prevented.

(b) Any process will request for a new resources only when it has none i.e., if a process has two resources and requires three more resources, then the process can request for these three resources after releasing two resources it is holding. After releasing the resources, it will request for new resources. If they are busy, the process waits without holding any resources. Suppose, if the requested resources are free, then they can be allocated to the process.

Q8. Write a brief note on deadlock detection in a system containing a single instance of each resource type.

Answer :

Deadlock detection in a system containing a single instance of each resource type is given below,

A deadlock detection algorithm that makes use of a variant of the resource allocation graph, called the wait-for graph is defined for a system containing only a single instance for all the resources.

An edge from a node P_i to node P_j exists in a wait-for graph, if and only if, the corresponding RAG contains two edges, one from node P_i to some resource node R_a and the other from the resource R_a to node P_j. The presence of a cycle in the wait-for graph indicates the existence of a deadlock.

An algorithm that is used to detect a cycle in the graph requires a total of n^2 operations, where n is the number of vertices in the graph.

Q9. What is deadlock avoidance?

Model Paper-II, Q6

Answer :

Deadlock avoidance does not impose any rules but, here each resource request is carefully analyzed to see whether it could be safely fulfilled without causing deadlock. The drawback of this scheme is its requirement of information in advance about how resources are to be requested. Different algorithms require different type and amount of information like some require maximum number of resources that each process requires etc.

Q10. What are the difficulties that may arise when a process is rolled back as a result of a deadlock?

Answer :

Deadlock refers to a situation in which one process is waiting for a resource which is currently under the control of some other process. Hence, it results in the permanent blocking of the processes. This situation can be avoided by rolling back some of the processes.

However, this may lead to a starvation issue wherein a runnable processor is made to wait indefinitely although it is capable of executing the process effectively.

PART-B

ESSAY QUESTIONS WITH ANSWERS

3.1 PROCESS SYNCHRONIZATION

3.1.1 Inter-Process Communication: Critical Section, Race Conditions, Mutual Exclusion, Peterson's Solution

Q11. What is process synchronization and discuss about race condition.

Answer :

Process Synchronization

There are several situations, where different processes need to interact with each other to achieve a common goal. The interaction can be done either by sharing data or by sending messages. When data is shared by several independent processes then there is a chance of data becoming inconsistent. For example, let us consider two processes P_1 and P_2, both share a variable named 'counter'. If both P_1 and P_2 execute simultaneously, with P_1 incrementing the counter while P_2 decrementing it. Then at the end, the result of this variable will be such that, it is not expected by either P_1 or P_2 because it became inconsistent. This is often called race condition. Hence, a synchronization mechanism is needed to avoid inconsistency among several processes.

Basically process synchronization is implemented by making a process to wait until another process performs an appropriate action on shared data. It is also called signalling where one process waits for notification of an event that will occur in another process.

Race Condition

A race condition refers to the situation that results when many processes or threads reads and writes data items in a way that the final result generated is in accordance with the order of instructions execution in multiple processes.

Example

Let P1 and P2 be the two processes that shares a global variable 'x'. During the execution, if at some point process P1 updates the value of 'x' to 5 and at some other point updates it to 10. Thus, a race among the two processes starts for changing the value of 'x' and a process that performs an update operation last determines the final value of 'x'.

Consider another situation in which two processes P3 and P4 shares two global-variables 'y' and 'z' whose initial values are 5 and 10 respectively. During some point in execution, if process P3 executes an assignment statement

$y = y + z$ and P4 executes $z = y + z$.

The final values of 'y' and 'z' depends on the order in which the two assignment statements are executed. If process P3 is executed prior to P4, then the final values of 'y' and 'z' are 15 and 25 respectively, on the other hand, if execution of P4 precedes P3, then the resultant values of 'y' and 'z' are 20 and 15 respectively.

Q12. How are pipes created? Explain the IPC between related processes using ordinary pipes.

Answer : Model Paper-II, Q13(a)

Pipes

Unix system uses pipes, to establish inter process communication. A pipe provides an unidirectional flow of data. A pipe is created using the **pipe( )** function.

Syntax

```
#include<unistd.h>

int pipe(int fd);
```

A *pipe* ( )function returns two file descriptors, fd[0] and fd[1]. The fd[0] is open for reading from the pipe and fd[1] is open for writing to the pipe. The data flows from one end of the pipe to the other end.

The *pipe* function returns '0' on success or –1 on error.

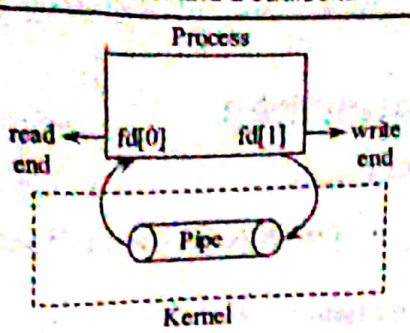


Figure (1): Pipe within a Single Process

Using the pipes within a single process does not make any sense, since the same process reads the data from the pipe that is written by itself.

Pipe between Two Processes

The application of pipes, is to exchange the data between two different processes. First, a process creates a pipe, then calls a *fork* ( ) function to create a child process. This creates an IPC channel, between the parent and the child process. Each process gets two file descriptors fd[0] and fd[1], for reading and writing respectively. This is shown in the below figure (2),

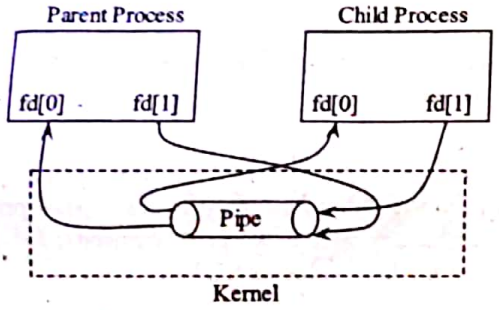


Figure (2): Pipe after a Fork of Child Process

After a fork, the data can flow in either direction from parent to child or from child to parent.

To provide a one-way flow of data from parent to child, the parent process closes the read end (i.e., fd[0]) of the pipe and the child process closes the write end (i.e., fd[1]) of the pipe. Similarly, for one-way flow of data from child to parent, the child closes the read end of the pipe and the parent closes the write end of the pipe. This is shown in the below figures (3) and (4),

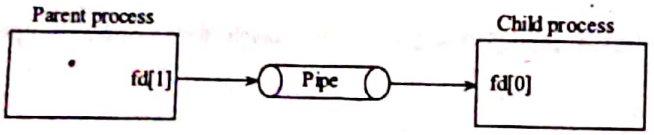


Figure (3): Pipe from Parent to Child

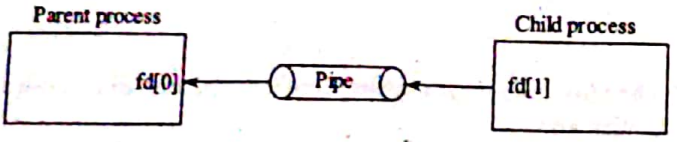


Figure (4): Pipe from Child to Parent

The following rules are applied, when one end of a pipe is closed.

(i) Reading from a pipe with its write and closed returns a value 0 if all the data has been read and end-of-file is reached.

(ii) Writing to a pipe with its read end closed, generates the SIGPIPE signal.

If either a signal is ignored or caught and the signal handler returns, the *write* ( ) function returns an *errno* to EPIPE.

The constant PIPE_BUF specifies the buffer size of the kernel's pipe. If there are multiple writes to the same pipe and data is less than this PIPE_BUF size, then data will not be interleaved with data from the other writers. Otherwise, the data is larger than PIP_BUF size the data might be interleaved.

Example

The following program shows how to create a pipe from the parent to the child and use it to write then read the data.

```
#include<unistd.h>
int main( )
{
        int n, fd[2];
        pid_t pid;
        char buff[100];
        char msg[100] ="Hello World";
            /*Create a pipe */
        if(pipe(fd) = = –1)
            printf("can't create the pipe");
            /*Create a child process*/
        if((pid=fork( )) = = –1)
            printf("Can't create a child");
    else if(pid > 0) /*Parent process*/
    {
      close(fd[0]); /*parent close read end of pipe*/
        /*Parent writes to the pipe*/
        write(fd[1], msg, sizeof(msg));
    }
    else            /*Child process*/
    {
      close(fd[1]); /*Child closes write end of pipe*/
                /*Child reads from the pipe*/
          n=read(fd[0], buff, size of (buff));
              /*Display on the standard output*/
        write(STDOUT_FILENO, buff, n);
    }
    exit(0);
}
```

In the above program, a pipe is created using the *pipe* function. The process then creates a child process using *fork* function.

The parent process closes its read end (i.e., *fd*[0]), before it writes to the pipe. Then, the child process closes its write end (i.e., *fd*[1]), before it reads from the pipe. Thus the parent and child, exchange data between themselves. Finally, the child displays the message "Hello world" onto the standard output.

Q13. Explain in detail about inter process communication using named pipes.

Answer :

FIFOs are also called named pipes. They provide unidirectional (half-duplex) flow of data. In FIFOs, data is read in first in first out order, i.e., it is read in the order in which it was written. The difference between pipes and FIFO is that, pipes do not have pathnames whereas, FIFOs have pathname so that unrelated processes can read or write data to it. A FIFO is created, using either mknod system call or mknod command.

Syntax

int mknod (char * pathname, int mode, int dev);

The first argument "pathname", specifies the path of a FIFO file being created. The second argument 'mode', specifies the access permission. The last argument "dev", is not considered for FIFOs.

The mknod command can be used as follows,

/etc/mknod name P.

After a FIFO is created, it must be opened for read or write operation using open system call. It can also be opened using the standard I/O functions, fopen or freopen.

The O_NDELAY flag for a FIFO, can be set in following two ways,

1. The O_NDELAY flag can be set, when open system call is invoked.
2. If file is already opened, then fcntl is called to enable the O_NDELAY flag.

The rules for reading from or writing to a pipe or FIFO are as follows,

(i) If a process requests to read data that is less than is currently available in the pipe or FIFO, then only the requested amount of data is returned.

(ii) If a process requests to read data that is more than that in the pip or FIFO, then only the available amount of data is returned.

(iii) If a process reads empty pipe or empty FIFO with no writer process connected to it, then a value of zero is returned, which indicates the end of file.

3. If the number of bytes to write is less than pipe's capacity, then write operation to a pipe or FIFO is said to be atomic i.e., if two processes, say P_1 and P_2 try to write data to a pipe at the same time, then data of process P_1 is written followed by the data of process P_2 or data of process P_2 is written, followed by the data of process P_1. The data of two processes is not mixed.
4. If the number of bytes to write exceed pipe's capacity then write operation to a pipe or FIFO is not atomic
5. If a process writes data to a pipe or FIFO with no reader process connected to it, then the SIGPIPE signal is generated. A process does not catch the notification of SIGPIPE signal, then default action of terminating the process is performed. If a process either ignores the SIGPIPE signal or if it catches the signal and returns from its signal handler, then write operation returns a value of zero with errno set to EPIPE.

Q14. Explain in detail how concurrent processes come into conflict with each other when they are competing for the use of the same resource.

Answer :

Process Competition for Resources

Conflicts arises among the various concurrently executing processes when they are competing for the same resource.

Let us consider a situation in which two or more processes want to access a resource. Each of these concurrently executing processes is unaware of the presence of other processes and the execution of one process doesn't cause any affect on the execution of the other process. Hence, the state of the resources used, remains unaffected.

For instance consider that information is not exchanged between these processes and the execution of one process causes a significant affect on the behavior of the other competing processes i.e., if two processes want to access a single resource then the OS grants the resource access to only one process and let the other process to wait, as a result of which the blocked process may never get the resource and terminates in an inappropriate manner.

Three problems dominates in case of competing processes

(i) The need for mutual exclusion

(ii) The occurrence of deadlock and

(iii) The problems of starvation.

(i) Need for Mutual Exclusion

Consider a situation in which two or more processes need access to a single non-sharable resource (Example: printer). During the execution process, each process sends the commands to I/O devices or sends and receives data or receives status information etc. Such an I/O device is said to be a critical resource and the portion of a program that uses it is called a critical section. An important point to be considered here is that only one program is permitted to enter into the critical section at any time.

(ii) Occurrence of Deadlocks

The major cause for the occurrence of a deadlock is the imposition of the mutual exclusion.

Consider an example, granting of two resources R1 and R2 to two processes P1 and P2. Further suppose that each of these processes want to access both the resources in order to execute some function. A situation may occur in which an operating system assigns the resource R1 to process P2 and resource R2 to process P1. Hence, each process is waiting for one of the resources and will not release the acquired resource till it gets the other resource i.e., a process needs both these resources in order to proceed. This leads to deadlock.

(iii) The Problem of Starvation

Two or more processes are said to be in starvation, if they are waiting perpetually for a resource which is occupied by another process. The process that has occupied the resource may or may not present in the list of processes that are starved.

Let P1, P2 and P3 be the three processes, each of which requires a periodical access to resource R. If access to resource 'R' is granted to the process P1, then the other two processes P2 and P3 are delayed as they are waiting for the resource 'R'. Now, let the access is granted to P3 and if P1 again needs 'R' prior to the completion of its critical section. If the OS permits P1 to use 'R' after P3 has completed its execution, then these alternate access permissions provided to P1 and P3 causes P2 to be blocked.

Here, we need to illustrate whether starvation is possible or not in algorithms like FCFS, SPN, SRT and priority.

Consider FCFS (First Come First Served) algorithm, in this starvation is not possible. The reason is CPU picks the process according to arrival of its burst time and runs the process till its completion.

Now, consider SPN (Shortest Processing Next) algorithm, in this starvation is possible with the processes that has long burst time. The reason is CPU picks the process that has shortest next burst time. Here, we can overcome starvation problem by using preemptive SPN algorithm, which prompts the currently running process.

Next, SRT (Shortest Remaining Time) algorithm, in this starvation is possible with the processes that has shortest remaining time. The reason is CPU picks the process that has shortest remaining time. Here, we can overcome the problem of starvation by giving chance to processes that are waiting for a long period of time. Finally, consider priority algorithm, in this starvation is possible with low priority processes. The reason is, CPU picks the process with highest priority.

We can overcome starvation problem by a technique called aging. This technique increases the priority of the processes that waiting for long period of time.

Q15. Write about the following.

(a) Critical resource

(b) Critical section.

Answer :

(a) Critical Resource

A resource that cannot be shared between two or more processes at the same time is called a critical resource. There may be a situation where more than one process requires to access the critical resource. Then, during the execution of these processes they can send data to the critical resource, receive data from the critical resource or they can just get the information about the status of the critical resource by sending related commands to it. An example of a critical or a non-sharable resource is 'printer'. A critical resource can be accessed only from the critical section of a program.

(b) Critical Section

A *critical section* is a segment of code present in a process in which the process may be modifying or accessing common variables or shared data items. The most important thing that a system should control is that, when one process is executing its critical section, it should not allow other processes to execute in its critical sections.

Before executing critical section the process should get permission to enter its critical section from the system. This is called an *entry section*. After that process executes its critical section and comes out of it this is called *exit section*. Then, it executes the remaining code called *remainder section*.

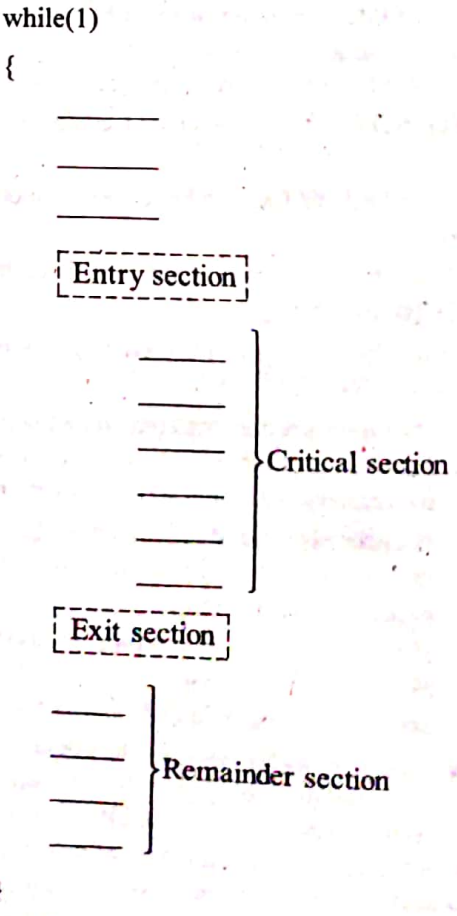


Figure: Structure of a Process

Q16. What are the three requirements that a solution to the critical section problem must satisfy?

Answer :

The following are the important properties that should be satisfied by critical section implementation or solution,

1. **Mutual Exclusion**

When a process P1 is in its critical section, then no other process can be executing in their critical section.

2. **Progress**

When a critical section is not in use and other processes is requesting for it, then it should be granted to only that process which is not executing in its remainder section enter its own critical section.

3. **Bounded Waiting**

A limit or bound is fixed for each process to enter critical section beyond which it is not allowed to enter in critical section.

There are two general ways for handling critical sections in operating systems. They are,

(i) **Preemptive Kernel**

It allows a Kernel mode process to be preempted (i.e., interrupted) during execution.

(ii) **Non-preemptive Kernel**

It doesn't allow a Kernel mode process to be preempted during execution, the process will execute until it exits Kernel mode or voluntarily leaves control of the CPU. This approach is helpful in avoiding race conditions.

The preemptive Kernel is used in real-time system, where a process executing in Kernel mode can also be preempted which makes the Kernel more responsive. Windows XP, Windows 2000 and traditional Unix are non-preemptive Kernels whereas Linux of Kernel version 2.6 is preemptive Kernel.

Q17. Explain preemptive kernels and non-preemptive kernels. Also explain why would any one favour a preemptive kernel over a non-preemptive one.

Answer :

Preemptive Kernel

A Kernel that permits a process to be preempted or interrupted during its execution is called preemptive kernel. In preemptive kernel, every task is designed as an independent entity that has total control over the CPU. However, the task that is ready to run and has the highest priority is executed first by the kernel.

The figure below depicts the flow of a preemptive kernel with three tasks *A*, *B*, and *C*.

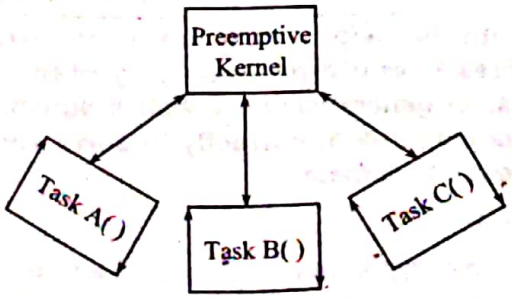


Figure: Preemptive Kernels Program Flow

Furthermore, in any process execution, a task can be in either of the following three states,

1. Running and waiting
2. Waiting
3. Idle.

1. **Running and Waiting**

A task will be in a running, waiting state when it is not ready to run.

2. **Waiting**

A task will be in a waiting state when it is ready to run but cannot do so due to the execution of a higher priority task.

3. **Idle**

A task is considered to be idle when no process has a task that is ready to be executed. This task is a special purpose entity which has the lowest priority and is usually incorporated in all Kernel programs.

Operation of Preemptive Kernel

The following figure shows the program context for a preemptive Kernel.

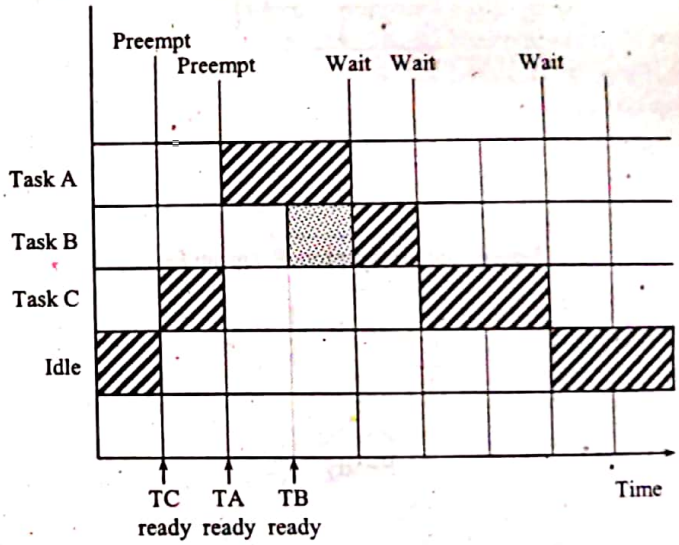


Figure: Preemptive Kernel Program Context

Where,

Denotes 'Running'.

Denotes 'Ready waiting'.

Denotes 'Not ready waiting'.

Here, there are three tasks *A*, *B* and *C* with priorities $A > B > C$. Hence, when task *C* is ready to run, the Kernel interrupts its idle task and begins the execution of task *C*. And, when task *A* is ready to run, the Kernel interrupts task *C* and starts executing task *A*. However, when task *B* is ready to run, the Kernel does not halt or preempt task *A* due to its' higher priority.

A this stage all the three tasks are in ready state, but *B* and *C* wait for *A* to finish. When this is done *A* goes into a waiting state until it is invoked again. The control is then transferred to *B* as it hold as higher priority than *C*. Therefore, when *B* finishes the control is transferred to *C* to complete it's execution.

Advantage and Disadvantage

The major advantage with preemptive kernel is that it allows the task to be designed independently. However, this makes it very complex and consumes more memory resources.

Non-preemptive Kernel

A Kernel that does not permit a process to be preempted or interrupted during its execution is called non-preemptive kernel. In a non-preemptive kernel, every task has to assist another task in its completion by giving up the CPU in an appropriate manner.

The following figure depicts a non-preemptive kernel along with its three cooperative tasks *A*, *B* and *C*.

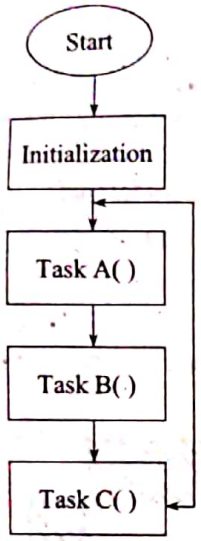


Figure: Non-preemptive Program Flow

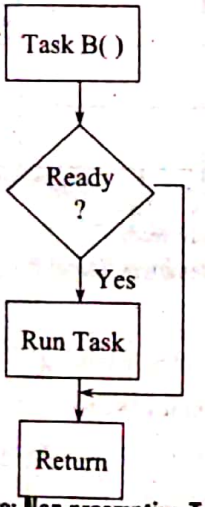


Figure: Non-preemptive Task

Here, the non-preemptive kernel acts as a periodic scheduler which serially executes every task. However, every task must assist each other by running just once and then returning to the scheduler loop. This is because, if any task gets implemented as an endless loop, then the scheduler will never get to other tasks.

Operation of Non-preemptive Kernel

The figure below shows the program context of a non-preemptive kernel.

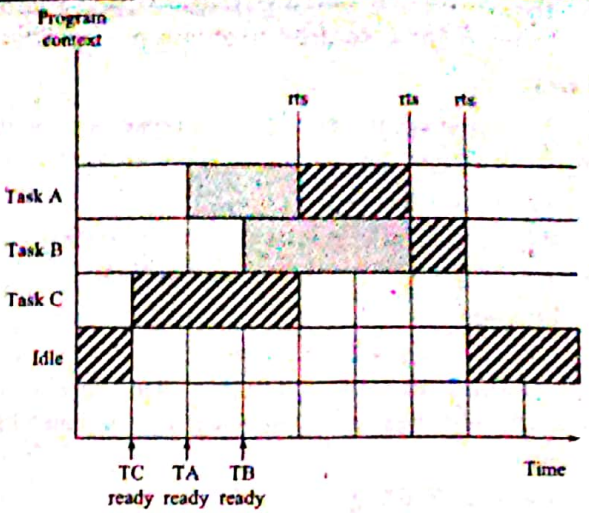


Figure: Non-preemptive Kernel Program Context

Where,

= 'Ready waiting'.

= 'Running'.

= 'Not ready waiting'.

Here, the task with the highest-priority waits for the completion of lowest-priority task followed by a normal execution (according to priority). Therefore, control to task *A* is passed only after the completion task *C* followed by task *B*.

However, this scheme is adopted by only a few schedulers. This is because, a round-robin method is widely used in non-preemptive schedulers which produces the same response times for all the tasks.

Advantage and Disadvantage

Non-preemptive Kernels are very easy to be designed and consume less memory resources. However, they have a relatively slower response time for higher priority tasks and are complex to be written.

Favouring a Preemptive Kernel over a Non-preemptive One

The reasons for favouring a preemptive kernel over a non-preemptive one are,

(a) Preemptive kernel can permit a real-time process to interrupt a process running in the Kernel. Hence, it is much appropriate for real-time programming.

(b) Preemptive kernel does not allow a kernel-mode process to run for a longer period of time without assigning a process to the processor for execution. Hence, it has a higher response time than non-preemptive Kernel.

Q18. With the help of neat examples, explain any three types of errors (using signal and wait) that can be generated easily when programmers use semaphores in correctly to solve the critical-section problem.

Answer :

The three types of error (using signal and wait) that can be generated easily when programmers use semaphores incorrectly to solve the critical section problem are as follows,

1. Error Generated by Interchanging the Execution Order

This type of error occurs whenever the order of executing wait and signal operations (defined on semaphore mutex) are being interchanged by process.

Example

Consider the following execution code, wherein the signal operation is being executed prior to wait operation.

```
Signal (mutex)
    ⋮
Critical section
    ⋮
wait (mutex)
```

The execution of above code results in simultaneous execution of multiple processes in their critical section , there by violating the mutual exclusion requirement.

2. Error Generated by Interchanging the Execution Order

This type of error occurs when the signal operation defined on semaphore mutex is replaced by wait operation.

Example: Consider the following code

```
Wait (mutex)
⋮
Critical section
⋮
Signal (mutex)
```

Now, if the signal (mutex) is replaced by wait (mutex) then, the code becomes,

```
Wait (mutex)
⋮
Critical section
⋮
Wait (mutex)
```

The execution of above code results in occurrence of deadlock.

3. Error Generated by Omitting the Execution of Operation

This type of error occurs when execution of signal (mutex) or wait (mutex) or both are omitted by the process.

Example

Consider the following execution codes wherein the signal operation is omitted or wait operation is omitted or both (i.e., signal and wait operation) is omitted .

(i)
```
Wait (mutex)
⋮
Critical section
⋮
//omitted signal (mutex)
```

(ii)
```
//omitted wait (mutex)
⋮
Critical section
⋮
Signal (mutex)
```

(iii)
```
//omitted wait (mutex)
⋮
Critical section
⋮
//omitted signal (mutex)
```

The execution of any of the above code result in violating mutual exclusion requirement or occurrence of deadlock.

Q19. Explain Peterson's solution for critical section problem.

Answer :

Peterson's Solution

Peterson's solution is a software based solution to the critical section problem that satisfies all the requirements like mutual exclusion, progress and bounded waiting. It provides alternate execution of critical sections of two processes named P_0 and P_1. We use the notation P_i for P_0 and P_j for P_1 where $i = 0$ and $j = 1 - i$ (i.e., $1 - 0 = 1$). Here, two data structures are used as follows,

```
int turn;
boolean flag[2];
```

The variable '*turn*' indicates the *turn* of the process, whose turn is to execute its critical section. For example, if turn ==*j*, then process P_j is allowed to enter its critical section and execute. The 'flag' array indicates whether a process is ready to enter its critical section or not. For example, if flag[*j*] is true, then it means process P_j is ready to enter its critical section.

The algorithm of Peterson's solution is as follows.

```
Repeat
    flag[i]: = true                          Entry section
    turn: = j
    while(flag[ i] and turn = j) do no-op

        Critical section

    flag[i]: = false;                        Exit section

        Remainder section

Until false;
```

Figure: Process 'P_i' in Peterson's Solution

Process P_i sets flag[i] as 'true' to indicate that it is ready to enter its critical section. Then it allows P_j to enter its critical section (if it wishes) by assigning the value of variable 'turn' as 'j' (turn:=j).

The while loop waits infinitely doing no-operation (no-op) until the value of '*turn*' is equal to 'j'. That is the value of 'turn' should be equal to 'i', then only it comes out of while loop and enters the critical section of 'i' and after executing that, it exits the critical section by disabling flag (as flag[i] := false). Later it executes the remainder section.

Mutual exclusion is preserved, because P_i can enter its Critical Section (CS) only if either flag[j] == false or turn == i. If both the cases doesn't hold, then control will be blocked in the while loop. Also the value of 'turn' which is a boolean variable can be either 0 or 1 but not both, which implies that either P_0(P_i) or P_1(P_j) will execute CS at a particular time, but not both at the same instance.

Progress and bounded waiting requirements are satisfied by using the condition in the blocking while loop. It consists of two conditions flag[j] = = true and turn = = j. If process P_j is not ready to enter its CS, then the value of flag[j] will be false, then P_i can execute its CS. If it is not the case i.e., P_j is ready, then execution of CS depends on the value of 'turn'. If it is equal to i, then P_i executes or else if it is equal to j, then P_j executes. That is, there is always progress and waiting time is also bounded.

3.1.2 Classical Problems of Synchronization: The Bounded Buffer Problem, Producer/ Consumer Problem, Reader's and Writer, Problem, Dining Philosopher's Problem

Q20. Explain the bounded buffer problem and how semaphores can be used as a solution to this problem.

Answer : **Model Paper-I, Q13(a)**

The Bounded Buffer Problem (or) Producer-Consumer Problem

Let us consider that, there is an array of 'n' buffers, each can hold a single item. There are two processes one is producer, which creates items and places in buffers and the other is consumer which takes one item at a time from the array.

The problem is to give mutually exclusive access to these processes i.e., at a particular instance of time either producer or consumer should be allowed to access the buffer but not both. And also to ensure that producer should not produce, if all buffers are full and consumers should not consume if all buffers are empty.

To implement a solution for the above problem, we will create three semaphores named *mutex*, *empty* and *full*. Mutex is used to employ exclusive access to array of buffers, empty and full semaphores count the number of empty and full buffers respectively. In the beginning, empty is initialized to the value of 'n' (i.e., size of array of buffers) and full is initialized to zero. The code for producer and consumer processes is as shown below,

```
while(1)
{
   /*Produce item A*/

   wait(empty);          <-- Entry section
   wait(mutex);

      /*Add item A to buffer*/   <-- Critical section

   signal(mutex);        <-- Exit section
   signal(full);

}
```

Figure: Producer Process's Pseudocode

```
while(1)
{                                  Entry section
   wait(full);                    <--
   wait(mutex);                                Critical section
                                               <--
      /*Take out item A from buffer*/

   signal(mutex);        <-- Exit section
   signal(empty);                              Remainder section
                                               <--
      /*Use item A or consume*/

}
```

Figure: Pseudocode for Consumer Process

Q21. Explain the state of the process queue for the Readers/Writers problem and get the solution to the same by using message passing.

Answer :

Readers/Writers Problem

The readers/writers problem can be defined as follows,

A data area which can be a file, a main memory block or a bank of processor registers can be shared by multiple processes. All the processes of the system can be categorized into 'readers' and 'writers' based on whether they are reading from the data area or writing to the data area.

The following conditions must be satisfied,

1. Multiple readers can simultaneously read a file.
2. Only one writer can write to a file at any time.
3. No reading of a file must occur when writing operation is in progress.

Generally, in Readers/Writers problem, no reader is allowed to write to the data area and no writer can read the data area.

Readers having Priority

The following code shows one possible solution to the Readers/Writers problem. It consists of a single instance for a reader and a writer. No changes can be made to the solution in case of multiple readers and writers. The writer process is simple and the semaphore *wrsem* is used to impose mutual exclusion condition. The reader process also makes use of *wrsem* semaphore to enforce mutual exclusion. In case if, at least one character is read by a reader, other reader processes must not wait before entering into the data area. A global variable called the *readcount* is maintained that can monitor the number of reader processes and a semaphore '*r*' is used to assure that the variable *readcount* is updated properly.

The program for Readers/Writers problem with readers having priority is as follows,

```
int readcount;
semaphore r=1, wrsem=1;
void READER( )
{
while(true)
{
semWait(r);
readcount = readcount + 1;
if(readcount = = 1)
semWait(wrsem);
semSignal(r);
READING( );
semWait(r);
readcount = readcount – 1;
if(readcount = = 0)
semSignal(wrsem);
semSignal(r);
}
}
void WRITER( )
{
while(true)
{
semWait(wrsem)
WRITING( );
semSignal(wrsem);
}
}
void main( )
{
readcount = 0;
READER( );
WRITER( );
}
```

Writers having Priority

If a single reader is accessing the data area, it enables multiple upcoming readers to retain control of the data area as long as at least one reader is reading the data area. This leads to the starvation of writers who can't access the data because of the presence of readers. This problem can be solved using the following code which assures that no new reader can be permitted to read the data when at least one writer issues a desire to write. Hence, the following variables and semaphores are added to the already defined variables and semaphores.

1. A semaphore *rdsem* is maintained which prevents all the readers from reading when at least one writer issues a desire to access the data area.
2. A variable called the *writecount* that is assigned with a task of controlling the settings of *rdsem*.
3. A semaphore '*w*' is used to assure that the variable *writecount* is updated properly.

An additional semaphore called '*Z*' is maintained, which permits only one reader to queue on *rdsem* while all the other readers queuing on '*Z*' are waiting on *rdsem*.

The program for Readers/Writers problem with writers having priority is as follows,

```
int writecount, readcount;
semaphore r=1, w=1, z=1, wrsem=1, rdsem=1;
void READER( );
{
while(true)
{
semWait(z);
semWait(rdsem);
semWait(r);
readcount = readcount+1;
if(readcount = = 1)
semWait(wrsem);
semSignal(r);
semSignal(rdsem);
semSignal(z);
```

```
READING( );
semWait(r);
readcount = readcount – 1;
if(readcount = = 0)
semSignal(wrsem);
semSignal(r);
}
}
void WRITER( )
{
while(true)
{
semWait(w);
writecount = writecount + 1;
if(writecount = =1)
semWait(rdsem);
semSignal(w);
semWait(wrsem);
WRITING( );
semSignal(wrsem);
semWait(w);
writecount = writecount – 1;
if(writecount = = 0)
semSignal(rdsem);
semSignal(w);
}
}
void main( )
{
readcount = 0;
writecount = 0;
READER( );
WRITER( );
}
```

Q22. Explain the solution for the Dining Philosophers problem using semaphore.

Answer : Model Paper-III, Q13(a)

The Dining Philosophers Problem

Consider the following situation, there are five philosophers who have only two jobs in this world like think and eat. Each philosopher is sitting on one chair laid around a circular table. There is a plate of noodles placed in the center and five single chopsticks are placed on the table as shown in figure.

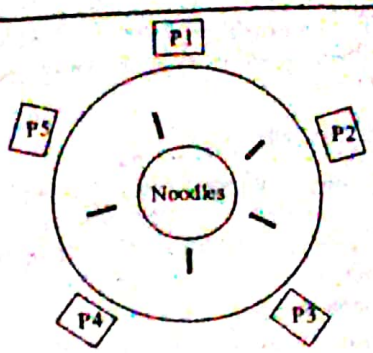


Figure: Dining Philosophers Problem

The philosopher thinks independent of each other and when he get hungry he eats. But, in order to eat, he needs two chopsticks. There is a restriction that philosopher has to take chopsticks of his left and right neighbour only and also he cannot pickup a chopstick which are currently in the hand of his neighbour. If philosopher gets both chopsticks, he eats and after that puts the chopsticks on the table and starts thinking again.

The problem is to ensure that all philosophers peacefully thinks and eats and no philosopher should starve of hunger (i.e., no starvation) and the chopstick should be given mutually exclusive from each other..

For solving this problem using semaphores, each chopstick has to be represented as a semaphore. Hence, we have an array of five chopsticks. To take a particular chopstick the philosopher has to execute a wait( ) operation on that semaphore and while putting down the chopsticks, it executes a signal( ) operation. Consider the following pseudocode for a philosopher '*X*'.

```
semaphore chopstk[5];
while(1)
{
    wait(chopstk[x]);
    wait(chopstk[(x + 1)%5]);
                          critical section
    /*perform eating*/
    signal(chopstk[x]);
    signal(chopstk[(x + 1)%5]);
                          Remainder section
    /*perform thinking*/
}
```

Figure: Structure of Philosopher 'X'

The above solution may lead to deadlock, consider a situation where all philosophers has grabbed their left chopsticks, now every body would try to grab right chopsticks but will be delayed forever. There are several other solutions for diming philosopher's problem which are deadlock free. One of the technique for the above situation is 'Hold-n-wait'. No philosopher should be allowed to hold a chopstick and wait for another. It should grab chopsticks only if both are available. Another solution is to use asymmetric order, in which an even philosopher (*P2* or *P4*) is allowed to pick their right chopstick first then left chopstick. In the same way each odd philosopher (*P1*, *P3*, *P5*) are allowed to take left chopstick first and then right chopstick.

3.1.3 Semaphores, Event Counters

Q23. Discuss in detail about Semaphores.

Answer : **Model Paper-I, Q13(b)**

Semaphore

Signals provide simple means of cooperation between two or more processes in such a way that a process can be forcefully stopped at some specified point till it receives the signal. For signalling between the processes a special variable called semaphore (or counting semaphore) is used. For a semaphore 'S', a process can execute two primitives as follows,

(i) **semSignal(S)**

This semaphore primitive is used to transmit a signal through semaphore 'S'.

(ii) **semWait(S)**

This semaphore or counting semaphore primitive is used to receive a signal using semaphore 'S'. If the corresponding transmit signal has not yet been sent then the process is suspended till a signal is received.

Hence, semaphore or counting semaphore is actually an integer variable, consisting of three operations, defined as follows,

1. A non-negative value can be used to initialize the semaphore.
2. Each *semWait* operation causes a decrementation in the semaphore value and when the value becomes negative, the process gets blocked, else the process execution proceeds in a regular manner.
3. Each *semSignal* operation causes an incrementation in the semaphore value and when the value becomes less than or equal to zero, the process is unblocked which was initially blocked by the *semWait* operation.

The two semaphore primitives *semWait* and *semSignal* can be defined as follows,

```
struct semaphore
{
int C;
queueType que;
};
void semWait(semaphore S)
{
S.C = S.C–1;
if(S.C<0)
{
keep the process in S.que;
block the process;
}
}
void semSignal(semaphore S)
{
S.C = S.C+1;
if(S.C<=0)
{
Remove a process from S.que;
place the process on the ready list;
}
}
```

The two semaphore primitive operations defined above are atomic.

Example

Consider an example of the semaphore operation consisting of three processes P1, P2 and P3. These processes depend on the result generated by process P4. The execution steps are shown in figure.

1. In the beginning, process P1 is running and processes P2, P3 and P4 are in ready state. The semaphore count value is 1, specifying that one of the results produced by process P4 is now available.

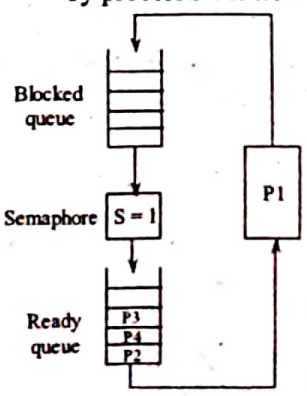


Figure (ii)

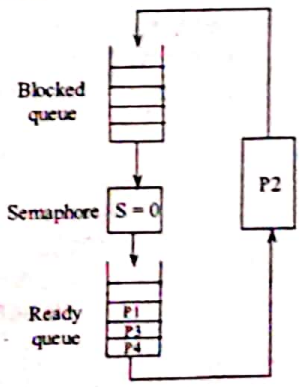


Figure (i)

2. Now, process P1 issues a *semWait* instruction on semaphore S which decrements its value to '0' thereby allowing process P2 to run. Process P1 now rejoins the ready queue as shown in figure (ii).

3. Process P2 now sends out a *semWait* instruction and gets blocked thereby permitting process P4 to run.

4. After the completion of process P4, a *semSignal* instruction is issued which allows process P2 to shift to the ready queue.

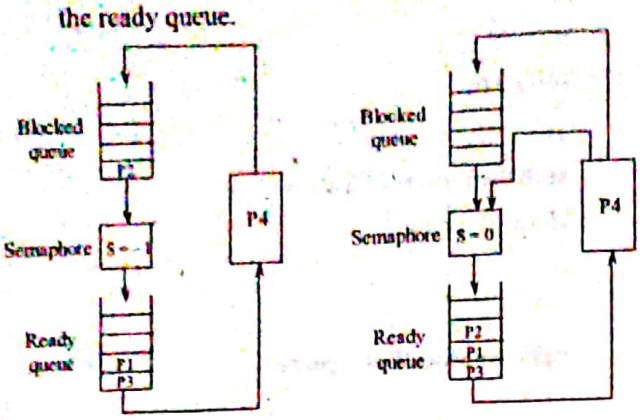


Figure (iv) Figure (iii)

5. Process P4 is again placed in the ready queue and P3 starts running, this is shown in figure (v).

6. Process P3 gets blocked on issuing a *semWait* instruction. Processes P1 and P2 run in a similar manner and are blocked, allowing process P4 to resume its execution.

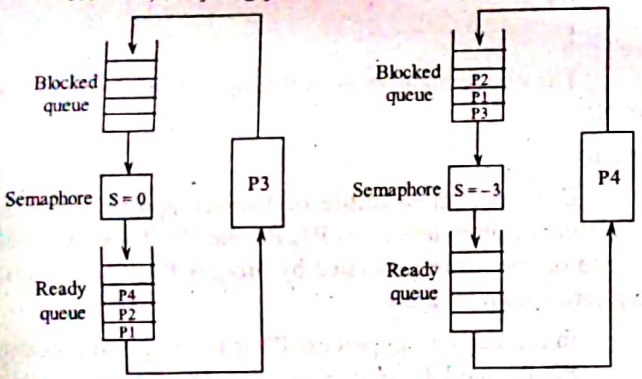


Figure (v) Figure (vi)

7. When process P4 produces a result, a *semSignal* instruction is issued and the process P3 is transferred to the ready queue. This process is repeated till the processes P1 and P2 become unblocked.

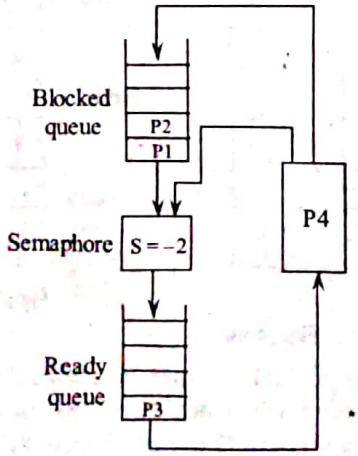


Figure (vii)

Q24. How semaphores can be used to control access to a given resource consisting of finite number of instances?

Answer :

Counting Semaphore

Counting semaphore or simply semaphore can be used to control access to a given resource consisting of finite number of instances. This semaphore (say *S*) will be assigned an integer value based on the number of resource instances available, that is, if the number of resource instances are 6 then, the semaphore value will be 6. All the processes that need to access the resource must perform the following semWait( ) operation on the semaphore.

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

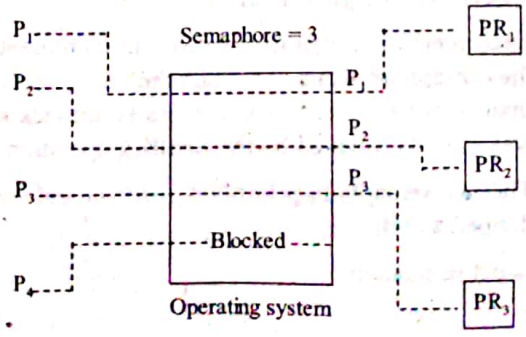
When semWait(S) operation is performed, the semaphore value gets decremented for each resource instance allocated to the process. When the semaphore value reaches zero (i.e., all the resources are assigned), all the subsequent processes that wish to utilize the same resource instance have to wait until the semaphore value bounces back to 1 or more. A process will perform the following semSignal( ) operation on the semaphore, whenever it releases a resource instance.

```
semSignal(semaphore S)
    {
        S++;
    }
```

When the above semSignal(S) operation is performed, the semaphore value gets incremented thereby allowing other processes to request for the resource.

Example

Consider four processes P_1, P_2, P_3 and P_4 running concurrently. Suppose that we have the three printer instances PR_1, PR_2, PR_3



Figure

If process P_1 needs to access the printer, it performs the semWait( ) operation on semaphore S. Thus, the S value will be decremented by 1 (i.e., S = 2) and the printer will be assigned to P_1. If processes P_2 and P_3 also require to print some files, they too will perform semWait (S) operation. Now, the S value will be zero because, all the printer instances are busy. During this time if any other process (say P_4), requests for the printer instance then, it will be blocked until the semaphore value bounces back to 1 or more.

On the other hand, if either of the processes P_1, P_2 or P_3 completes its printing task, then it performs semSignal( ) operation and releases the printer instance. The semSignal( ) operation increments the semaphore value, thereby making room for other processes to request for the printer instance.

Q25. How semaphores can be used to solve the synchronization problems?

Answer :

Semaphores can also be used for solving different synchronization problems.

Example

Consider two concurrently running processes P_1 and P_2. The process P_1 has a statement S_1 and process P_2 has a statement S_2. Suppose that we require S_1 to be executed before the execution of S_2. Which can be implemented by the following certain steps. They are,

1. We need to have a common semaphore 'synch' for both the statements S_1 and S_2.
2. Initialize the semaphore to zero.
3. Insert the following statements in process P_1.

```
S1;
semSignal(synch);
```

4. Insert the following statements in process P_2.

```
do
    {
    semWait(mutex);
    critical section
    semSignal(mutex);
    remainder section
    }
    while(1);
semWait(synch);
    S2;
```

If the above steps are followed, then the semaphore 'synch' will be initialized to zero. As a result, P_2 will execute S_2 only when the process P_1 invokes the semSignal(synch), which is after the statement S_1 has been executed.

Some of the synchronization problems that can be solved using semaphores are,

1. Bounded-buffer problem
2. Readers-writers problem
3. Dining philosophers problem.

Q26. Explain binary semaphore and counting semaphore.

Answer :

Binary Semaphore

A binary semaphore is a restricted semaphore and can have only two values '0' and '1'. It can execute three semaphore operations as follows,

1. Either '0' or '1' can be used to initialize the binary semaphore.
2. The **semWaitB** operation inspects the semaphore value and if it is '0', the process is blocked else if, the value is '1', then it is changed to '0' in order to proceed with the normal process execution.
3. The **semSignalB** operation determines if any of the processes are blocked on this semaphore. If it finds any blocked process then it changes its state to unblocked. If no blocked processes are present then the semaphore is set to a value '1'.

The two binary semaphore primitives can be defined in the following manner,

```
struct bin_semaphore
{
enum(zero, one) val;
queueType que;
};
void semWaitB(bin_semaphore S)
{
if (S.val = = 1)
{
S.val=0;
}
else
{
keep the process in S.que;
block the process;
}
}
void semSignalB(bin_semaphore S)
{
if(S.que is empty( ))
{
S.val=1;
}
else
{
remove a process from S.que;
place the process on the ready list;
}
}
```

Counting Semaphore

For answer refer Unit-III, Q24.

Q27. Explain in brief about bounded waiting.

Answer :

Whenever a process sends a request to enter into its critical section, a bound or limit must be provided to the process, which specifies the number of times it can enter the critical section before granting that request.

The following algorithm is presented to assure that both critical section and bounded waiting requirements are met,

```
do
{
waiting[x] = TRUE;
key = TRUE;
while(waiting[x] && key)
key = TestAndSet(& lock);
waiting[x] = FALSE;
//critical section
y = (x + 1) % n;
while((y ! = x)&& !waiting[y])
y = (y + 1) % n;
if(y == x)
lock = FALSE;
else
waiting[y] = FALSE;
//remainder section
}
while(TRUE);
```

The above algorithm is explained using TestAndSet( ) instruction. The common data structures used in this algorithm are,

```
boolean waiting[n];
boolean lock;
```

Initially, the data structures waiting[n] and lock are initialized to a false value. If the variables waiting[x] and key are false, then only a process P_x can be allowed to enter into the critical section. This condition helps in satisfying all the requirements of mutual exclusion. When the instruction TestAndSet( ) is executed, the key value become false. To begin the execution of TestAndSet( ) instruction, the key value is set to false and all the remaining prooesses are allowed to wait. When a process leaves the critical section, then only a single waiting[x] variable is set to false value. This will maintain the requirement of mutual exclusion.

A process before entering into its critical section may set either a lock or waiting[y] variable to false value. And, when this process leaves the critical section, the other waiting process is allowed to enter into its critical section. This is done by scanning an array of process variables that are waiting in a cyclic order (i.e., $x + 1, x + 2 \ldots, n - 1, 0, \ldots x - 1$). Scanning is used to designate the process, which is in the entry section and sets the waiting[y] variable to true so as to allow the next waiting process to enter the critical section. Thus, the processes that are waiting must perform $n - 1$ iterations to enter into the critical section.

Therefore, the bounded waiting mutual exclusion algorithm with TestAndSet( ) instruction fulfills both the bounded-waiting and mutual exclusion requirements.

Q28. Explain the implementation of mutual-exclusion semaphores.

Answer :

Mutual Exclusion Semaphores

The problem of mutual exclusion can be solved by using semaphores. For this, consider an array *a[i]* carrying '*n*' number of processes in the system. All the processes of a system want to access the same resource. Each process has a critical section for accessing the shared resource. An operation called *semWait*(S) is executed by all the processes prior to entering into its critical section. The current state of a process depends on the value of 'S'.

For a negative value of 'S', the process gets blocked and when S = 1, its value is decremented to '0' and the process is allowed to enter into the critical section immediately. All the other subsequent processes are then prevented from entering into their respective critical sections.

An initial value of the semaphore is '1'. When a process executes *semWait* operation, it immediately enters into the critical section by changing the value of 'S' to 0. All the other processes, wishing to enter into the critical section, gets blocked thereby setting the value of 'S' to –1 and this value is further decremented upon each attempt. Whenever a process leaves the critical section, the value of 'S' is incremented by 1 and a blocked process is removed from the blocked queue (a queue containing all the blocked processes) and is put in the ready queue. On scheduling, it may enter into the critical section.

Example

Consider three processes X, Y, Z sharing access to some resource which is protected by a semaphore '*sem*'. Process 'X' executes *semWait*(sem) because '*sem*' has a value 1 at the time of *semWait* operation which immediately permits process 'X' to enter into its critical section. Once it enters, the semaphore's value is decremented and becomes '0'. During the time when 'X' is inside the critical section, the other two processes 'Y' and 'Z' executes semWait operation and are blocked. Upon X's exit, it performs *semSignal*(sem) and puts the first process in the blocked queue i.e., 'Y' is now allowed to enter into its critical section.

Sometimes more than one process can be permitted to enter into the critical section at any time. This can be achieved by initializing the semaphore to some particular value. The value of *S.count* can be evaluated in the following manner,

S.count ≥ 0

S.count refers to the total number of processes that executes *semWait*(S) without any suspension or blocking. This feature allows the semaphore to support synchronization and mutual exclusion.

S.count < 0

The magnitude of S.count specifies the number of suspended processes in *S queue*.

3.1.4 Monitors, Message Passing

Q29. What is a monitor? Discuss about message Passing.

Answer :

Monitor **Model Paper-II, Q13(b)**

A monitor is a construct in a programming language which consists of procedures, variables and data structures. Any process can call the monitor procedures but access to the internal data structures is restricted. At any time, a monitor contains only one active procedure.

The condition variables can be used for blocking and non-blocking.

or

A monitor refers to the software module which consists of one or more procedures, an initialization sequence and local data.

Characteristics of Monitor

1. Access to the local variables can be granted only to the monitor's procedures but not to any external procedure.
2. Any process can be allowed to enter into the monitor by invoking one of its procedures.
3. At any time only one process can execute inside the monitor and during its execution if any other process invokes, it blocked till the monitor becomes available.

Monitor makes use of condition variables for providing synchronization. It can be operated by using two functions, They are,

(i) **cwait(c)**

Upon executing this function, a calling process is suspended and the monitor becomes available for use by any other process.

(ii) **csignal(c)**

One of the blocked processes resumes its execution upon executing this function.

Comparison between a Semaphore and a Monitor

	Semaphore		Monitor
1.	Semaphores can be used anywhere within the program but can't be used inside a monitor.	1.	Monitor makes use of condition-variables anywhere inside it.
2.	The caller is not always be in a blocked state when *wait*( ) condition is executed.	2.	The *wait*( ) function always blocks the caller.
3.	*signal*( ) releases the blocked thread, if it exists, or increases the semaphore count value.	3.	*signal*( ) releases the blocked thread, if it exists, or is lost as if it never occurs.
4.	Upon releasing a blocked thread by the *signal*( ) function, the caller and the released thread can continue with their executions.	4.	Upon releasing a blocked thread by the *signal*( ) function, only one among the caller or the released thread can continue, but not both.

Monitors with Notify and Broadcast

According to Hoare's definition, a process (if any exist in a condition queue must be executed immediately when some other process issues a csignal for that condition. SOme demerits associated with this approach are,

1. If the process issuing the *csignal* has not yet completed using the monitor, two process switches may occur for,
   (i) Blocking this process and
   (ii) Resuming some other process when the monitor becomes available.
2. Scheduling of the processes must be done in a perfectly reliable manner.

Once a *csignal* is issued, a process in a condition queue must immediately be set to the ready state and the scheduler is assigned a task of ensuring that no other process can enter into the monitor. In MESA, *csignal* is replaced with *cnotify* with an interpretation that a process executing in a monitor must execute *cnotify*(x), which causes 'x' condition queues to be notified.

Example

```
void append_elem(char i)
{
while(count = = n)
cwait(empty);
buf(next_inelem) = i;
next_inelem = (next_inelem+1)% n;
count = count+1;
cnotify(full);
}
void take_elem(char i)
{
while(count = = 0)
cwait(full);
i = buf(next_outelem);
next_outelem=(next_outelem+1)% n;
count=count–1;
cnotify(empty);
}
```

The broadcast signal associated with the semaphore makes all the processes waiting on a condition to be kept in a ready state.

Message Passing

In message passing system the job of operating system is to perform both tasks i.e., providing memory space and performing communication. The main function of message passing system is to allow processes to communicate with each other without making use of shared variables. A process can send a message of variable or fixed sizes. If two processes wants to communicate then they must send and receive messages from each other. Thus, a communication link is established between them which can be implemented in different ways.

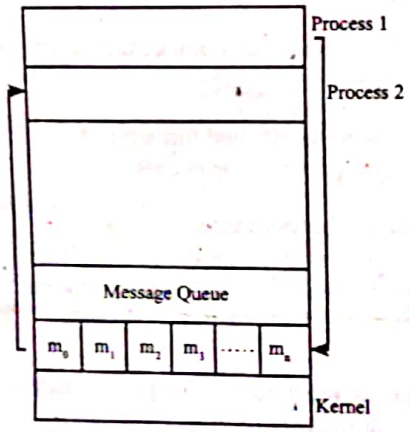


Figure: Message Passing System

In message system if two processes wish to communicate with each other, then they can communicate in the following ways,

(a) Direct and indirect communication.

(b) Symmetric and asymmetric communication.

(c) Automatic and explicit buffering.

(d) Send by referred and send by copy.

Q30. Explain how you can implement a monitor using semaphores.

Answer :

Implementation of a Monitor using Semaphores

Semaphores can be used to produce the same effect as that of monitors. A semaphore 'sem_mutex' is created for each monitor. Before entering or leaving a monitor every process should execute wait(sem_mutex) and signal (sem_mutex) respectively.

Another semaphore 'sem_nxt' is declared with initial value zero. This is used to suspend the signalling process themselves. An integer variable named nxt_count is provided to count number of suspended processes through semaphore 'sem_txt'. However, each procedure coding needs slight modification to include synchronization code. Any external procedure now contains,

```
wait(sem_mutex);
Body of procedure
  –
  –
if(nxt_count > 0)
        signal(sem_nxt);
else
        signal(sem_mutex);
```

Thus, mutual exclusion within a monitor is achieved.

To implement the wait( ) and signal( ) operation for a condition 'cn', we need to create a semaphore 'sem_cn' and an integer variable named 'cn_count' both with initial value as zero. The operation cn.wait( ) is implemented as,

```
wait( )
{
    cn_count++;
    if(nxt_count > 0)
        signal(sem_nxt);
    else
        signal(sem_mutex);
    wait(sem_cn);
    cn_count – –;
}
```

The operation cn.signal( ) has the following code,

```
signal( )
{
    if(cn_count > 0)
    {
        nxt_count++;
        signal(sem_cn);
        wait(sem_nxt);
        cn_count – –;
    }
}
```

Q31. How do you resume process within a monitor?

Answer :

When multiple processes are suspended on a single condition (cn) with a signal operation cn.signal( ) then it leads to the confusion in selecting a process to be resumed among these suspended processes. The simplest solution available for this problem is to use FIFO approach but, in most of the situations, this solution cannot be considered as effective. Therefore a new approach is designed which uses 'conditional-wait' construct which is of the form cn.wait(x);

In this format, integer expression 'x' is computed by executing wait( ) operation and is referred as priority number. It is usually stored with the name of its associated process which is in suspended state. The process with the smallest priority number is resumed first immediately after execution of cn.signal( ) operation.

For instance, consider a monitor for allocating a single resource using a resource allocator 'ResAlloc'. It allocates the resource based on the maximum time a process needs to use the resource and hence, the shortest timed process is allocated first. Corresponding pseudocode for the above process is

```
Monitor ResAlloc
boolean busy;
condition cn;
void grab (int t)
{
if(busy)
cn.wait(t);
busy = true;
}
void release( )
{
busy = False;
cn.signal( );
}
initialize_code( )
{
busy=False;'
}'
}
```

In the above code 't' represents time

However, there exist certain problems in using this method. These includes,

- A resource might be accessed without getting permitted.
- A resource might be acquired forever once accessed
- A resource which is never requested, a process might try to release it.
- An already acquired resource might be requested by the same process.

3.2 DEADLOCKS

3.2.1 Definition, Necessary and Sufficient Conditions for Deadlock

Q32. Define deadlock. What are the four conditions that create deadlock. Explain with an example.

Answer : Model Paper-III, Q13(b)

A situation in which a process waits indefinitely for requested resources and that resources are held by other processes in a waiting state. This situation results in disallowing the process to change its state which is called as a deadlock situation.

Conditions for the Deadlock

A deadlock can occur if the following four conditions hold simultaneously in a system,

1. **Mutual Exclusion**

   In a non-sharable environment where not more than a single process can be allocated with a particular resource at a time is referred to as mutual exclusion. In such an environment, a resource in use if requested by some other process, it is kept on hold until the release of that resource.

2. **Hold and Wait**

   As the name implies that process is already holding a resource and requires some of the additional resources in use by other processes. Hence this situation is known as hold and wait.

3. **No Preemption**

   A resource allocated to one process can be allocated to other only when the process holding it deallocates it after completion. This means that we cannot preempt the resources.

4. **Circular Wait**

   There exists a list of waiting processes ($P_0, P_1, ..., P_n$) such that process P_0 is waiting for a resource currently under the usage by process P_1, P_1 is waiting for a resource that is held by P_2, P_2 is waiting for a resource that is held by P_3 and so on. Finally, a process P_n is waiting for the resource held by P_0.

Out of the four conditions described above, the first three conditions are necessary but not enough for the existence of a deadlock. The fourth condition actually results from the first three conditions i.e., the first three conditions results in a sequence of events that finally leads to an unresolved circular wait which is actually the main cause for the occurrence of deadlock.

Example

The conditions for deadlock can be clearly understood by the illustration of the following example.

Let us consider an example of a bridge crossing as shown in figure. In this example, the traffic flow is only in one direction and at a time only one vehicle can pass. Consider each section of the bridge as a resource.

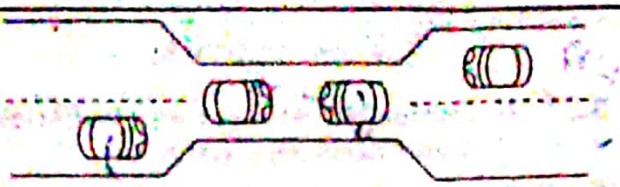


Figure: Bridge Crossing

The four necessary conditions for dead lock hold in this example due to the following reasons,

(i) When two vehicles meet on the bridge, they can't pass because each of the vehicle on the bridge is holding a non-sharable resource i.e., the vehicles are occupying the portion of the bridge, which they cannot share with the other vehicles. This is called mutual exclusion which is the first condition for dead lock.

(ii) Each vehicle occupied some space resource of the bridge and is waiting for the space in front of them to be freed by the other waiting vehicles. This is called hold and wait, which is the second condition for dead lock.

(iii) A section of the bridge occupied by a vehicle cannot be taken away from it by the another one unless it takes back up. The third condition for deadlock called no preemption is present because none of the vehicle can give up their resource.

(iv) These three conditions lead to the fourth one called "circular wait" where each vehicle involved in the stand-off situation wait for another to release the space resource in order to continue on and reach the destination.

3.2.2 Methods for Handling Deadlock: Deadlock Prevention

Q33. Discuss the methods for handling deadlock.

Answer : **Model Paper-II, Q14**

A deadlock can be handled in the following ways,

(i) Avoiding the entry of a deadlock by using various deadlock prevention and avoidance techniques.

(ii) In case that a deadlock has entered the system, we can implement various detection and recovery techniques.

(iii) In this method, non of the method are used to detect, recover, prevent (or) avoid and hence the deadlock is simply ignored.

Deadlock Prevention

For answer refer Unit-III, Q34.

Deadlock Avoidance

For answer refer Unit-III, Q36.

Deadlock Recovery

For answer refer Unit-III, Q43.

Q34. Briefly explain about deadlock prevention methods with examples of each.

Answer :

Deadlock Prevention

Deadlock prevention means placing restrictions on resource requests so that deadlock cannot occur. Deadlock can be prevented by denying at least any one of the condition of deadlock as follows,

(i) Mutual Exclusion

Only for non-sharable resources, the mutual exclusion concept should be applied.

Example

Printer is a non-sharable resource. If one program is using printer then other programs must wait for the printer. This waiting may lead to indefinite waiting, but to overcome this, the non-sharable resource may be made because for sharable resource, the concept of mutual exclusion is not required. However, by making the printer as sharable, we get error output. In general, it is not possible to prevent deadlock by denying mutual exclusion principle, since some resources should be maintained as non-sharable resource i.e., when one program is accessing it other programs cannot access it.

(ii) Preventing Hold and Wait

The hold and wait condition preludes a process from holding some resources while requesting others.

There are two protocols to prevent hold and wait.

(a) Request all the resources before starting an execution. Any program must request all the resources before starting the execution, acquire them, use them and release them once execution is completed. In this way, no process will wait for the resources during execution. Thus, hold and wait is prevented.

(b) Any process will request for a new resources only when it has none i.e., if a process has two resources and requires three more resources, then the process can request for these three resources after releasing two resources it is holding. After releasing the resources, it will request for new resources. If they are busy, the process waits without holding any resources. Suppose, if the requested resources are free, then they can be allocated to the process.

Example

Consider a process that needs to copy data from DVD drive to the disk file. If the data need to be printed, it is copied, sorted and is sent to the printer. For doing all these things the process either needs to request all the resources at the beginning of the process or obtain them when needed. Both the cases prevents the hold and wait condition.

(iii) **Preventing No Preemption**

If a system follows no preemption, then the resources which are once allocated are not taken back from a process involuntarily. Any system with above behaviour can lead to hold and wait condition and thereby to deadlock. Hence, no preemption condition should not be applied in order to prevent deadlock.

If a process P_1 request any resource R_1 which is currently held by some other process P_2 then, P_2 is preempted and R_1 is given to P_1.

Example

Consider a car occupying some part of the street which cannot be taken by other car until and unless the first car has been moved. This situation is known as No preemption. An occurrence of this condition can be prevented only if the first car has been moved forcibly giving the other car a chance to place itself and finish its task.

(iv) **Preventing Circular Wait**

Circular wait occurs, when there are a set of n processes $\{P_j\}$, that hold units of a set of n different resources $\{R_j\}$ such that, P_j holds R_j while it requests units of a different resources in the set. In other words, each of the n resources are held by the n processes, but each process then requests unavailable units of one of the resource types held by another process. A circular wait is reflected by the resource process relationships (represented wholly within a system state), so the state-transition model does not help in the study of this problem.

To prevent circular wait, we impose ordering of all resource type i.e., a unique integer number is assigned to each resource. This concept is applied only for resources of higher numbers or higher "id". If a process is requesting for resources of higher "id", it must be released.

If a process acquires all of the resources, it needs at one time, then it will never be in a situation, where it is holding a resource and waiting for another resource. This will prevent deadlock.

Example

Consider that the resources has been assigned positive integers as follows,

Printer → 1

Tap drive → 2

Card punch → 3

Card reader → 4

Plotter → 5

Now, the process must request the resources in numerical order. For example, the process can request the printer first followed by the card punch and card reader (1, 3, 4). It cannot request card reader first and then printer.

Q35. Consider the deadlock situation that could occur in the dining philosopher's problem when the philosophers obtain the chopsticks one at a time. Discuss the four conditions for deadlocks indeed hold in the setting. Discuss how deadlocks could be avoided by eliminating any one of the four conditions?

Answer :

Conditions for Deadlocks in the Dining Philosopher's Problem

The four conditions that lead to a deadlock situation are as follows,

1. Mutual exclusion
2. Hold and wait
3. No preemption
4. Circular wait.

For remainng answer refer Unit-V, Q14.

In terms of mutual exclusion, dining philosophers problem holds a deadlock situation when only one chopstick can be accessed by a philosopher at a given instance of time, while others have to wait until its release. In terms of hold and wait, a deadlock can occur where a philosopher has one chopstick in hand and waiting for the other chopstick. This in because a philosopher can not forcibly take chopstick from the others, thus leading to no preemption.

There is a possibility of the occurence of circular wait situations in dining philosopher's problem. For example, assume that a philosopher A is waiting for a chopstick that is currently being used by the philosopher B. And, the philosopher B is waiting for chopstick that is being used by philosopher C who in turn is waiting for chopstick acquired by philosopher A.

Avoiding Deadlocks

Deadlocks can be avoided by overcoming the aforementioned conditions in the following manner.

(a) **Mutual Exclusion**

Only for non-sharable resources, the mutual exclusion concept should be applied. That is, philosophers should share the chopsticks simultaneously. However, it is quite difficult to implement this method practically.

(b) **Prevent Hold and Wait**

This situation can be avoided by enabling the philosopher to hand over the first chopstick to other philosopher, if he/she is unable to obtain the other chopstick.

(c) **Prevent No Preemption**

Frequent occurrence of deadlocks is mainly due to no preemption condition. Hence, it should be avoided in order to prevent deadlock. This is possible by allowing the philosopher to forcibly take away the chopstick from another philosopher if it is being used for a long period of time.

(d) Prevent Circular Wait

To prevent circular wait, numbering of chopsticks should be enforced i.e., a unique number must be assigned to each chopstick. It must be ensured that the lower numbered chopstick must be obtained first rather than the higher numbered one.

3.2.3 Deadlock Avoidance: Banker's Algorithm

Q36. Explain about deadlock avoidance.

Answer :

Deadlock Avoidance

Deadlock avoidance does not impose any rules but, here each resource request is carefully analyzed to see whether it could be safely fulfilled without causing deadlock. The drawback of this scheme is its requirement of information in advance about how resources are to be requested. Different algorithms require different type and amount of information like some require maximum number of resources that each process requires etc.

The following are the various deadlock avoidance algorithms,

1. Safe State

Consider a system consisting of several processes like $< P_1, P_2, P_3, ..., P_n >$. Each of them requires certain resources immediately and also specifies the maximum number of resources that they may need in their life time. Using this information, we need to build a "safe sequence" which is a sequence of processes where their resource request can be satisfied without having deadlock to occur. If there exists any such safe sequence, then system is said to be in "safe state" during which deadlock cannot occur. An unsafe state may lead to deadlock but not always. There are some sequences in unsafe state which can lead to deadlock.

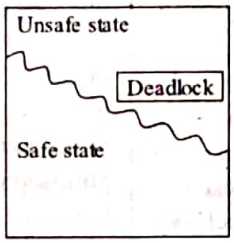


Figure: States in a System

For example, consider a system with 24 printers and three processes i.e., P_1, P_2 and P_3 requiring 14, 8 and 13 printers respectively. This is the maximum need they require, it is not always the case that they require all of them at once. At a particular time t_0, P_1 needs only 9 printers, P_2 and P_3 needs 6 each. The table (1) shows the same.

Process	Current Need	Maximum Need
P_1	9	14
P_2	6	8
P_3	6	13

Table (1)

If the processes are executed in the sequence i.e., $<P_2, P_1, P_3>$ then the safety condition can be satisfied. The table (2) shows the sequence of resource allocation and release.

	P_2	P_1	P_3	Total (24) Printers Remaining
At t_0 resources allocated according to current needs.	6	9	6	3
P_2 is allocated its maximum requirements.	6 + 2 = 8	9	6	1
P_2 completes and returns all resources.	–	9	6	9
P_1 is allocated its maximum requirements.	–	9 + 5 = 14	6	4
P_1 completes and returns all resources.	–	–	6	18
P_3 is allocated its maximum requirements.	–	–	6 + 7 = 13	11
P_3 completes and releases all its resources.	–	–	–	24

Table (2)

The above table (2) shows one of the safe sequence, there may be many safe sequences for the same example. In the beginning, system will be in safe state then processes are allocated resources according to their current need. Thereafter, whenever a process requests, the algorithm must decide whether the allocation is safe or unsafe and accordingly the action should be taken.

2. Resource Allocation Graph

It is a directed graph, given as $G = (V, E)$ containing a set of vertices V and edges E. The vertices are divided into two types, i.e., a set of processes, $P = \{P_1, P_2, P_3, \ldots, P_n\}$ and a set of resources, $R = \{R_1, R_2, R_3, \ldots, R_n\}$. An edge from process ($P_i$) to resource ($R_j$) ($P_i \rightarrow R_j$) indicates that P_i has requested for resource R_j. It is called as 'request edge'. Any edge from a resource R_j to a process P_i ($R_j \rightarrow P_i$) indicates that R_j is allocated to process P_i. It is called as "assignment-edge" when a request is fulfilled it is transformed to assignment edge. Processes are represented as circles and resources as rectangles. The following is an example of graph where process P_1 has R_1 and requests for R_2.

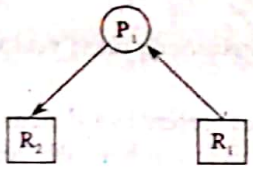


Figure: Resource Allocation Graph

For avoiding deadlocks using resource allocation graph, it is modified slightly by introducing a new edge called "claim edge". It is an edge from process P_i to resource R_j ($P_i \rightarrow R_j$) indicates that in future, P_i may request for R_j. The direction of the arrow is same as request edge but with dashed line.

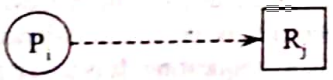


Figure: Claim Edge

To describe the usage of this graph in deadlock avoidance, let us consider the following example graph consisting of two processes (P_1 and P_2) and two resources (R_1 and R_2) such that P_2 has resource R_2 and requests for R_1 and P_1 has resource R_1 and may claim for R_2 in future. This action can create a cycle in the graph which means deadlock is possible and system is in unsafe state. Hence, allocation should not be done.

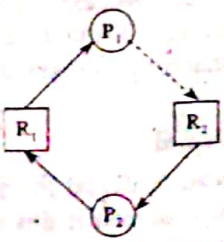


Figure: Resource Allocation Graph Representing Unsafe State

3. Banker's Algorithm

It is used to avoid deadlocks when multiple instances of each resource types are present. This is not possible, using the methods like safe state and resource allocation graphs. It is similar to a banking system where a bank never allocates cash in such a way that it could not satisfy the needs of all its customers and also it cannot allocate more than what is available. Here, customers are analogous to processes, cash to resources and bank to operating system.

A process must specify in the beginning the maximum number of instances of each resource type it may require. It is obvious that this number should not be more than the available. When process request resources, system decides whether allocation will result in deadlock or not. If not, resources are allocated otherwise process has to wait.

The following are the various data structures which has to be created to implement Banker's algorithm.

Where, n = Number of processes

m = Number of resources.

(a) Max

A $n \times m$ matrix indicating the maximum resources required by each process.

(b) Allocation

A $n \times m$ matrix indicating the number of resources already allocated to each process.

(c) Need

A $n \times m$ matrix indicating the number of resources required by each process.

(d) Available

It is a vector of size m which indicates the resources that are still available (not allocated to any process).

(e) Request

It is a vector of size m which indicates that process P_i has requested for some resource.

Each rows of matrices "allocation" and "need" can be referred as vectors. Then "allocation$_i$" indicates the resources currently allocated to process P_i and "need$_i$" refers to resources required by P_i.

The following algorithm is used to determine whether the request can be safely granted or not.

Step1

If Request$_i \leq$ Need$_i$, then proceede to step (ii), otherwise raise an exception saying process has exceeded its maximum claim.

Step2

If Request$_i \leq$ Available, then proceed to step (iii), otherwise block P_i because resources are not available.

Step3

Allocate resources to P_i as follows,

Available: = Available – Request$_i$

Allocation$_i$: = Allocation$_i$ + Request$_i$

Need$_i$: = Need$_i$ – Request$_i$

4. Safety Algorithm

The job of banker's algorithm is to perform allocation, it will not see whether this allocation has resulted in safe or unsafe state. It is the safety algorithm which is called immediately after banker's algorithm to check for the system state after allocation. The following is the safety algorithm which requires $m \times n^2$ operations to find system state.

Step1

Assume work and finish as vectors of length m and n respectively.

Work : = Available

Finish[i] : = 'false'.

Step 2

Find 'i' such that

Finish[i] : = false

Need$_i \leq$ Work

If no such i is found jump to step (iv).

Step 3

Work : = Work + Allocation

Finish[i] : = 'true'

Jump to step (ii)

Step 4

If finish[i] : = True for all then system is in safe state.

Q37. What is deadlock avoidance? Explain process initiation denial and resource allocation denial in detail with an example.

Answer :

Deadlock Avoidance

For answer refer Unit-III, Q36, Topic: Deadlock Avoidance.

Process Initiation Denial

In process initiation denial, a process is not started if the total demands of the process might lead to deadlock.

Let the total number of processes in the system be n and the total number of resources be m. Now, we define the following vectors and matrices.

(a) Resource Vector

It describes the total amount of each resource present in the system.

It is denoted as $R = (R_1, R_2, \ldots R_m)$.

(b) Available Vector

It describes the quantity of each resource which has not yet been assigned to each process.

It is denoted as $V = (V_1, V_2, \ldots V_m)$.

(c) Claim Matrix

It describes the maximum demand of process i for resource j. Each row corresponds to one process.

It is denoted as,

$$C = \begin{bmatrix} C_{11} & C_{12} & C_{13} & \ldots\ldots & C_{1m} \\ C_{21} & C_{22} & C_{23} & \ldots\ldots & C_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ C_{n1} & C_{n2} & C_{n3} & & C_{nm} \end{bmatrix}$$

(d) Allocation Matrix

It describes the current allocation of resource j to process i.

It is denoted as,

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \ldots\ldots & A_{1m} \\ A_{21} & A_{22} & A_{23} & \ldots\ldots & A_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ A_{n1} & A_{n2} & A_{n3} & & A_{nm} \end{bmatrix}$$

The relationships among these vectors and matrices are as follows,

(i) All the resources present in a system are either allocated or available. This is symbolically represented as,

$$R_q = V_q + \sum_{p=1}^{n} A_{pq}, \forall q$$

(ii) The demand of each process for the resources must always be less than the total amount of resources available in a system. It is symbolically expressed as,

$$C_{pq} \leq R_q, \forall p, q$$

(iii) The allocation of resources to the process must not exceed the process need i.e., no process is assigned more resources than it is actually requesting. It is given as,

$$A_{pq} \leq C_{pq}, \forall p, q$$

Hence, a deadlock can be avoided by preventing the process execution if its requirement (for the resources) leads to deadlock.

A new process P_{n+1} is started only if,

$$R_q \geq C_{(n+1)_q} + \sum_{p=1}^{n} C_{pq}, \forall q$$

The process execution begins only if the maximum demand from all the current processes plus the demand from new process can be satisfied. This technique is inefficient as it makes an assumption that all processes demands together for maximum resources.

Resource Allocation Denial

In resource allocation denial the increment request to a resource by a process is denied if this allocation leads to a deadlock. The strategy used by resource allocation denial is Banker's algorithm.

For remaining answer refer Unit-III, Q36, Topic: Banker's Algorithm.

Example

For answer refer Unit-III, Q39.

Q38. Consider a system with 5 processes P_0, P_1, P_2, P_3, P_4 and 3 resources A, B and C. Resource A has 10 instances, resource B has 5 instances, resource C has 7 instances, the maximum and allocation matrices are as follows,

	A	B	C
P_0	7	5	3
P_1	3	2	2
P_2	9	0	2
P_3	2	2	2
P_4	4	3	3

MAX

	A	B	C
P_0	0	1	0
P_1	2	0	0
P_2	3	0	2
P_3	2	1	1
P_4	0	0	2

ALLOCATION

1. **Is the present state safe, if so give the safe sequence.**
2. **If P_1 is requesting for [1 0 2], can the request be granted, if so give the safe sequence.**
3. **If P_4 is requesting for [3 3 0], can the request be granted, if so give the safe sequence.**
4. **If P_0 is requesting for [0 2 0], can the request be granted, if so give the safe sequence.**

Answer :

1. Here NEED is equal to MAX–ALLOCATION.

$$\begin{bmatrix}7&5&3\\3&2&2\\9&0&2\\2&2&2\\4&3&3\end{bmatrix} - \begin{bmatrix}0&1&0\\2&0&0\\3&0&2\\2&1&1\\0&0&2\end{bmatrix} = \begin{bmatrix}7&4&3\\1&2&2\\6&0&0\\0&1&1\\4&3&1\end{bmatrix}$$

Maximum – Allocation = Need

It is given that there are 10 resources of type *A*, 5 resources of type *B* and 7 resources of type *C*.

Therefore, after allocation is over the available resources are [10 – (2 + 3 + 2), 5 – (1 + 1), 7 – (2 + 1 + 2)]

= [3 3 2]

Work= Available = [3 3 2]

Finish= [F F F F F]

Is there any process which satisfy (Need ≤ work) and ($finish_i$ = f)	Work = Work + Allocation	Finish
P_1	[332]+[200]=[532]	[FTFFF]
P_3	[532]+[211]=[743]	[FTFTF]
P_4	[743]+[002]=[745]	[FTFTT]
P_0	[745]+[010]=[755]	[TTFTT]
P_2	[755]+[302]=[1057]	[TTTTT]

Since finish vector contains all trues the current state is safe and the safe sequence is (P_1, P_3, P_4, P_0, P_2).

2. P_1 is requesting for [1 0 2] i.e. $Request_1$ = [1 0 2].

Banker's Algorithm

Check whether $Request_1$ <= Need

[1 0 2] ≤ [1 2 2] is satisfied.

Compare $Request_1$ with available

Request ≤ Available

[1 0 2] ≤ [3 3 2] is satisfied.

Perform Allocation.

i.e., $Allocation_1$ = $Allocation_1$ + $Request_1$

$Need_1$ = $Need_1$ – $Request_1$

∴ Allocation and Need matrices are modified as,

$$\begin{bmatrix}0&1&0\\3&0&2\\3&0&2\\2&1&1\\0&0&2\end{bmatrix} \quad \begin{bmatrix}7&4&3\\0&2&0\\6&0&0\\0&1&1\\4&3&1\end{bmatrix}$$

Allocation Need

Available = [2 3 0]

Safety Algorithm

Initially Work = Available = [2 3 0]

Finish = [F F F F F]

Is there any process which satisfy (Need ≤ work) and ($finish_i$ = f)	Work = Work + Allocation	Finish
P_1	[230]+[302]=[532]	[FTFFF]
P_3	[532]+[211]=[743]	[FTFTF]
P_4	[743]+[002]=[745]	[FTFTT]
P_0	[745]+[010]=[755]	[TTFTT]
P_2	[755]+[302]=[1057]	[TTTTT]

Therefore, the Finish vector contains all trues, the Request can be accepted. Hence, safety algorithm is satisfied and the safe sequence is [P_1, P_3, P_4, P_0, P_2].

3. P_4 is requesting for [3 3 0]

i.e. $Request_4$ = [3 3 0]

Check whether $Request_4$ ≤ $Need_4$

[3 3 0] ≤ [4 3 1] is satisfied. Hence, compare $Request_4$ with Available.

$Request_4$ ≤ $Available_4$

[3 3 0] ≤ [3 3 2] is satisfied. Hence, perform allocation.

i.e. $Allocation_4$ = $Allocation_4$ + $Request_4$

$Need_4$ = $Need_4$ – $Request_4$

Available = Available – $Request_4$

Available = [3 3 2]

$$\begin{bmatrix} 0 & 1 & 0 \\ 2 & 0 & 0 \\ 3 & 0 & 2 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix} \qquad \begin{bmatrix} 7 & 4 & 3 \\ 1 & 2 & 2 \\ 6 & 0 & 0 \\ 0 & 1 & 1 \\ 4 & 3 & 1 \end{bmatrix}$$

Allocation Need

After allocation the Need and Available matrices are as follows.

$$\begin{bmatrix} 0 & 1 & 0 \\ 2 & 0 & 0 \\ 3 & 0 & 2 \\ 2 & 1 & 1 \\ 3 & 3 & 2 \end{bmatrix} \qquad \begin{bmatrix} 7 & 4 & 3 \\ 1 & 2 & 2 \\ 6 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

Allocation Need

Available = [0 0 2]

Initialization

Work = Available = [0 0 2]

Finish = [F F F F F].

Is there any process which satisfy Need ≤ work and $finish_i$ = F	Work = Work + Allocation	Finish

No process is satisfying two conditions, the state is unsafe. Hence donot grant request to process P_4.

4. Process P_0 is requesting for [0 2 0] i.e., $Request_0$ = [0 2 0]

Check whether $Request_0 \leq Need_0$

= [0 2 0] ≤ [–7 4 3] is satisfied.

Hence, compare $Request_0$ with available

∴ [0 2 0] ≤ [3 3 2] is satisfied.

Hence, perform allocation.

i.e., $Allocation_0 = Allocation_0 + Request_0$

$Need_0 = Need_0 – Request_0$

$$\begin{bmatrix} 0 & 1 & 0 \\ 2 & 0 & 0 \\ 3 & 0 & 2 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix} \qquad \begin{bmatrix} 7 & 4 & 3 \\ 1 & 2 & 2 \\ 6 & 0 & 0 \\ 0 & 1 & 1 \\ 4 & 3 & 2 \end{bmatrix}$$

Allocation Need

Available = [3 3 2]

After Allocation

$$\begin{bmatrix} 0 & 3 & 0 \\ 2 & 0 & 0 \\ 3 & 0 & 2 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix} \qquad \begin{bmatrix} 7 & 2 & 3 \\ 1 & 2 & 2 \\ 6 & 0 & 0 \\ 0 & 1 & 1 \\ 4 & 3 & 1 \end{bmatrix}$$

Allocation Need

Available = [3 1 2]

Initialization,

Work = Available = [3 1 2]

Finish = [F F F F F]

Is there any process which satisfy (Need ≤ work) and (finish$_i$ = f)	Work = Work + Allocation	Finish
P_1	[3 1 2] + [2 1 1] = [5 2 3]	[F F F T F]
P_3	[5 2 3] + [2 0 0] = [7 2 3]	[F F F T F]
P_2	[7 2 3] + [3 0 2] = [10 2 5]	[F F T T F]
P_0	[10 2 5] + [0 3 0] = [10 5 5]	[T F T T F]
P_4	[10 5 5] + [0 0 2] = [10 5 7]	[T F T T T]

Therefore, the Finish vector contains all trues. Hence, Request can be granted and the safe sequence is,

$[P_3, P_1, P_2, P_0, P_4]$.

Q39. Consider the following snapshot of a system. Answer the following questions using the bankers algorithm.

Processes	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	1	5	2	0
P1	1	0	0	0	1	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	6	5	2				
P4	0	0	1	4	0	6	5	6				

(a) What is the content of matrix need?

(b) Is the system in a safe state?

(c) If a request from process P1 arrives for (0, 4, 2, 0) can the request be granted immediately.

Answer :

Given snapshot of the system,

Processes	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	
P0	0	0	1	2	0	0	1	2	1	5	2	0
P1	1	0	0	0	1	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	6	5	2				
P4	0	0	1	4	0	6	5	6				

(a) Content of the matrix 'Need'

$$\text{Matrix Need} = \text{Max} - \text{Allocation}$$

$$\text{Matrix Need} = \begin{bmatrix} 0 & 0 & 1 & 2 \\ 1 & 7 & 5 & 0 \\ 2 & 3 & 5 & 6 \\ 0 & 6 & 5 & 2 \\ 0 & 6 & 5 & 6 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 1 & 2 \\ 1 & 0 & 0 & 0 \\ 1 & 3 & 5 & 4 \\ 0 & 6 & 3 & 2 \\ 0 & 0 & 1 & 4 \end{bmatrix}$$

∴ The content of matrix Need = $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 7 & 5 & 0 \\ 1 & 0 & 0 & 2 \\ 0 & 0 & 2 & 0 \\ 0 & 6 & 4 & 2 \end{bmatrix}$

(b) Safe State

The system is said to be in safe state if the allocation of resources to the processes satisfies the bankers algorithm.

Initially Work = Available = 1520, Finish vector = {F F F F F}

Is there any process satisfying (Need$_i$ ≤ work) and (Finish[i] = false)	Need = (Max – Allocation)	Work = Work + Allocation	Finish Vector
P0	0 0 0 0	[1 5 2 0] + [0 0 1 2] = 1 5 3 2	TFFFF
P3	0 0 2 0	[1 5 3 2] + [0 6 3 2] = 2 1 6 4	TFFTF
P4 not satisfying the condition	0 6 4 2	–	TFFTF
P1 not satisfying the condition	0 7 5 0	–	TFFTF
P2 not satisfying the condition	1 0 0 2	–	TFFTF

Since, Finish vector do not contains all trues we can say that the system is not in a safe state.

(c) Granting Request

Given, P1 Requests for (0, 4, 2, 0)

It is known that if the Request ≤ Available then the request can be granted immediately.

∴ Checking the condition(Request ≤ Available)

(0, 4, 2, 0) ≤ (1, 5, 2, 0) is satisfied.

Hence, the request can be granted immediately.

Q40. Consider the following snapshot of a system,

Processes	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	2	1	0	0
P1	2	0	0	0	2	7	5	0				
P2	0	0	3	4	6	6	5	6				
P3	2	3	4	5	4	3	5	6				
P4	0	3	3	2	0	6	5	2				

Answer the following questions using the Banker's algorithm,

(a) What is the content of the matrix need?

(b) Is the system in a safe state? Why?

(c) Is the system currently deadlocked? Why or why not?

(d) Which process, if any, may become deadlocked if this whole request is granted immediately?

Answer :

Given that,

Processes	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	2	1	0	0
P1	2	0	0	0	2	7	5	0				
P2	0	0	3	4	6	6	5	6				
P3	2	3	4	5	4	3	5	6				
P4	0	3	3	2	0	6	5	2				

(a) Content of the Matrix Need

Matrix 'Need' = Max – Allocation

$$\text{Matrix Need} = \begin{bmatrix} 0 & 0 & 1 & 2 \\ 2 & 7 & 5 & 0 \\ 6 & 6 & 5 & 6 \\ 4 & 3 & 5 & 6 \\ 0 & 6 & 5 & 2 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 1 & 2 \\ 2 & 0 & 0 & 0 \\ 0 & 0 & 3 & 4 \\ 2 & 3 & 4 & 5 \\ 0 & 3 & 3 & 2 \end{bmatrix}$$

$$\text{Matrix Need} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 7 & 5 & 0 \\ 6 & 6 & 2 & 2 \\ 2 & 0 & 1 & 1 \\ 0 & 3 & 2 & 0 \end{bmatrix}$$

∴ The content of a matrix Need is,

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 7 & 5 & 0 \\ 6 & 6 & 2 & 2 \\ 2 & 0 & 1 & 1 \\ 0 & 3 & 2 & 0 \end{bmatrix}$$

(b) System in a Safe State

According to the Banker's algorithms the system is said to be in safe state if the allocation of resources to the processes satisfies the criteria (Finish[i] = True for all i).

Hence, processing the steps of safety algorithm in order to check whether finish vector contains all trues or not.

Initially,

Work = Available = 2100

Is there any process satisfying			Finish vector				
(Need$_i$ ≤ Work) and (Finish = False)	Need$_i$ = (Max – Allocation)	Work = Work + Allocation	P0	P1	P2	P3	P4
P0	0 0 0 0	Work = 2100 + 0012 = 2112	T	F	F	F	F
P3	2 0 1 1	Work = 2112 + 2345 = 4457	T	F	F	T	F
P4	0 3 2 0	Work = 4457 + 0332 = 4789	T	F	F	T	T
P1	0 7 5 0	Work = 4789 + 2000 = 6789	T	T	F	T	T
P2	6 6 2 2	Work = 6789 + 0034 = 6823	T	T	T	T	T

Since, Finish vector contains all true according to algorithm, we can say that the system is in safe state and the sequence <P0, P3, P4, P1, P2>.

(c) No the system is not currently deadlocked since it is in a safe state and the safe sequence is <P0, P3, P4, P1, P2>.

(d) Whole Request = Max = $\begin{bmatrix} 0 & 0 & 1 & 2 \\ 2 & 7 & 5 & 0 \\ 6 & 6 & 5 & 6 \\ 4 & 3 & 5 & 6 \\ 0 & 6 & 5 & 2 \end{bmatrix}$

Work = Available = 2100

The process *Pi* will be in deadlocked state if it does not satisfy the condition Request ≤ Work. Therefore, checking the condition for each process,

P0: 0012 ≤ 2100 not satisfying

P1: 2750 ≤ 2100 not satisfying

P2: 6656 ≤ 2100 not satisfying

P3: 4356 ≤ 2100 not satisfying

P4: 0652 ≤ 2100 not satisfying.

Therefore, if the whole request is granted immediately then every process of the system becomes deadlocked.

3.2.4 Deadlock Detection and Recovery

Q41. Explain all the strategies involved in deadlock detection and how it is recovered.

Answer :

Deadlock Detection Strategies

Deadlocks can be prevented by employing two techniques as follows,

1. Deadlock prevention
2. Deadlock avoidance.

If either of the above two techniques is not applied then a deadlock may occur and a system must provide,

(i) An algorithm that can monitor the state of the system to detect the occurrence of a deadlock.

(ii) A recovery algorithm to regain from the deadlocks state.

Deadlock Detection in a System Containing a Single Instance of each Resource Type

A deadlock detection algorithm that makes use of a variant of the resource allocation graph, called the wait-for graph is defined for a system containing only a single instance for all the resources.

An edge from a node P_i to node P_j exists in a wait-for graph, if and only if, the corresponding RAG contains two edges, one from node P_i to some resource node R_q and the other from the resource R_q to node P_j. The presence of a cycle in the wait-for graph indicates the existence of a deadlock.

An algorithm that is used to detect a cycle in the graph requires a total of n^2 operations, where n is the number of vertices in the graph.

Example

- Consider five processes P_1, P_2, P_3, P_4 and P_5 and five resources R_1 to R_5. The Resource Allocation Graph (RAG) for such a system is shown in figure.

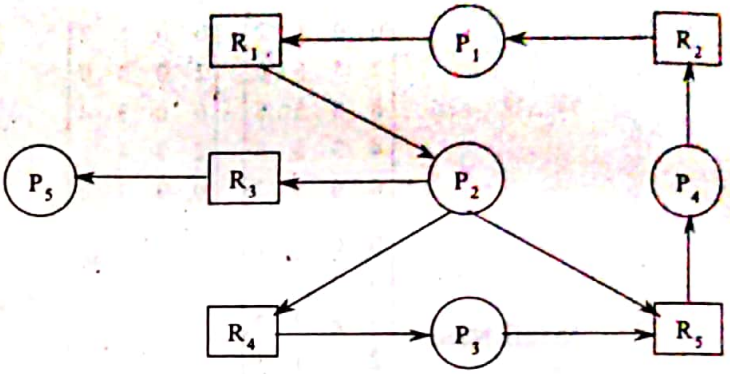


Figure (a): Resource Allocation Graph

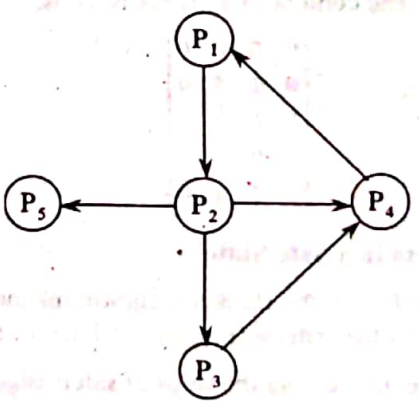


Figure (b): Wait-For Graph for the Given RAG

Deadlock Detection in a System Containing Multiple Instances of a Resource Type

The wait-for graph can't be used for a resource allocation system containing multiple instances of each resource type. Hence, a different algorithm is employed which carries certain data structures.

The data structures in the algorithm are,

Available

It is a vector of length *m* that can specify the number of resources of each type that are available.

Allocation

It is an $n \times m$ matrix that is used to define the number of resources of each type presently assigned to each process in a system.

Request

It is an $n \times m$ matrix which specifies the current request made by each process.

If Request $[i, j] = k$, then process i is currently requesting for k additional instances of resource j.

The deadlock detection algorithm for every possible resource allocation sequence is given below.

Consider two vectors work and finish whose lengths are m and n respectively.

1. Initialize work = Available.
2. Determine allocation for each i, where $i = 1, 2, ... n$. If allocation $i \neq 0$ then set Finish [i] to false else, Finish [i]is assigned a value true.
3. Determine an index i for which both Finish [i] = false and Request $i \leq$ Work. If no such i value is available then jump directly to step 5.
4. Set Work = Work + Allocation and Finish [i] = true and iterate through step 2.
5. If Finish [i] = False, for some i, $1 \leq i \leq n$ then the system is in deadlock state and the process P_i is deadlocked.

Hence, $m \times n^2$ operations are required to determine whether the system is in deadlocked state.

Deadlock Recovery

For answer refer Unit-III, Q43.

Q42. Write the deadlock detection algorithm. Illustrate through an example snap shot of a systems.

Answer :

Deadlock Detection Algorithm

An algorithm for the deadlock detection make use of the 'allocation matrix' which describes the current resource allocation and the 'available vector' that describes the total amount of each resource not allocated to any process. In addition to the allocation matrix and the available vector, a request matrix Q is defined in such a way that Q_{ij} specifies the amount of resources of type j requested by a process i.

The processes that are not under the deadlocked state are marked. All the processes are unmarked at the beginning of an algorithm. The execution proceeds as follows,

1. Each process, having a row of all zeroes in the allocation matrix is marked.
2. A temporary vector W is initialized with the available vector.
3. Determine an index i for which a process i is currently unmarked and the i^{th} row of $Q \leq W$ i.e., $Q_{ik} \leq W_k$ for $1 \leq k \leq m$. Stop the algorithm if no such row is found.
4. After finding such a row, process i is marked and the associated row in the allocation matrix is added to W i.e., set $W_k = W_k + A_{ik}$, for $1 \leq k \leq m$. Go back to step3.

After executing the algorithm if any unmarked processes are present then a deadlock exists. This algorithm finds a process whose requests for the resources can be satisfied with the available resources and it is assumed that those resources are allocated to it and the process completes its execution thereby releasing all its resources. Another process is then looked up by the algorithm and the whole process is repeated. This algorithm does not assures deadlock prevention but instead it determines the existence of deadlock.

Example

Consider the given allocation and request matrices and resource and available vectors.

	R_1	R_2	R_3	R_4	R_5
P_1	0	1	0	0	1
P_2	0	0	1	0	1
P_3	0	0	0	0	1
P_4	1	0	1	0	1

Request Matrix, Q

	R_1	R_2	R_3	R_4	R_5
P_1	1	0	1	1	0
P_2	1	1	0	0	0
P_3	0	0	0	1	0
P_4	0	0	0	0	0

Allocation Matrix

R_1	R_2	R_3	R_4	R_5
2	1	1	2	1

R_1	R_2	R_3	R_4	R_5
0	0	0	0	1

Available vector Resource vector When the deadlock detection algorithm is applied it proceeds as follows,

1. Mark the process P_4 in the allocation matrix as it has no allocated resources.
2. Set $W = (0\ 0\ 0\ 0\ 1)$.
3. As request mode by the process P_3 is less than or equal to W, so it is marked and W is set to,

   $W = W + (0\ 0\ 0\ 1\ 0)$

   $W = (0\ 0\ 0\ 1\ 1)$.
4. As no other unmarked process in Q contains a row which is less than or equal to W. So, the algorithm will be terminated.

The algorithm execution ends, leaving the processes P_1 and P_2 in an unmarked state. Hence, these processes are deadlocked.

Usage of Deadlock Detection Algorithms

The usage of deadlock detection algorithm depends on the following factors,

(i) Frequent occurance of deadlocks

(ii) Effects of deadlock on other process.

Invocation of deadlock detection algorithm highly depends on how frequently deadlocks occur. In case that the deadlocks occurring frequently, it must be used as frequently as their occurence. This is done because when a process is affected by deadlock, all its allocated resources can not be released and hence, can not be used by other processes. In addition to this, it is possible the processes in the ready queue might increase.

In the second case, the detection algorithm is used every time when a process requested for a resource and it is not allocated immediately. By implementing this, it is possible to grab the process with which a deadlock occurred and the set of processes associated with it. However, using the algorithm on every process request takes a lot of time for computing overall processes.

One method to overcome the above said drawback is to use the detection algorithm on certain regular (custom time intervals (for example, every 30 minutes). However with this method, it is difficult to identify the process with which a deadlock occurred. This is because, within that time interval, the resource graph might carry many deadlock cycles.

Q43. State and explain the deadlock recovery methods.

Answer :

Recovery from Deadlock

When a deadlock has been detected in the system by deadlock detection algorithms, then it has to be recovered by using some recovery mechanism. The brute force approach is to reboot the computer, but it is inefficient because it may loose complete data and waste computing time. Hence, other techniques are used to recover from deadlock. They are broadly classified into two types. They are,

1. Process termination
2. Resource preemption.

1. **Process Terminatfon**

Here one or more processes are terminated to eliminate deadlock. It has two methods as follows,

(i) Terminate all deadlocked processes which will break deadlock immediately, but it is a bit expensive because there may be some processes which have been executing for a long time consuming considerable CPU time and their termination will result in wasting those CPU cycles.

(ii) In order to overcome drawback of the above method, this method terminates one process at a time until deadlock is recovered. However, it has some overhead since, after terminating each process, detection algorithm has to be executed for deciding to further terminate the processes or not. This method is slower than the first one.

2. **Resource Preemption**

In this method, resources are deallocated or preempted from some processes and the same are allocated to others until deadlock is resolved. We have three important issues to implement this scheme. They are,

(i) **Selection of Victim Process**

We need to decide which process or which resources are to be preempted, the decision is based on cost factor which includes the number of resources, a deadlocked process is holding and CPU time consumed by it etc.

(ii) **Rollback**

The process which was preempted cannot continue normal execution, because its resources are taken back. Hence, we need to rollback to some previous checkpoint or total rollback to start it from the beginning.

(iii) **Starvation**

We should ensure that a particular process should not starve every time preemption is done.

MEMORY MANAGEMENT AND VIRTUAL MEMORY

PART-A
SHORT QUESTIONS WITH ANSWERS

Q1. Write about shared libraries.

Answer :

While fixing bugs in libraries, there can be two types of modifications i.e., major and minor. Major modifications such as change in the program addresses typically changes (increments) the version number of the library whereas minor bug fixes do not change it.

When dynamic linking is used the latest version installed is just referenced whereas in the absence of dynamic linking, they need to be relinked. There exist multiple version of libraries as there can be programs that might use older version of libraries (those that were installed before updating the library). This system where multiple versions of shared libraries exist is known as shared libraries.

Q2. What is meant by swapping? List various reasons to perform swapping.

Model Paper-II, Q7

Answer :

Swapping

In a multiprogramming environment, there are several processes that are executed concurrently. A process needs to be present in main memory for execution, but its capacity is not enough to hold all active processes. Hence, sometime processes are swapped-out and stored on disk to make space for others and later they are swapped-in to resume execution. This process of swapping-in and swapping-out is called as swapping.

Reasons for Performing Swapping

There are several reasons to perform swapping. For example,

- If time quantum of a particular process is expired.
- If some high priority process pre-empts a particular process.
- When an interrupt occurs and makes this process to wait.
- Process is put in wait-state for performing some input/output operations.

Q3. Give the relative advantages and disadvantages of contiguous memory allocation.

Answer :

Model Paper-I, Q7

Advantages

(i) The implementation of contiguous memory allocation is simple.

(ii) It supports fast sequential and direct access.

(iii) It provides a good performance.

(iv) The number of disk seek required is minimal.

(v) It does not require expertise to understand and use such a system.

Disadvantages

(i) It does not utilize the memory efficiently.

(ii) It does not utilize the processors efficiently.

(iii) Fragmentation.

(iv) The user's process is limited to the size of the available main memory.

Q4. What is fixed partitioning?

Answer :

Model Paper-III, Q7

In fixed partitioning, main memory is divided into a number of fixed sized blocks called as partitions. When a process has to be loaded in main memory, it is allocated a free partition, which it releases while terminating and that partition becomes free to be used by some other process. The major drawback of this scheme is internal fragmentation. It arises when a memory allocated to a process is not fully utilized. For example, consider a system having fixed partitions of size 100 KB, if a process of 50 KB in allocated to one such partition, then it uses only 50 KB and remaining 50 KB is unnecessarily wasted. The figure (1) shows the problem with internal fragmentation.

Q5. Discuss in brief about segmentation.

Answer :

Model Paper-I, Q8

The programmer is allowed to view memory as consisting of multiple address spaces or segments through the concept of segmentation. Segments may be of unequal size. Memory references consists of a (segment number, offset) form of address.

The organization has a number of advantages. They are as follows,

(i) It simplifies handling of growing data structures.

(ii) It allows programs to be altered and recompiled independently, without requiring entire set of programs to be relinked and reloaded.

(iii) It lends itself to sharing among processes.

(iv) It lends itself to protection.

Q6. Define paging.

Answer :

Model Paper-II, Q8

It is a non-contiguous memory allocation scheme. It divides the physical memory into fixed-sized blocks called as frames and logical memory into pages. The page and block are of the same size. Hence, one logical page fits exactly in one physical block.

Each process 'P_i' residing on disk is composed of several pages. Whenever 'P_i' has to be executed, its pages are brought into main memory's frames. There is no restriction of pages being contiguous, they can be fragmented, here and there in main memory. Each process maintains a table which maps its page numbers to the block numbers they are residing in.

Q7. Differentiate between page and frame.

Answer :

	Page		Frame
1.	The process is split into several equal and fixed sized blocks called "pages".	1.	The main memory is split into several equal and fixed size blocks called "frames".
2.	Pages in fixed sized blocks are associated with logical memory.	2.	Frames are associated with physical memory.
3.	Pages are loaded into frames.	3.	Frames store pages.
4.	The logical address of page contain the pair (Page number, offset).	4.	The physical address of frame contain the pair (Frame number, offset).
5.	The page in logical address is identified by page number.	5.	The frame in physical address space is identified by frame address.

Q8. What are the merits and demerits of paging scheme?

Answer :

Merits of Paging

1. It avoids external fragmentation which means free frame can be allocated to a process which needs it.
2. It provides memory protection i.e., malicious tasks cannot harm the kernel.
3. It saves memory, when used for DLLs.
4. It conserves memory and prevents overload on CPU through demand loading.
5. It helps in running the process whose virtual address space is larger than physical memory.

Demerits of Paging

1. It consumes extra resources.
2. It incurrs memory overhead for storing page tables.
3. It incurs translation overhead.

Q9. What is the concept behind virtual memory?

Answer :

Virtual memory is a concept of giving programmers an illusion that they have a large memory at their disposal even though they have very small physical memory. Programmers can write programs that takes more memory than the available physical memory (RAM). This is achieved by storing their big programs in secondary memory or disk storage and then portions of these programs are brought into main memory whenever needed for execution. Virtual memory also allows processes to share files, libraries etc. The two fundamental techniques for implementing virtual memory are paging and segmentation.

Q10. What is Thrashing?

Answer : Model Paper-III, Q8

Thrashing refers to a situation wherein the operating system wastes most of its crucial time accessing the secondary storage, looking-out for the referenced pages that are unavailable in the memory. This situation (i.e., thrashing) can arise in both the demand paging system as well as in circular job stream.

In this situation, the OS swaps in the referenced page from secondary storage to primary while swapping out certain pages from the memory. In other words, thrashing is referred to a situation where the processor spends most of the time in the swapping of pages instead of executing the instructions.

PART-B

ESSAY QUESTIONS WITH ANSWERS

4.1 MEMORY MANAGEMENT

4.1.1 Basic Concept, Logical and Physical Address Map

Q11. Write in brief about background of memory management strategies.

Answer :

Memory is the control part of the computer system. it consists of huge array of bytes each with address. The CPU is responsible for fetching the instructions from memory based on program counter contents. Additional operations such as loading from memory and storing to memory need to done as per instructions.

An instruction execution cycle will initially fetch the instruction from the memory. It decodes the instructions, fetches the operands from memory and store the result back in memory after the execution is done on operands. Memory unit contains a set of memory addresses sequentially.

Memory can be managed in number of ways by using the memory management strategies such as paging, segmentation etc. Selecting a particular technique for a system will be based on various factors specifically on the system design.

Memory management has several issues such as basic hardware, binding of symbolic memory addresses to actual physical addresses and difference between the logical and physical addresses.

Address Binding

A program that is placed on a disk must be in the form of a binary executable file. For executing this program, it must be placed with a process in the memory. Based on the usage of memory, the process may be allowed to move between the memory and disk. An input queue is maintained for those processes that are waiting for execution in memory. Normally, only one process can be executed at a time. During execution, a process fetches instructions and data from memory and when the process terminates, its related space in memory becomes free so that the next process is brought into the memory for execution.

Generally, a user process can be placed in any part of the physical memory with the addresses assigned to it. Though the starting address of the physical address space is 00000, the first address cannot be stored at this location. Hence, the addresses used by the user program gets affected.

Logical Versus Physical Address Space

Logical address is defined as the address which is generated by CPU and physical address is defined as the actual memory address where data instruction is present. Both these address are common in certain address binding methods including compile-time and load-time methods whereas for exeuction-time they carry different addresses.

When the logical and physical addresses are different, the logical address is commonly called as virtual address. The term logical address space is usually referred to the group of addresses associated with a program whereas a logical address space is referred to the group of addresses associated with the logical addresses.

Usually, mapping can be done by using various methods but for run-time mapping of addresses from logical to physical is carried out through MMU (Memory Management Unit). The base register used in this case is referred to as relocation register because it stores its value in all the logical address spaces. This is done at the time of locating address in the memory. A typical MS-DOS operating system carries four register of this kind.

As the user program always uses logical address, it does not carry any information related to the physical addresses. To map to a physical address, it creates a pointer that carries the location of the register which keeps on comparing it with the other addresses.

In case of memory mapping the conversion of logical addresses to physical is done through hardware. User programs assume that they are associated with logical addresses only but they need to be allocated with physical addresses before accessing the memory.

Q12. Write short notes on,

(i) Dynamic loading

(ii) Dynamic linking

(iii) Shared libraries.

Answer :

(i) Dynamic Loading

A process needs the program and its associated data to be present in the physical memory of the system for its successful execution and hence, its size cannot exceed the size of physical memory. To overcome this, dynamic loading is used which stores the data associated with a process in main memory in the format that can be relocated. This approach results in executing the program routine only when its is called. Incase, that an executed routine wants to call other routine then it verifies that the desired routine already exist in the loaded routines. If it already exists, it directly executes it and if not, relocatable linking loader comes into action and the desired routine is loaded.

This approach avoids the loading of routines that are not required and hence, it is very useful in handling large amounts of code like error routines.

At the users point of view, there is no support provided by the operating system to dynamic loading. Therefore, to take the maximum advantage of its, they must design it in an efficient way.

(ii) Dynamic Linking

The concept of dynamic linking is similar to the concept of dynamic loading. The difference is that instead of postponding the loading of routines, it postponeds the linking of libraries until they are called. This feature eliminates the requirement of including language library associated with the program in the executable image.

With use of dynamic linking, unnecessary wastage of memory and disk can be avoided. This can be done by using a small code called stub in every library routine which is responsible for pointing out the location of memory resident library associated with the called routine. In addition to this, it is also responsible for checking the existence of routine in the memory and loading it if necessary. When the routine is to be executed it is done by placing its address at the place of stub and hence, this particular routine is executed directly from the next execution.

Dynamic loading is independent of operating systems help whereas dynamic linking checks the availability of needed routine in the memory space of to other processes with the help of operating system only. This is the case where each of the process is protected from every other.

(iii) Shared Libraries

While fixing bugs in libraries, there can be two types of modifications i.e., major and minor. Major modifications such as change in the program addresses typically changes (increments) the version number of the library whereas minor bug fixes do not change it.

4.1.2 Memory Allocation: Contiguous Memory Allocation, Fragmentation and Compaction

Q13. Explain briefly about the contiguous memory allocation.

Answer : Model Paper-4, Q15(a)

Contiguous Memory Allocation

The main memory of computer is divided into two major sections, one contains the Operating System (OS) and other is left for user processes. The OS is usually placed in starting locations or low memory area and an Interrupt Vector Table (IVT) is stored before OS.

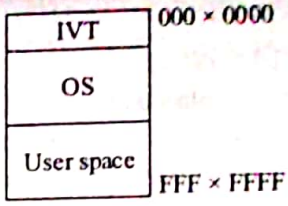


Figure: Main Memory Structure

Memory allocation means to bring the waiting processes from the ready queue to user space in main memory. When each process is allocated a single contiguous memory section then it is called as contiguous memory allocation. There are two variations in this scheme.

For remaining answer refer Unit-IV, Q15.

Memory Mapping and Protection

This is one of the important issue that arise during contiguous memory allocation. It refers to protecting the process from addressing an unspecified location which can be done using limit register and relocation register. Limit register contains the range of valid logical addresses and relocation register contains the starting physical address.

If the logical address entered by user is less than the value of limit register then it is a valid address and it is added with relocation address to map a physical location. If it is greater than limit, an addressing error is raised. Figure (2) shows memory mapping and protection.

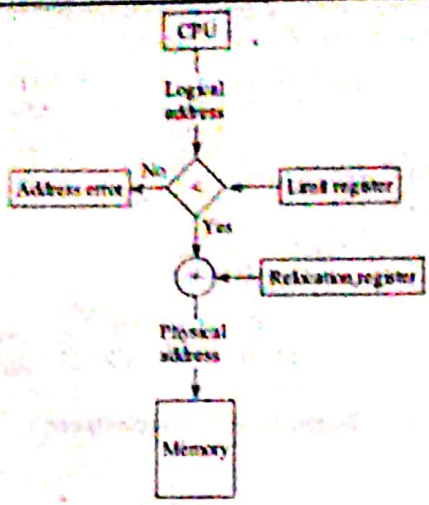


Figure: Memory Mapping and Protection

Q14. Explain the following allocation algorithms,

(a) First fit

(b) Best fit

(c) Worst fit.

Answer :

(a) First Fit Algorithm

In first fit algorithm, the memory manager scans along the linked list until it finds a hole that is large enough to store the program. Searching starts at the beginning of a set of hole and we can stop searching as soon as we find a free hole that is large enough. This algorithm is the fastest because it searches as little as possible.

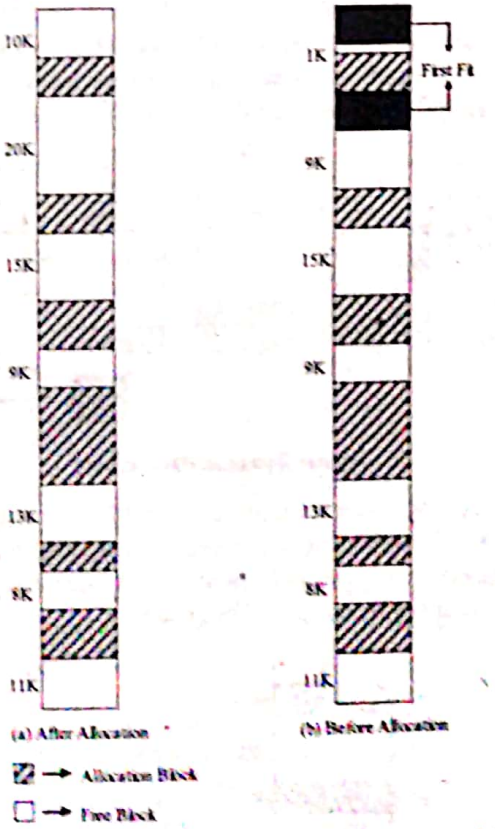


Figure: Example of Memory Configuration Before and After Allocation of 11 KB and 9 KB Block (Based on First Fit Algorithm)

Let us consider a linked list containing the following holes in the order specified below,

10 20 15 9 13 8 11

Assume that there are two programs waiting to enter into memory. The program sizes are 11 and 9 respectively. The memory manager allocates hole 20 to program of size 11 and hole 10 for program of size 9. Although, there are holes of sizes 11 and 9 in the linked list, the memory manager will not scan the entire linked list. It starts searching from the beginning of the linked list until it finds a hole whose size is greater than or equal to program size. Once, such a hole is found, the rest of the linked list is not scanned.

(b) Best Fit Algorithm

In best fit algorithm, the memory manager searches the entire linked list and takes the smallest hole that is adequate to store the programs.

Suppose, for the program of size 11, it scans the entire linked list and allocates the hole 11 for the program of size 11. For the program of size 9, it searches the entire linked list and allocates the hole of size 9.

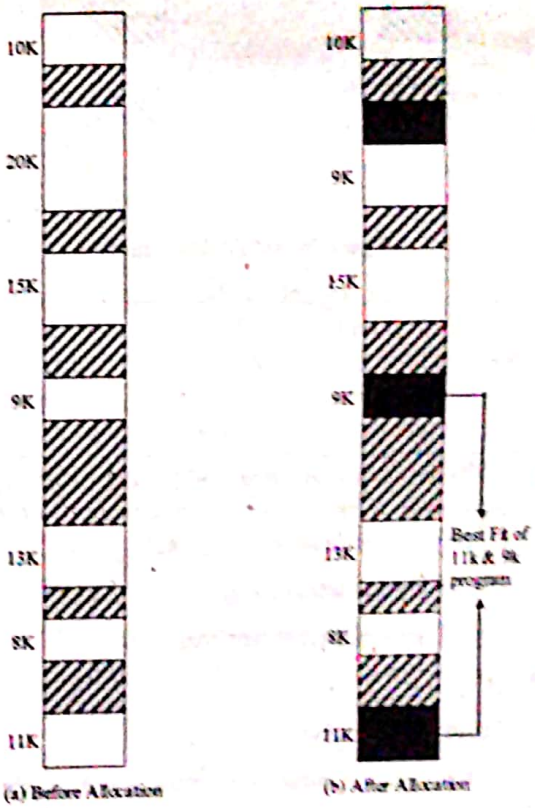


Figure: Example of Memory Configuration Before and After Allocation to 11 KB and 9 KB Block

Best fit algorithm is slow, because every time algorithm is called it scans the entire linked list. However, it results in minimal internal fragmentation because it allocates the best hole for the program. However, first fit algorithm may result in more internal fragmentation because it does not scan the entire linked list.

(c) Worst Fit Algorithm

In worst fit algorithm, the memory manager scans the entire linked list and allocates the largest hole to the program. For the program of size 11, it allocates the hole of size 20 which is maximum and for the program of size 9, it allocates hole of size 15 which is next to the largest size. Thus, the entire linked list is scanned twice. After allocation is performed, the remaining part of the hole can be used to allocate another program.

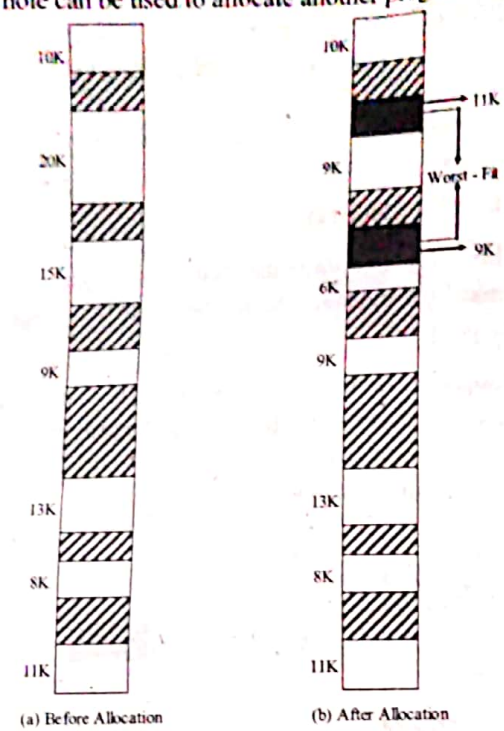


Figure: Worst Fit Algorithm

When best fit searches a list of holes from smallest to largest, as soon as it finds a hole that fits, it knows that the hole is the smallest one and that will do the job. No further searching is needed, as it is with the single list scheme. With a hole list sorted by size, first fit and best fit are equally fit and next fit is point less.

Q15. Write in brief on the following memory management techniques comparing their relative strengths and weaknesses,

(i) Fixed partitioning

(ii) Dynamic partitioning.

Answer :

(i) Fixed Partitioning

In fixed partitioning, main memory is divided into a number of fixed sized blocks called as partitions. When a process has to be loaded in main memory, it is allocated a free partition, which it releases while terminating and that partition becomes free to be used by some other process. The major drawback of this scheme is internal fragmentation. It arises when a memory allocated to a process is not fully utilized. For example, consider a system having fixed partitions of size 100 KB, if a process of 50 KB in allocated to one such partition, then it uses only 50 KB and remaining 50 KB is unnecessarily wasted. The figure (1) shows the problem with internal fragmentation.

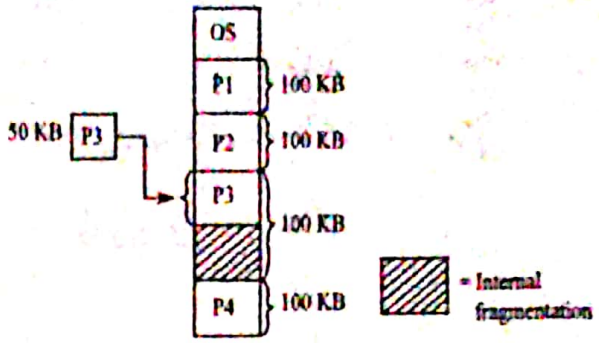


Figure: Internal Fragmentation

Strengths

- Fixed partitioning is simple and easy to implement.
- It helps in efficient utilization of processor.
- It supports multiprogramming.

Weakness

- Inefficient use of memory due to internal fragmentation.
- The number of partitions specified at the time of system generation limits the number of active processes.

(ii) Dynamic Partitioning

In dynamic partitioning, partitions are created dynamically depending on the size of process being loaded. The partition size is exactly equal to the size of process being loaded. For example, initially the memory will be empty, then a process of 200 KB is loaded into memory and is allocated 200 KB. Then some other process of 500 KB is loaded which is also allocated 500 KB. Likewise more two processes are loaded of size 300 KB and 100 KB. Figure (2) shows the allocation sequence.

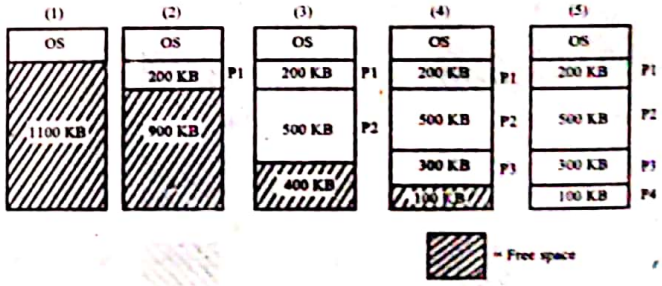


Figure: Dynamic Partitioning

Consider that process P1 and P3 are completed and terminates by releasing the memory occupied by them, then the memory status would be as shown in figure (3). Two holes or free memory sections will be created.

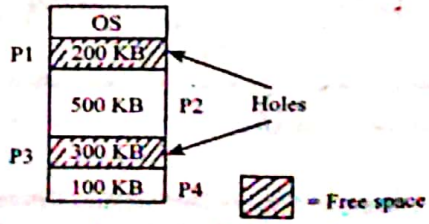


Figure: External Fragmentation

Now, suppose that a new process P5 of size 500 KB wants to get loaded, but that cannot be done because we don't have a single contiguous 500 KB free space. Although we have 500 KB free space but it is fragmented and not contiguous hence, cannot be allocated. This problem is called external fragmentation, which is a major drawback of dynamic partitioning.

One of the solution to external fragmentation is to apply memory compaction where kernel shuffles the memory contents in order to place all free memory partitions together to form a single large block. Compaction is performed periodically and it requires dynamic relocation of program. If compaction is applied on the previous example then memory status will be as shown in figure (4).

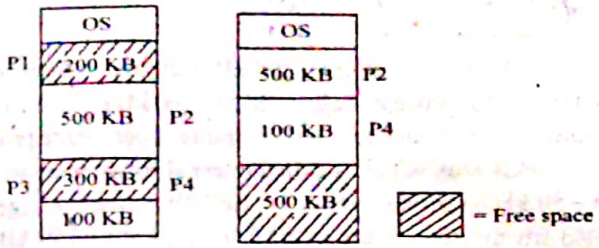


Figure: Memory Before and After Compaction

Strengths

- There is no internal fragmentation in dynamic partitioning.
- Main memory can be used more efficiently.

Weakness

Due to external fragmentation and use of memory compaction processor can not be used efficiently.

Q16. Explain different forms of fixed partitioning scheme with its advantages and disadvantages.

Answer :

Forms of Fixed Partitioning Scheme

The operating system occupies some fixed portion of main memory and the rest of the memory is available for user programs. The simplest scheme is to partition the memory into regions with fixed boundaries.

There are two forms of fixed partitioning. They are as follows,

1. Equal Size Partitions

This contains memory with partitions of equal size. A process whose size is less than or equal to the size of available partition can be allocated to it. In case that all the available partition get filled with various processes and none of them is active one of the processes is swapped out and a new process in inserted to make the processors busy all the time.

Advantages

(i) Supports multiprogramming.

(ii) Efficient utilization of processor and I/O.

(iii) Requires no special hardware.

(iv) Simple and easy to implement.

Disadvantages

(i) If a program is too large to set into that partition i.e., if program's size is larger than partition's size then, we cannot load that program into memory.

For example, let us assume a program of 600 KB. But, as shown in the following figure, partition size is 512 KB due to which we cannot load a 600 KB program.

(ii) Any program though too small occupies an entire partition. Thus, resulting in wastage of memory and leading to inefficient usage of memory.

For example, let us take a program of size 112 KB. If we load this program into a partition of size 512 KB as shown in the below figure, result is wastage of 400 KB memory which is undesirable. In this case, there is a wasted space internal to a partition, which is referred to as internal fragmentation.

2. Unequal Size Partitions

The problems of equal size partitioning can be partially overcome using unequal size partitioning.

As shown in the figure below, main memory is divided into partitions of unequal sizes. Here, a process is loaded into partition such that it minimizes wastage of memory. For example, if we take a process of size 120 KB, it can be loaded into 1 MB or 512 KB or 250 KB or either in 128 KB. But, the objective is to reduce memory wastage. So, we load that process into 128 KB partition as it results in 8 KB of memory wastage, which is more for other cases.

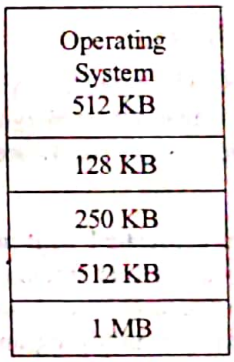


Figure: Unequal Size Partition

Advantages

In the fixed partitioning, unequal sized partitions usually provide a higher level of flexibility. The larger processes can be handled by providing one or two larger partitions. Further, the leftover partition space can be divided into multiple partitions, each of which is dedicated to a smaller sized process. Thus, supporting multiprogramming. This in turn reduces the internal fragmentation, because smaller partitions will be assigned to smaller processor.

Disadvantages

(i) The number of partitions specified at the time of system generation limits the number of active processes.

(ii) Since, partition sizes are present at the time of system generation, small jobs do not use partition space efficiently.

Q17. In the context of disk space allocation, what is fragmentation and compaction.

Answer :

Fragmentation

External fragmentation occurs when a request for a new file cannot be satisfied because the largest available continuous chunk of blocks is not sufficient enough for the file. Therefore even though there is enough space available for the file, but it is not utilized because it is contagious.

Compaction

External fragmentation can be resolved with the help of a scheme called "compaction". Compaction involves rearranging all free memory locations in a sequential order so that contagious disk space can be allocated for new files instead of space getting wasted. Compaction is carried out in both off line and online modes. In offline mode all operations are suspended and the file system is unmounted and finally compaction is performed. In this strategy a lot of time is wasted and hence it is avoided. In online mode compaction is performed along with other system operations but it effects business performance and reduces time wastages.

Q18. What is the difference between internal and external fragmentation?

Answer : **Model Paper-III, Q14(b)**

Memory Fragmentation

Fragmentation is defined as a wastage of memory space. It is a problem that occurs in dynamic memory allocation system when some blocks are too small to satisfy a request.

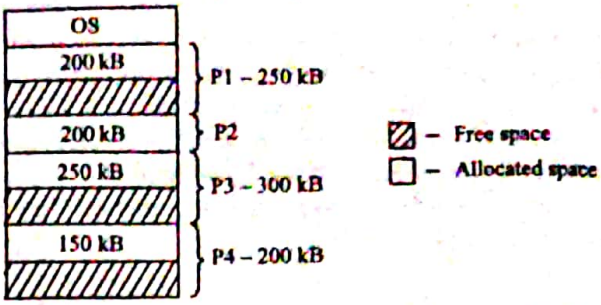
The problem associated with memory fragmentation particularly in allocating (or) freeing up the disk space is much similar with the problems associated with variable partitioning that exist in multiprogramming systems while allocating primary memory. Continuous allocation and clean up of memory fragments could result in the creation of huge number of fragments in the memory that can result in various performance problems.

(i) Internal Fragmentation

Internal fragmentation means wastage of memory space when a partiton is allocated with a process whose size is less than the partition. For example, in multiprogramming, with fixed number of partitions, a partition of 400 KB becomes free, then operating system scans through the queue and selects the largest job i.e., 385 KB job and loads it into 400 KB partition. In this case 15 KB memory is being wasted. This is called internal fragmentation. In general, if there is a partition of 'm' bytes and there is a program of size 'n' bytes where $m > n$ and a program is loaded into the partition, then internal fragmentation is equal to $(m - n)$ bytes.

(ii) External Fragmentation

When dynamic partitioning is used for the allocation of processes, some of the memory space could be left over after the allocation of each and every frame. For instance consider the example.



In the above example, 200 kB is alloted to a partition of 250 kB, 250 kB process is allocated to a 300 kB partition and 150 kB process is allocated to a 200 kB frame. These three processes waste 50 kB each which is called internal fragmentation. Now, these 50 kB partitions cannot be used by a process larger than 50 kB irrespective of the overall free space of 150 kB. This problem is referred to as external fragmentation.

If there is a partition of size 'm' bytes and there are no programs in the queue of size $\leq m$ bytes, then the entire partition is left empty until a job of size $\leq m$ bytes arrives into the queue. Thus, external fragmentation is equal to 'm' bytes i.e., the entire partition.

For example, if 100 KB partition becomes free and there are no programs in the queue of size $\leq$ 100 KB, then the entire 100 kB partition is left empty. This is called external fragmentation. One solution to the problem of external fragmentation is compaction.

4.1.3 Paging: Principle of Operation - Page Allocation - Hardware Support for Paging, Structure of Page Table, Protection and Sharing, Disadvantages of Paging

Q19. What is paging? Explain the basic method for implementing paging.

Answer : **Model Paper-III, Q14(a)**

Paging

It is a non-contiguous memory allocation scheme. It divides the physical memory into fixed-sized blocks called as frames and logical memory into pages. The page and block are of the same size. Hence, one logical page fits exactly in one physical block.

Each process 'P_i' residing on disk is composed of several pages. Whenever 'P_i' has to be executed, its pages are brought into main memory's frames. There is no restriction of pages being contiguous, they can be fragmented, here and there in main memory. Each process maintains a table which maps its page numbers to the block numbers they are residing in.

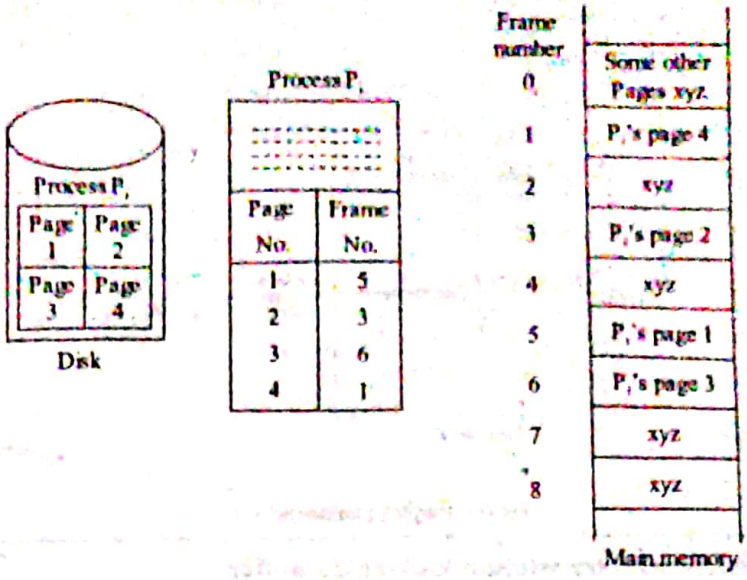


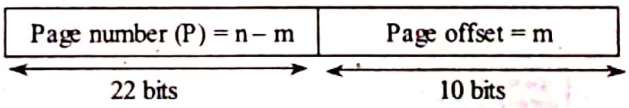
Figure: Mapping Pages to Frames

Paging Implementation

In basic implementation of paging, the physical memory is divided into fixed -sized blocks called frames and logical memory into pages.

There is a page table available which stores the base address of each page available in main memory and the offset act as descriptor within the page. The base address is combined with offset to get address of a physical memory location.

The system makes use of a paging table to implement paging. When a process is to be executed, its pages are loaded into free frames in the physical memory. The information about frame number, where a page is stored is entered in the page table. During the process execution, CPU generates a logical address that comprises of page number (*P*) and offset within the page (*d*). The page number *p* is used to index into a page table and fetch corresponds frame number. The physical address is obtained by combining the frame number with the offset. Logical address consists of page number and page offset



P = Index into the page table

D = Displacement within the page

Size of the logical address = n = 32

Page size = $2^m = 2^{10}$ = 1024 bytes

Number of bits to represent page offset = m = 10

Number of bits to represent page number n – m = 22

The lower order bits of a logical address represent page offset and higher order bits represent page number. Max size of logical address space is 2^{32} bytes i.e., 4 G bytes.

So the maximum length of a page table of a process = 4 m entries, each entry being 4 bytes so a page table would occupy 16 M bytes in RAM.

There is a page table available which stores the base address of each page available in main memory and the offset act as descriptor within the page. The base address is combined with offset to get address of a physical memory location. The figure (2) shows the hardware requirement of paging scheme.

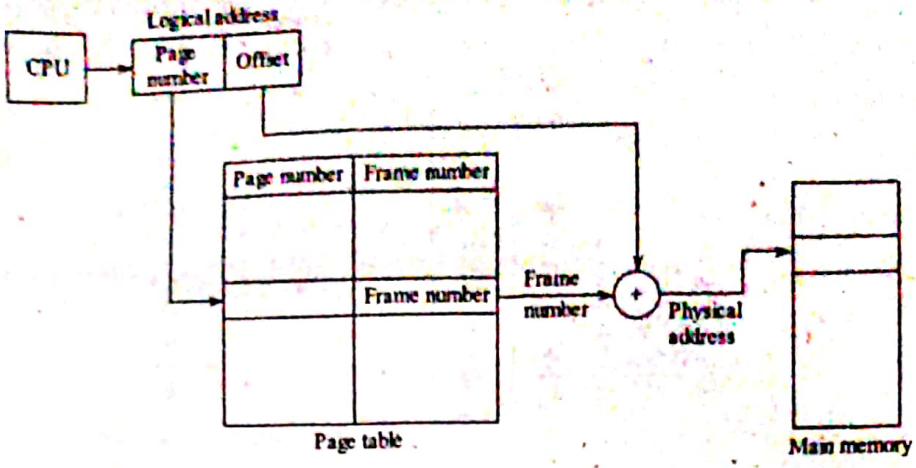


Figure: Paging Implementation

Q20. Explain paging hardware with translation look-aside buffer.

Answer :

Hardware Support

The page table implementation is done in several ways by various operating systems. The simplest way is to use a set of registers with a high speed logic to translate addresses easily and efficiently. But such implementation cannot store more than 256 page table entries whereas today's computers require nearly 1 million page table entries which is infeasible to be implemented in hardware. Some systems store page table in memory and store its address in a special register called as Page Table Base Register (PTBR).

One of the feasible solution to this problem is to use a fast, small, associative cache memory called as Translation Look-aside Buffer (TLB) to look up and translate addresses. A TLB entry is divided into two parts, a key and a value. When key is provided to TLB it looks up for it, simultaneously in all entries (i.e., a typical property of associative memory) and returns its corresponding value field if found (TLB hit). Otherwise, if it is not found (TLB miss), then a page table present in main memory is used to map that logical address to physical address. The following figure shows the implementation of paging using TLB.

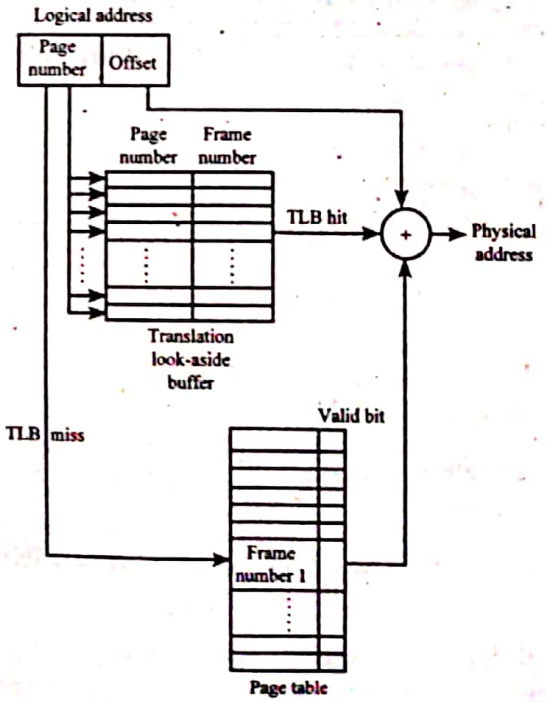


Figure: Paging using TLB

Q21. Write short notes on page table structure.

Answer :

Translation from logical address consisting of page number and offset into a physical address consisting of frame number and offset should be done in order to read a word from memory. Because page table is of variable length, it cannot be held in registers and must be in main memory.

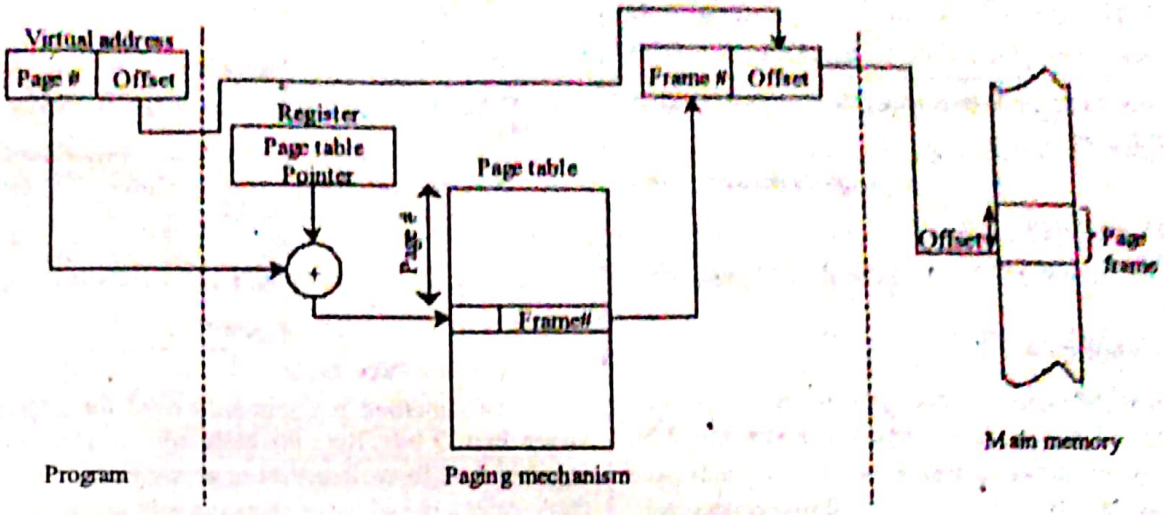


Figure: Address Translation in a Paging System

When a particular process is being executed, a register holds the starting address of page table for that process. Page number of a virtual address is used to index the table and look up the corresponding frame number. This is combined with the offset portion of virtual address to produce desired real address as shown in the figure. Typically, the page number field is longer than the frame number field ($n > m$).

Here, the disadvantage is that the amount of memory used up by the page tables alone can be high. To overcome this, virtual memory schemes store page tables in virtual memory than in real memory and so the page tables are also subjected to paging. When a process is running, a part of the page table including page table entry of the currently executing page must be in main memory. Some processors make use of a two level scheme to organize large page tables. In this, there is a page directory in which each entry points to a page table is present. So, if length of page directory is 'x' and if maximum length of it is 'y', a process will contain upto $x * y$ pages. Usually, length of page table is restricted to be equal to one page.

Q22. What elements are typically found in a page table entry? Briefly define each element.

Answer :

Page Table Entry

A page table is the basic requirement of each and every process. The elements incorporated in the page table entry are as follows,

(i) Frame number

(ii) Present bit

(iii) Modify bit.

(i) Frame Number

Frame number can be identified as the sequential number assigned to each and every individual page in the main memory.

(ii) Present Bit

Present bit is identified as an element that assures the presence of a page in the main memory. In other words, since the pages in the main memory are confined to a limited extent, each page table entry is provided a bit that indicates the presence of a particular bit in the main memory, thereby indicating even the frame number of that particular page.

(iii) Modify Bit

Modify bit of the main memory can be identified as the element that specifies the modifications (if any) performed on the contents of the page. However, if there are no modifications done on the page, then there is no requirement of performing the write operation on the page while it is being replaced into the frame.

Q23. Explain various techniques for structuring of the page table.

Answer : Model Paper-III, Q15(a)

Structure of Page Table

There are several techniques to implement page table. Some of them are,

1. Hierarchical Paging

Logical address space in modern computer is very vast like 2^{32} to 2^{64} bytes. Let us take a computer which supports 32-bit logical addresses with a page size of 4 KB (2^{12}). Then its page table may contain nearly 1 million entries. However, this huge page table cannot be stored in contiguous memory location. It will be decomposed into smaller pieces and managed in several ways as follows,

- In a two-level paging algorithm, the page table is mapped into another page table. The logical address is divided as shown in figure (1),

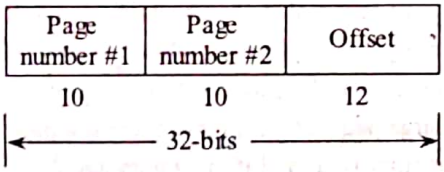


Figure (1)

Page number is indexed into an outer table and its result is indexed into an inner page table. This method is also called as forward-mapped page table. The following figure (2) shows the same.

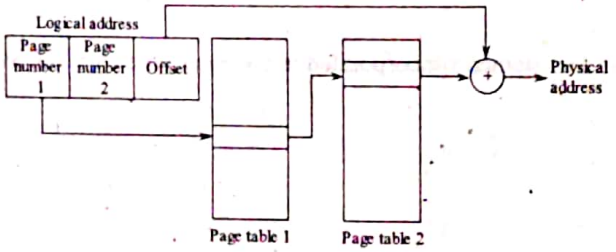


Figure (2)

- The VAX processor divides the 32-bit logical address into three parts as follows,

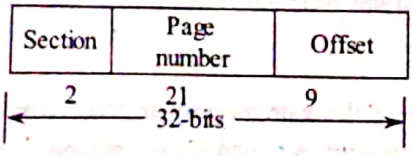


Figure (3)

- For systems with logical-address space more than 32-bits, more levels of paging is used. A 64-bit logical address is divided as follows,

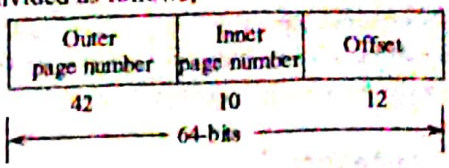


Figure (4)

Since, the outer page is still large, we divide it further as follows.

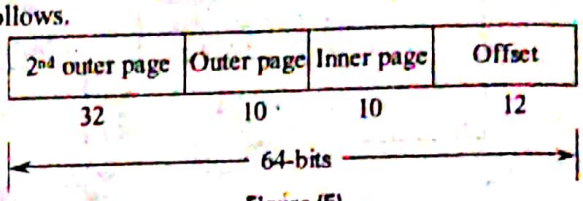


Figure (5)

2. Hashed Page Table

This method is commonly used for address spaces larger than 32 bits. Here the hash value is assumed to be the page number. To avoid collision, we use hashing with chaining that creates a linked list of elements with the same value. Each element has three fields. They are,

(i) A page number
(ii) The value of mapped page frame
(iii) A pointer to next element.

The page number is given to a hash function which indexes to a particular entry in hash table. The first element of that entry is compared with page number if it is matched, its mapped frame value is taken otherwise we move to next element by using its link part. Then at the next element same procedure is applied until the value is matched.

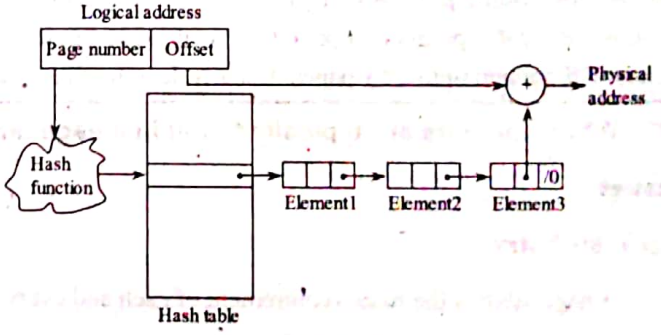


Figure (6)

3. Inverted Page Table

Normally, each process owns certain pages and a separate page table to map these logical page addresses to physical addresses. But each page table may consists of millions of entries out of which a process would be using a few entries. If multiple processes are present then the overhead of maintaining multiple page tables also increases. This representation may consume huge memory space unnecessarily.

Hence, inverted page table is used, where we have only one page table keeping track of all pages in physical memory. Each entry in inverted page table stores process-id (pid) of the process to which it belongs. The format of virtual address is as follows,

Process-id	Page number	Offset

The process-id and page number is used to index into page table and get the frame number which is then combined with offset to generate a physical address. The figure (7) shows the same,

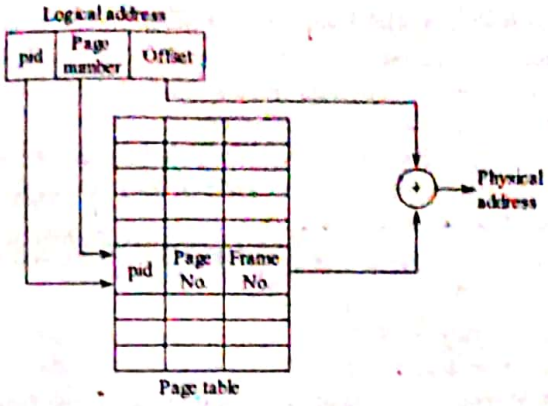


Figure (7)

Q24. Explain how sharing of pages is accomplished in a paged environment.

Answer :

Sharing of Pages

Sharing is one of the important advantage of using paging scheme. The reentrant code (or pure code) is one which is non-modifying code. For example in a compiler application, we will submit our source program to it and it generates equivalent object code, but the compiler IDE is not modified. Such reentrant code can be shared among several processes, other applications that can be shared includes compilers, run-time libraries, database system etc.

Consider an example, where we have two processes which are executing a compiler application of size three pages. Since, the compiler code is reentrant we can share it between them.

In addition to this, each process has a non-shareable page to store their source program. The following figure shows sharing of pages by two different processes. Here we need five pages instead of eight, thereby saving three pages.

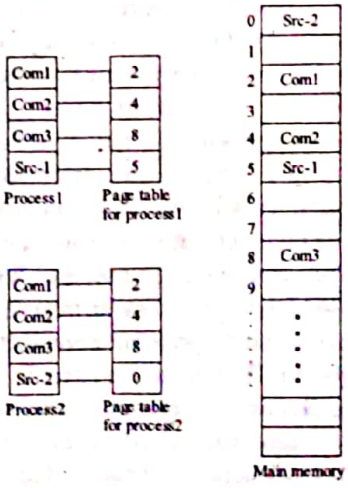


Figure: Sharing Pages among Processes

Q25. What is meant by paging? Explain its advantages and disadvantages.

Answer :

Paging

For answer refer Unit-IV, Q19, Topic: Paging.

Advantages of Paging

1. It eliminates segmentation
2. It shares memory between tasks and allows high degree of multiprogramming.
3. It increases memory and processor utilization.
4. It reduces CPU load.
5. It also eliminates the compaction overhead required for the relocatable partition scheme.

Disadvantages of Paging

1. The cost of the computer get increases due to page address mapping hardware.
2. The most of the memory is used to store the various tables such as, page table and memory map table.
3. It is translation overhead.
4. If the number of available block is not sufficient for the address spaces of the tasks to be executed, then some memory space will remain un used.

4.2 VIRTUAL MEMORY

4.2.1 Basics of Virtual Memory

Q26. Explain briefly about virtual memory.

Answer : Model Paper-II, Q15(a)

Virtual Memory

Virtual memory is a concept of giving programmers an illusion that they have a large memory at their disposal even though they have very small physical memory. Programmers can write programs that takes more memory than the available physical memory (RAM). This is achieved by storing their big programs in secondary memory or disk storage and then portions of these programs are brought into main memory whenever needed for execution. Virtual memory also allows processes to share files, libraries etc. The two fundamental techniques for implementing virtual memory are paging and segmentation.

The virtual address space is the set of logical addresses used by a process. Physical address space is the set of effective memory address of an instruction or data byte where they are actually located. The Memory Management Unit (MMU) maps the virtual addresses to respective physical addresses.

Consider the following figure that shows the dynamic memory allocation of virtual memory.

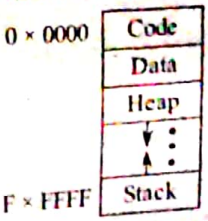


Figure: Organization of Virtual Address Space

The above organization allows heap to grow downwards and stack to upwards. The empty space (hole) between stack and heap is the virtual address space, such type of space is also known as sparse address space. The advantage of using virtual memory is sharing of pages and leads to following benefits,

- Several processes can share system libraries by mapping them into virtual address space. These libraries are stored as pages in physical memory.
- Inter process communication can be done by sharing virtual memory among several processes.
- Process creation can speed up by sharing pages with fork( ) system call.

4.2.2 Hardware and Control Structures

Q27. Why is the principal of locality crucial to the use of virtual memory? What is accomplished by page buffering?

Answer :

Principle of locality is very important for increasing the performance of two level memory. According to the principle, memory references tends to cluster (a group of things of similar type that grow or appear close together). These cluster changes over a long duration of time and remain fixed over a short period of time.

Virtual memory that was based on either paging or segmentation has become an important component of operating system. Let us consider an example that specifies why virtual memory is essential?

Consider a process that is very large, that consists of larger program and number of arrays of data. If the time period is very short only small sub routines are executed and one or two arrays of data are only accessed. In this case, it is more waste to load large number of pieces when few pieces are enough for completing the execution of program. It is better to make use of main memory that stores only few pieces of process. If the required data item is not present in the main memory, a page fault occurs because of this fault, the operating system is informed to upload the desired page from the secondary memory.

Using the scheme of paging, time is saved and more processes can be stored in the main memory. Operating system is responsible for managing paging scheme.

One of the drawback of paging is thrashing, in which the system waste much of the time in swapping in/out the pages rather than executing instructions.

One way to avoid thrashing problem is to make use of recent history that specifies which pages are not likely to be used in the near future. This is based on the principle of locality which states that the program and data references within a process tend to cluster. Thus makes the assumption valid that only few pieces of process will be required over short period of time.

In order to avoid thrashing, operating system must be careful about swapping out the pieces based on their history. Principle of locality can be confirmed by examining the performance of process in the virtual memory.

Thus, principle of locality states that virtual memory schemes may work for making virtual memory practical, two important components are required.

1. Hardware support is required both for paging and segmentation.
2. A software support for managing the movement of pages or segment to/from the secondary memory and main memory is required that should be included in the operating system.

Q28. What are the three main issues of implementing a virtual memory system?

Answer :

Virtual memory is a concept which has the ability to address a storage space much larger than that available in the primary storage of a particular computer system. The idea behind virtual memory is that since, the user programs exceeds memory capacity, operating system keeps the parts of program currently in use in main memory and the rest on disk.

The key to virtual storage concept is disassociating the address referenced in a running process from the address available in primary storage. It is the separation of user logical memory from physical memory. Thus separation allows extremely large virtual memory to be provided if memory is available. Virtual memory makes the task of programming much easier because the programmer no longer needs to worry about the amount of physical memory available.

Virtual memory is present in the paging system. The paging system divides the program into a number of small parts and this division is done based on sorting the logical addresses, so the programmer does not have to make any decisions about how to divide the program. In fact, paging is entirely transparent to the programmer. The hardware is already checking each memory reference and picking up the page base address to use. Using the protection bits, it is already checking each page reference for validity. To achieve virtual memory, we simply need to drop the requirement that every page should be present in the memory while a program is running. When a process references a page that is not in memory, an interrupt is generated. This interrupt does not end the program instead, it is intercepted by the operating system. The operating system figures out which page was found to be missing, reads it from the disk, changes the page table to indicate that the page is now in memory and restarts the program instruction that caused the exception. The program doesn't even know that this happened.

For virtual memory to be effective :

(i) Hardware support must be there for paging or segmentation to be employed.

(ii) Operating system include software for managing movement of pages/segments between secondary memory and main memory.

Paging

For answer refer Unit-IV, Q19, Topic: Paging

Page Table Structure

For answer refer Unit-IV, Q23.

Q29. Write short note on translation look-aside buffer.

Answer :

Model Paper-I, Q14(a)

Translation Look-aside Buffer (TLB)

A reference to virtual memory causes two physical memory accesses-one to fetch the appropriate page table entry and one to fetch the desired data, which doubles memory access time. To overcome this, a special high-speed cache for page table entries called Translation Look-aside Buffer (*TLB*) is used. This *TLB* functions in the same way as memory cache and contains the page table entries which have been most recently used. Given a virtual address, processor first examines the *TLB*. If the desired page table entry is present (*TLB* hit), then frame number is retrieved and real address is formed and if it is not present (*TLB* miss), then processor uses the page number to index the process page table and examines the corresponding page table entry. If "present bit" is set, then page is in main memory and processor retrieves the frame number from page table entry to form the real address. Also, it updates, the *TLB* to include this new page table entry. If "present bit" is not set, then desired page is not in main memory and memory access fault called page fault is issued.

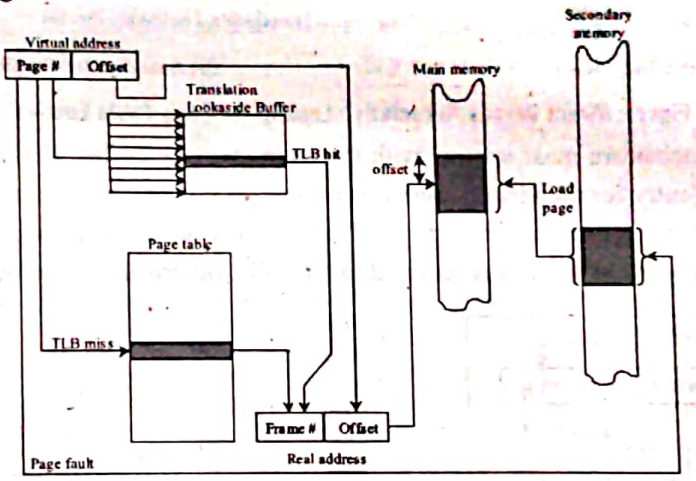


Figure: Use of Translation Look-aside Buffer

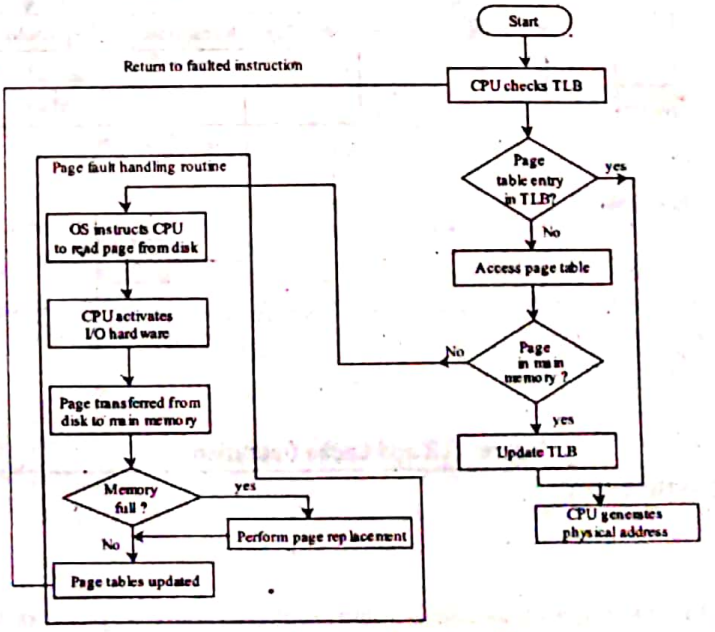


Figure: TLB and Paging Operation

The flow chart shows the use of *TLB*. It shows that if the desired page is not in main memory, a page fault interrupt causes page fault handling routine to be invoked. According to the principle of locality, most virtual memory references will be to locations of recently used pages. So, most references will involve page table entries in cache.

A *TLB* cannot be indexed based on page number as it contains some of the entries in a full page table. Each entry should include the page number as well as complete page table entry. The processor is equipped with hardware that allows it to simultaneously interrogate number of *TLB* entries to determine if there is a match on page number. This technique is referred to as associative mapping which is contrasted with direct mapping. The design of *TLB* must also consider the way in which entries are organized in *TLB* and which entry to replace when a new entry is brought in.

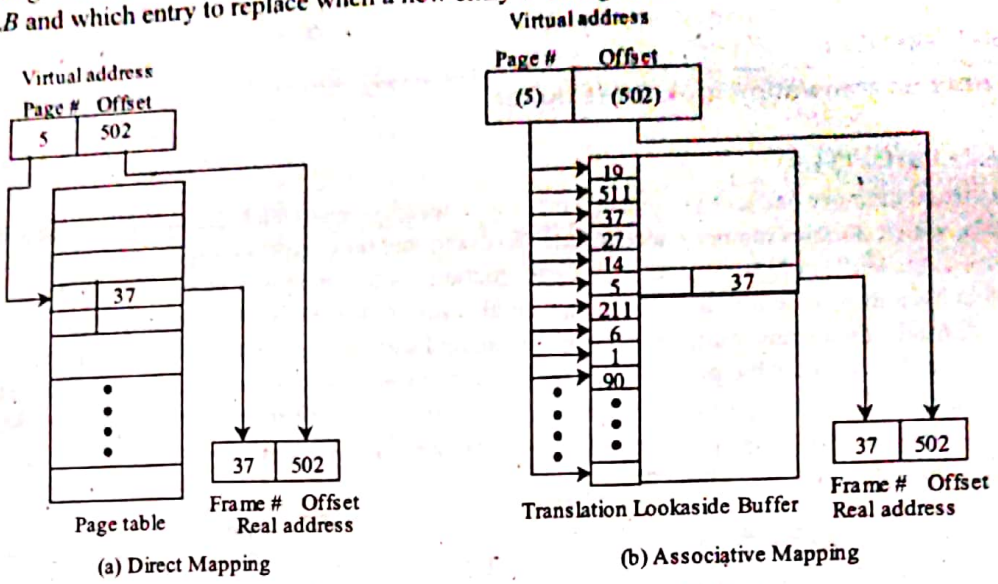


Figure: Direct Versus Associative Lookup for Page Table Entries

Finally, virtual memory mechanism must interact with the main memory cache. The memory system consults the *TLB* to see if corresponding page table entry for the virtual address is present. If it is present, real address is generated by combining frame number with offset. If not, entry is accessed from page table. Once the real address is generated, the cache is consulted to see if block containing that word is present. If so, it is returned to the CPU, otherwise it is retrieved from main memory.

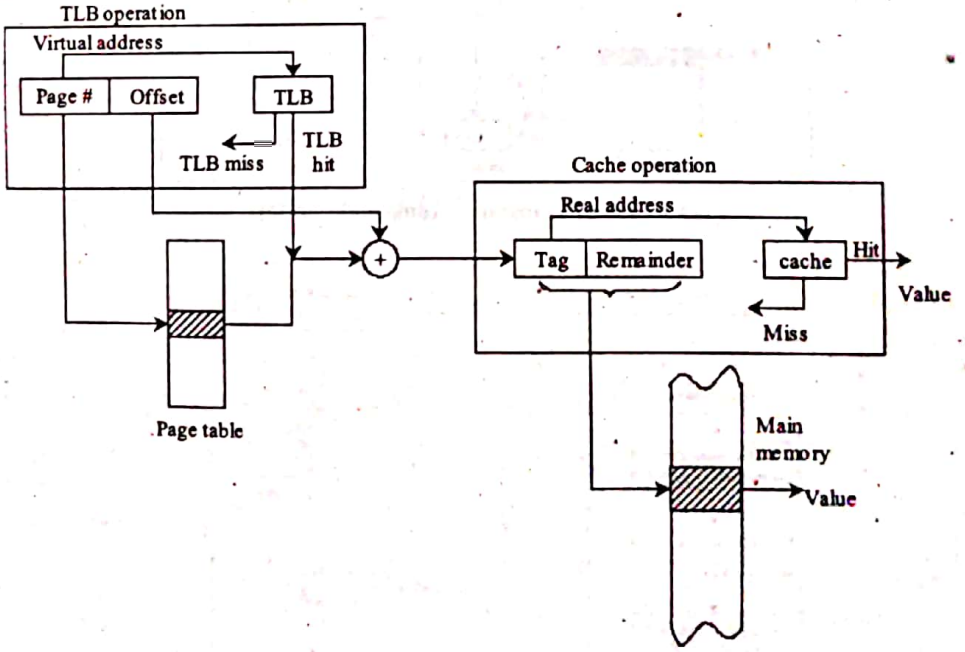


Figure: TLB and Cache Operation

Q30. Write short note on segmentation.

Answer :

Segmentation

Model Paper-I, Q15(b)

The programmer is allowed to view memory as consisting of multiple address spaces or segments through the concept of segmentation. Segments may be of unequal size. Memory references consist of a (segment number, offset) form of address.

The organization has a number of advantages. They are as follows,

(i) It simplifies handling of growing data structures.

(ii) It allows programs to be altered and recompiled independently, without requiring entire set of programs to be relinked and reloaded.

(iii) It lends itself to sharing among processes.

(iv) It lends itself to protection.

For a virtual memory scheme based on segmentation, there is a unique segment table for each process. Because only some of the segments of a process may be in main memory, a bit is needed to indicate if corresponding segment is present in main memory and if present, then the entry also includes starting address and length of the segment. There is also a modify bit indicating whether the contents of the segment have been altered, since the segment was last loaded into main memory. If protection or sharing is managed at segment level, there are other control bits for them.

Segment table is of variable length and so cannot be held in registers but must be held in main memory. When a particular process is running, the starting address of the segment table for that process is held in a register. The segment number of a virtual address is used to index the table and lookup the corresponding main memory address from the start of segment. This is added to the offset portion of virtual address to produce desired real address.

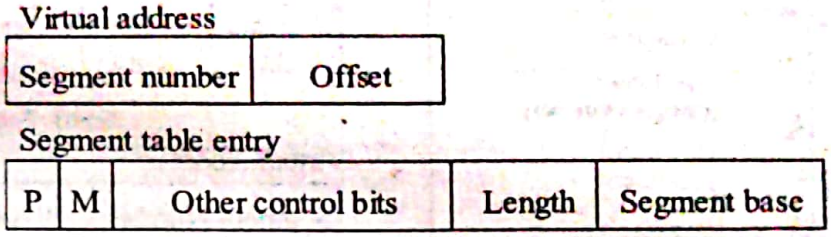


Figure: Memory Format for Segmentation

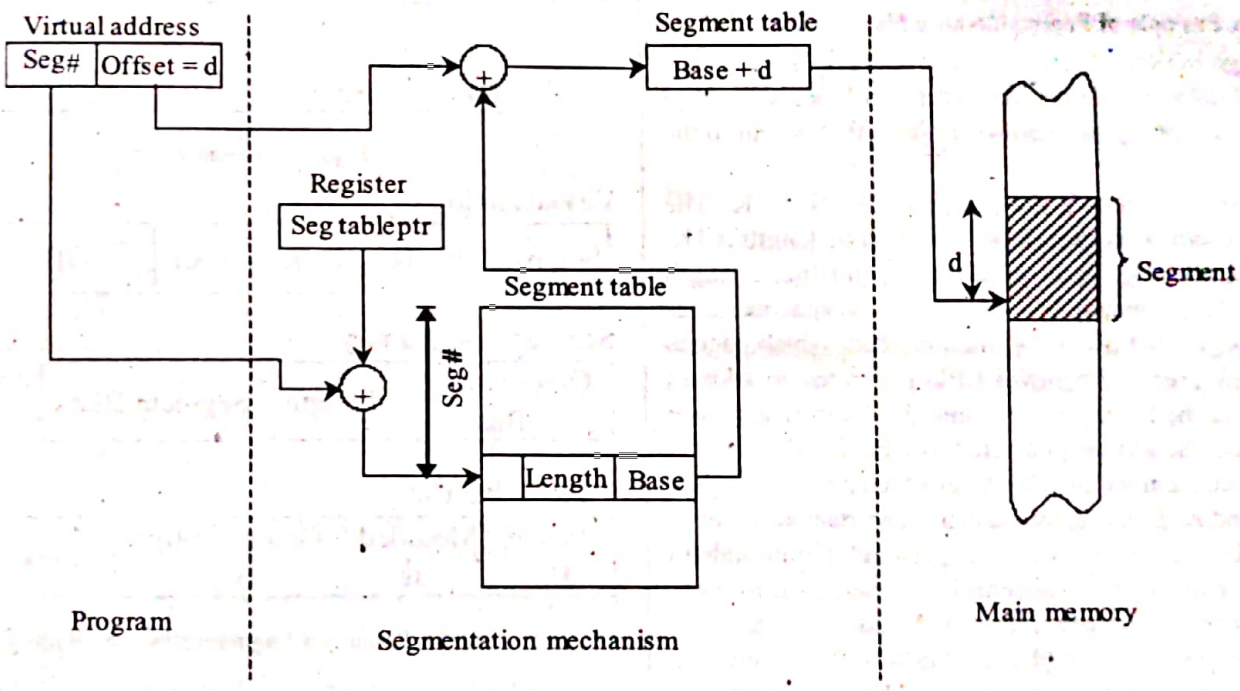


Figure: Address Translation in a Segmentation System

Q31. What is the difference between simple paging and virtual memory paging?

Answer :

Model Paper-II, Q15(b)

Simple Paging

Paging is a memory management scheme in which memory is allocated to the job at noncontiguous locations without combining them. In paging user develops programs thinking that he is developing a program for contiguous memory locations. However, memory is allocated to the program non-contiguously. Here, logical memory is defined as the view of the memory the user has, physical memory is defined as the actual memory in the system. For a program to run successfully, a mapping should exist between logical and physical memory. The user's job in the logical memory is divided into pages, each of equal length. Physical memory is divided into blocks called frames. The page length and frame length should be compatible. Memory allocation for a job is done through frames. At any point of time a frame may be free or busy.

Paging Memory Management

The memory manager maintains the total free frames available for allocating the frames. As we know that one page needs one frame, whenever a program is to be executed, the memory manager checks if the total number of free frames is greater than the number of pages in program. If so, the program is loaded into physical memory at free frames. The free frames may or may not be contiguous i.e., the program is stored in memory in noncontiguous frames. Once, the program is stored in physical memory, the details of allocation are stored in a table called 'page table'.

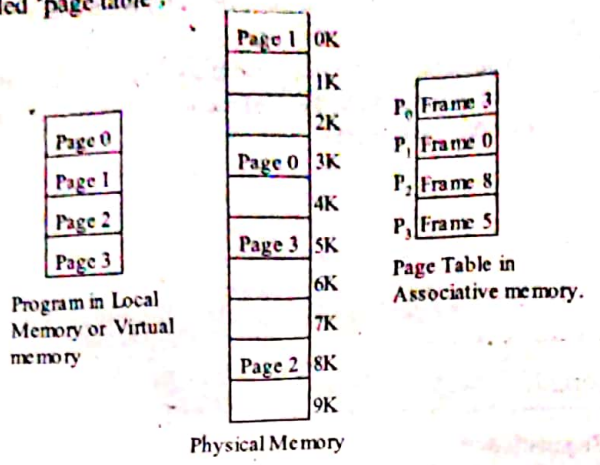


Figure: Example of Paging Memory Management

The page table contains the base address of each page in physical memory. This base address is combined with page offset to define the physical memory address that is sent to the memory unit.

In the above example the program size is '*n*' K. The program is split into '*n*' pages of 1K. Since, page length is 1K, frame length is also 1K since, page length and frame length must be same. The program is stored in memory in noncontiguous locations, the page table gives information about which page is present in which frame. Whenever CPU generates an address during execution, the main memory unit checks in which page this address falls. The address generated by CPU is called logical address. The main memory unit checks in which page this logical address falls and finds the displacement. The displacement is nothing but difference between logical address and starting address of the page. The main memory unit checks the page table to see in which frame this page is present. The starting address of the frame is added to displacement to get physical address. Thus, physical address is the starting address of frame in which the required page and displacement are present.

Displacement = Logical address – Starting address of required page

For example, let there be a logical address of 3092 falls in page 3. The starting address be 3072

Hence, displacement = 3092 – 3072 = 20.

Page 3 falls in frame 5. Starting address of frame 5 is 5K. Hence, physical address = 5K + 20 = 5120 + 20 = 5140. The physical address of logical address 3092 is 5140.

Paging

For answer refer Unit-IV, Q19, Topic: Paging.

Q32. Explain using illustrations typical memory management formats.

Answer : Model Paper-I, Q14(b)

Memory Management Formats

There are three memory management formats.

(i) Paging

(ii) Segmentation

(iii) Combined segmentation and paging.

These formats can be seen from the following figures,

Virtual Address

Page Number	Offset

Page Table Entry

Present Bit	Modified Bit	Other Control Bits	Frame Number

Figure: Paging

Virtual Address

Segment Number	Offset

Segment Table Entry

Present Bit	Modified Bit	Length	Segment Blue

Figure: Segmentation

Virtual Address

Segment Number	Page number	Offset

Segment Table Entry

Other Control Bits	Length	Segment Blue

Page Table Entry

Present Bit	Modified Bit	Other Control Bits	Frame Number

Figure: Combined Segmentation and Paging

These formats are described in detail as follows,

(i) Memory Management Format for Paging

For a virtual memory scheme based on paging, a unique page table is associated with each process. Since, only few pages of a process are in the main memory, a bit is present in each page table entry to indicate if the corresponding page is present in main memory or not. It is denoted as '*P*'. If the page is in the memory then the entry also includes frame number of the page. There is also another control bit in page table entry called the modify bit which indicates if the contents of this corresponding page have been altered since the page was last loaded in the main memory. If the paging or sharing is managed at page level, other control bits for that purpose may be present.

Where,

P = Present bit

M = Modify bit.

(ii) Memory Management Format for Segmentation

The programmer is allowed to view memory as consisting of multiple address spaces or segments, through the concept of segmentation, segments may be of unequal size. Memory references consist of a (Segment number, Offset) form of address. For a virtual memory scheme based on segmentation, there is a unique segment table for each process. 'P' bit is needed to indicate if corresponding segment is present in main memory and if present, the entry also includes starting address and length of the segment. 'M' bit indicates whether the contents of the segment have been altered, since the segment was last loaded into main memory. If protection or sharing is managed at segment level, there are other control bits for them.

(iii) Memory Management Format for Combined Paging and Segmentation

In a combined paging and segmentation system, the user's address space is broken up into a number of fixed-size pages which are equal in length to main memory frame. If segment has less than that of a page segment occupies just one page. A logical address consists of a segment number and a segment offset from the programmer's point of view, whereas from system's point of view, the segment offset is seen as a page number and page offset for a page within the specified segment. With each process, a segment table and a number of page tables are associated. A register holds the starting address of segment table when a process is running. The process uses the segment number portion to index into the process segment table to find page table for the segment. Then the page number is used to index the page table for the segment, then the page number is used to index the page table and lookup the corresponding frame number. This is combined with the offset portion of the virtual address to produce desired real address. Segment table entry contains length of segment and a base field which refers to a page table. For the purpose of sharing and protection, other control bits maybe used. Each page number is mapped into a corresponding frame number if page is present in main memory.

Q33. Write in brief on the following memory management techniques comparing their relative strengths and weaknesses.

(i) Fixed partitioning

(ii) Simple segmentation

(iii) Virtual memory paging

(iv) Dynamic partitioning.

Answer :

(i) Fixed Partitioning

In fixed partitioning, main memory is divided into a number of static partitions at system generation time. A process may be loaded into a partition of equal or greater size. There are two forms of fixed partitioning.

1. Equal size partitions
2. Unequal size partitions.

Strengths

1. Fixed partitioning is simple and easy to implement
2. It helps in efficient utilization of processor
3. It supports multiprogramming.

Weakness

1. Inefficient use of memory due to internal fragmentation
2. The number of partitions specified at the time of system generation limits the number of active processes.

(ii) Simple Segmentation

In simple segmentation, each process is divided into a number of segments. A process is loaded by loading all of its segments into dynamic partitions that need not be contiguous.

Strengths

1. In simple segmentation, there is no internal fragmentation. Hence, the memory can be used efficiently.
2. Improved memory utilization and reduced over head compared to dynamic partitioning.

Weakness

1. Unequal size segments can lead to complications.
2. Inefficient use of the processor.

(iii) Virtual Memory Paging

The main memory is divided into a number of equal size frames. Each process is also divided into a number of equal size pages which are of the same length as frames. In virtual memory paging, a process is loaded by loading the pages into available frames. Non resident pages which may be needed later are brought automatically.

Strengths

1. In virtual memory paging, there is no external fragmentation
2. Higher degree of multiprogramming is possible in this memory management technique.
3. Large virtual address space is created which can be used efficiently.

Weakness

Virtual memory paging can sometimes lead to complex memory management.

(iv) Dynamic Partitioning

In dynamic partitioning, partitions are created dynamically. Each process is loaded into a partition of exactly the same size as that process.

Dynamic partitioning was introduced in order to overcome the drawbacks of fixed partitioning.

Strengths

1. There is no internal fragmentation in dynamic partitioning.
2. Main memory can be used more efficiently.

Weakness

Inefficient use of the processor due to the need for compaction to counter external fragmentation.

Q34. (a) Enumerate the reason for allowing two or more processes to, all have access to particular region of memory.

(b) In a fixed partitioning scheme, what are the advantages of using unequal size partitions?

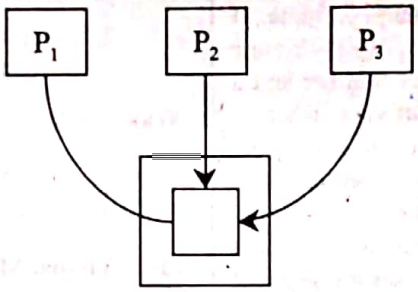
(c) What is the difference between internal and external fragmentation?

Answer :

(a) Sharing

While keeping in mind the protection mechanisms, the memory management scheme must also allow sharing the same portion of main memory by a number of processes i.e., being able to access the same memory by a number of processes.

Any protection mechanisms that are implemented must have the flexibility to allow several processes to access the same portion of main memory. For example, if a number of processes are executing the same program, it is advantageous to allow each process to access the same copy of the program rather than to have its own separate copy. Processes that are cooperating on some task may need to share access to the same data structure. The memory management system must therefore, allow controlled access to shared areas of memory without compromising essential protection.



It will be advantageous to share the same copy of a program rather than a separate copy for each process that requires this program. Controlled access to shared memory must be possible without losing any protection.

(b) Fixed Partitioning

For answer refer Unit-IV, Q16

(c) Memory Fragmentation

For answer refer Unit-IV, Q18.

Q35. Explain paging scheme for memory management, discuss the paging hardware and paging model.

Answer :

Paging Memory Management

For answer refer Unit-IV, Q31, Topic: Paging Memory Management.

Paging

For answer refer Unit-IV, Q19, Topic: paging.

Paging Hardware

Generally, CPU generates an address, which consists of two parts,

1. Page number (p)
2. Page offset (d).

Page number is used in a page table so as to provide an index to a particular page. Page table includes the base address, which is merged with the page offset. This merging is done in order to define the address of main memory that is transferred to the memory unit.

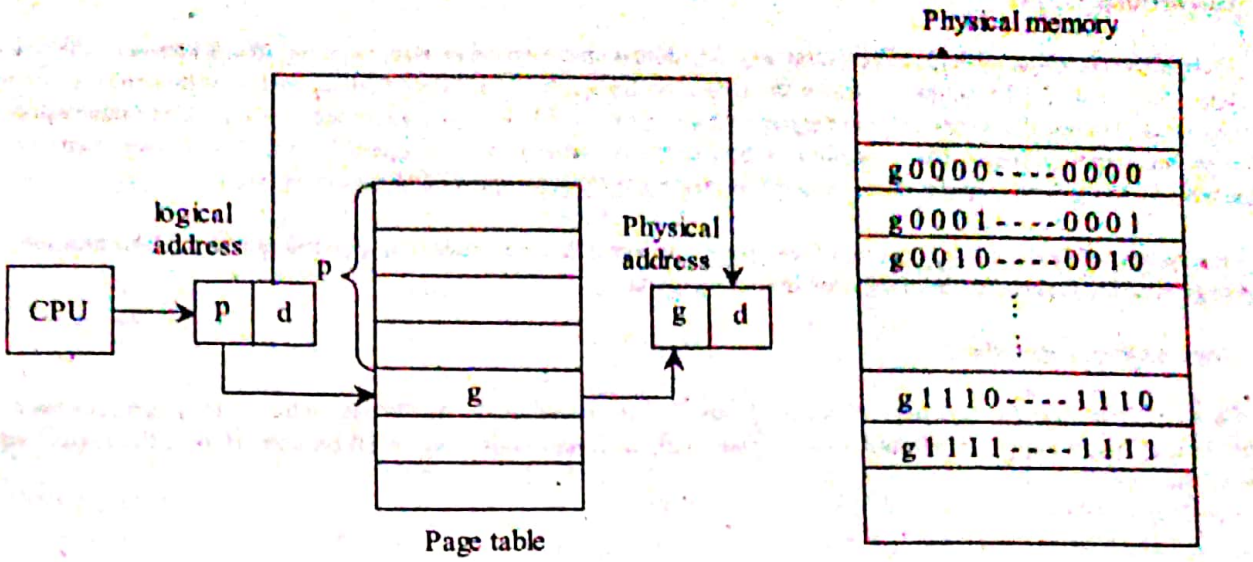


Figure: Hardware Support for Paging

Paging Model

The size of individual page is defined by hardware. The page size ranges from 512 bytes to 16 megabytes and varies from computer to computer depending on its architecture. It is expressed in power of 2 so as to perform easy translation of logical address into the corresponding page number and page offset.

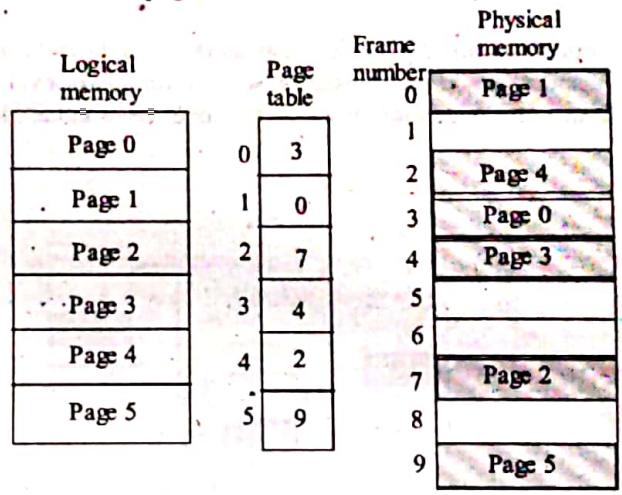
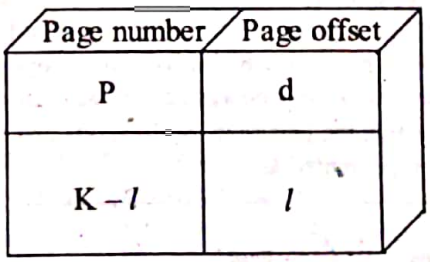


Figure: Logical and Physical Memory Paging Model

Consider logical address of size 2^k and page size of 2^l, then page number is represented by '$k - l$' high-order bits and page offset is represented by 'l' low-order bits.



Q36. Explain any two techniques for structuring the page table. Discuss with suitable examples.

Answer :

The most commonly used techniques for structuring the page table are,

1. Hierarchical paging and
2. Inverted page tables.

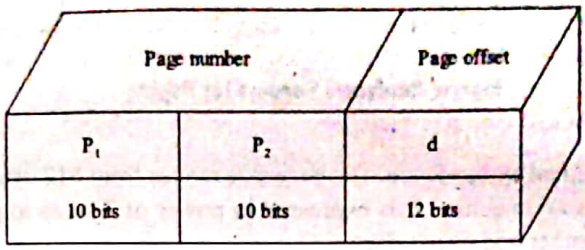
1. Hierarchical Paging

Most of the computer systems have a large logical address space stored in main memory, which increases the size of page table. Here, consider a 32-bit machine, in which the size of each page is 4 kB such that, at least 1 million entries can be stored in the page table. Assume that every entry in the page table consists of 4 bytes and each process in the system may require at least 4 MB of physical address space. The allocation of the entire page table is not contiguously available in main memory. Indeed, the page table is divided into smaller pieces and this division can be accomplished in several ways.

One method of division is to use a two-level paging, wherein the page table is also paged. Since, a 32-bit machine consists of 4 kB page size, the logical address fragmented into two parts,

<page-number, page-offset>

The size of page-number and page-offset are 20 and 12 bits, respectively. As the page table is also paged, the page-number is in turn divided into a page-number and a page-offset with each part consisting of 10 bit size. Hence, the logical address is represented as,



From the above logical address representation, P_1 is referred as an index in the outer page table and P_2 is referred as the displacement of the outer page table. Consider an address translation scheme for a two-level 32-bit paging architecture. In this, the working of address translation is from outer page table towards the inner page table. Hence, this mapping is known as a forward-mapped page table.

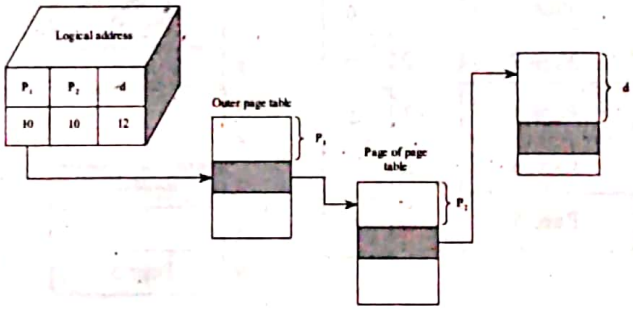
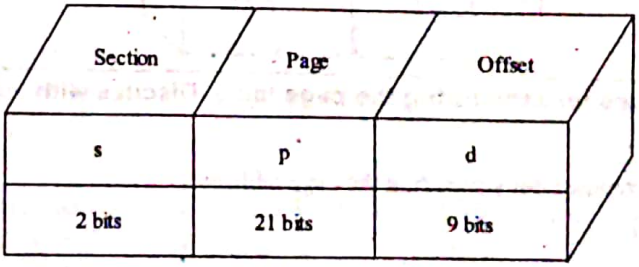


Figure: Forward-mapped Paging for a 32-bit Machine

A variation of two-level paging is also supported by VAX (Virtual Address Extension) architecture, which is also a 32-bit machine. In this, the size of each page is 512 bytes. A process containing logical address space is divided into four equal parts, wherein each section designates a different parts of process, which is of size 2^{30} bytes. In logical address, the first two most significant bits represent an appropriate section, the next 21 bits designate the page number and the final 9 bits designates an offset. When this kind of partitioning is done on page table, the partitions or divisions can be used only when a process requires them. A logical address on a VAX architecture can be represented as,



Where,

s denotes the section numbers

p denotes an index and

d denotes the displacement.

By implementing this scheme, the use of main memory is reduced since, one-level page table size for a VAX is 8 MB.

2. Inverted Page Tables

Each process in the system is associated with a page table, which in turn has an entry for each page. These pages are referred by their virtual addresses, which are translated to their respective physical address by operating system. The operating system then calculates the location of a physical address entry and use it by sorting the table according to the virtual address. The drawback of this method is that there exist some millions of entries in the page table, which may consume huge amount of physical memory space. The solution to this problem is to use an inverted page table. Unlike the other page tables, an inverted page table also consists of one entry for each page in it. The virtual address is stored in each entry of the desired page, along with the information of process. Therefore, there exist only one page table with one entry for each page in the system.

Usually, each page entry must contain an address-space identifier since, the page table consists of a set of addresses that are mapped to the physical memory. The storage of this identifier ensures that a page of logical address frame is mapped onto the corresponding page of the physical address frame. Some examples of inverted page tables includes UltraSPARC and PowerPC, which are 64-bit machine architectures.

The simplified version of the inverted table in IBM RT contains a process-id, a page-number and an offset, which are stored in the logical address. Each pair in the inverted page table in turn consists of a process-id and page-number. Here, process-id is considered as an address-space identifier. When there exists a reference of memory, then a pair of virtual addresses i.e., a process-id and a page-number is provided to the memory subsystem. A physical address <x,'offset> is generated when a match is found at an entry *x* after performing a search operation on the inverted page table. And, when there is no match, then the operation performed on the table is said to be an illegal access. Following figure shows the working of an inverted page table.

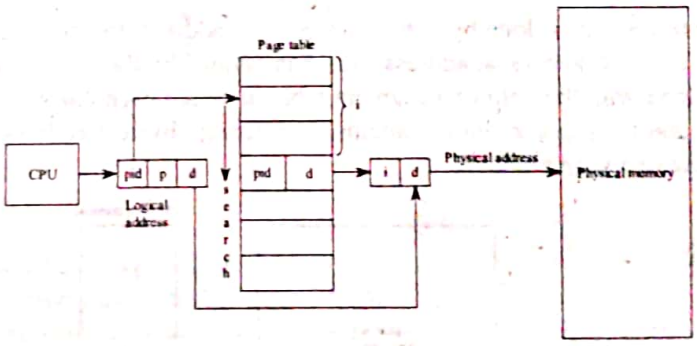


Figure: Inverted Page Table

This scheme reduces the storage amount of memory space. It increases the amount of time required for searching an entry to find a match for a page reference. The problem of space and time can be avoided by using a hash table technique. The virtual memory used in hash table requires at least two actual memory read operations, which are used for the entries in hash table and page table.

The inverted page table has only one virtual address that is mapped onto a single physical address, whereas, the shared memory requires multiple virtual addresses to be mapped onto the single physical address. When systems use inverted page tables, the implementation of shared memory becomes difficult. This difficulty can be overcomed by allowing the page table to have only a single mapping between a virtual address and a shared physical address. If there exists no mapping, then page fault occurs.

Q37. Explain segmentation scheme for memory management. Give the segmentation hardware.

Answer :

Memory management deals with the separation of user's view of memory and physical memory. The user's view is different from the actual physical memory. To differentiate between logical memory and physical memory, a mapping is done from user's view to the physical memory. The user's view of a program is represented in the figure (1).

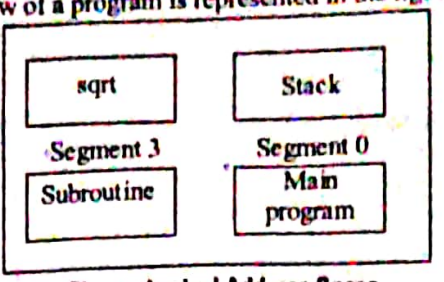


Figure: Logical Address Space

Segmentation is a scheme of memory management in which user's view of memory is considered. Typically, a logical address space contains a set of segments along with their specified name and length. Therefore, user specify the addresses of each segment by the following two attributes.

<segment-name, offset>

Instead of referring segment by a segment name, they are referred by a segment number. This helps in simplifying the implementation of segmentation. Hence, logical address can be defined by,

<segment-number, offset>

Usually, the user program is compiled by a C compiler, which in turn constructs the following segments to define an input program.

(i) The program code

(ii) Global variables

(iii) Heap memory

(iv) Stack memory

(v) Standard C libraries.

During compilation, if the linked libraries exist, then they may be assigned to separate segments. The loader helps in assigning the segment numbers to each of these segments.

Segmentation Hardware

Though the user's reference to objects is done by a two-dimensional address, the physical memory stores the addresses in a one-dimensional format. Thus, a two-dimensional address, which is defined by the user is mapped onto a one-dimensional physical address. This mapping is done with the help of a segment table in which each entry is defined by a segment base and a segment limit. The segment base and the segment limit contains the starting physical address and the length of the segment respectively. Figure (2) shows the use of a segment table.

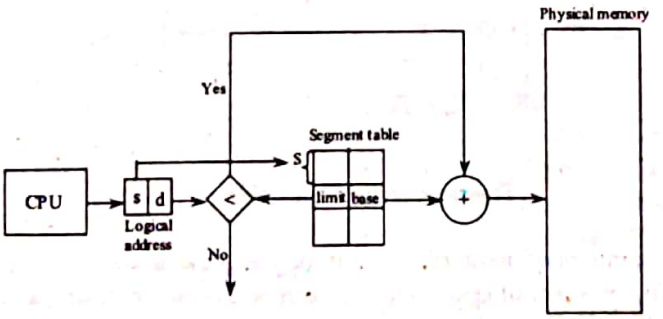


Figure: Trap Occurs to the Operating System

A logical address is defined by a segment number '*s*' and an offset '*d*', where '*s*' specifies an index of the segment table and '*d*' specifies the range between '0' and the segment limit. When the offset is not in the range, the operating system get trapped. And, when the offset is in the range, '*d*' is added to the base address for generating an address in the physical memory.

Consider a situation where segments are numbered from '0' through '3' in the logical address space. These segments are stored in the physical memory and each entry in the segment table specifies the starting address and limit of a segment. Figure (3) shows an example of segmentation.

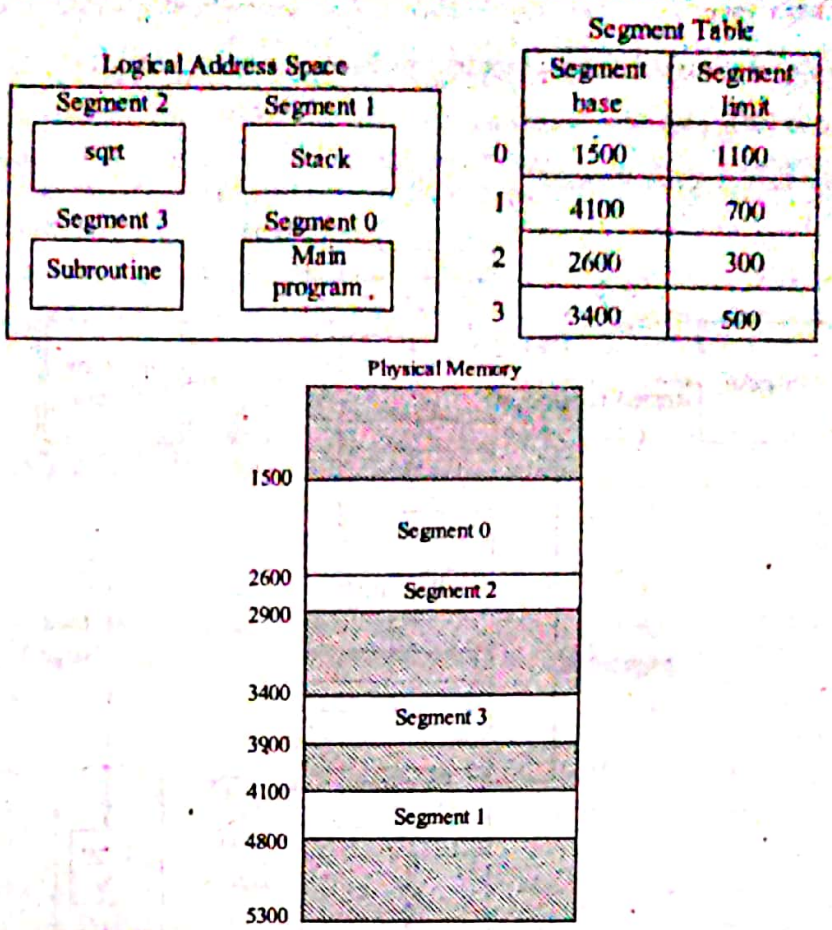


Figure: Segmentation

From the figure (3), segment 1 has a segment base (starting address) of 4100 and a segment limit of 700 bytes. Here, consider an offset of byte 23, which is less than the segment limit i.e., 700. Hence, this offset is added to the segment base 4100 + 23 = 4123 to generate an address of byte 23 in the physical memory. A reference to segment 3 of byte 765 is considered. Here, offset is beyond the segment limit i.e., 500. Hence, in this situation, a trap occurs to the operating system.

4.2.3 Locality of Reference, Page Fault, Working Set, Dirty Page/Dirty Bit

Q38. Describe briefly the following terms.

(i) Locality of reference

(ii) Dirty bit

(iii) Page Fault

Answer :

(i) Locality of Reference

Locality of reference gives the location of frequently executed/required instructions. When any of the computer programs are considered, their execution time is mainly due to the execution of routines present in them. These routines are nothing but simple loops, multiple loops or branch instructions etc. Hence, in any computer programs, these instructions are of primary focus, which gets repeatedly executed all the time and rest of the instructions are executed rarely. This behaviour of computer programs are often referred as locality of reference.

(ii) Dirty Bit

Dirty bit is a flag bit used to indicate the enhancement when a certain block of data is written only to cache memory, this enhancement is indicated by setting required bit of flag associated with the block.

(iii) Page Fault

When a process tries to access the page which is not fetched in memory then such condition is called 'page-fault' trap, which causes operating system to load the desired page into memory. The following are the sequence of steps are performed when a page-fault occurs,

1. The operating system notices the page-fault and verifies whether the reference is valid or not.

2. Then it searches for a free frame.
3. Schedules disk read operation to fetch desired page into the free frame found in step (2).
4. After successful read or fetch, page table is updated.
5. The process is restarted and it access the page.

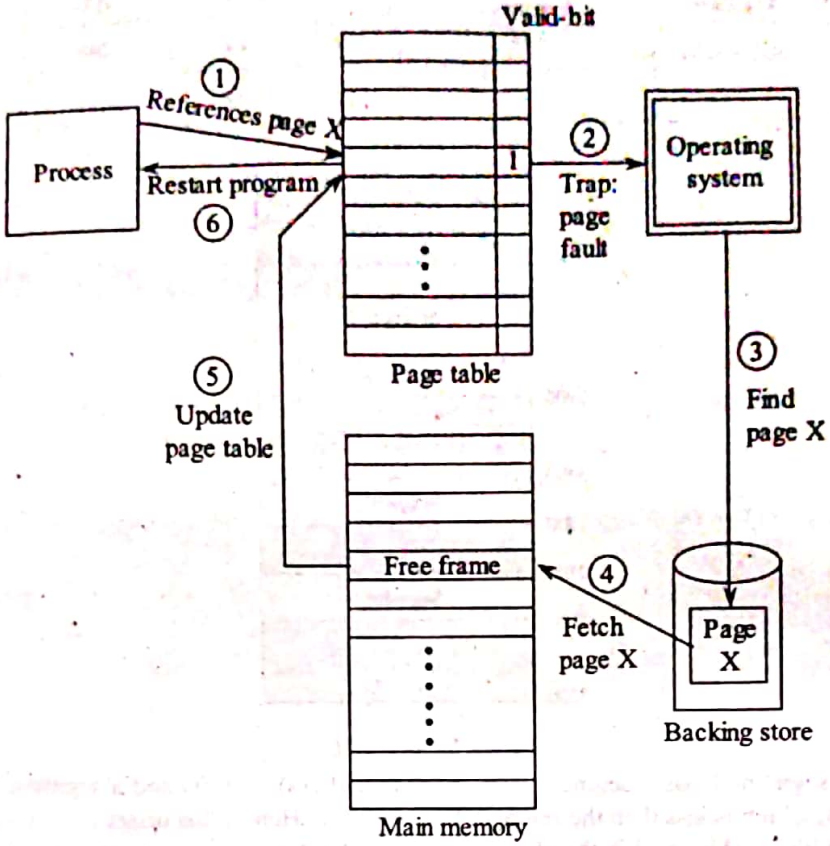


Figure: Demand Paging

Pure demand paging is a technique where a process starts execution without a single page in memory. As it proceeds execution, it gradually causes page-faults and those pages are fetched in memory. It never bring a page until it is required by some process.

The hardware that is necessary to implement demand paging is a page table for performing paging and swap space for performing swapping.

Q39 Define thrashing. Explain available methods to avoid thrashing.

Answer :

Working Set Model

Working set can be defined as the set of pages that a program is currently using (or) most recently used. It is denoted by 'Δ' parameter which carries the page references that are used recently. If a page reference in 'Δ' is not used for certain period of time, it is removed from the working set. For this reason, the working set is said to be the approximation of locality of program. This model is used to prevent thrashing.

The size of 'Δ' parameter is important for the accuracy of this model because if it carry very less number of page references, it will not consider overall pages and if it carry large number of references, pages might overlap. Moreover, if it carries infinite number of references, it refers to all the pages used during certain operation. For these reasons, the size of the working set model is considered as one of the most important property.

The demand for page frames (Δ) in a process (say i) can be given by the formula,

$$D = \Sigma W_{Si}$$

Where W_{si} is the size of working set of process i. Thrashing occurs if the D value exceeds the total number of frames due to their unavailability on some of the processes. The responsibility of allocating D value will be given to the operating system once Δ value is set. Based on availability of frames, the OS initiates the new process and if there are not enough frames, one (or) more processes will be terminated (or) suspended.

In case of suspending a process, it is swapped out and its associated frames are allocated to other processes and the suspended process gets swapped in when there are enough available frames. This increases the degree of multiprogramming and provide optimized CPU utilization.

One challenge in the working set model is to track the pages references because these references might change with every new reference while dropping the older reference. One solution to this is to use a timer interrupt with a fixed interval along with a reference bit.

4.2.4 Demand Paging

Q40. What is demand paging and how is it implemented?

Answer : **Model Paper-II, Q16(b)**

Demand Paging

Whenever a program has to be executed, its code which is stored in the form of pages in secondary memory has to be fetched into the main memory. The traditional fetching scheme is called prepaging which fetches all the pages with respect to that process irrespective of their need.

Another technique is called demand paging where pages are fetched into main memory only when they are needed by processes. Instead of fetching or swapping all the pages of a process, the lazy swapper here swaps only selected or required pages. It never swaps a page unless it is asked by some process. As the pages are getting swapped-in and swapped-out we use the word pager here instead of swapper.

When a process tries to access the page which is not fetched in memory then such condition is called 'page-fault' trap, which causes operating system to load the desired page into memory. The following are the sequence of steps that happen when a page-fault occurs,

1. The operating system notices the page-fault and verifies whether the reference is valid or not.
2. Then it searches for a free frame.
3. Schedules disk read operation to fetch desired page into the free frame found in step (2).
4. After successful read or fetch, page table is updated.
5. The process is restarted and it access the page.

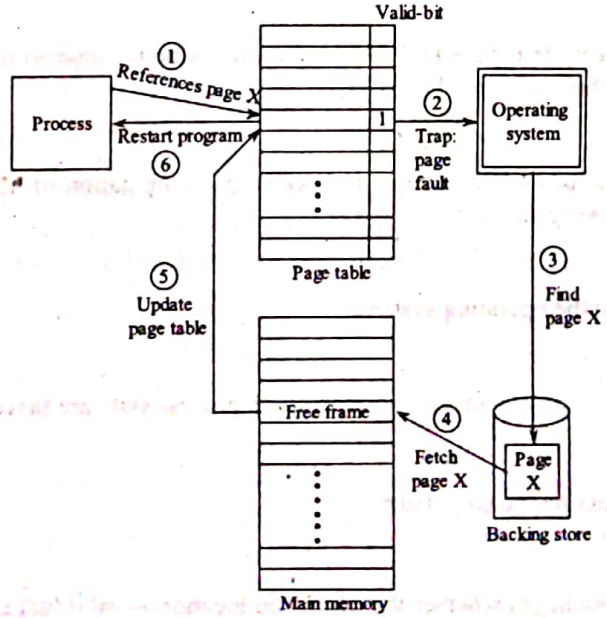


Figure: Demand Paging

Pure demand paging is a technique where a process starts execution without a single page in memory. As it proceeds execution, it gradually causes page-faults and those pages are fetched in memory. It never bring a page until it is required by some process.

The hardware that is necessary to implement demand paging is a page table for performing paging and swap space for performing swapping.

Implementation of Demand Paging

The implementation of demand paging can be done by keeping a valid bit in the page table. This bit signifies that the related page is either present in memory or disk. If the valid bit is 0, this indicates that the page is not present in the memory. It means that the page might be available on disk or it might be an invalid address. A page that has been requested must have its valid bit set to 1. If it is not then the OS must check whether the address is valid or not. If the page is not present in memory then the OS performs the following steps,

(a) It selects a page to replace using a page replacement algorithm

(b) It invalidates the old page present in the page table

(c) It loads a new page from the disk into the memory

(d) It does content switching to another process while the I/O is under operation

(e) It gets interrupted after the page is loaded completely

(f) It updates the page table entry

(g) It performs context switching back to the faulting process.

Q41. Discuss the performance of demand paging. Also list the steps involved in servicing a page fault.

Answer :

Performance of Demand Paging

The performance of system can be affected by demand paging technique whenever there exist a page fault. To spot the exact reason behind this, consider the computation of Effective Access Time (EAT) which is given by the formula,

$$EAT = (1 - P) \times MAT + P \times Pft$$

Where,

P is probability of page fault whose value is in the interval $(0 \leq P \leq 1)$

MAT is the memory access time whose typical value ranges from 10 to 200 nanoseconds.

Pft is the page fault time.

It is assumed that there exists very few page faults and hence, its value is assumed to be near to zero. When its value is equal to zero i.e., no page faults, EAT becomes equal to MAT.

Steps in Servicing a Page Fault

One of the important considerations while computing EAT is the computation of time taken for servicing a page fault which is done by performing the following steps,

Step1

Firstly, the issue is addressed to the operating system.

Step2

In this step, all the registers associated with the user along with process state are saved.

Step3

Now, operating system starts finding the page fault.

Step4

When the page fault is found, it checks whether the addressed location is valid (or) not, if not OS starts finding it in the memory.

Step5

Now, OS determines an empty frame and after the completion of read operation, the page is transferred to the empty frame.

Step6

During this read process, the CPU can be alloted to different user.

Step7

After the completion of read process, CPU is interrupted from the memory.

Step8

Now, CPU is made ready to be given back to the process of servicing page faults by saving the registers and process state associated with the user to whom the CPU is allocated in Step 6.

Step9

Memory location from which the interrupt request is made is determined.

Step10

Now, the page table and other tables associated with the relocated page are updated.

Step11

Finally, control is given back to the process and saved register and process states associated with the current user are restored and the instruction is continued.

Overall time can be improved by skipping step1 and step3 which can be done by careful coding. Typically, a disk takes 8 milliseconds to carryout paging. These 8 milliseconds include 5 milliseconds for a seek, 3 milliseconds for latency and 0.05 milliseconds for transfer. As already discussed that a typical MAT value lies in between 10 and 200 nanoseconds. Considering its maximum value i.e., 200 with which EAT can be computed as,

$$EAT = (1 - P) \times 200 + P \times 80,00,000$$

[$\therefore$ 8 milliseconds = 80,00,000 nanoseconds]

$$= (200 - 200P) + 80,00,000P$$

$$= 200 + (80,00,000 - 200)P$$

$$= 200 + 79,99,800P$$

With use of demand paging, the system will be slowed down to around 40 percent for every one thousand page-fault occurrence. To drop this percentage to less than 10 percent, by restricting the slow down and page-fault level. For example say EFT in the above system is restricted to less than 220. Therefore the probability of page falut will be,

$$220 > 200 + 79,99,800P$$

$$\Rightarrow \quad 79,99,800P < 220 - 200$$

$$\Rightarrow \quad 79,99,800P < 20$$

$$\Rightarrow \quad P < \frac{20}{79,99,800}$$

$$\therefore \quad P < 0.0000025$$

Therfore, the probability of page-fault is very close to zero.

4.2.5 Page Replacement Algorithm

Q42. What is a system call? Explain how system programs are developed using system calls.

Model Paper-III, Q15(b)

Answer :

System Call

For answer refer Unit-I, Q30, Topic: System Calls.

Development of System Programs Using System Calls

The development of system programs using system calls can be performed in the following steps.

Step 1

In this step, the input and output files are named. After acquiring the names, the writing of prompting message is done. The procedure involves the following,

The process of naming the input and output files follows various methods largely depending upon the layout of the operating system. However, when it comes to an interactive system series of system calls is required. In first system call, it asks the user to name the input and output files and in order to accomplish this, another system call is required in which the characters are read from the keyboard to define the two files. In addition to this, another approach includes mouse-based and icon-based systems in which filenames are displayed in a menu form with in the window. Therefore, the user has multiple options to choose the source name and destination name. Each sequence requires series of I/O system calls.

Step 2

In this step, input file is open in order to display the output file. However, few discrepancies are involved corresponding to the existence and non-existence of the file.

After the files are named, the first system call opens the input file and the second system call is required in order to generate the output file. However, each of these operations has its related error conditions such as in certain situations when a program attempts to open a non-existing input file or try to gain access to protected file. This problem is alleviated again by a series of system calls that carries the messages printed by the program and another system call is used to terminate the file abruptly. Consequently, if the input file exist, its new output file is generated. However, if the same problem arises connected to the existence of the output filename, the problem is addressed by a new system call where the program is either aborted or may result in the deletion of existing file which requires another system call. This deletion is followed by the creation of new file which again requires system call. Also, the user adopts an interactive system wherein with the help of series of system calls, the user is asked whether or not to replace the existing file or to abort the program. So, one system call outputs the prompting message and another system call reads the response from the terminal.

Step 3

This step is concerned with the loop wherein reading and writing in the file is performed. After setting up both the files one system call reads the data from the input file within the loop and another system call is used that writes output to the file. However, status information is kept in order to keep track of what each write or read operations are doing and if there is nothing to read, it sends error condition. Also, the write operations can face various errors depending on the output devices such as less disk space or no papers in the printer.

At last, after the completion of task a system call closes both the files. This is performed by writing a message to window which is possible via series of system calls and another system is used so as to terminate the program normally. Therefore, each program makes heavy use of OS frequently wherein innumerable system calls are executed every second.

Q43. Discuss the design goals of operating system and also discuss on separation of policy from mechanism.

Answer :

Design Goals of Operating System

Designing a system requires in depth understanding of its goals and specification. However, it is very difficult to understand overall requirements because of the existence of different types of systems and hardware platforms. For this reason, the requirements are divided interms of user goals and systems goals.

From the user's point of view, the system must be,

- Fast and secure
- Easy to learn and use
- It must posses reliability and much more.

However, there is no particular way to define what design should be adopted to fulfill these goals.

From the system's (system designers) point of view, the design and implementation of the system must be carried out in an easy manner along with the features like flexibility, reliability etc. As there exist number of options for designing such a system, these design goals are also considered as vague.

Therefore, there is a no particular way of defining the overall requirements for designing the system. Rather, there exist some general principles such as that of Software Engineering that can be considered.

Separation of Policy from Mechanism

Separation of mechanism and policy is a principle which states that the mechanisms i.e, how the operations are performed should be separated from policies i.e, what needs to be done (which operations to be performed). For instance, providing protection to the CPU using a timer construct is a mechanism whereas the decision of time allottment is considered as a policy.

As policies are subjected to change with respect to time and location, its corresponding mechanism also needs to be changed. If the policies and mechanisms are properly separated, the changes in policies will no longer effect the mechanism and hence very few parameters in the systems need to be changed which provides flexibility.

In Micro-Kernel based systems, it is considered as a fundamental approach. In these systems, certain building blocks are used that allow user-level processes to add mechanisms and policies ih a policy free environment. Consider solaris systems as an example in which scheduling is performed by loading the tables. This provides flexibility in changing the type of system i.e, real time, time shared, batch processing etc.

Some other systems like Windows and Mac x carry inbuilt interfaces in the Kernel that are almost the same for all the programs.

Q44. Write about operating system implementation.

Answer :

The operating system is ready to be implemented after its design is complete. Operating systems initially written in assembly language. It can be written in multiple languages. The top levels of kernel are written in C and the remaining levels are written in assembly language. Other than this the system programs can be in C, C++, PERL, python or in shell scripts. When these languages are used for implementing the operating system the benefits are same when they are used for implementing application programs. Code is generated with higher speed, compact as well as easy to understand and debug. Consider the example of MSDOS which is written in Intel 8088 assembly language. It is run on Intex X86 family of CPU's. And the Linux operating system is written in C mostly. It is available on multiple CPU's along with Intel X86, Oracle SPARC and IBM PowerPC.

The drawbacks of implementing the operating system in higher level language are decrease in speed and high requirement of storage. But this is however covered by the expert programmers by generating small size routines and by using model compilers.

The latest processors contain deep pipelining and multiple functional units which are capable of handling the complex dependencies easily. The data structures and algorithms generate good results along with performance improvement than the assembly language code. The complex routines of operating systems are interrupt handler, I/O manager, high performance, memory manager and CPU scheduler. After the system is written accurately and it is working properly then the bottlenecks are identified and replaced with respective assembly language routines.

4.2.6 Thrashing

Q45. What is thrashing?

Answer : Model Paper-II, Q16(a)

Thrashing

Thrashing refers to a situation wherein the operating system wastes most of its crucial time accessing the secondary storage, looking-out for the referenced pages that are unavailable in the memory. This situation (i.e., thrashing) can arise in both the demand paging system as well as in circular job stream.

In this situation, the OS swaps in the referenced page from secondary storage to primary while swapping out certain pages from the memory. In other words, thrashing is referred to a situation where the processor spends most of the time in the swapping of pages instead of executing the instructions.

One of the cause of thrashing is decreased CPU utilization. In this case, OS increases the degree of multiprogramming by including a new process which requires extra page frames. Now, newly joined process takes the frames from active processses which results in the occurrence of page faults in these active processes as well. This leads to more decrement in the CPU utilization and results n a stucked situation.

This situation is expressed in the form of a graph as follows,

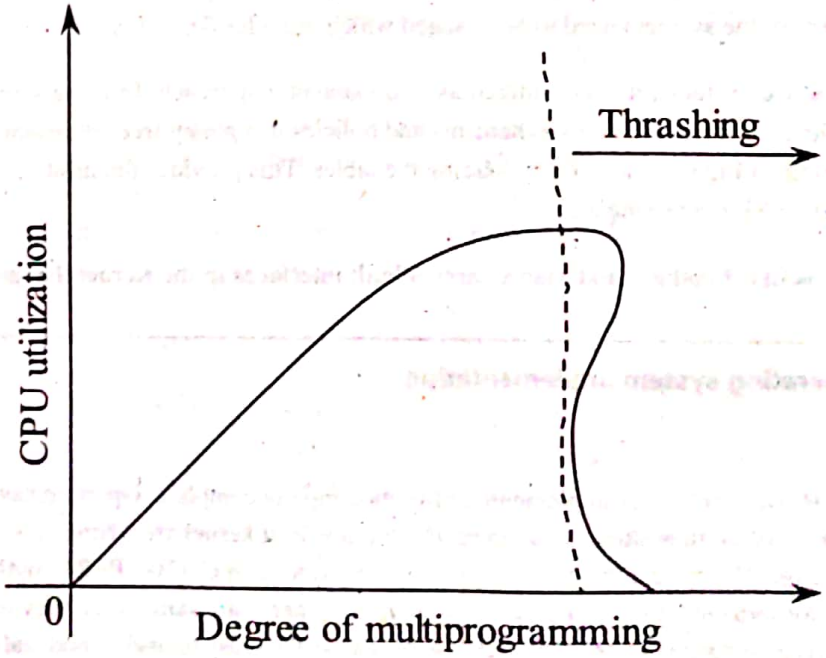


Figure: Effect of Thrashing

As it can be observed from the graph that CPU utilization increases with increase in the degree of multiprogramming but at a certain point after reaching maximum CPU utilization, it decreases sharply. That particular point is referred to as thrashing.

With use of local replacement algorithm of avoiding the use of pages associated with active processes, effects of thrashing can be limited to certain extent. To eliminate thrashing completely, a processes must be alloted with as many as page frames it require.

One of the effective methods of doing so is the use of locality model which allows the use of frames with multiple processes concurrently. It does so by creating a locality set of active frames and the processes are made to move from one locality to another.

UNIT 5

I/O HARDWARE, FILE MANAGEMENT AND SECONDARY - STORAGE STRUCTURE

PART-A

SHORT QUESTIONS WITH ANSWERS

Q1. Write short notes on file system.

Answer :

A file is grouping of similar records or related information together which is stored in secondary memory. Both the data as well as programs of all users are stored in files. A collection of files is called directory. Files and directories are the basic concepts of a file system. Directories are used to organize files.

Q2. Brief about disk structure.

Answer :

Disk drives uses a logical block which is nothing but a large one-dimensional array and is the smallest unit of transfer. Its size is usually 512 bytes or 1024 bytes. These logical blocks are mapped to the disk sectors sequentially. The first track sector is on the outermost cylinder or track is called as sector-0.

Q3. What are disk allocation methods?

Model Paper-I, Q9

Answer :

The disk allocation method are,

1. **Contiguous Allocation**

In a contiguous allocation method, all the files are arranged in sequential blocks of memory. Therefore, according to this technique, if a file size is k blocks and it starts from a blocks then it spans till k-1 block numbers. With this approach, accessing of files is much faster as all files occupy contiguous blocks.

2. **Linked Allocation**

This scheme overcomes the problem of contiguous allocation. Any file can be represented as a linked list of disk blocks. The directory entry stores the starting block number and last block number. By taking starting block number, that block can be accessed, the address of next block can be captured to move a head. Similarly, the whole file can be read by traversing the list.

3. **Indexed Allocation**

This scheme store pointers to all the blocks of a particular file at one location called as index block. It is a simple modification of linked allocation. The directory stores the address of this index block. Hence, after getting a particular index block the whole file can be accessed.

Q4. List any three disk scheduling algorithms.

Answer :

Model Paper-II, Q9

The following are the types of disk scheduling algorithms,

1. **First Come First Serve (FCFS) Scheduling**

It is simplest among all scheduling algorithms. Here, the request which comes first is served first. But it cannot promise fastest service always.

2. **Shortest Seek Time First (SSTF) Scheduling**

This algorithm serves the requests which are close to the current head position i.e., the next request is selected such that it has minimum seek time from current head position.

3. **SCAN Scheduling**

In this algorithm the disk arm moves in one direction, servicing all the requests that comes along that route until last track is reached and then it reverses the direction and moves to the other end servicing the requests along this way also.

Q5. Define seek time and latency time in disk.

Answer : Model Paper-III, Q9

Seek Time

The time taken by the disk arm to reach a particular track on the platter is known as seek time. Typically, it considers two components. They are,

(i) Startup time and

(ii) Traversal time i.e., time taken by the disk arm to traverse the tracks.

It is given by the formula

$$T_s = m \times n + s$$

Where,

T_s = Estimated seek time

m = Constant value associated with the disk

n = Number of traversed tracks

s = Startup time.

Latency Time

It is the amount of time taken by the read/write head to reach the desired sector in a particular track.

Q6. Why is rotational latency usually not considered in disk scheduling?

Answer :

The rotational position information of the disk is not shared with the host because the time duration for exchanging data between scheduler and the disk is variable. Moreover, the information is also correct. Hence the rotational latency is not considered in disk scheduling.

Q7. What is the advantage of bit-vector approach in free space management?

Answer :

The advantage of bit vector approach in free space management is to determine the initial free block or a set of free blocks. In free space management method, the bit vector implements free list. A free block is determined by bit 1 and occupied/allocated block is determined by bit 0. For example disk blocks 1, 5, 6, 8, 10 which are free, represents the bit vector as follows,

0100011 01010000

Q8. How does DMA increase system concurrency?

Answer :

The concurrency of the system can be increased when the DMA allows the CPU to perform its tasks there by transferring the system data using the system and memory buses. This happen when the DMA is integrated into the system and made as bus master.

Q9. Suppose that the disk rotates at 7200 r.p.m.

(a) What is the average rotational latency of the disk drive?

(b) Identify seek distance can be covered in the time?

Answer :

Given,

Number of rotations of disk per minute (r.p.m) = 7200.

(a) Average Rotational Latency

Average rotational latency can be determined as follows,

$$t = \left(\frac{1}{\frac{RPM}{60}}\right) * 0.5 * 1000$$

$$= \left[\frac{1}{\frac{7200}{60}}\right] * 0.5 * 1000$$

$$= \frac{60}{7200} * 0.5 * 1000 = 8.333 * 0.5 * 1000$$

$$= 0.00833 * 0.5 * 1000$$

[$\because$ 1 sec = 1000 milliseconds]

$$= 8.33 * 0.5$$

t = 4.167 ms.

(b) Identify Seek Distance in the Time

It is calculated that average rotational latency time t = 4.167. The equation to calculate seek distance that can be covered in time t = 4.167 is,

$$\boxed{t = C + d\sqrt{L}}$$

Where, C and d are constants for unknown data points, L is the track length to be covered.

$$4.167 = 0.7561 + 0.2439\ \sqrt{L}$$

$$4.167 - 0.7561 = 0.2439\ \sqrt{L}$$

$$3.4109 = 0.2439\ \sqrt{L}$$

$$\frac{3.4109}{0.2439} = \sqrt{L}$$

$$\sqrt{L} = 13.9848$$

$$L = (13.9848)^2$$

$$= 195.57$$

Hence, it is possible to seek 195 tracks with an average rotational latency.

Q10. List the major attributes and operations of a file.

Answer :

Major Attributes

Attributes of a file are as follows,

1. Name
2. Identifier
3. Type
4. Location
5. Size
6. Protection.

Major Operations

Operation of a file are as follows,

1. File creation
2. Writing to a file
3. Reading from a file
4. Seeking in a file
5. File deletion
6. Truncating a file.

Q11. Give a short note on Input and Output system.

Answer :

Input and output system refers to the process that allows communication with the information processing system like computer using various I/O devices. This can be done by providing input to the system and getting output from the system by I/O devices.

Q12. What are the operation that can be performed on directory?

Answer : Model Paper-I, Q10

The operation that can be performed on directory are,

1. **Searching Files**

   A file can be searched in the directory table, by finding a particular file or similar files or whose name matches a specified pattern or criteria.

2. **File Creation**

   It creates a file by inserting its entry in directory table.

3. **File Deletion**

   It deletes a file by deleting its entry from the directory table.

4. **Listing**

   It is used to list the files and other sub-directories present in their directory.

5. **Renaming Files**

   It renames the name of existing file by modifying its entry in directory table.

Q13. Differentiate between file and directory.

Answer : Model Paper-II, Q10

The differences between file and directory are as follows,

	File		Directory
1.	A file is a collection of similar records.	1.	A directory is a collection of files.
2.	It is used to arrange/organize date.	2.	It is used to arrange/organize files.
3.	It has file extension to represent the file type.	3.	It does have any file extension.
4.	It has different file organizations such as sequential series, index sequential and directory file organization.	4.	It has different directory organizations such as one directory per user, and multiple directory per user organization.
5.	It does not store other files.	5.	It store's other file/directories.

Q14. Enlist different types of directory structure.

Answer : Model Paper-III, Q16

The different types of directory structure are as follows,

1. **Single-level Directory**

Here, the volume contains a single directory and all files are stored in the same directory. Sub-directories cannot be created within this directory.

2. **Two-level Directory**

Here, each user of the system is given a separate directory called as user file directory (UFD) where all files of particular user is present. If there are '*n*' number of users then there will be "*n*" number of UFDs all of which are indexed in a Master File Directory (MFD).

3. **Tree-structured Directory**

This scheme allows user to create any number of their own directories within their User File Directory (UFD). It has a variable number of levels. It gives a better flexibility to manage files.

4. **Acyclic-graph Directories**

It is a technique which allows sub-directories and files to be shared. Acyclic-graph means a graph without cycle. The tree-structured directory method does not allow sharing of files and directories, but here that problem is resolved. The entry of a particular file is present in all directories that are sharing that file.

5. **General Graph Directories**

It is a tree-structured directory organization where one can add links to an existing directory. It is same as acyclic-graph structure except it allows cycles in graph whereas, acyclic-graph does not.

PART-B

ESSAY QUESTIONS WITH ANSWERS

5.1 I/O HARDWARE

5.1.1 I/O Devices, Device Controllers

Q15. Explain the different components of I/O hardware.

Model Paper-I, Q16(a)

Answer :

Components of I/O Hardware

The different components of I/O hardware are,

1. **Device Drivers**

The device controller provides an interface to be used by the high-level machine software as shown in figure. This software implements the device management part of the Operating System (OS) as a collection of device drivers. The device driver to controller interface may be relatively complex in the sense that there can be many parameters to set prior to starting the controller and many aspects of the status to check when each operation has completed. However, because of the "standardized" device driver to application software interface, application software can deal uniformly with different kinds of devices. Drivers for different devices implement a similar interface, so higher-level machine need not be especially concerned with the details of the particular device. The goal is to simplify the software interface to devices as much as possible, the reason being the differences between controllers produced by different manufacturers will be transparent as possible.

The presence of devices, controllers and drivers enables two parts of programs to operate at the same time one part using the CPU and the other part using the devices.

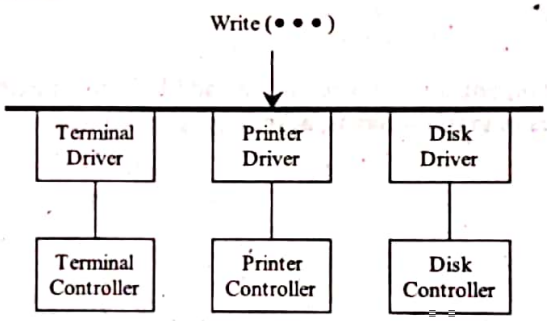


Figure: Device Driver Interface

One goal of an OS is to provide facilities that enable application programmers to take advantage of the physically separate devices so as to execute logically separate operations simultaneously. Unfortunately, most programming languages discourage such overlap through the semantic of high-level I/O instructions. Nevertheless, the OS can manage the processes so that CPU utilization by one process overlaps the I/O operation of another. While, this overlap does not increase other performances of either process, it makes the use of the hardware more effectively since two or more parts are used simultaneously rather than sitting idle.

The device driver is the OS entity that controls CPU-I/O parallelism. A device can be started by the device driver and the application program can continue operating parallel with the device operation. A problem arises in providing some OS mechanism to notify the part to the program that is executing on the CPU that the I/O operation has completed. Contemporary computer systems use interrupts to accomplish this notification.

2. **Interrupt Request Line**

The processor (CPU) handle interrupts using separate pins or wires called as "interrupt-request lines". The I/O devices which needs to perform I/O operation interrupts the CPU by activating the interrupt signal on interrupt-request line. Whenever, CPU detects an interrupt on interrupt request line, i.e, it saves the state of its current program, suspends it temporarily and jumps to execute the Interrupt Service Routine (ISR) of the respective interrupt. In latest processors an interrupt controller hardware is embedded which takes care of everything associated with interrupts including management of priorities among interrupts, differentiating interrupts as necessary or ignorable during critical processing etc.

The Intel 80 × 86 family of CPUs has two types of interrupts called "maskable interrupts" and "non-maskable interrupts". Non-maskable interrupts are those which has to be executed in any situation, they can't be ignored. For example, unrecoverable memory errors. Maskable interrupts are those that can be ignored by the CPU while it is executing critical instructions. Servicing a request with respect to device controllers typically employs maskable interrupts.

In order to service an interrupt, the interrupt handler needs the address of ISRs which are stored as offsets in interrupt vector table. The interrupt vector table of Intel 80 x 86 processors contains 32 non-maskable interrupts (from 0 to 31) and remaining from 32 to 255 contains maskable interrupts which can be used by device controllers.

Whenever an interrupt is triggered, the interrupt handler selects the appropriate ISR to be executed according to its interrupt vector number. The interrupt handler also manages the priority and allows the CPU to execute highest priority ISR first. A low priority interrupt can't preempt an interrupt of higher priority whereas, a higher priority interrupt can preempt a lower priority interrupt.

The interrupt handler also manages "exceptions" such as divide by zero, stack fault, page fault, accessing invalid memory location etc. Another type of low priority interrupts called as software interrupts or trap are present which are triggered during the execution of system calls.

3. Direct Memory Access (DMA)

For answer refer Unit-V, Q16, Topic: Direct Memory Access (DMA).

4. PCI Bus Architecture

The computer system communicates with devices by sending and receiving signals via some medium like cable or wireless channels. All the devices are connected to computers through a connection point often called as "port" (like serial port etc).

Usually, the devices which follows common protocols, like voltage level of signals, timing etc., are connected to the computer through a set of conductors called "bus". The most commonly used bus mechanism for connection of various components of computer is called as "PCI" bus (Peripheral Component Interconnect). The fast operating processor, memory various buses and device controllers are connected together via PCI bus.

There are various other buses like "expansion bus" which is used to connect slow peripheral devices like keyboard, serial and parallel ports etc. The SCSI bus connects storage devices like hard disks etc.

For controlling the operations of various buses and devices etc., a special electronic circuit called as "controller" is used, Various controllers that are shown in figure include the graphics controller to control the monitor, a disk controller to control various ATA or S-ATA disks. They also perform tasks like buffering, caching, bad sector mapping etc.

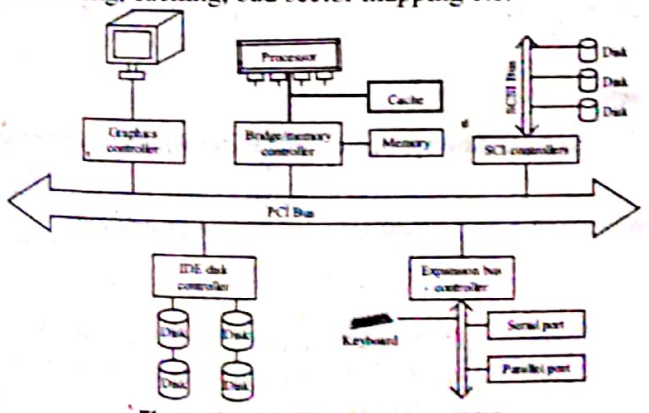


Figure: Bus Architecture-based PC

Each controller has a set of registers which the processor uses to communicate and instruct the various devices that are connected through these controllers. The processor does this by writing a certain bit pattern in these registers or reading the bit pattern from these registers

5. I/O Devices

Block devices typically refers to disk drives and require system calls such as read( ), write( ) and seek( ) for communication.

The interface associated with these devices can be divided into two types,

(i) Raw I/O

(ii) Direct I/O.

(i) Raw I/O

Applications including operating systems can access the data devices directly as a array of disk blocks. This type of accessing disk blocks is referred to as raw I/O and the type of disk drive (or) block is called Raw device. These devices eliminates the interruption of operating system to avoid certain conflicts such as locking services provided by both the application and operating system. One drawback associated with this method is that if it is employed, services associated with the operating systems cannot be resumed.

(ii) Direct I/O

This method is typically used in UNIX system to overcome the drawback associated with Raw I/O. This can be done by including the feature of disabling locking and buffering mechanisms in the operating system itself.

One improvement in this regard is the layering of memory-mapped I/O above these types of devices which replace the traditional read-write operations with access through array of bytes in main memory. This in turn returns a copy of the disk block instead of original data. The original data can only be modified in extreme situations using disk image.

Character Devices

The interface of character devices require system calls such as get( ) and put( ) for communication. An example of character oriented devices is a keyboard. These devices offer interactive features including line-at a time access, buffering and editing.

Layers of I/O Software

I/O software contains the following layers,

1. User-level software
2. Device-independent OS SW
3. Device drivers
4. Interrupt handlers.

1. User-level Software

The execution of I/O code is done at this layer. The arguments of unimportant library routines are shifted to specific registers and then a signal trap is used to call the appropriate system call for performing the desired functionality.

A printer is used to print can perform its task in two parts where one is for which the regular program using Lpr in unix and the other is for daemon processors using Lpr. In this, spooling concept is used for printing where a copy of file is created using lpr and later, the copy obtained is used by daemon. In E-mail system such a concept is called as querying.

2. Device-Independent OS SW

It contains the following functionalities.

(i) Buffering

A special storage place where data to be transmitted between systems (or) applications are stored is called as a buffer.

Example

1. Suppose speed of mode of transmission through which the data will be transmitted (modem) and where the data will be stored (secondary memory or hard disk) are different, then neither the data will be transmitted nor the data is stored properly. To avoid this, buffer captures all the data received from the modem and then writes it to the disk. Moreover in a situation where the modem needs to send more data, two buffers are created where the first buffer is writing its data to the disk and the second one captures incoming data from the modem. While performing this double buffering, when the second buffer gets full, by this time the first buffer must complete its write operations to avoid data loss.

2. Another situation where buffering is mostly used, when the capacity of systems transmitting the data is different. Suppose, one device has the capacity to split or fragment the data into various packets and transmitting it, but the other device might not have that much capacity to receive and rearrange the packets to get the original data. In this case the receiving device stores the packets in the buffer which rearranges fragment packets to get the original data.

3. Another situation is when a write operation is done using write( ) system call and when it returns to the buffer after performing its operation, the data in the buffer might get changed by the application. Therefore, to ensure the correct copy of data, copy semantics is used with which the data associated with the application is copied into the kernel buffer and write operation is performed from there itself. During this operation, the control is not given back to the application until the successful copying of data is accomplished.

(ii) Protection

Protection is a mechanism of controlling access of computer resources by users or processes. The mechanism should provide tools so that the administrator can define the restrictions or privileges of various users/processes and should have mechanism to enforce those restrictions. A protection enabled system can find the differences between authorized and unauthorized access or usage and can take measures to defend the system against misuse. If protection is not employed then errors may also occur among subcomponents of system. This happens usually when a defected subsystem interacts with healthy subsystem through its interface. Then healthy subsystem gets corrupted.

Goals of Protection

There are several reasons to provide security and protection in operating system. Some are as follows,

1. To avoid mischievous and notorious activities by users and others.
2. To prevent intentional damage and violation of access-restriction by a user.
3. To ensure that each program should use the system resources in a consistent and desired manner only.
4. To improve reliability by detecting errors in subsystems that can propagate further. This can be detected at interface of each subcomponent.
5. To defend the system from unauthorized and incompetent user.
6. Provide mechanism to specify the protection policy and enforce it. It should be a flexible mechanism which can be extended in future.
7. The protection mechanism vary over time, hence application programmers should also implement protection.

3. Device Drivers

For answer refer Unit-V, Q15, Topic: Device Drivers.

4. Interrupt Handlers

For answer refer Unit-V, Q15, Topic: Interrupt handlers.

5.1.2 Direct Memory Access

Q16. Write about direct memory access (DMA).

Answer :

Work load on the CPU can be decreased by distributing its work among special processors designed for various purposes. One of the such processors is Direct Memory Access (DMA) controller which can be initiated simply by writing a command block into the memory. This blocks carries I/O instructions of a particular DMA register with which the transfer continues till the whole block is transferred.

The CPU sends the following information through the data bus to initialized the DMA.

1. The starting address of the memory block is where data is to be written or from where data to be read.
2. The word count i.e., the number of words contained within the given memory block.
3. A control that specifies whether to perform a read or a write operation.
4. Another control that starts the DMA transfer.

It typically employes a handshaking method among device controller and DMA controller with use of two signals namely DMA-request and DMA-acknowledgment. The process initiates with a DMA-request generated by device controller which is placed on line controlling DMA. When DMA controller gets this request, it generates DMA acknowledgment signal confirming that it is ready for the transfer. When this acknowledgments received at the device controller side, it places the data on the transfer wire while removing the DMA-request.

During this process, the CPU gets restricted for accessing main memory to avoid decrease in the computation speed. However, the CPU can still access some of the primary and secondary memory caches. This access is called cycle stealing. One transformation in this method is Direct Virtual Memory Access (DVMA) which uses virtual memory addresses instead of primary addresses. These virtual addresses are nothing but pointers (or) translations of actual physical address. This process eliminates the need of CPU (or) accessing of memory for the transfer.

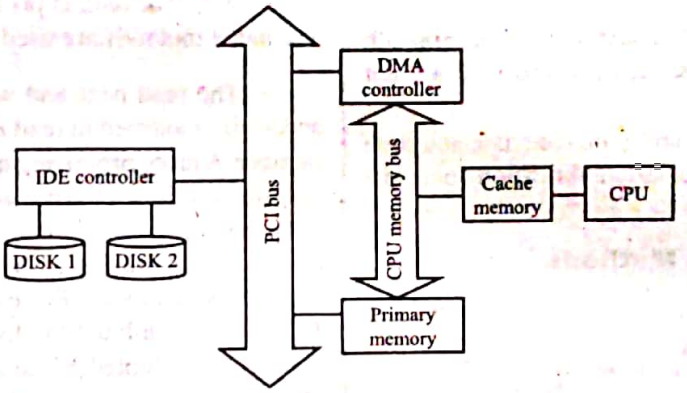


Figure: DMA Transfer

5.1.3 Principles of I/O Software, Goals of Interrupt Handlers, Device Drivers, Device Independent I/O Software

Q17. Explain the different layers of I/O software, goals of interrupt handlers, device drivers and device independent I/O software

Answer :

For answer refer Unit-V, Q15.

5.2 FILE MANAGEMENT

5.2.1 Concept of File

Q18. What is a file? Discuss its attributes.

Model Paper-I, Q16(b)

Answer :

File

A file is grouping of similar records or related information together which is stored in secondary memory. Both the data as well as programs of all users stored in files. A collection of files is called directory. Files and directories are the basic mechanism of a file system. Directories are used to organize files.

In order to store data on secondary memory we need to create a file and input data into that which is then stored in secondary memory. Without a file we cannot store data in secondary memory. The following are various attributes of file.

Attributes of File

Each file has several attributes. They are,

- **Name**
  It is a symbolic name of the file which gives convenience to users to refer to that file. It is a string of characters like myfile.txt, resume.doc etc.
- **Identifier**
  It is a unique number which is used to identify a particular file by the file system. It is not in user-readable form.
- **Type**
  There are different types of files depending on the type of data they store like text, executable code, sound, video, image etc. This attribute tells the type of data that file stores.
- **Location**
  It specifies the physical address of the file located on a particular storage device.
- **Size**
  It indicates the size of a file usually measured in bytes. It can also specify the maximum size allowed to a file.
- **Protection**
  This attribute determines the access control information i.e., who are allowed to use this file and who are not allowed.

Other miscellaneous information include date and time at which the file was created, last modification done, last usage etc.

5.2.2 Access Methods

Q19. Write short notes on,

(a) Sequential access

(b) Direct access

(c) Indexed access

(d) Indexed sequential access.

OR

What are the various ways of accessing a file?

Answer : **Model Paper-II, Q17(a)**

(a) Sequential Access

Among all the access methods, it is considered as the simplest method. As the name itself suggest that it is the sequential (step by step) processing of information present in a file. Due to its simplicity most of the compilers, editors etc., uses this method.

Processing is carried out with use of two operations namely, read and write. Read operation is responsible for reading the portion present next to the file pointer which in turn proceeds automatically and tracks the I/O location. Write operation is responsible for writing at the end of the file and proceeds towards the new end.

In this type of access while processing the records sequentially, some of the records can be skipped in both the directions (either forward (or) backward) and can also be reset to the head of the file (beginning). The following figure shows a tape model of sequential access file

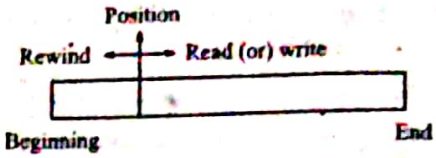


Figure: Sequential Access of a File

(b) Direct Access

This access method is also called as realtime access where the records can be read irrespective of their sequence. This means they can be accessed as a file which is accessed from a disk where each records carry sequence number. For example: block 40 can be accessed first followed by block 10 and then block 30 and so on. This eliminates the need of sequential read (or) write operation.

A major example of this type of file access is the database where the corresponding block of the query is accessed directly for instant response. This type of access saves a lot of time when large amount of data is present. In such cases, hash functions and index methods are used to search for a particular blocks.

The read next and write next operations of sequential access are modified to read *n* and write *n* where '*n*' is the block number. A more promising approach is to use a direct access to map the position of the file and sequential access for performing read next operation in that block. This can be done by using relative block numbers which are nothing but index related to the access of blocks. For example, relative block numbers 0, 1, 2, ... can be alloted to block numbers 72, 1423, 20 etc. This approach is adopted by some of the operating systems while others use either sequential (or) direct access.

(c) Indexed Access

This methods is typically an advancement in the direct access method which is the consideration of index. A particular record is accessed by browsing through the index and the file is accessed directly with use of a pointer.

To understand the concept consider a book store where the database contains a 12-digit ISBN and a four digit product price. If the disk can carry 2048(2 kb) of bytes per block then 128 records of 16 bytes (12 for ISBN and 4 for price) can be stored in a single block. This results in a file carrying 128000 records to be reduced to 1000 blocks to be considered in index each entry carrying 10 digits. To find the price of a book binary search can be performed over the index with which the block carrying that book can be identified.

A drawback of this method is, it is considered as ineffective in case of larger database with very large files which results in making the index too large.

(d) Indexed Sequential Access

To overcome the drawback associated with indexed access, this method is used where an index of index is created. Primary index points to the secondxary index and secondary index points to the actual data items. An example of such a method is ISAM (Indexed-Sequential Access Method) of IBM which carry two types of indexes. They are a master index and a secondary index. Master index carry pointers to secondary index whereas, the secondary carry blocks that points directly to the disk blocks. Two binary searches are performed to access a data item. The first one is performed on a master index and the second one on the secondary index. This type of method is a two direct access read method.

5.2.3 File Types, File Operation

Q20. Define the term 'lock'. What are the two types of it? Discuss file types.

Answer :

Lock

If multiple processes are accessing a particular file then simultaneous operations on this file could make its contents inconsistent. Hence, some operating systems uses locking mechanism to prevent others from gaining access while the file is in use.

Types of Lock

The two types of locks are,

(i) Shared Lock

Several processes can acquire this lock concurrently. They are allowed only to read the file.

(ii) Exclusive Lock

Only one process can acquire this layer of lock at a particular instant of time. This lock is applied when a write operation has to be performed on the file.

File Types

There are various types of files depending on the types of data they store. This type distinction is useful for operating system to determine whether they support that format or not and if it recognizes the type it can treat it in respective manner. Normally, the file type is specified in the filename itself. The filename consist of two parts firstly the user-defined name and other is an extension separated by a period (.). An example of a text document in windows operating system is Myfile.txt.

Where "myfile" is the filename and the extension ".txt" defines that this is a text file. The following are various popular file type extension.

Extension	File type
exe, com, bin,	Executable or ready-to-run program files.
obj, o	Object files generated by compilers.
c, cc, java, htm or html, Vb etc.	Source files belonging to various languages.
asm, a	Assembly language files.
bat, sh	File containing sequence commands executable at command prompt or shell.
doc, txt	Text documents or files.
wp, rtf, doc, tex	Various word processor application files.
lib, a, so, dll	Various library files.
pdf, jpg, ps	Format files that can be printed out using a printer.
rar, zip	Archive files where some related files are grouped together.
mpeg, mov, rm, mp3, avi, 3gp, divx, axxo	Files containing multimedia data like video, audio etc.

Different operating systems may have different extensions for the same type of file. Another advantage of using extension is that operating system can associate with each file the type of application which is needed to open that file. Whenever user opens a particular file by double-clicking its icon (obviously in GUI environment) the operating system starts the applications that support that file format implicitly.

Unix uses a concept called *magic number* to indicate the type of file.

Q21. Discuss about the below system calls,

(i) open( )

(ii) read( )

(iii) write( )

(iv) close( ).

Answer :

(i) open( )

The open function is used to open (or) create a file.

Syntax

```
int open (const char *pathname, int flag, ....
    /*mode_t mode*/);
```

Return Value

Open returns file descriptor on success and –1 on error.

Example

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <error.h>
main (void)
{
 int fd;
 if(fd = open ("/usr/Econe.txt", O_RDWR)< 0) .
 perror("open error");
}
```

The open( ) has two arguments, which are filename and mode of operation. Here the user is trying to open a file in read and write mode. If the file is existing and opened it returns the file descriptor, otherwise returns –1, which causes an error.

The different file opening mode constants are,

O_RDONLY open for reading only

O_WRONLY open for writing only

O_RDWR open for reading and writing.

The optional constants are,

O_APPEND, O_CREAT, O_EXCL, O_TRUNC, O_NOCTTY, O_NONBLOCK, O_SYNC

(ii) read( )

Data is read from an opened file by calling the read function.

Syntax

```
#include<unistd.h>
ssize_t read(int filedes, void *buffer, size_t nbytes);
```

The read( ) function reads the number of bytes specified by "nbytes" from the file pointed to by the 'filedes' into the buffer 'buff'. On successful the completion function returns the number of bytes actually read. It returns 0 if an end of file is reached and –1 on error. read( ) starts reading from the file's current offset. On successful completion, before the function returns, the offset is incremented by the number of bytes actually read. There are many cases in which the number of bytes requested these are as follows.

- ❖ While reading from a file, if the end of file is reached. This occurs if the request to read 'n' number of bytes is more than the number of bytes remaining until the end of file. For example, if the request is for 100 bytes and there are only 40 bytes remaining until the end of file then the read function returns only 40 bytes.
- ❖ Reading from a terminal device is normally upto one line at a time.
- ❖ When reading from a network, buffering within the network may cause the number of bytes read to be less than the requested number of bytes.
- ❖ Some record-oriented devices return a single record at a time example, magnetic tape.

(iii) write( )

Data is written to an opened file by calling the write function.

Syntax

```
#include<unistd.h>
ssize_t write(int fields, const void *buff, size_t.nbytes);
```

The write( ) function writes the number of bytes specified by 'nbytes' from the buffer "buff" into the file pointed to by 'filedes' the functions returns the number of bytes written on successful completion and –1 on error.

For a regular file, the write( ) function starting writing at the file's current offset. If the O_APPEND option was specified at the time of opening a file then the file's offset is set to the current end of the file before each write operation begins. After the write operation the file's offset is incriminated by the number of bytes actually written.

A write( ) function fails if the disk is full or the file size limit is exceeding for a given process.

(iv) close( )

The close( ) function closes an open file. It has the following syntax.

Syntax

```
# include <unistd.h>
int close(int fd);
```

The argument fd is the file descriptor of an open file that needs to be closed.

Closing a file frees unused file descriptors, releases any record locks on the file and deallocate system resources (e.g., buffer and file table entries).

When a process is done using the file it must close all open files. So that the file descriptors associated with the open files can be reused by some other process to reference other files. However when a process terminates without closing the open files, the kernel closes those files for the process.

Q22. Write about seek( ) and unlink( ).

Answer :

seek( )

seek( ) is a file function that is used to move the file pointer to the desired location in the file. It is generally used when a particular part of the file is to be accessed. The main purpose of file seek( ) is to position the file pointer to any location on the stream.

Syntax

seek(fileptr, offset, position);

'fileptr' is a file pointer.

Offset specifies the number or variable of type long to reposition the file pointer. It tells the number of bytes to be moved. Offset can either be positive or negative number where positive integer is used to reposition the pointer forward and negative integer is used to move the pointer backward.

Position

It specifies the current position.

The values that can be assigned to position are,

Value	Position
0	Beginning of file
1	Current position
2	End Of File (EOF)

The seek( ) returns zero, if all the operations are performed successfully or –1 if an error occurs such as EOF is reached.

unlink( )

unlink( ) system call deletes, the hard link of a file from the directory.

Syntax

int unlink (const char* fileName)

Return Value

If successfully unlinks, it returns '0' otherwise '–1'.

Example

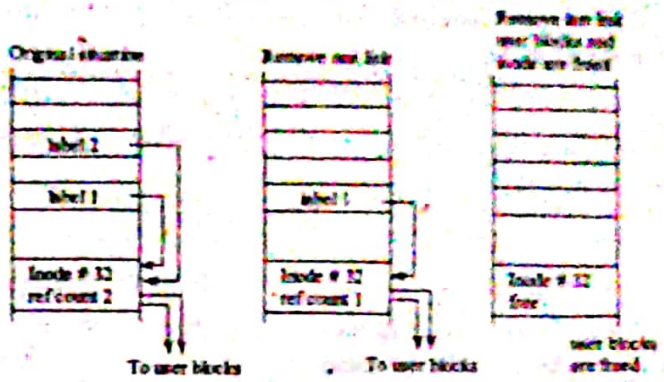
int strdInput = FALSE;

if(strdInput) unlink(oraName);

In the above example unlink function removes oraName, which is the link of oracle file.

Description

unlink( ) system call deletes the hard link and decrements the hard link count. If the hard link count of a file decremented to zero, all of its resources are deallocated. If any process file descriptors are using that original file, first they have to be closed, then only deallocation of resources is possible.



Notice the above diagram links are deleted one by one till the hard link count dropped to zero. Once all links are removed, the user blocks and inodes of original file are freed.

5.2.4 Directory Structure

Q23. Explain different directory structures with neat diagram.

Answer : Model Paper-II, Q17(b)

Directory Structure

Normally, the disk is divided into various parts known as partitions or volumes. Each of which contains file system that stores device directory or system volume information table. It contains information such as name, locations, size, type etc., about all files stored in that volume.

A directory is used to organize files. It can be thought as a symbol table that converts the given filenames into their directory entries thereby we can get all the information about that particular file including its location. The following operations can be performed on directories.

- **Searching Files**

  By reading the directory table, we can find a particular file or similar files or whose name matches a specified pattern or criteria.

- **File Creation**

  When a new file is created an entry is inserted in directory table.

- **File Deletion**

  When a particular file has to be deleted, its entry is deleted from directory table.

- **Listing**

  We can list the files and other sub-directories present in their directory.

❖ **Renaming Files**

The name of existing file can be changed by modifying its entry in directory table.

Schemes for Directory Structure

The following are the common schemes used to defined directory structure.

1. Single-level Directory

Here, the volume contains a single directory and all files are stored in the same directory. We cannot create sub-directories within that directory.

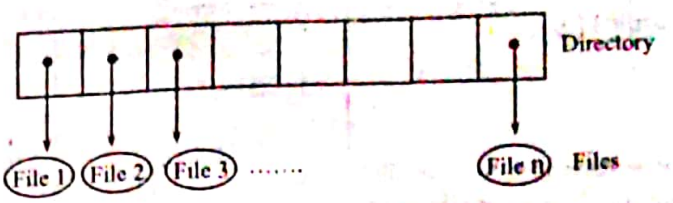


Figure (1): Single-level Directory Structure

There are various limitations to this scheme. All filenames has to be unique, because they reside in the same container. As users increases, files also increases and given unique file names to all those files may become complicated.

2. Two-level Directory

Here, each user of the system is given a separate directory called as user file directory (UFD) where all files of particular user is present. If there are '*n*' number of users then there will be "*n*" number of UFDs all of which are indexed in a Master File Directory (MFD).

When a user wants to search any file '*x*' then it is searched in his UFD only. The filenames within the UFD should be unique. But, two or more users can have same filenames because their directories are different. UFD is created whenever a new user is created.

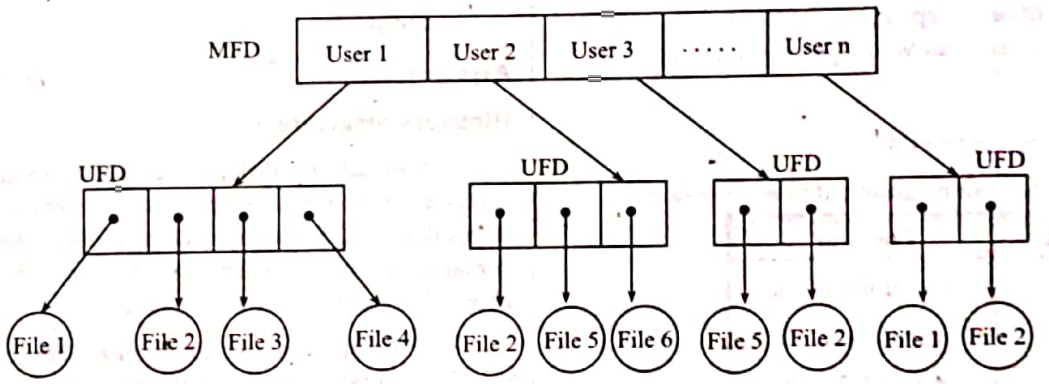


Figure (2): Two-level Directory Structure

This directory structure overcomes the problem of unique filename by isolating or separating users in different directories. However, this solution creates problem when several users require to share some files.

Each file in the system is identified by a path name. The syntax of specifying pathname differs from operating system to operating system. For example in MS-DOS and windows a colon(:) is used to specify a volume and a back slash(\) to specify a directory. The path name is created by using these symbols and the directory names. For example, if we want to access file 2 of user 2 the path would be,

C:\user2\ file2

Similarly if file2 of user3 has to be accessed then path would become,

C:\user3\file2

3. Tree-structured Directory

This scheme allows user to create any number of their own directories within their User File Directory (UFD). It has a variable number of levels. It gives a better flexibility to manage files.

A sub-directory is treated as a file. A special bit is used which defines whether the entry is file (0) or sub-directory (1). A current directory is normally a directory from where process is executing and carry almost all the associated files of currently executing process. When process tries to access a particular file it is searched in current directory. If it is not there, then user has to specify the pathname of that file or change the current directory to that path which can be done using a system call which considers the path name as a parameter and redefines the current directory.

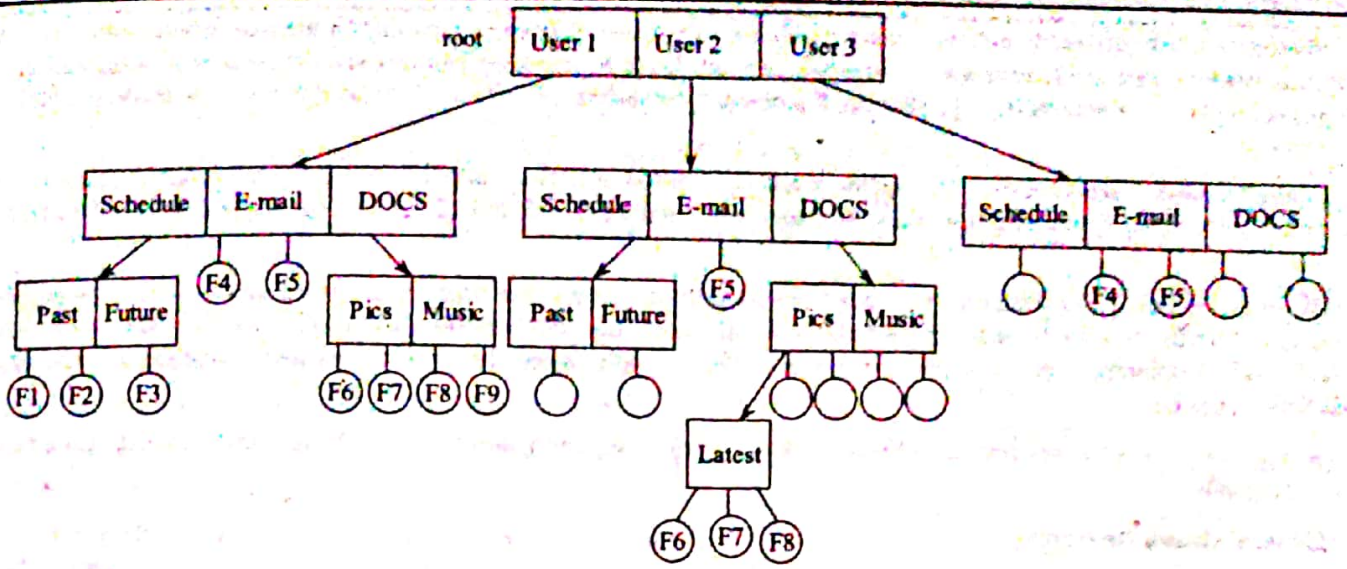


Figure (3): Tree-structured Directory Structure

The two types of pathnames are,

(i) Absolute Pathname

It gives full address of file starting from 'root' directory through all directories till the filename. For example, the absolute pathname of file "F9" in the figure (3) is,

root\User1\DOCS\Music\F9.

(ii) Relative Pathname

It gives address of file from current directory. For example, if current directories is "root\User1\DOCS\", the relative path of file "Music\F9" would be,

root/user/DOCS\MUSIC\F9".

4. Acyclic-graph Directory

It is a technique which allows sub-directories and files to be shared. Acyclic-graph means a graph without cycle. The tree-structured directory method does not allow sharing of files and directories, but here that problem is resolved. The entry of a particular file is present in all directories that are sharing that file.Figure (4) shows a file "main library" shared by two directories.

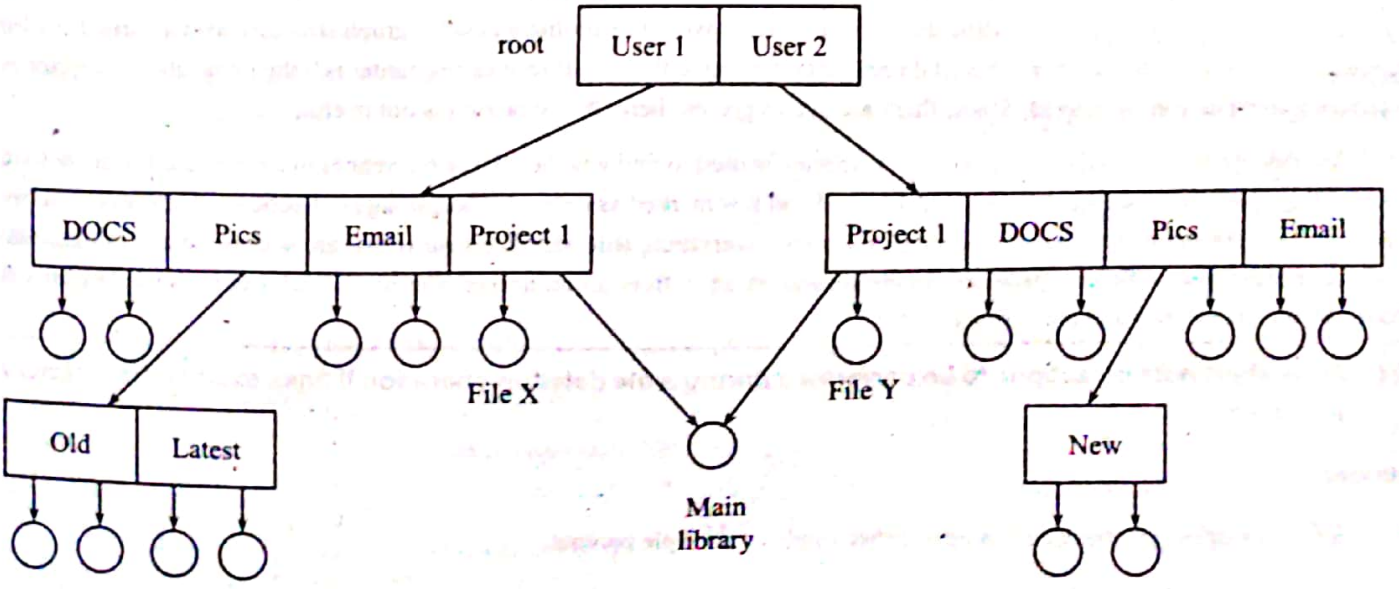


Figure (4): Acyclic-graph Directory Structure

Sharing is totally different from maintaining multiple copies and it is very important in the case where more than one programmer working on a single program. In this case, changes made by one programmers to a file need to be informed to the others instantly which can only be done by sharing. If a directory is shared, every new file created in it will be made available to all its shared users.

Sharing in unix is accomplished by creating a directory called link which acts as a pointer to the original directory/file. When we try to access the file present in a shared directory it is marked as a link and the name of the original file will be attached to it.

Acyclic-graph directory structure gives flexibility as well as sharing. The deletion of shared files is a complex problem here, because if we delete a file by searching it through a path, then entry will be deleted in that directory, but what about other directories who are pointing to the same file? Entry will not be deleted in those. This will create dangling pointers to file which actually does not exist.

Hence, some operating systems like MS-DOS, does not allow acyclic-graph structure. It uses simple tree structure rather than acyclic-graph.

5. General Graph Directory

It is a tree-structured directory organization where we can add links to an existing directory. It is same as acyclic-graph structure except it allows cycles in graph whereas, acyclic-graph does not.

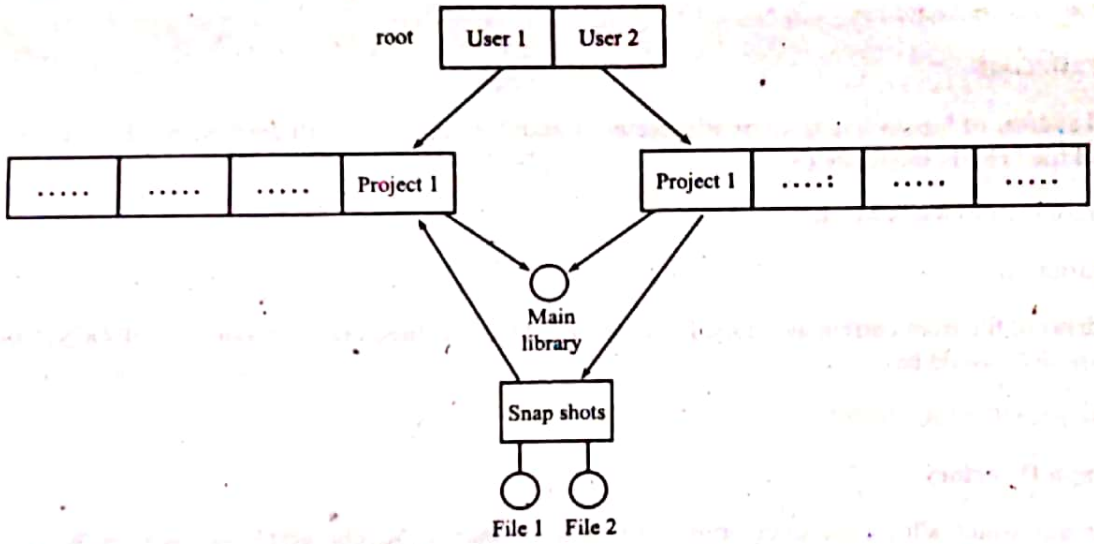


Figure (5): General Graph Directory Structure with Cycles

Since, cycles are present in graph, we should avoid searching a particular file more than once. This organization also suffers the dangling pointers problem while deletion of files. To avoid this problem acyclic-graph structure uses a variable called reference counter which stores the number of directories referring to this file. If reference counter is 0 then it means no directories are referring to it and can be deleted. Since, there are cycles present here, this approach is not useful.

Another approach called garbage collection scheme is used to find whether all the references to a particular file are deleted or not, so that space occupied by that file is deallocated and it is marked as deleted. The garbage collection scheme works in two phase. Firstly, it traverses the whole file system and marks everything (file and directories) that are accessible and ensures that it is marked only once (without reputation). In the second phase, it frees all unmarked files and directories because they are not referred by any directories and are garbage.

Q24. Write short note on actions to be performed during a file deletion operation if links exist in the directory structure.

Answer :

A file in a directory structure can have either single or multiple parents.

When a file to be deleted has a single parent. It is easily deleted and its entry is removed from its parent directory. If a file has multiple parents then deleting it is a complex task as it will create dangling pointers.

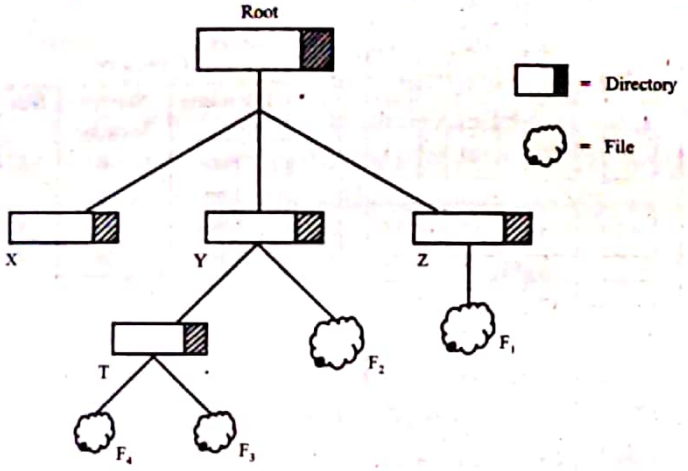
However, its entry still can be removed from parent directory present in the access path of delete command.

The process of checking for multiple parents is a very complex. The complixity is reduced by maintaining a reference count for every file. When a new file is created the count is set to one and whenever a link points to the file the count is incremented by one.

On contrary, when a file deletion attempt is made its reference count is decremented by one. Further, its parent directory provided in the access path of delete command's is entry is deleted. Finally, when reference count of the file becomes zero actual file is deleted.

The reference count strategy does not work when a directory structure contain cycles.

A cycle is developed when a link is made between a directory and its grand parent directory as shown in below (1).



Figure

In the above figure, directory *T* is linked to directory Y and its grand parent directory root.

If directory Y is deleted then its entry is root directory is deleted as it has one as reference count. Further, directory Y and its file become unreachable from root directory therefore there is no use to retain the directory.

To solve this unreachable problem some cycle detection techniques must be applied on formation of cycles must be prevented.

5.2.5 File System Structure

Q25. Write about layered file system structure.

Answer : **Model Paper-III, Q16(a)**

File System Structure

A file system is an essential part of the operating system which is responsible for allowing efficient access to the disk also storing reading data to and from the disk. Every design of a file system faces two major problems. They are,

1. How should the file system appear to the user i.e., a user should be able to easily create a file, specify file permissions and arranging it in a directory structure.
2. Writing efficient algorithms or programs to associate logical files and directory names with the actual stored data on the physical storage devices.

A typical file system uses a layered approach where each layer uses the session of its lower layer to perform required functions for upper layers. In a layered file system, the lowest layer consists of devices that includes all secondary storage devices. The layer above the devices is the "I/O control" that uses device drivers and interrupt routines to communicate with the under lying devices. The I/O control gets commands from the higher layers which is in high level languages such as "eject drive A". These high level commands are translated using device driver into a form such that the devices can understand.

The "basic file system" is responsible for giving high-level commands as an using device driver input to the respective device driver for performing disk related operations like read and write data blocks to the disk. The file organization module keeps track of all information associated with files. The information includes logical name and address of the file, file size etc. This basic information is used to compute the actual physical address of the file required for performing disk operations. A part of file-organization module called free-space manager is responsible for maintaining information about the unallocated or blocks space on the disk. Disk space is measured in blocks which is the least possible unit of storage. The free space manager arranges disk space for creation of new files by the file organization module.

The logical file system is viewed by the user through application programs and the operating system for managing files and folders created by him. Logical file system maintains all the metadata and logical information of the file. Meta data is the data about actual data. Which includes file names, file sizes, starting addresses etc. The logical file system helps the file-organization module with all the file related information when requested. It maintains the directory structure with the help of file control blocks (FCB) that contains file information like file permissions, starting address etc. Other functions of logical file system are ensuring file protection and security by enforcing user permissions, groups etc.

The layered approach enables code receivability as code related to a layer can also be used in other file systems. This eliminates unnecessary repetition of code. Every operating system has its own file system. For example, Unix has a file system called UFS (Unix File System). Windows supports FAT, FAT32 and NTFS file systems. Linux uses a file system called "Extended File System" apart from forty other file systems. Removable media like CD-ROM, DVD-ROM and floppy disks have specific file systems which are mostly supported by all the operating systems.

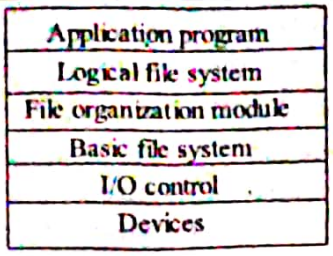


Figure: Layered File System

Q26. Write short notes on file structure.

Answer :

Different types of files have different structure inside. For example the source file of a particular programming language has a structure which matches the expectation of the compiler that reads it. In the same way a binary file is expected to have series of 0's and 1's. Modern operating systems support a variety of file types like text, images, video, audio etc. The default filetype that every operating system must support is an executable file so as to load and run programs.

When an operating system does not support a particular file type, then we have to write new application programs that are capable of reading and understanding the desired file structure. For example, windows XP does not support a "rm" filetype, hence, we have to install application such as real player that can read it.

The files in Mac OS consist of two parts, a resource fork and a data fork. The resource fork include information like labels on buttons etc., which can be relabled in some other language (like Arabic, French etc.) by using tools provided by Mac OS. The data fork consist of program code and data.

The internal file structure consist of a number of variable-sized logical blocks which are combined into one or more fixed size physical block(s) of disk. This is called as *packing technique* which can be done either by user's program or by operating system.

In some operating systems like Unix, all files are treated as stream of bytes and we can address each byte using its offset from the starting or end of the file.

5.2.6 Allocation Methods

Q27. Explain the three allocation methods in file system implementation. Illustrate with proper diagram.

Answer :

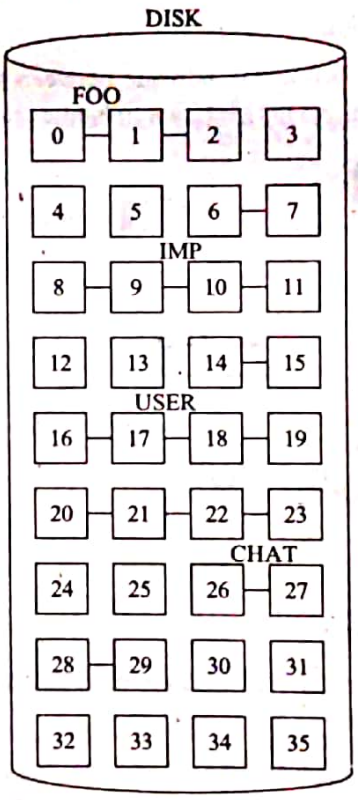
Allocation Methods

An allocation method is considered to be efficient if it allocates space such that no disk space is wasted and accessing of the files takes less time. The three most important file allocation methods are,

1. Contiguous Allocation

In a contiguous allocation method all the files are arranged in a sequential blocks of memory. Therefore, according to this technique, if a file size is *k* blocks and it starts from a block *s* then it spans till still k-1 block numbers. With this approach file access is much faster as all files occupy contiguous blocks. For sequential access of files the physical address of the last referred block is noted, so that file access can start from the next block to avoid repeated accesse of the previous blocks. Direct access is also supported since only the starting address and the block numbers are required.

Though access time is minimal, contiguous allocation does not make efficient use of the disk space. This means that the allocation of space for a new file should be efficient enough so that disk space is not wasted.



Directory

File name	Starting location	Size
Foo	0	3
Imp	6	7
User	14	6
Chat	26	4

Figure: Contiguous Allocation

During storage allocation a question arises that which free blocks or holes should be selected. This problem is similar to the dynamic storage allocation problem. The most common strategies used for this purposes are best fit, worst fit and first fit. First fit is considered to usually be faster than other two strategies. However, best fit is also considered to be efficient than worse fit in terms of both time and disk space utilization.

Even if any of these strategies are used, all of them suffer with external fragmentation. External fragmentation occurs when a request for a new file cannot be satisfied because the largest available continuous chunk of blocks is not sufficient enough for the file. Therefore, even though there is enough space available for the file, but it is not utilized because it is not contiguous.

External fragmentation can be resolved with the help of a scheme called "compaction". Compaction involves rearranging all free memory locations in a sequential order so that contiguous disk space can be allocated for new files instead of space getting wasted. Compaction is carried out in both off line and online modes. In offline mode, all operations are suspended and the file system is unmounted and finally compaction is performed. In this strategy a lot of time is wasted and hence it is avoided. In online mode, compaction is performed alongwith other system operations, but it effects business performance and reduces time wastages.

However, contiguous allocation scheme also requires to know the size of the new file for storage allocation. This requirement looks simple but it might be difficult to determine the size of an output file after its execution. Also if the size of the file is determined in advance and the file takes a lot of time to reach its final size, then a lot of space is wasted when the file has not reached its final size. Therefore, a modified contiguous allocation scheme is used that allocates some fixed amount of space for the first time and if the file requires more space then another chunk of contiguous blocks is allocated. The chunks of free blocks allocated after the first allocation is called "extent" and these two chunks of contiguous blocks are linked to one another.

2. Linked Allocation

This scheme overcomes the problem of contiguous allocation. Any file can be represented as a linked list of disk blocks. The directory entry stores the starting block number and last block number. By taking starting block number we read that block gets the address of next block and move there. Similarly, the whole file is read by traversing the list.

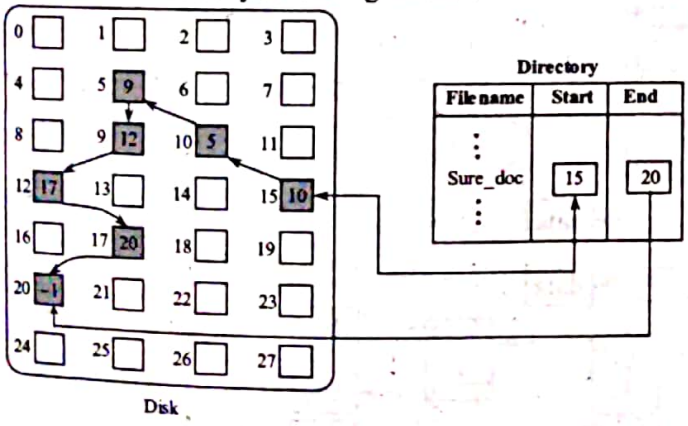


Figure: Linked Allocation Method

Creation of new file involves a new entry in the directory which points to the first block with its pointer value as nil and field size = 0. Thus an empty block is allocated for the new file.

It is resistant towards external fragmentation. The file size may be variable and it can grow and shrink without any difficulty. There are two major disadvantages of this scheme,

- It is suitable for sequential-access-files only. We cannot access any random (i^{th}) block directly. We have to find it by traversing the whole list.
- Some space is wasted for storing the pointers within the blocks.

One solution to the above problem is to combine multiple blocks forming a cluster and allocating it instead of blocks. It helps in improving disk access time and reduces some space requirement, but it is prone to internal fragmentation.

Another important solution is the **File Allocation Table (FAT)** which is used in popular operating systems like MS-DOS, OS/2 etc. FAT is stored in the beginning of each volume and it has one entry for each disk block. The directory entry consist of starting block number of the file and the FAT entry of that block contains address of next block. This linking continues until last block which contains End-Of-File (EOF) table is reached.

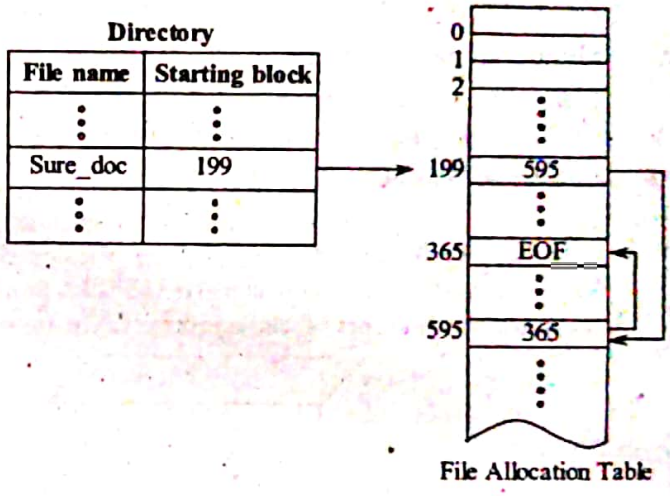


Figure: File Allocation Table (FAT)

3. Indexed Allocation

This scheme stores pointers to all the blocks of a particular file at one location called as index block. It is a simple modification of linked allocation. The directory stores the address of this index block. Hence, after getting a particular index block we can access the whole file.

When the file is created, OS provides an index block to it which contains nothing. When a i^{th} block is written, its address is stored in the i^{th} entry of index block. This scheme is resistant to fragmentation but some space is wasted to store the index blocks for each file. The size of index block can not be determined. If large index blocks are used, then for small files most of their entries will be empty hence space will be wasted. If small index blocks are used then they may not be able to accommodate all pointers of file. Hence, the following schemes are used,

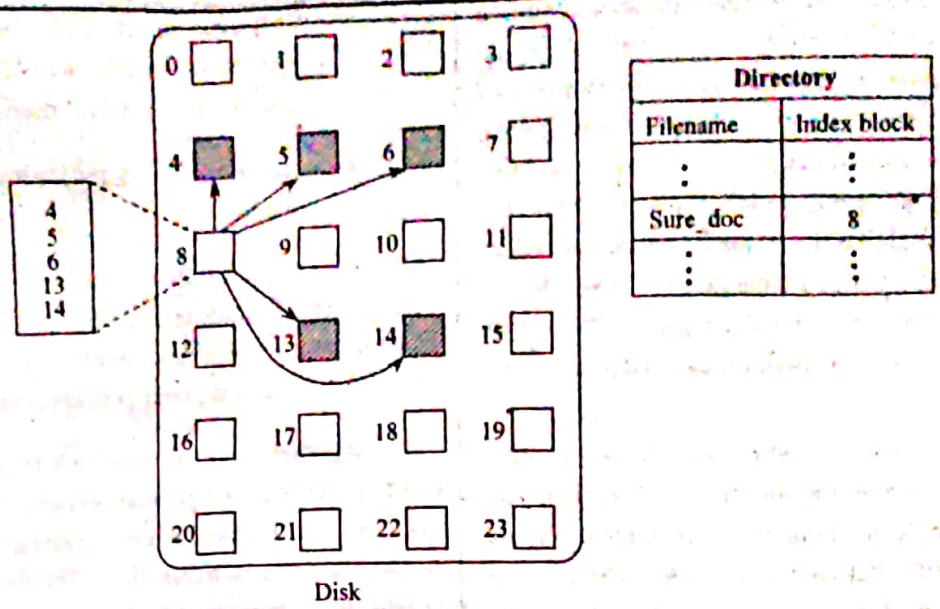


Figure: Indexed Allocation Scheme

(a) Linked Scheme

It creates a single disk block for accommodating small files. As their size increases, it links together several index blocks.

(b) Multilevel Index

In this scheme multiple levels of index blocks are used. The first level index block points to the second level index blocks which points to actual data blocks. Similarly, we can have many levels of index blocks present.

(c) Combined Scheme

This scheme is implemented in unix file system. Each inode of unix stores 15 pointers. The first 12 are called **direct blocks** because they point directly to data blocks. The next pointer is **single indirect block** which is a pointer pointing to a index block that contains address of actual data blocks. Similarly, we have two and three levels of index blocks for **double direct block** and **triple direct block** respectively. The following figure shows the same.

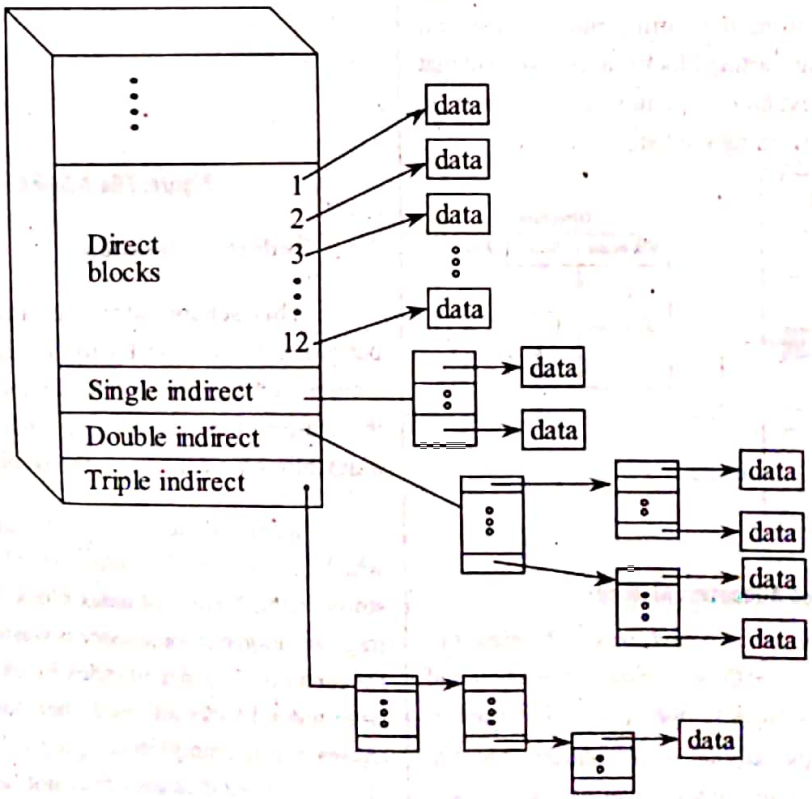


Figure: Index Node of Unix File System

5.2.7 Free Space Management

Q28. State and explain four approaches to free space management.

Answer :

Free Space Management

When files are stored, disk space is consumed and the same is released when they are deleted. The system should maintain a free-space list to record all disk locations that are not allocated to any file or directory. Whenever any file is created, space is allocated to it from free space list and the allocated block numbers are removed from this list. There are several methods to keep track of free-space in disk.

1. Bit Vector

In this method a bit map or bit vector is maintained where for each block, one bit is present. If that bit value is 1, block is supposed to be free, otherwise if it is '0' block is in use.

Example

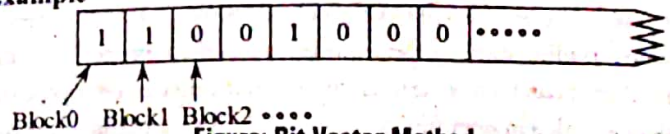


Figure: Bit Vector Method

The above figure depicts the blocks 0, 1 and 4 are free and blocks 2, 3, 5, 6 and 7 are allocated to some file. This method is simple to implement and free space can be found easily by using simple bit manipulation instructions of Intel × 86, Motorola 68000 series and many popular processors.

The bit vector has to be kept in main memory so as to improve efficiency and a backup copy of it should be written on disk periodically to recover if any loss occurs in future. A hard disk of 40 GB with 1 kB sized blocks needs only 5 MB of memory for storing its bit vector.

2. Linked List

In this approach all free blocks of disk are linked together using pointers. Each block has a pointer which points to the next free block.

This scheme requires considerable amount of I/O time to traverse the list, but it is not frequently done because the operating system usually uses the first free blocks and does not need to traverse the list.

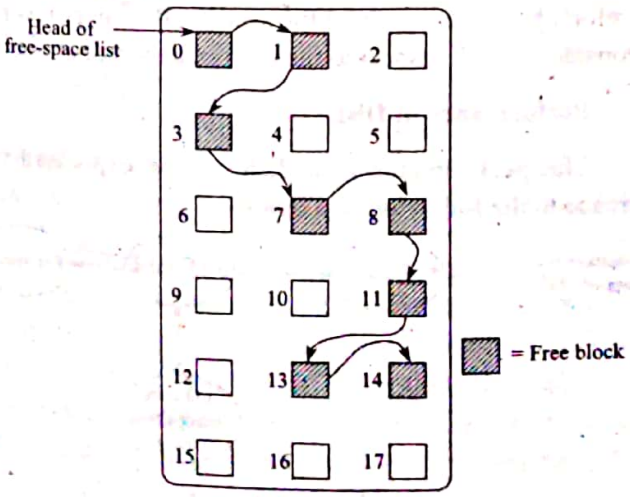


Figure: Linked List Method

3. Grouping

It is a modified version of linked list method. Here a group of free blocks are linked together. In each group the last block contains the address of next group of free blocks. Memory can be allocated in chunks of blocks and large number of free blocks can be found easily.

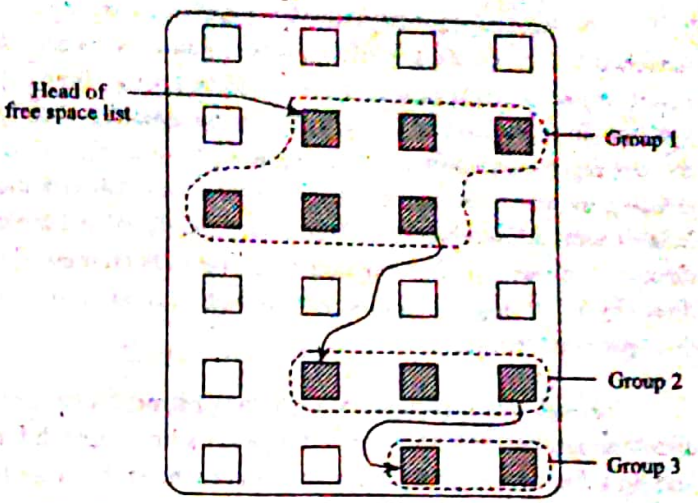


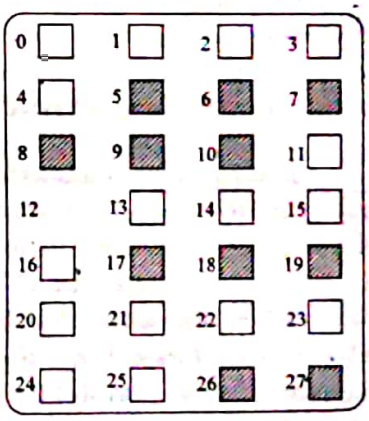
Figure: Grouping of Free Space

4. Counting

If contiguous memory allocation scheme is used then many contiguous blocks will be allocated and freed simultaneously. '*n*' number of contiguous blocks may be free at several places. In counting scheme the address of first free block is taken and a count of successive free blocks is stored. Hence, a table is maintained which consist of two columns, one the address of first free block and second the number of successive free blocks following that address.

Starting block number	Count
5	5
17	2
26	1

Free-space table



Disk

Figure: Free Space Management Using Counting Method

5.2.8 Directory Implementation

Q29. Describe the linear list and hash table directory implementation methods.

Answer :

Directory Implementation

Implementation of a directory structure involves selection of efficient directory allocation and management algorithms from the available algorithms.

1. Linear List

This is the most simplest of all directory implementation methods. It is easy to implement, but it takes a lot of time to execute. It simply maintains a sequential list of file names pointing to their corresponding data blocks.

This kind of implementation requires a filename to be searched before creating a file with that name. Similarly, a delete operation also requires a linear search for the directory and then disallocate the space occupied by the directory. Finally the corresponding entry is removed from the list. Instead of deleting the entry we can name the entry as unused with the help of used-unused bit, or include it into the list of available directory entries. Another method could be to decrement the directory length and transfer the directory entry contents to any free spaces on the disk.

Though this approach is simple to implement it has few disadvantages as well. This method requires a linear search for finding a file before performing any operation. This makes it slow and users would experience this property very frequently. Even if a binary search technique is used instead of linear search, though it improves average search time, but it still needs a sorted list of directory entries to search.

2. Hash Table

In this technique a hash table is used only with the linear list for storing directory entries. The hash table makes use of a hash function that takes an input value based on the filename and produces an output as a reference to the corresponding directory entry in the linear list. Therefore, all file operations consume very less time to execute. However necessary arrangement should be made made to handle collisions. A collision is a situation where more than one file name is hashed to the same location.

Disadvantages of this technique are that a hash table is usually of fixed size and the performance of the hash functions is also dependent on the size of the hash table. Therefore, whenever a new file is to be added after all the available free entries have been used the hash table should be expanded to accommodate addition of new files and the existing directory entries should be organized in such a way that the new hash function also maps input values to there corresponding directory entries.

A solution to the above problem could be to use a chained-overflow hash table. A chained-overflow hash table consists of a linked list that stores all hashed entries. Now, if more than one filename hashes to the same entry it can be added as another node to the linked list. Though this approach eliminates all the disadvantages of linear list directory implementation, searching directory names can consume sometime since, it involves traversing the linked list. However, this technique is considered to be efficient than linear list directory implementation method.

5.2.9 Efficiency and Performance

Q30. Comment on the efficiency and performance of a disk.

Answer :

Efficiency and Performance of a Disk

(i) Efficiency of Disk

The efficiency of the disk in various operating systems is based on two things. They are,

(a) Capacity of the disk.

(b) Various procedures that are required to keep disk efficient.

Let us take an example of UNIX operating system where efficiency of its disk is decided with the way the "Index nodes" are allocated. These nodes play very important role in UNIX systems, because if allocation of these nodes is not done properly for the disk, then it can have a serious impart on the disk like suppose the disk assembled in operating system of UNIX is new and obviously there will be no data in it, but still its capacity will be decreasing because of index nodes not properly assigned for the disk. So, before putting any new disk into the UNIX operating system, it will be much better to have an idea of the way the index nodes should be allocated for the disk.

Another important point to be noted is the data which is stored in such a disk. There will be different users who will be working on their files and after work they will store the file in the same directory. So, if one user wants to retrieve its file, it will be very hard as there will be a cluster of files. To avoid this, inconvenience UNIX systems consists of two important properties of the file which are, the date on which user last accessed or read the file i.e., Last Access Date and next the date on which user did the last write operation i.e., Last Write Date.

These two properties makes the directory entry to be edited every time when a file is used which can be considered as inefficient in case of files that are accessed most oftenly. Therefore, before designing a file system all these points must be considered with respect to the performance and efficiency.

(ii) Performance of Disk

The performance of a disk can be explained with reference to the following two figures,

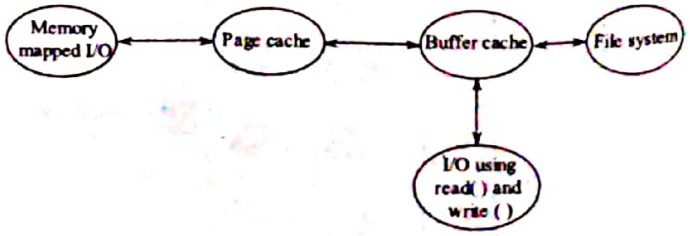


Figure (1)

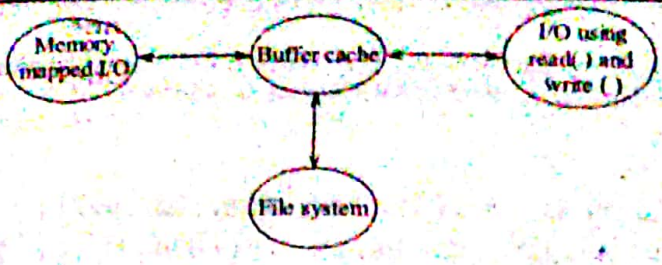


Figure (2)

From the figures (1) and (2) it can be observed that there are similarities and differences between them. The similarities are that both figures contain memory mapped I/O, buffer cache, file system, I/O using read( ) and write( ) where read( ) and write( ) are system calls. The difference between them is that there is no page cache in figure (2).

Let us see working process of each of the figures in detail.

Working Process Done for Improving Performance in Figure (1)

In figure (1), concept called memory mapping is used in which the data is first taken from the file system and is saved in a cache called buffer or block cache of the file system. In most disk processes, virtual memory concept is used. In this case, it is not used. For this reason, another cache is employed by the file system which is called page cache where data sent by the buffer cache will be stored. This procedure is called double caching since two cache's are used.

Working Process Done for Improving Performance in Figure (2)

The working process of figure (2) is almost similar to figure (1). The difference is that virtual memory concept is used in figure (2). Therefore only one cache is used by file system which is buffer cache and not page cache.

In both cases of caching disk block and a page. least recently used algorithm was preferred till the advent of solaries page-caching algorithm. These algorithms provide the feature of sharing free space present in the disk by various processes and page cache. Due to this reason, it might be possible that all the free space an be utilized by the page cache and hence the advanced version of solaries i.e., sloaries and uses limiting of available memory to both page cache and process.

Performance of a system also depends on the synchronous and asynchronous writes to the file system. This is because a synchronous write is carried in the sequence decided by subsystem (mainly in receiving sequence) whereas an asynchronous write allots the control to the called program which occurs in most of the situations. To avoid this, flag is used with which writes occur synchronously.

5.3 SECONDARY - STORAGE STRUCTURE

5.3.1 Disk Structure, Disk Scheduling Algorithms, Disk Management

Q31. Explain briefly about disk structure.

Answer : Model Paper-I, Q17(a)

Disk Structure

Disk drives uses a logical block which is nothing but a large one-dimensional array and is the smallest unit of transfer. Its size is usually 512 bytes or 1024 bytes. These logical blocks are mapped onto the sectors of the disk sequentially. The first sector of the first track is on the outermost cylinder or track is called as sector-0.

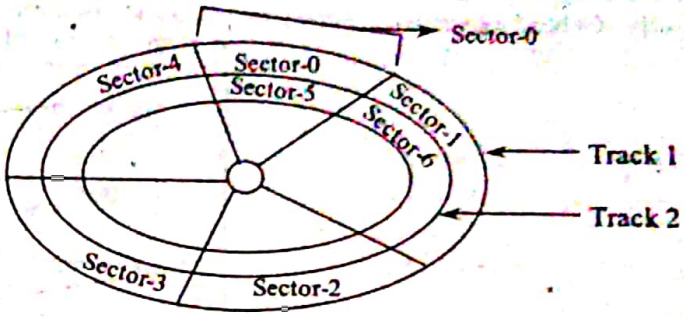


Figure: Disk Structure

The mapping goes on until all sectors on that track are covered, then it moves to the inner track and continues mapping. As we move from outer tracks to the inner tracks, the density of sectors per track decreases i.e., the density of bits in outermost tracks is greater than density of bits in inner tracks. So, Constant Angular Velocity (CAV) is used to keep the transfer rate constant. In CAV, the drive increases the speed of spindle motor when head is on outer tracks and decreases the speed when head is on inner tracks.

Some disk drives also uses Constant Linear Velocity (CLV) if the density of bits per track is uniform. Now-a-days the number of sectors per track is increasing due to advanced opto-magnetic technology. Thereby, allowing gigabytes and terabytes of storage in small space.

Q32. Explain various disk scheduling algorithms with an example.

Answer : Model Paper-III, Q17

Types of Disk Scheduling Algorithms

The various disk scheduling algorithms are,

1. FCFS (First Come First Serve)
2. SSTF (Shortest Seek Time First)
3. SCAN or Elevator
4. C-SCAN (Circular SCAN)
5. LOOK
6. C-LOOK (Circular Look).

1. First Come First Serve (FCFS) Scheduling

It is the simplest among all scheduling algorithms. Here, the request which comes first is served first. But it cannot promise fastest service always. Consider the following example in which the requests for the disk block comes as follows.

Example

104, 189, 43, 128, 20, 130, 71, 73.

Suppose, the head is initially at 59, it will move to 104, then to 189, 43, 128, 20, 130, 71 and finally reads block 73.

Total head movements = |59 – 104| + |104 – 189| + |189 – 43| + |43 – 128| + |128– 20| + |20 – 130| + |130 – 71| + |71 – 73|

= 45 + 85 + 146 + 85 + 108 + 110 + 59 + 2 = 640

Average seek length = 640/8

= 80

It is not efficient scheme, consider the servicing mechanism for requests 128, 20, 130. As 128 is first head moves from 128 to 20 (108 movements), then it moves back to 130 (130 – 20 = 110 movements). It would have been efficient if 128 and 130 are serviced together (128 – 130 = 2 movements) followed by 20 (130 – 20 = 110 movements). This approach would have avoided nearly 106 head movements, thereby improving the performance.

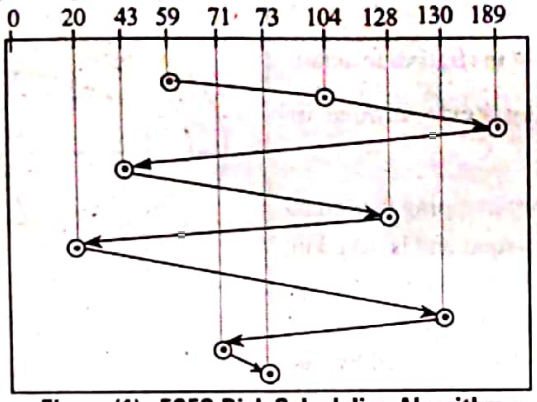


Figure (1): FCFS Disk Scheduling Algorithm

2. Shortest Seek Time First (SSTF) Scheduling

This algorithm serves the requests which are close to the current head position i.e., the next request is selected such that it has minimum seek time from current head position.

Let us apply the SSTF algorithm on the previous example, initially head is on 59 then it goes to 71 and next to 73. From here 43 is closer than 104, hence 43 is served. Then 20, 104, 128, 130 and 189 are served.

Total head movements = |59 – 71| + |71 – 73| + |73 – 43| + |43 – 20| + |20 – 104| + |104 – 128| + |128 – 130| + |130 – 189|

= 12 + 2 + 30 + 23 + 84 + 24 + 2 + 59

= 236

Average seek length = 236/8

= 29.5

This scheme provides a good improvement over FCFS, but it may lead to starvation if requests keeps on coming. Consider an example that if incoming requests are closer to head then, they are served and farther requests are kept pending causing starvation.

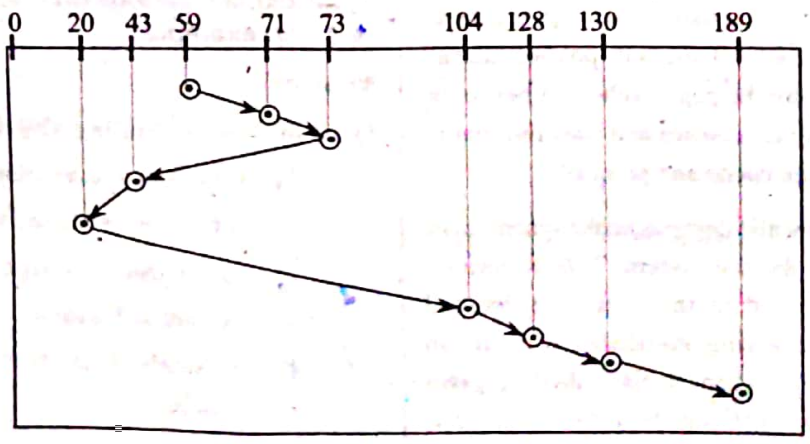


Figure (2): SSTF Disk Scheduling Algorithm

3. SCAN Scheduling or Elevator Algorithm

In this algorithm the disk arm moves in one direction, servicing all the requests that comes along that route until last track is reached and then it reverses the direction and moves to the other end servicing the requests along this way also. This action is similar to that of an elevator (or lift) of a building and hence it is also called as *elevator* algorithm.

Let us consider, the previous example of disk requests 104, 189, 43, 128, 20, 130, 71, 73 and initial head position be 59. If we apply SCAN scheduling algorithm, then from 59 the head moves towards the first track i.e., 0 servicing the requests 43 and 20 and finally reaches to 0. Then the disk arm will reverse and starts moving towards the other end servicing requests 71, 73, 104, 128, 130 and 189. Hence, the sequence of servicing requests is now,

59, 43, 20, 0, 71, 73, 104, 128, 130, 189

Total head movements = |59 – 43| + |43 – 20| + |20 – 0| + |0 – 71| + |71 – 73| + |73 – 104| + |104 – 128| + |128 – 130| + |130 – 189|

= 16 + 23 + 20 + 71 + 2 + 31 + 24 + 2 + 59 = 248

Average seek length = 248/8 = 31

The limitation of this scheme is that the waiting time of some requests increases. Most of the requests are present on the other side of the disk so, it would be good if that side is scanned first.

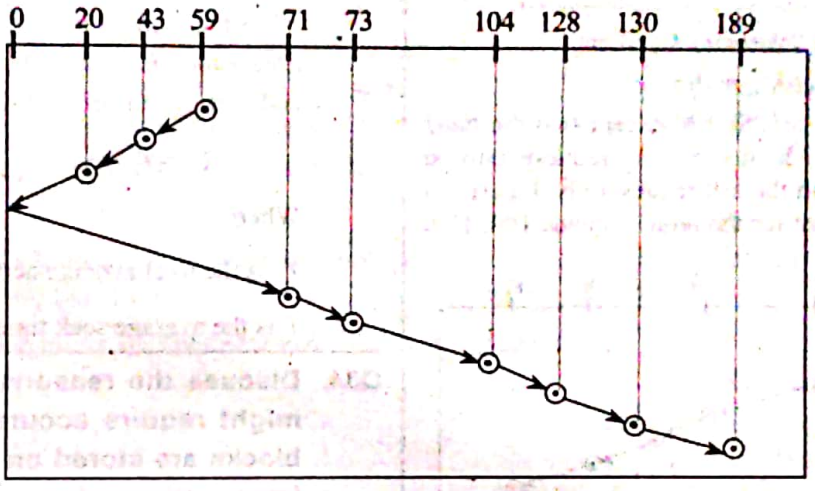


Figure (3): SCAN Scheduling Algorithm

4. C-SCAN Scheduling

It is a modified version of SCAN scheduling which overcomes the limitation of SCAN and provides uniform wait time. It is same like SCAN except that the requests are serviced only in one direction or trip.

Let us apply C-SCAN to our previous example with head initially at 59. As there are more requests on the right side, the head starts moving towards that side servicing 71, 73, 104, 128, 130, 189 and finally reaches the end. Then, it immediately returns to the other end without servicing any requests in this direction and starts at 0 again. The following figure (4) depicts the same.

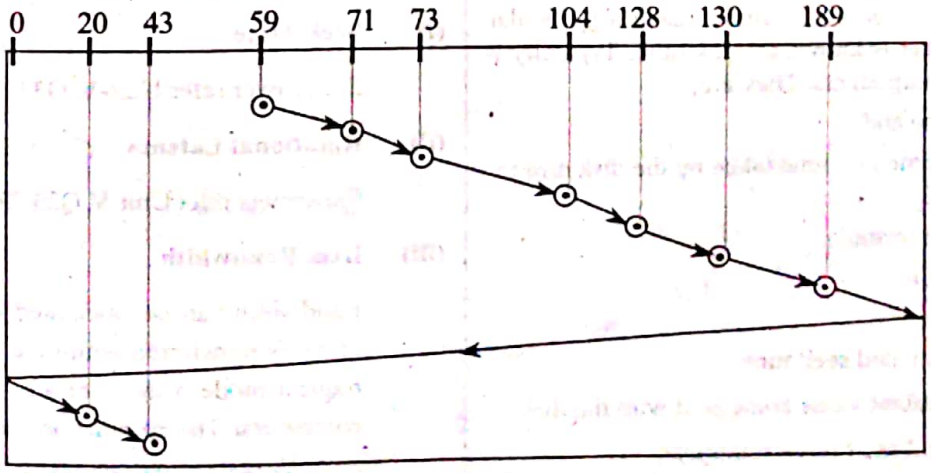


Figure (4): C-SCAN Scheduling Algorithm

5. LOOK Scheduling

It is a modified version of SCAN which overcomes its limitations by not visiting the extreme ends unnecessarily. It looks for the last request in one particular direction and then reverses from there. It services requests in both directions. Figure (5) shows the head movement when this algorithm is applied to the previous example.

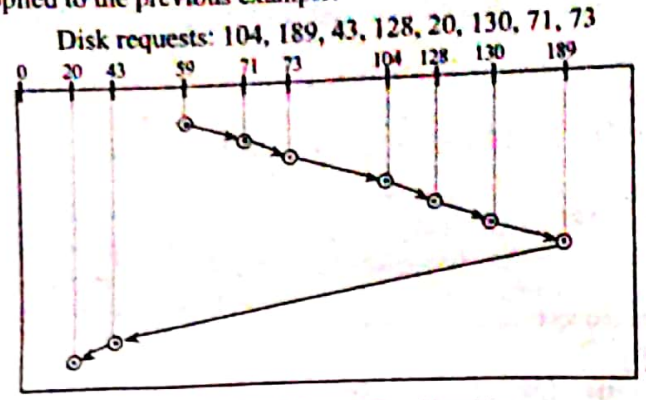


Figure (5): LOOK Scheduling Algorithm

6. C-LOOK Scheduling Algorithm

C-LOOK is similar to C-SCAN except that the head does not visit the extreme ends. It serves the requests only in one direction and ends up till the last request only. Figure (6) describes the head movement for the request queue 104, 189, 43, 128, 20, 130, 71, 73.

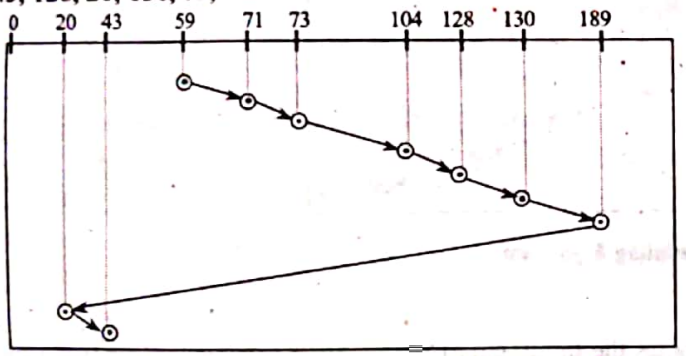


Figure (6): C-LOOK Scheduling Algorithm

Q33. Explain various disk performance parameters.

Answer : Model Paper-I, Q17(b)

(a) Seek Time

The time taken by the disk arm to reach a particular track on the platter is known as seek time. Typically it considers two components. They are,

(i) Startup time and

(ii) Traversal time i.e., time taken by the disk arm to traverse the tracks.

It is given by the formula

$$T_s = m \times n + s$$

Where,

T_s = Estimated seek time

m = Constant value associated with the disk

n = Number of traversed tracks

s = Startup time.

(b) Rotational Latency

It is also called as rotational delay which is nothing but the time taken by the disk head to reach a position where it can perform read/write operation on a particular area. Typical value of rotational delay in 2 ms for all the disks except floppy disk whose delay ranges from 100 to 50 ms.

(c) Transfer Time

The time taken by the disk head to perform read/write operations (or) to transfer the data is called as transfer time. Its value depends on the rotation speed of the disk and it can be given by the following equation,

$$T_t = \frac{b}{rN}$$

Here, T_t is the transfer time, 'b' is the total number of bytes to be copied to the disk 'r' is rotation speed and 'N' is the number of bytes present on the track. Access time which is nothing but the sum of rotational delay and seek time is given by,

$$T_a = T_s + \frac{1}{2r} + \frac{b}{rN}$$

Where,

T_a is the total average access time and

T_s is the average seek time.

Q34. Discuss the reasons why operating system might require accurate information on how blocks are stored on a disk. How can the OS improve the performance of the file system with this knowledge?

Answer :

The main reason for the operating system to have an accurate information about the blocks storage on disk is to use the hardware efficiently. This can be accomplished if the hard disk has fast access time and large disk bandwidth.

(i) Seek Time

For answer refer Unit-V, Q33, Topic: Seek Time.

(ii) Rotational Latency

For answer refer Unit-V, Q33, Topic: Rotational Latency.

(iii) Disk Bandwidth

Bandwidth can be calculated by dividing the number of bytes transferred with time delay occurred between request mode b the user and the most recent request completion. The information needed by the process while requesting the input/output operation to or from the disk is,

(a) Information about the type of operation (i.e, input or output)

(b) The disk address for information transfer

(c) The memory address for information transfer

(d) The number of sectors that are to be transferred.

If the request could not serviced due to the unavailability of disk drive and controller. It is placed in a queue of pending requests specially in multiprogramming system with multiple processes. This lets the operating system to decide which pending request to be serviced next which can be done using any of the disk scheduling algorithms available.

Selecting of the disk scheduling algorithm should be made carefully. These algorithms help operating system in improving the performance of the system. There are several algorithms for disk scheduling depending on the seek distances. They are,

For remaining answer refer Unit-IV, Q34.

Q35. Discuss about N-step-SCAN policy for disk scheduling.

Answer :

N-step-SCAN Policy

N-step-SCAN is a disk scheduling algorithm that scans N number of disk request queue at a time. This queue is fragmented into subqueues, which are processed by using a SCAN policy. Each subqueue is of length N where $N = 1, 2, 3,.., n$. During the processing of a queue, if some new requests arrive, then they must be placed in some other queue. At the completion of a scan, if the requests that are less then N are left, then they must be processed with the other scan. If the value of N is large i.e., a large number of requests occur in a subqueue, then the performance of N-step-SCAN policy become similar to the SCAN policy. And if the value of N is 1, then FIFO policy is used for processing the requests.

Example

Let the requested tracks from a moving head disk with 200 tracks, which are numbered from 0 to 199 be in the following order: 122, 90, 160, 24, 102, 89, 143, 18, 67.

For $N = 2$, the requested tracks are divided into the following subqueues.

subqueue 1 = {122, 90}

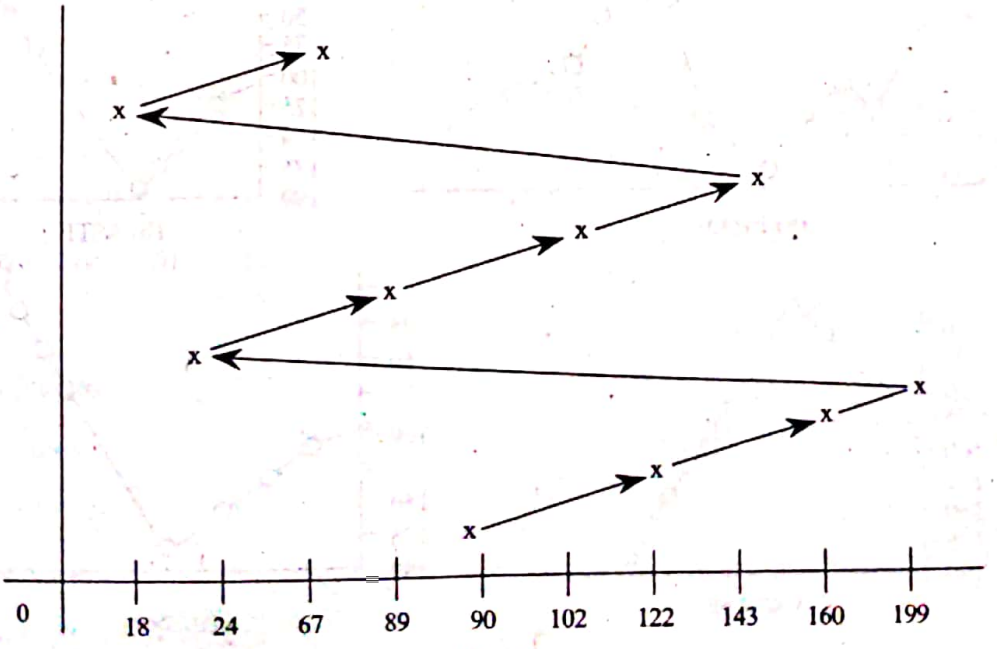
subqueue 2 = {160, 24}

subqueue 3 = {102, 89}

subqueue 4 = {143, 18}

subqueue 5 = {67}

Using N-step-SCAN policy, the tracks are processed as shown in the following figure.



Figure

Initially, subqueue 1 containing the request tracks 122 and 90 is chosen for disk scheduling. As the track 90 is near to 0, it is processed first. After processing the track 90, the track 122 is processed. Now, the disk head is located at track 122. A track that is nearer to 122 from the subqueue 2 is selected for processing. Hence, the track 160 is selected. As there are no more tracks to be processed between 160 and 199, the head moves to last track i.e., 199 and changes its direction so as to continue with the processing of request for track 24. Then, the subqueue 3 will be processed, which contains the request tracks 102 and 89. As 89 is nearer to 24, it is processed first followed by 102. Similarly, the requests in subqueue 4 and subqueue 5 are processed.

Q36. Suppose we have a disk with 200 tracks. The disk head starts at track 100 and moving in the direction of decreasing track number. For the following sequence of disk track requests 27, 129, 110, 186, 147, 41, 10, 64, 120. Compute the average seek time for the following disk scheduling algorithms, FIFO, SSTF, Scan, C-Scan. Queue contains the following request in order at time 0, 27, 129, 110, 186, 147, 41, 20, 64, 120. Compute the average time to service a request for the disk head scheduling algorithm FCFS.

Answer :

Seek time formula is,

$$T_S = m \times n + S$$

Where, T_S – Estimated seek time

n – Number of tracks traversed

m – Constant that depends on the disk drive

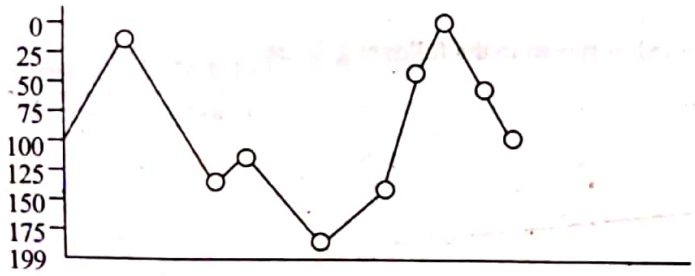
S – Start-up time.

Here, in our problem the estimated values for m, S are,

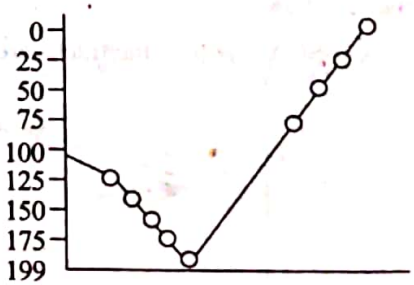
m – 0.3 msec

S – 20 msec.

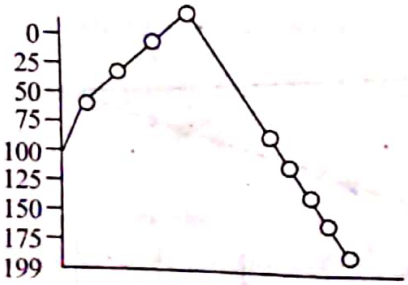
We have to first compute the value of 'n' for different disk scheduling policies FIFO, SSTF, SCAN, C-SCAN, Queue.



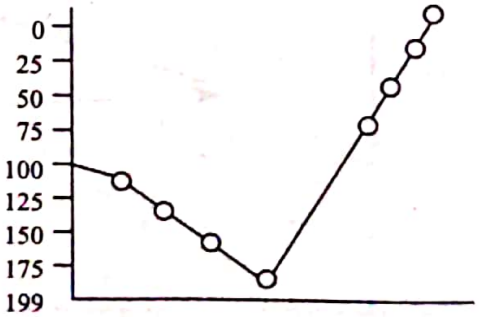
(a) FIFO



(b) SSTF



(c) SCAN



(d) C-Scan

Figure

(a) FIFO (Starting at track 100)		(b) SSTF (Start at track 100)		(c) Scan (Start at track 100 in decreasing track direction)		(d) C-Scan (Start at track 100 in increasing track direction)	
Next track accessed	Number of tracks traversed	Next track accessed	Number of tracks traversed	Next track accessed	Number of tracks traversed	Next track accessed	Number of tracks traversed
27	73	110	10	64	36	110	10
129	102	120	10	41	23	120	10
110	19	129	9	27	14	129	9
186	76	147	18	10	17	147	18
147	39	186	39	110	100	186	39
41	106	64	122	120	10	64	122
10	31	41	23	129	9	41	23
64	54	27	14	147	18	27	14
120	56	10	17	186	39	10	17
Average seek length	61.8	Average seek length	29.1	Average seek length	29.5	Average seek length	29.1

Calculation of average seek-time for different scheduling algorithms is shown in table.

Average seek-time for different scheduling algorithms,

FIFO

$m = 0.3$ msec

$S = 20$ msec

$n = 61.8$

$T_S = m \times n + S$

$T_S = 0.3 \times 61.8 + 20 = 38.54$

SSTF

$m = 0.3$ msec

$S = 20$ msec

$n = 29.1$

$T_S = m \times n + S$

$T_S = 0.3 \times 29.1 + 20 = 28.73$

Scan

$m = 0.3$ msec

$S = 20$ msec

$n = 29.5$

$T_S = m \times n + S$

$T_S = 0.3 \times 29.5 + 20 = 28.85$

C-Scan

$m = 0.3$ msec

$S = 20$ msec

$n = 29.1$

$T_S = m \times n + S$

$T_S = 0.3 \times 29.1 + 20 = 28.73$

Queue contains 0, 27, 129, 110, 186, 147, 41, 20, 64, 120 The table for FCFS is as follows.

Starting	Number of	track tracks traversed
0	0	
27	27	
129	102	
110	19	
186	76	
147	39	
41	106	
20	21	
64	44	
120	56	
Average seek length	49.0	

$$T_s = m \times n + S$$

$$= 0.3 \times 49 + 20$$

$$= 34.7$$

So, the average time to service a request is 34.7 using a FCFS scheduling.

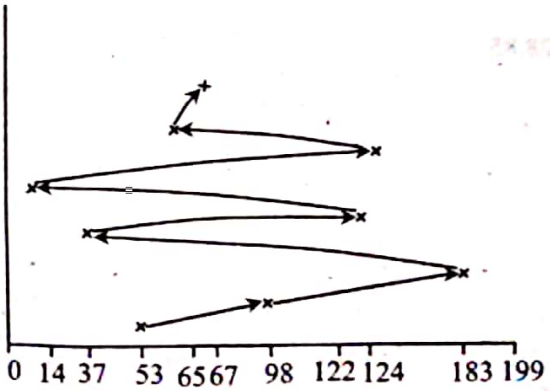
Q37. Consider the following disk queue with requests for I/O to block on cylinders 98, 183, 37, 122, 14, 124, 65, 67 in that order, using FCFS algorithm of the disk head is initially at cylinder 53, find the total head movement in cylinders. Also provide the necessary diagram to show the head movement for the above queue.

Answer :

The given order is : 98, 183, 37, 122, 14, 124, 65, 67

Also given,

The disk head is initially at cylinder 53. The following diagram shows the movements of disk head using FCFS (First Come First Serve) algorithm.



Figure

Head Movements

From 53 to 98 = 45
98 to 183 = 85
183 to 37 = 146
37 to 122 = 85
122 to 14 = 108
14 to 124 = 110
124 to 65 = 59
65 to 67 = 2

∴ The total head movements = 640.

Q38. Suppose the head of a moving-head disk with 200 tracks, numbered 0 to 199, is currently serving a request at track 143 and has just finished a request at track 125. If the queue of requests is kept in FIFO order: 86, 147, 91, 177, 94, 150, 102, 175, 130. What is the total head movement to satisfy these requests for the following disk scheduling algorithms?

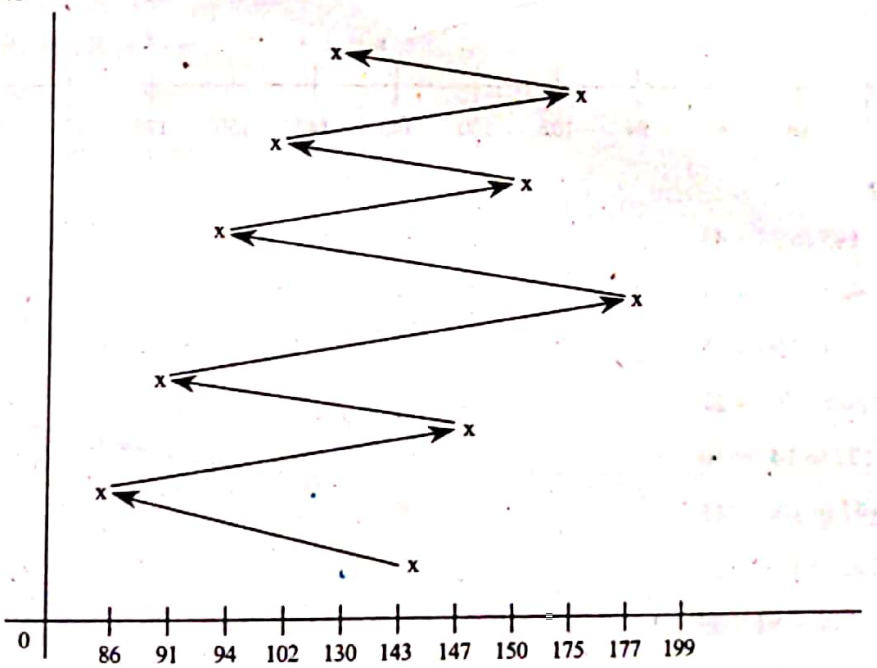
(a) FCFS
(b) Random
(c) PRI
(d) SCAN
(e) SSTF
(f) C-SCAN.

Answer :

The given FIFO order is: 86, 147, 91, 177, 94, 150, 102, 175, 130.

(a) FCFS (First Come First Serve)

Head is at track: 143



Head Movements

From 143 to 86 = 57
86 to 147 = 61
147 to 91 = 56
91 to 177 = 86
177 to 94 = 83

94 to 150 = 56

150 to 102 = 48

102 to 175 = 73

175 to 130 = 45

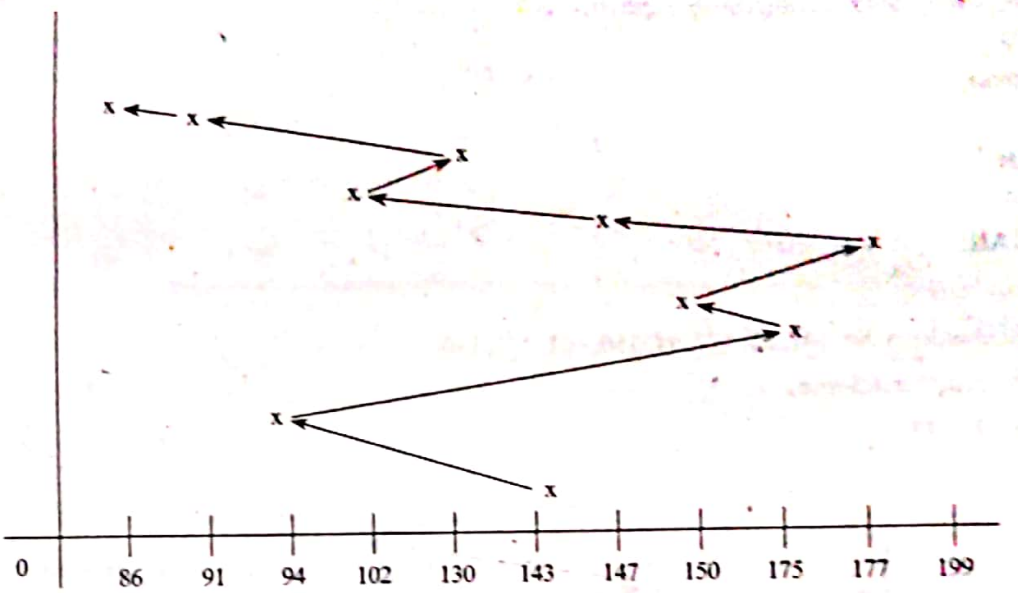
∴ Total head movements = 565

(b) Random

In this disk scheduling algorithm, the head moves to random positions.

Let the random order be: 143, 94, 175, 150, 177, 147, 102, 130, 91, 86

Head is at track: 143



Head Movements

From 143 to 94 = 49

94 to 175 = 81

175 to 150 = 25

150 to 177 = 27

177 to 147 = 30

147 to 102 = 45

102 to 130 = 28

130 to 91 = 39

91 to 86 = 5

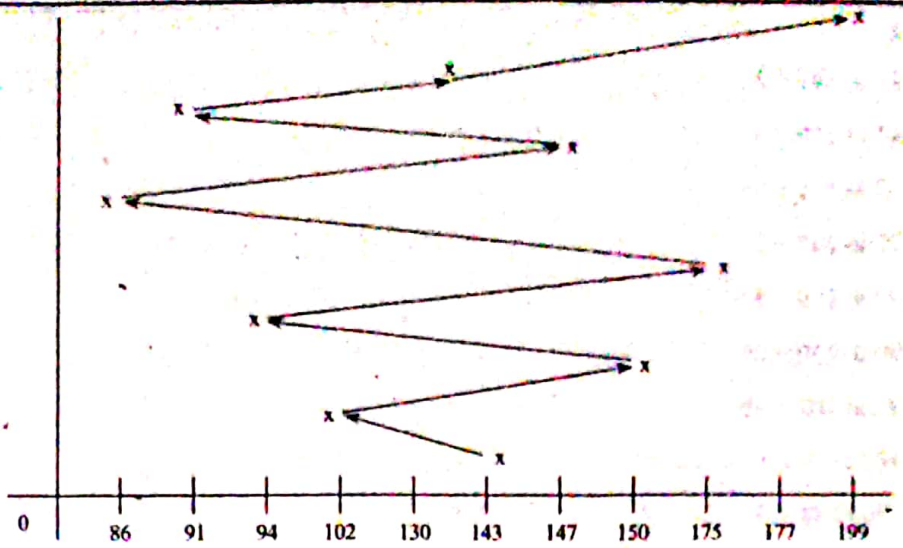
∴ Total head movements = 329

(c) PRI (Priority)

Let the priority order of the given tracks be:

102, 150, 94, 175, 86, 147, 91, 130, 177.

Head is at track: 143



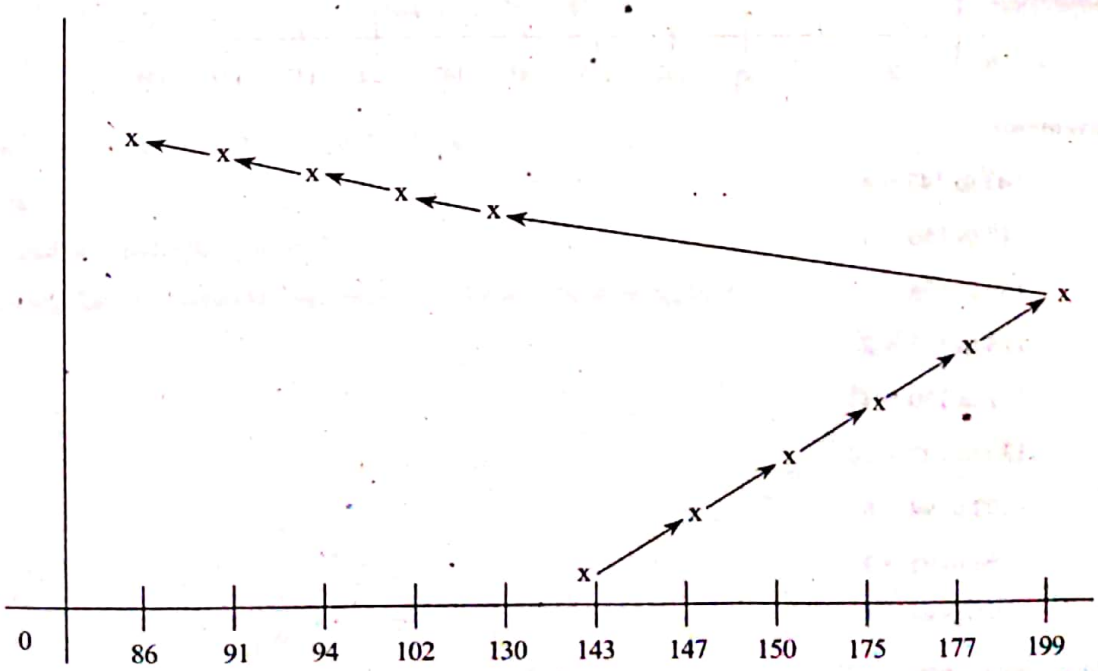
Head Movements

From 143 to 102 = 41
102 to 150 = 48
150 to 94 = 56
94 to 175 = 81
175 to 86 = 89
86 to 147 = 61
147 to 91 = 56
91 to 130 = 39
130 to 177 = 47

∴ Total head movements = 518

(d) **SCAN**

Head is at track: 143



Head Movements

From 143 to 147 = 4

147 to 150 = 3

150 to 175 = 25

175 to 177 = 2

177 to 199 = 22

199 to 130 = 69

130 to 102 = 28

102 to 94 = 8

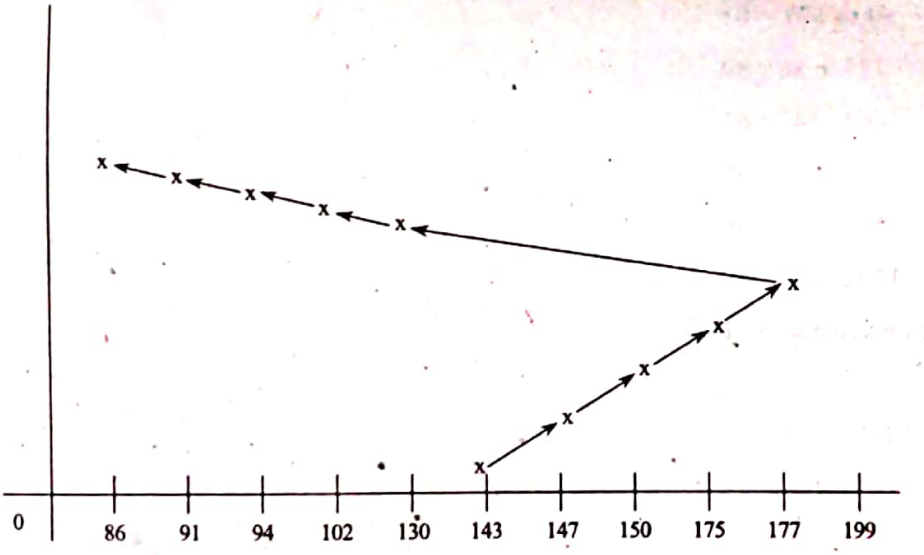
94 to 91 = 3

91 to 86 = 5

∴ Total head movements = 169

(e) SSTF (Shortest Seek Time First)

Head is at track : 143



Head Movements

From 143 to 147 = 4

147 to 150 = 3

150 to 175 = 25

175 to 177 = 2

177 to 130 = 47

130 to 102 = 28

102 to 94 = 8

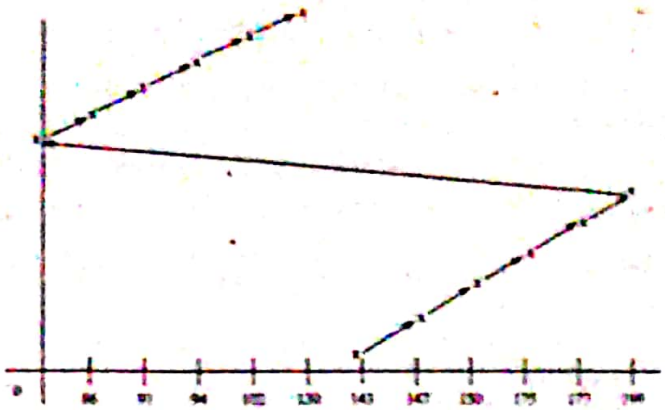
94 to 91 = 3

91 to 86 = 5

∴ Total head movements = 125

(f) C-SCAN

Head is at track : 143



Head Movements

From 143 to 147 = 4

147 to 150 = 3

150 to 175 = 25

175 to 177 = 2

177 to 199 = 22

199 to 0 = 199

0 to 86 = 86

86 to 91 = 5

91 to 94 = 3

94 to 102 = 8

102 to 130 = 28

∴ Total head movements = 385

Q39. Explain about SCAN disk scheduling algorithm. Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 37, 122, 14, 124, 65, 67 in that order. Assume the disk head is initially at cylinder 53. Show diagrammatically the SCAN disk scheduling. Find out the total head movements for SCAN Algorithm.

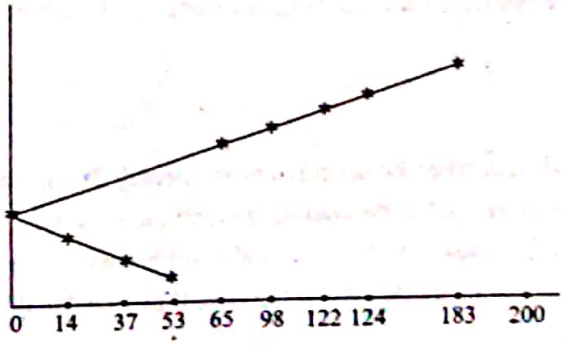
Answer :

The given order is: 98, 183, 37, 122, 14, 24, 65, 67.

Also given,

The disk head is initially at cylinder 53.

The following diagram shows the movements of the disk head using SCAN disk algorithm.



Figure

Head Movements

From 53 to 37 = 16
37 to 14 = 23
14 to 0 = 14
0 to 65 = 65
65 to 67 = 2
67 to 98 = 31
98 to 122 = 24
122 to 124 = 2
124 to 183 = 59

∴ The total head movements = 236.

Q40. Briefly explain,

(i) Disk formatting

(ii) Boot block

(iii) Bad blocks.

Answer :

Disk Management

It concerns with initialization of disk, booting from disk and bad block recovery etc.

(i). Disk Formatting

A new uninitialized magnetic disk is just a platter coated with magnetic material. This blank disk needs to be initialized so that, it is capable of storing and retrieving data. It is done by using a process called *formatting* which divides the total disk into tracks and sectors. It is also called as *low-level formatting* or *physical formatting* and it is usually done in the factory by the manufacturer of that disk. In this formatting each sector is created composing of three sections, they are a header, a trailer and a data area (like 512 bytes or 1024 bytes etc.,). The header and trailer stores information about that sector like sector number and Error Correcting Code (ECC) which is used to correct bits when some data is corrupted.

But for using this to store files, operating system still performs two more steps. They are,

(a) Partitioning

The whole disk is divided into one or more groups of cylinders and operating system can use each partition as a separate disk.

(b) Logical Formatting

The operating system creates a file system by storing its data structures like tables of free and allocated area, *i*-nodes and an empty initial directory.

(ii) Boot Block

Whenever the computer is switched on or rebooted, an initial program called bootstrap program is executed which initializes the CPU registers, checks the required hardware and helps operating system to be loaded into main memory from hard disk. The bootstrap program is usually stored in ROM chips, which is read only and hence cannot be effected by computer viruses but it has a limitation that once it is stored, it cannot be modified and the only thing we can do is to change the hardware chip which could be expensive.

Hence, now-a-days bootstrap program is split into two parts, one is called tiny bootstrap that is stored in ROM and its only job is to bring the second part called full bootstrap that is stored in "the boot block", which is a fixed location on the disk. The boot block is present on a separate partition called a *boot partition* or *boot disk* or *system disk*. The full bootstrap program is complicated and it loads the entire operating system from a non-fixed disk location.

The boot code in windows 2000 is stored in the first sector on the hard disk which is also known as Master Boot Record (MBR). It contains the boot code and the partition table consisting of addresses of the various partitions of the hard disk, from here it selects the address of boot partition where operating system is present and loads it.

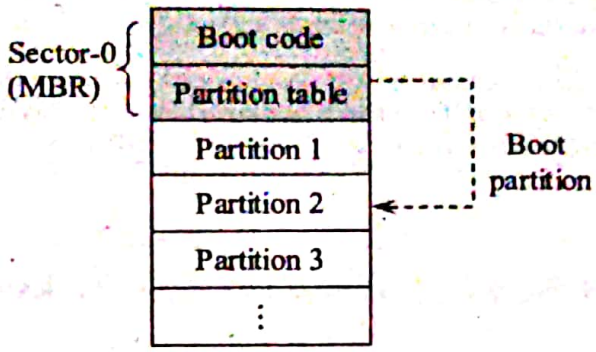


Figure (1): Boot Disk

(iii) Bad Blocks

Some times one or more sectors of the disk becomes defective often if the read-write head comes in contact with the disk platter (also called as head crash), such defective sectors are called as *bad blocks*. Sometimes, the crash is so severe that total disk has to be replaced and sometimes only a few sectors are damaged and they are managed by disk controller.

If disk uses simple IDE controller, then bad blocks have to be managed manually by performing logical formatting. For example, the FORMAT command in MS-DOS automatically scans the disk for bad blocks and if it finds one, it writes a value in FAT table not to use that block. The data that was present in bad block will be lost forever. The checkdisk utility (CHKDSK) also, finds bad blocks and locks them.

In sophisticated disk controllers like SCSI disks, the controller itself carries a list of bad blocks on the disk. This list is initialized by manufacturer during low level formatting and is updated whenever controller finds bad blocks.

Some low level formatting techniques also creates certain spare sectors which are hidden from operating system. When controller finds a bad block, it replaces it with a spare block. This is called as *sector sparing* or *forwarding*. And next time when the controller wants to read or write from that block, it is redirected to the new spare block given to it on behalf of the old damaged block. Such redirections will have adverse effects on the disk scheduling algorithms.

Hence, some controllers uses sector slipping in which all blocks from the bad block are moved or slipped one step down till spare block. For example, consider figure (2), block-5 got damaged, and there are 20 blocks followed by a spare block. B5 is moved to 6, 6 to 7, 7 to 8 and so on till block-20 is moved to spare block.

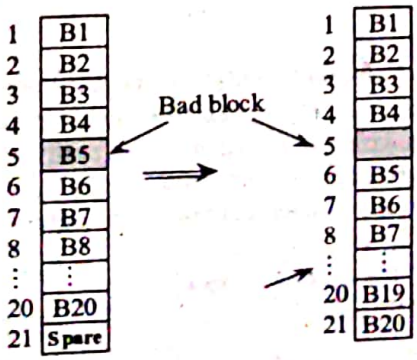


Figure (2): Sector Slipping

5.3.2 RAID Structure

Q41. Explain in detail about RAID structure.

Model Paper-III, Q16(b)

Answer :

RAID Structure

RAID stands for Redundant Array of Inexpensive Disks. The idea here is to use multiple disks and keep redundant data or copies of data so as to improve performance and reliability. The simplest RAID is to copy a whole disk to another disk. Thus, if the original disk fails, its duplicate can be used to restore the data. Hence, reliability is increased.

Here both the original and duplicate disks can be used concurrently to access data. Consider an example, of reading 16-blocks of data from a disk and it requires 16 milliseconds. If first eight blocks are read from one disk and remaining eight from another simultaneously, then the time required will be only 8 milliseconds i.e., half of the traditional approach. Hence, performance is improved.

RAID Levels

There are certain techniques as mirroring and striping that are employed in RAID concept each performing their own functions.

Mirroring is a technique of making a duplicate copy of the complete disk so that in case of disk crash, it can be used. Hence, it increases reliability. Striping is a technique of splitting the bits of each byte across multiple disks. In other words blocks of a file are splitted across multiple disks. This increases performance because all these disks will be active in parallel.

As mirroring provides reliability and striping provides efficiency, several combinations of striping and mirroring are used to get various RAID levels as follows.

RAID Level 0

In this scheme, block striping is done by splitting blocks across several disks without employing any redundancy.

Figure (1): RAID 0 (Non-redundant Stripping)

RAID Level 1

In this scheme disk mirroring is done by creating duplicate copies of disks.

C = Copy

Figure (2): Mirrored Disks

RAID Level 2

It is also called as Memory Style Error Correcting Code (ECC) organization. In this level, a special type of bits called parity bits which are associated with every byte in the memory system are used. A parity bit is used to detect errors in bits contained in a byte, so that ECC can reconstruct the data which is damaged. These parity bits associated for every byte are stripped across other disks that are embedded with the ECC. Hence, a higher level of reliability is obtained with improved performance using striping. The following figure shows for four disks of data, but there is a requirement of only three disks of parity.

P = Parity

Figure (3): Memory Style Error Correcting Code

RAID Level 3

It is also called as bit interleaved parity organization. Unlike memory systems, disk controller could detect the memory sector or bytes that are not read correctly. Then by comparing the parity bits of the damaged sector with other mirrored sectors across other disks, damaged sector is corrected by using a separate parity bit. Hence, it uses only one parity bit for error correction, thereby decreases the overload of disks. This level of RAID uses only one disk for storing parity of four disks.

Figure (4): Bit Interleaved Parity

Another advantage is that, the transfer rate of reading and writing is improved since data is striped across several disks and each operates parallely. A part from these advantage a performance problem of this level and all RAID levels that uses parity bit is that they involve overhead of reading and writing parity.

RAID Level 4

It is also called as block interleaved parity organization. In this scheme block level striping is performed and for each block it stores a parity block in a separate disk.

Figure (5): Block Interleaved Parity

In case of a disk failure, the parity disks come into action failed and the failed blocks are restored from the other disks. The transfer rate of single data block is slow because only a single disk can be accessed by a block. However, the overall transfer rate with respect to read access is higher because it supports multiple read accesses to be processed simultaneously.

One major drawback of this level is that, as there is a single parity disk, the failure of this disk results in the failure of entire system.

RAID Level 5

It is also called as block interleaved distributed parity. It doesn't store parity in a single separate disk but distributes the parity across several disks. If there are *N* disks to store the data, parity is distributed among N+1 disks.

The parity blocks of one disk are stored in some other disk or in other words the parity blocks of a particular disk are not stored in the same disk because if that disk fails, then its parity bits are also lost. Whereas, if its parity bits are present in some other disk, then it could be possible to recover the data.

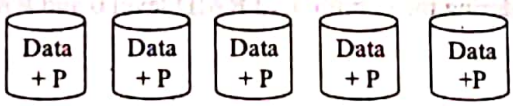


Figure (6): Block Interleaved Distributed Parity

The limitation of this scheme is that if multiple disks fail simultaneously, then it would be impossible to restore the whole data. Hence, next level is used.

RAID Level 6

It is also known as $P + Q$ redundancy scheme, which is very much similar to level 5 except that it stores additional redundant information to overcome multiple disk failures. It stores extra parity bits and sometimes advanced error correcting codes like reed-solomon codes are used. For every 4-bits of data 2-bits of redundant data is stored with which it can tolerate two disk failures. It requires one extra disk than level 5.

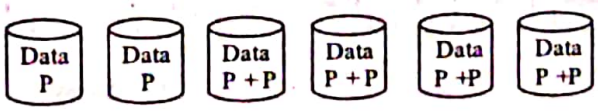


Figure (7): P + Q Redundancy

RAID Level 0 + 1

It is a combination of RAID level 0 and RAID level 1 i.e., it provides both performance and reliability. This level is better than level 5 but the disadvantage is that the number of disks needed is more compared to other levels. First striping is performed than mirroring of those strips are done as shown in figure (8) below.

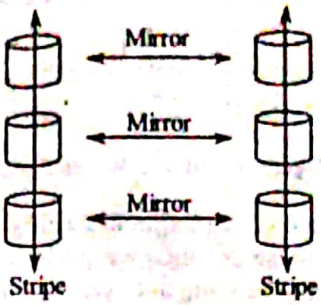


Figure (8): RAID Level 0 +1

Raid Level 1 + 0

It is similar to the above level (i.e., 0 +1) but here the disks are mirrored first then striping is performed on the mirrored pairs.

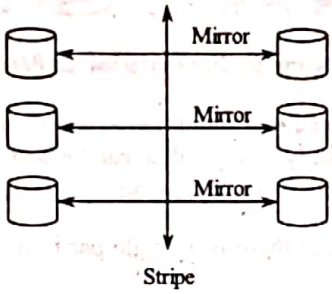


Figure (9): RAID Level 1 + 0

Q42. Could RAID level 1 organization achieve better performance for read requests than a RAID level 0 organization? If so, how? explain.

Answer :

Yes, a RAID level 1 organization can achieve better performance for read requests than a RAID level 0 organization. The reason for this can be explained by considering the definition of RAID level 0 and RAID level 1, as given below,

RAID Level 0

In this scheme, block striping is done by splitting blocks across several disks without employing any redundancy.

Figure (1): RAID 0 (Non-redundant Stripping)

RAID Level 1

In this scheme disk mirroring is done by creating duplicate copies of disks.

C = Copy

Figure (2): Mirrored Disks

From the above two definitions, it can be understood that RAID level 0 does not provide any redundancy while storing data in disks unlike RAID level 1, which provides redundancy.

This concept of redundancy enables RAID level 1, organization to choose between the duplicates copies of required data when a read operation is requested. It select certain copy of data which is closest to the current location of the disk head in order to optimize the performance of system.

Since, there is no redundancy in RAID level 0, this optimization of the system performance cannot be achieved thereby making RAID level 1 organization to achieve better performance for read requests.

IMPORTANT QUESTIONS

UNIT-I

SHORT QUESTIONS

Q1. What is operating system? Discuss briefly the objectives of operating systems. *(Refer Unit-I, Q1)*

Q2. List various types of operating systems. *(Refer Unit-I, Q2)*

Q3. Define system call. *(Refer Unit-I, Q4)*

Q4. What are the types of system calls? *(Refer Unit-I, Q5)*

Q5. List the advantages of virtual machines. *(Refer Unit-I, Q9)*

ESSAY QUESTIONS

Q6. Define OS. Discuss in brief about the various objectives of an operating system. Also discuss its history. *(Refer Unit-I, Q10)*

Q7. What is an operating system? State and explain various types of operating system. And also discuss about operating system services. *(Refer Unit-I, Q13)*

Q8. Explain with a neat example how system calls are used. *(Refer Unit-I, Q15)*

Q9. Discuss various approaches of designing an operating system. *(Refer Unit-I, Q18)*

Q10. Explain briefly about VM ware with respect to its architecture. *(Refer Unit-I, Q21)*

Q11. Explain in detail about java virtual machine. *(Refer Unit-I, Q22)*

UNIT-II

SHORT QUESTIONS

Q1. List the attributes of the process. *(Refer Unit-II, Q1)*

Q2. List the advantages of threads. *(Refer Unit-II, Q3)*

Q3. Describe the differences among short-term, medium-term and long-term scheduling. *(Refer Unit-II, Q5)*

Q4. What is meant by process preemption? Explain with examples. *(Refer Unit-II, Q8)*

Q5. Define Process Control Block. *(Refer Unit-II, Q9)*

ESSAY QUESTIONS

Q6. State and explain the various states that a process can be in. *(Refer Unit-II, Q13)*

Q7. Write short notes on context switching. *(Refer Unit-II, Q16)*

Q8. Define thread. What are the advantages of threads? *(Refer Unit-II, Q17)*

Q9. Identify the different states of a thread. *(Refer Unit-II, Q20)*

Q10. Write about multithreading. *(Refer Unit-II, Q24)*

Q11. Explain preemptive and non-preemptive scheduling algorithms. *(Refer Unit-II, Q28)*

UNIT-III

SHORT QUESTIONS

Q1. Define pipes. *(Refer Unit-III, Q1)*

Q2. What is critical resource? *(Refer Unit-III, Q2)*

Q3. Define semaphore. *(Refer Unit-III, Q3)*

Q4. Define monitor. *(Refer Unit-III, Q4)*

Q5. What is deadlock avoidance? *(Refer Unit-III, Q9)*

ESSAY QUESTIONS

Q6. How are pipes created? Explain the IPC between related processes using ordinary pipes. *(Refer Unit-III, Q12)*

Q7. Explain the bounded buffer problem and how semaphores can be used as a solution to this problem. *(Refer Unit-III, Q20)*

Q8. Explain the solution for the Dining Philosophers problem using semaphore. *(Refer Unit-III, Q22)*

Q9. Discuss in detail about Semaphores. *(Refer Unit-III, Q23)*

Q10. What is a monitor? Compare it with semaphore. Explain in detail a monitor with notify and broadcast using an example. *(Refer Unit-III, Q29)*

Q11. Discuss the methods for handling deadlock. *(Refer Unit-III, Q33)*

Q12. Briefly explain about deadlock prevention methods with examples of each. *(Refer Unit-III, Q34)*

Q13. Explain about deadlock avoidance. *(Refer Unit-III, Q36)*

Q14. State and explain the deadlock recovery methods. *(Refer Unit-III, Q43)*

UNIT-IV

SHORT QUESTIONS

Q1. What is meant by swapping? List various reasons to perform swapping. *(Refer Unit-IV, Q2)*

Q2. What is fixed partitioning? *(Refer Unit-IV, Q4)*

Q3. Discuss in brief about segmentation. *(Refer Unit-IV, Q5)*

Q4. Differentiate between page and frame. *(Refer Unit-IV, Q7)*

Q5. What is the concept behind virtual memory? *(Refer Unit-IV, Q9)*

Q6. What is Thrashing? *(Refer Unit-IV, Q10)*

ESSAY QUESTIONS

Q7. Explain briefly about the contiguous memory allocation. *(Refer Unit-IV, Q13)*

Q8. What is the difference between internal and external fragmentation? *(Refer Unit-IV, Q18)*

Q9. What is paging? Explain the basic method for implementing paging. *(Refer Unit-IV, Q19)*

Q10. Explain various techniques for structuring of the page table. *(Refer Unit-IV, Q23)*

Q11. Explain briefly about virtual memory. *(Refer Unit-IV, Q26)*

Q12. Write short note on translation look-aside buffer. *(Refer Unit-IV, Q29)*

Q13. Write short note on segmentation. *(Refer Unit-IV, Q30)*

Q14. Explain using illustrations typical memory management formats. *(Refer Unit-IV, Q32)*

Q15. What is demand paging and how is it implemented? *(Refer Unit-IV, Q40)*

Q16. What is a system call? Explain how system programs are developed using system calls. *(Refer Unit-IV, Q42)*

Q17. What is thrashing? *(Refer Unit-IV, Q45)*

UNIT-V

SHORT QUESTIONS

Q1. Write short notes on file system. *(Refer Unit-V, Q1)*

Q2. Brief about disk structure. *(Refer Unit-V, Q2)*

Q3. What are disk allocation methods? *(Refer Unit-V, Q3)*

Q4. List any three disk scheduling algorithms. *(Refer Unit-V, Q4)*

Q5. Enlist different types of directory structure. *(Refer Unit-V, Q14)*

ESSAY QUESTIONS

Q6. Explain the different components of I/O hardware. *(Refer Unit-V, Q15)*

Q7. Write about direct memory access (DMA). *(Refer Unit-V, Q16)*

Q8. Explain different directory structures with neat diagram. *(Refer Unit-V, Q23)*

Q9. Explain the three allocation methods in file system implementation. Illustrate with proper diagram. *(Refer Unit-V, Q27)*

Q10. Explain briefly about disk structure. *(Refer Unit-V, Q31)*

Q11. Explain various disk scheduling algorithms with an example. *(Refer Unit-V, Q32)*

Q12. Explain in detail about RAID structure. *(Refer Unit-V, Q41)*

FACULTY OF ENGINEERING

B.E. V-Semester (AICTE) (Main) Examination

MODEL PAPER 1

OPERATING SYSTEMS

(Computer Science and Engineering)

Time: 3 Hours | Max. Marks: 70

Answer All Question from PART-A

Each Question carries equal marks

Answer any five Questions from PART-B

PART-A (10 × 2 = 20 Marks)

1. What is operating system? Discuss briefly the objectives of operating systems. (Unit-I / Q1)
2. List the advantages of virtual machines. (Unit-I / Q9)
3. List the attributes of the process. (Unit-II / Q1)
4. Describe the differences among short-term, medium-term and long-term scheduling. (Unit-II / Q5)
5. Define pipes. (Unit-III / Q1)
6. Compare semaphore and monitor. (Unit-III / Q5)
7. Give the relative advantages and disadvantages of contiguous memory allocation. (Unit-IV / Q3)
8. Discuss in brief about segmentation. (Unit-IV / Q5)
9. What are disk allocation methods? (Unit-V / Q3)
10. What are the operation that can be performed on directory? (Unit-V / Q12)

PART-B (50 Marks)

11. (a) State and explain the various types of system calls in detail. (Unit-I / Q17)

    (b) State and explain operating system services that provide functions that are helpful to the user. (Unit-I / Q14)

12. (a) State and explain the various states that a process can be in. (Unit-II / Q13)

    (b) Explain in detail about scheduling in a multiprocessor system. (Unit-II / Q31)

13. (a) Explain the bounded buffer problem and how semaphores can be used as a solution to this problem. (Unit-III / Q20)

    (b) Discuss in detail about Semaphores. (Unit-III / Q23)

14. (a) Write short note on translation look-aside buffer. (Unit-IV / Q29)

    (b) Explain using illustrations typical memory management formats. (Unit-IV / Q32)

15. (a) Explain briefly about the contiguous memory allocation. (Unit-IV / Q13)

(b) Write short note on segmentation. (Unit-IV / Q30)

16. (a) Explain the different components of I/O hardware. (Unit-V / Q15)

(b) What is a file? Discuss its attributes (Unit-V / Q18)

17. (a) Explain briefly about disk structure. (Unit-V / Q31)

(b) Explain various disk performance parameters. (Unit-V / Q33)

FACULTY OF ENGINEERING

B.E. V-Semester (AICTE) (Main) Examination

MODEL PAPER 2

OPERATING SYSTEMS

(Computer Science and Engineering)

Time: 3 Hours | Max. Marks: 70

Answer All Question from PART-A

Each Question carries equal marks

Answer any five Questions from PART-B

PART-A (10 × 2 = 20 Marks)

1. Write the services of operating system. (Unit-I / Q3)
2. Write a brief note on privileged instructions and the concept of virtual machine. (Unit-I / Q8)
3. List the advantages of threads. (Unit-II / Q3)
4. What is meant by process preemption? Explain with examples. (Unit-II / Q8)
5. Define semaphore. (Unit-III / Q3)
6. What is deadlock avoidance? (Unit-III / Q9)
7. What is meant by swapping? List various reasons to perform swapping. (Unit-IV / Q2)
8. Define paging. (Unit-IV / Q6)
9. List any three disk scheduling algorithms. (Unit-V / Q4)
10. Differentiate between file and directory. (Unit-V / Q13)

PART-B (50 Marks)

11. (a) What is an operating system? State and explain various types of operating system. And also discuss about operating system services. (Unit-I / Q13)

    (b) Explain with a neat example how system calls are used. (Unit-I / Q15)

12. (a) Write about process control block. (Unit-II / Q14)

    (b) Write about multithreading. (Unit-II / Q24)

13. (a) How are pipes created? Explain the IPC between related processes using ordinary pipes. (Unit-III / Q12)

    (b) What is a monitor? Compare it with semaphore. Explain in detail a monitor with notify and broadcast using an example. (Unit-III / Q29)

14. Discuss the methods for handling deadlock. (Unit-III / Q33)

15. (a) Explain briefly about virtual memory. (Unit-IV / Q26)

(b) What is the difference between simple paging and virtual memory paging? (Unit-IV / Q31)

16. (a) What is thrashing? (Unit-IV / Q45)

(b) What is demand paging and how is it implemented? (Unit-IV / Q40)

17. (a) What are the various ways of accessing a file? (Unit-V / Q19)

(b) Explain different directory structures with neat diagram. (Unit-V / Q23)

FACULTY OF ENGINEERING

B.E. V-Semester (AICTE) (Main) Examination

MODEL PAPER 3

OPERATING SYSTEMS

(Computer Science and Engineering)

Time: 3 Hours

Max. Marks: 70

Answer All Question from PART-A

Each Question carries equal marks

Answer any five Questions from PART-B

PART-A (10 × 2 = 20 Marks)

1. What are the features of system call? (Unit-I / Q6)
2. Write about design goals of operating system. (Unit-I / Q7)
3. Explain the following transitions,
   (a) Blocked → Blocked/Suspended
   (b) Blocked/Suspended → Ready/Suspended
   (c) Ready/Suspended → Ready. (Unit-II / Q2)
4. Define Process Control Block. (Unit-II / Q9)
5. Define monitor. (Unit-III / Q4)
6. What is critical resource? (Unit-III / Q2)
7. What is fixed partitioning? (Unit-IV / Q4)
8. What is Thrashing? (Unit-IV / Q10)
9. Define seek time and latency time in disk. (Unit-V / Q5)
10. Enlist different types of directory structure. (Unit-V / Q1

PART-B (50 Marks)

11. (a) Define OS. Discuss in brief about the various objectives of an operating system. Also discuss its history. (Unit-I / Q10)
    (b) Discuss various approaches of designing an operating system. (Unit-I / Q18)
12. (a) Define thread. What are the advantages of threads? (Unit-II / Q17)
    (b) Explain preemptive and non-preemptive scheduling algorithms. (Unit-II / Q28)
13. (a) Explain the solution for the Dining Philosophers problem using semaphore. (Unit-III / Q22)
    (b) Define deadlock. What are the four conditions that create deadlock. Explain with an example. (Unit-III / Q32)

14. (a) What is paging? Explain the basic method for implementing paging. [illegible]

(b) What is the difference between internal and external fragmentation? [illegible]

15. (a) Explain various techniques for structuring of the page table. [illegible]

(b) What is a system call? Explain how system programs are developed using system calls. [illegible]

16. (a) Write about layered file system structure. [illegible]

(b) Explain in detail about RAID structure. [illegible]

17. Explain various disk scheduling algorithms with an example. [illegible]