MAILAM Engineering College

Approved by AICTE, New Delhi, affiliated to Anna University, Chennai, Accredited by NBA & TCS

UNIT I INTRODUCTION

Computer System - Elements and organization; Operating System Overview - Objectives and Functions - Evolution of Operating System; Operating System Structures – Operating System Services - User Operating System Interface - System Calls – System Programs - Design and Implementation - Structuring methods.

PART A

1. **What is an Operating System?**
   - An operating system is a program that manages the computer hardware. It also provides a basis for application programs and act as an intermediary between a user of a computer and the computer hardware.
   - It controls and coordinates the use of the hardware among the various application programs for the various users.

2. **Can traps be generated intentionally by a user program? If so, for what purpose?**
   - A trap is a software-generated interrupt. An interrupt can be used to signal the completion of an I/O to obviate the need for device polling.
   - A trap can be used to call operating system routines or to catch arithmetic errors.

3. **Mention the objectives of an operating system. (or) List any two objectives of an operating system. (April/May 2023)**
   - Convenience – makes computer user friendly.
   - Efficiency - allows computer to use resources efficiently.

- Ability to evolve - constructed in a way to permit effective development, testing and introduction of new functions without interfering with service.

4. What is SYSGEN and system boot?

- The SysGen process runs as a series of jobs under the control of the operating system.
- The process of bringing up the operating system is called booting.

5. Consider a memory system with a cache access time of 10ns and a memory access time of 110ns assume memory access time includes the time to check the cache. If the effective access time is 10% greater than the cache access time. What is the hit ratio H?

Cache access time Tc = 100 ns

Memory access time Tm = 500 ns

If the effective access time is 10% greater than the cache access time, what is the hit ratio H?

Effective access time =cache hit ratio*cache access time+ cache miss ratio *(cache access time +main memory access time)

Effective access time =10% greater the cache access time ==>110

let cache hit ratio is h

110 = h*100ns +(1-h) (100+500)

110= 100h+600-600h

500h= 490

h= 490/500= .98 = 98%

6. How does an interrupt differ from a trap?

- An interrupt is a hardware-generated change-of-flow within the system. An interrupt handler is summoned to deal with the cause of the interrupt; control is then returned to the interrupted context and instruction.
- A trap is a software-generated interrupt. An interrupt can be used to signal the completion of an I/O to obviate the need for device polling. A

trap can be used to call operating system routines or to catch arithmetic errors.

7. What are the advantages and disadvantages of multiprocessor systems?

Advantages of multiprocessor systems

- Increased Throughput
- Economy of Scale
- Increased Reliability

Disadvantage of multiprocessor systems

- If one processor fails then it will affect in the speed.

8. What are the advantages of peer-peer system and client server system?

- Peer-Peer is easy to install
- All the resources and contents are shared by all the peers without server help, where Server shares all the contents and resources.
- Peer-Peer is more reliable as central dependency is eliminated. Failure of one peer doesn't affect the functioning of other peers. In case of Client –Server network, if server goes down, whole network gets affected.

9. What is the purpose of system programs?

- System programs can be thought of as bundles of useful system calls. They provide basic functionality to users so that users do not need to write their own programs to solve common problems.
- The system program serves as a part of the operating system.
- It lies between the user interface and the system calls.

10. Do timesharing differ from Multiprogramming? If so, How?

Time Sharing: Here, **OS assigns time slots to each job**. Here, each job is executed according to the allotted time slots.

Job1: 0 to 5

Job2: 5 to 10

Job3: 10 to 15

Multi-Tasking: In this operating system, **jobs are executed in parallel by the operating system.** But we can achieve this multi-tasking through multiple processors (or)multicore CPU only.

CPU1: Job1

CPU2: Job2

CPU3: Job3

11. Why API's need to be used rather than system calls?

- API is easier to migrate your software to different platforms.
- API usually provides more useful functionality than the system call.
- API can support multiple versions of the operating system.

12. Define operating system and list its objectives.

- An operating system is a program that manages a computer's hardware.
- An operating system is a program that controls the execution of application programs and act as an interface between applications and the computer hardware.

Objectives:

- Convenience
- Efficiency
- Ability to evolve

13. Why is the Operating System viewed as a resource allocator & control program?

- A computer system has many resources that may be required to solve a problem.
- The operating system acts as the manager of these resources.
- The OS is viewed as a control program because it manages the execution of user programs to prevent errors and improper use of the computer.

14. Define SMP. List the advantages of SMP.

- SMP (symmetric multiprocessing) is the processing of programs by multiple processors that share a common operating system and memory.
- There are two or more similar processors of comparable capability.
- All processors can perform the same functions.

List the advantages of SMP.

- Performance
- Availability
- Incremental growth
- Scaling

15. Define Multicore computers.

A multicore computer, also known as a chip multiprocessor, combines two or more processors on a single piece of silicon. Each processor consists of all of the components of an independent processor, such as registers, ALU, pipeline hardware and control unit, and an instruction and data caches.

16. What is serial processing and its problems?

- In serial processing, processor does only one process at a time. (Requests, gets it, and fetches it). Therefore this kind of processing is slow;
- While in parallel processing, CPU does more than one process at a time, so it's faster & consumes less time.

Problems in Serial processing:

- Scheduling
- Setuptime

17. Write a note on the working of simple batch systems.

- Users submit jobs to operator
- Operator batches jobs
- Monitor controls sequence of events to process batch

- When one job is finished, control returns to Monitor which reads next job
- Monitor handles scheduling.

18. Define Time sharing systems.

- The processor time is shared among multiple users.
- In a time-sharing system, multiple users simultaneously access the system through terminals, with the OS interleaving the execution of each user program in a quantum time.

19. Difference between the Batch system and time sharing system. (or) List any four differences between batch system and time-sharing system. (Nov/Dec 2024)

Batch system	Time sharing system
• Jobs are grouped into batches and processed sequentially without user interaction.	• Multiple users interact with the system simultaneously, and CPU time is shared among them.
• No direct interaction with the user; jobs are submitted and executed automatically.	• Users can interact with programs in real time.
• High response time since jobs are processed in order.	• Low response time as the system switches between tasks quickly.
• CPU utilization is high but not always optimal due to job dependencies.	• CPU utilization is optimized by rapidly switching between multiple tasks.

20. Define Single Processor system.

- On a single-processor system, there is one main CPU capable of executing a general-purpose instruction set, including instructions from user processes.
- They may come in the form of device-specific processors, such as disk, keyboard, and graphics controllers.

21. What is Multiprocessor Systems?

- Multiprocessor systems (also known as parallelsystems or multicore systems) have two or more processors in close communication, sharing the computer bus and sometimes the clock, memory, and peripheral devices.
- Recently, multiple processors have appeared on mobile devices such as smart phones and tablet computers.

22. What is graceful degradation and fault tolerance?

- The ability to continue providing service proportional to the level of surviving hardware is called graceful degradation.
- Fault tolerance requires mechanism to allow the failure to be detected, diagnosed and if possible, corrected.

23. Distinguish between symmetric and asymmetric multiprocessor systems.

Symmetric multiprocessor systems	**Asymmetric multiprocessor systems**
• Here each processor runs an identical copy of the operating system and these copies communicate with one another as needed.	• Here each processor is assigned a specific task. A master processor looks to slave processors for predefined instruction.
• Symmetric multiprocessor systems means thatall processor are peers and no master slave relationship exists between processor. • Ex: Encore version of Unix for MultimaxComputer	• It defines master slave relationship. • The master processor schedules and allocates work to the slaveprocessors. • Ex:Sun OS version

24. Distinguish between trap and interrupt

25. What is trap?

- A **trap** (or an **exception**) is a software-generated interrupt caused either by an error or by a specific request from a user program that an operating-system service be performed.

26. What is Dual mode operation and what are the two type's modes?

- The dual mode operation provides us with the means for protecting the operating system from wrong users and wrong users from one another.

Transition from user to kernel mode

- Two separate *modes* of operation:
  1. User mode or supervisor mode
  2. kernel mode orsystem mode or privilegedmode.

27. Define System call. (or) What are system calls? Give some examples. (April/May 2024)

System calls provide an interface to the services made available by an operating system.

Examples of system calls

- **Open**: Opens or creates a file or device
- **Read**: Reads data from a file or device
- **Write**: Writes data to a file
- **Close**: Closes an open file or device
- **Fork**: Creates a copy of a process, called a child process

28. What are the various types of system calls?

- Process control
- File manipulation
- Device manipulation
- Information maintenance
- Communications
- Protection.

29. What are the various types of communication models?

- In the message-passing model, the communicating processes exchange messages with one another to transfer information.

- In the shared-memory model, processes use shared memory create () and shared memory attach () system calls to create and gain access to regions of memory owned by other processes.

30. Differentiate tightly coupled systems and loosely coupled systems.

Loosely coupled systems	Tightly coupled systems
• Each processor has its own memory	• Common memory is shared by many processors
• Each processor can communicate with other all through communication lines.	• No need of any special communication lines.

31. Differentiate system calls and system programs.

- The system calls are list of the function that will be call in section between user and kernel.
- But the system program is the program that can do system works like: Change system settings.

32. Distinguish between Multicore and Multiprocessor.

- The main difference between multicore and multiprocessor is that the multicore refers to a single CPU with multiple execution units while the multiprocessor refers to a system that has two or more CPUs.
- Multicores have multiple cores or processing units in a single CPU. A multiprocessor contains multiple CPUs.

33. What is dual mode operation and what is the need of it?

- The dual mode of operation provides us with the means for protecting the OS from errant users-and errant users from one another.
- If an attempt is made to execute a privileged instruction in user mode, the hardware does not execute the instruction but rather treats it as illegal and traps it to the OS.

34. What is the Kernel?

- A more common definition is that the OS is the one program running at all times on the computer, usually called the kernel, with all else being application programs.

35. What is meant by Mainframe Systems?

- Mainframe systems are the first computers developed to tackle many commercial and scientific applications.
- These systems are developed from the batch systems and then multiprogramming system and finally time sharing systems.

36. What is meant by Batch Systems?

- Operators batched together jobs with similar needs and ran through the computer as a group.
- The operators would sort programs into batches with similar requirements and as system become available, it would run each batch.

37. OS is designed in different ways. In what ways modular kernel approach is similar to the layered approach? In what ways they differ? (April/May 2024)

- The modular kernel approach and the layered approach are both architectural design patterns used in operating systems.
- They share similarities in terms of organizing the system into separate modules or layers, each with a specific set of responsibilities.
- Both approaches aim to improve system modularity, maintainability, and flexibility.
- However, they differ in the way they handle dependencies between modules or layers.

38. Write any two usages of fork and exec system calls. (Nov/Dec 2024)

1. Process Creation

- **fork()** is used to create a new child process by duplicating the parent process.
- **exec()** replaces the current process image with a new program, allowing execution of a different program within the same process.

2. Running a New Program

A parent process can use **fork()** to create a child process and then use **exec()** within the child to run a new program, such as launching a shell command or executing a system utility.

PART B

1. Explain in detail about the Computer System Overview.

Key Points

1. Basic elements of a Computer
2. Computer System Organization
3. Storage Structure
4. I/O Structure

1. Basic elements of a computer

- **Processor:** Controls the operation of the computer and performs its data processing function.
- **Main memory:** Stores data and programs. This memory is volatile. Main memory is also called as real or primary memory.
- **I/O modules:** Move data between the computer and its external environment including secondary memory devices (eg: disks), communications equipment's and terminals.
- **System bus:** Provides for communication among processors, main memory, and I/O modules.
- **Program Counter(PC):** It holds the address of the next instruction to be executed.
- **Instruction Register(IR):** It specifies the recently fetched instruction or current instruction.
- **Memory Address Register(MAR):** It specifies the address in memory for the next read or write.
- **Memory Buffer Register(MBR):** It contains the data to be written into memory or which receives the data from memory.
- **I/O Address Register(I/O AR):** It receives a particular I/O device.
- **I/O Buffer Register(I/O BR):** It is used for the exchange of data between an I/O module and the processor.
- **Memory Module** consists of a bit pattern that can be interpreted as either an instruction or data.

- Each location contains a bit pattern that can be interpreted as either an instruction or data.
- I/O module transfers data from external devices to processor and memory and vice versa. It contains internal buffer for temporarily holding data until they can be sent on. Refer figure 1.1.

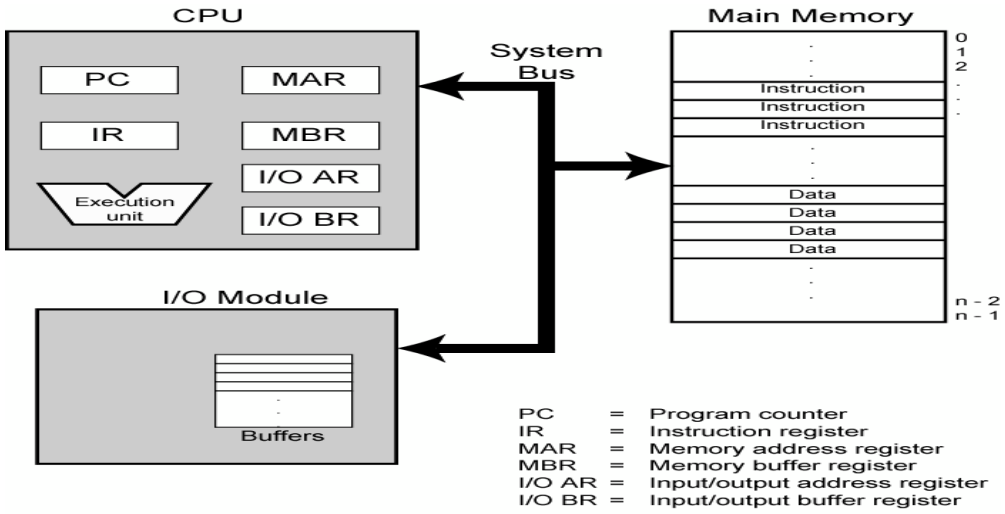


Fig.1.1 Computer Components

2. Computer System Organization

Computer System Operation

- Fig.1.2 refers A modern general-purpose computer system consists of one or more CPUs and a number of device controllers connected through a common bus that provides access to shared memory.
- Each device controller is in charge of a specific type of device (for example, disk drives, audio devices, or video displays).
- The CPU and the device controllers can execute in parallel, competing for memory cycles. To ensure orderly access to the shared memory, a memory controller synchronizes access to the memory.
- For a computer to start running—for instance, when it is powered up or rebooted—it needs to have an initial program to run.
- This initial program, or bootstrap program, tends to be simple. Typically, it is stored in read-only memory (ROM) or electrically erasable programmable read-only memory (EEPROM), known by the general term firmware, within the computer hardware.

- It initializes all aspects of the system, from CPU registers to device controllers to memory contents.
- The occurrence of an event is usually signaled by an interrupt from either the hardware or the software.
- Hardware may trigger an interrupt at any time by sending a signal to the CPU, usually by way of the system bus. Software may trigger an interrupt by executing a special operation called a system call (also called a monitor call).
- When the CPU is interrupted, it stops what it is doing and immediately transfers execution to a fixed location.

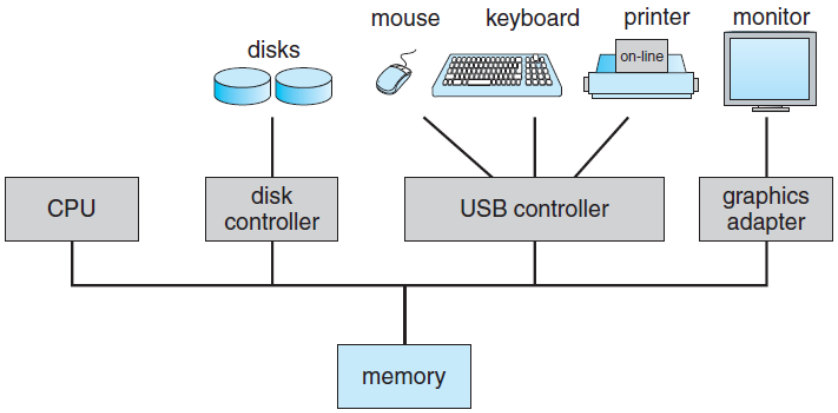


Figure 1.2 A modern computer system.

3. Storage Structure

- Basically we want the programs and data to reside in main memory permanently.
- This arrangement is usually not possible for the following two reasons:
  - Main memory is usually too small to store all needed programs and data permanently.
  - Main memory is a volatile storage device that loses its contents when power is turned off or otherwise lost.

There are two types of storage devices:-

Volatile Storage Device – It loses its contents when the power of the device is removed.

Non-Volatile Storage device – It does not loses its contents when the power is removed. It holds all the data when the power is removed.

- Secondary Storage is used as an extension of main memory. Secondary storage devices can hold the data permanently.
- Storage devices consists of Registers, Cache, Main-Memory, Electronic-Disk, Magnetic-Disk, Optical-Disk, Magnetic-Tapes.
- Each storage system provides the basic system of storing a datum and of holding the datum until it is retrieved at a later time.
- All the storage devices differ in speed, cost, size and volatility.
- The most common Secondary-storage device is a Magnetic-disk, which provides storage for both programs and data.

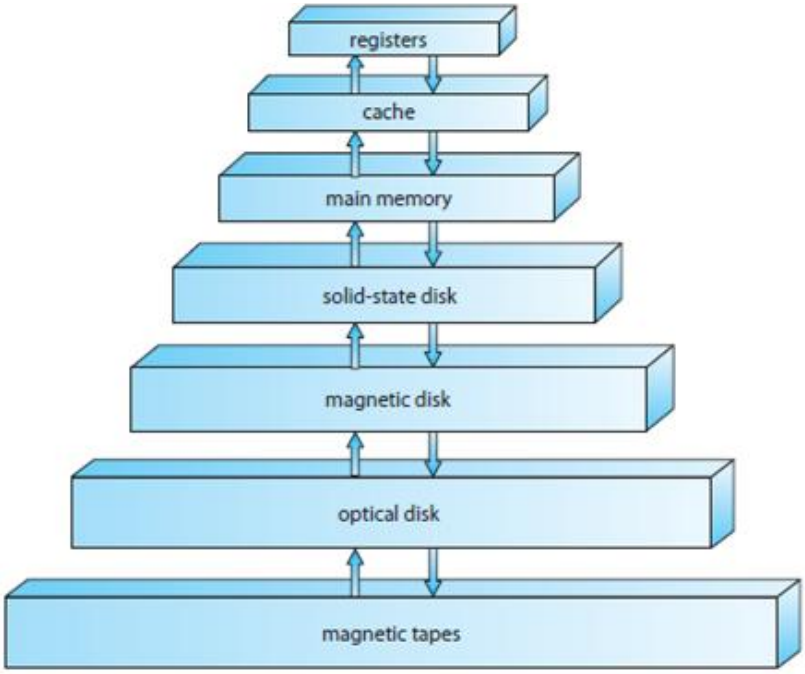


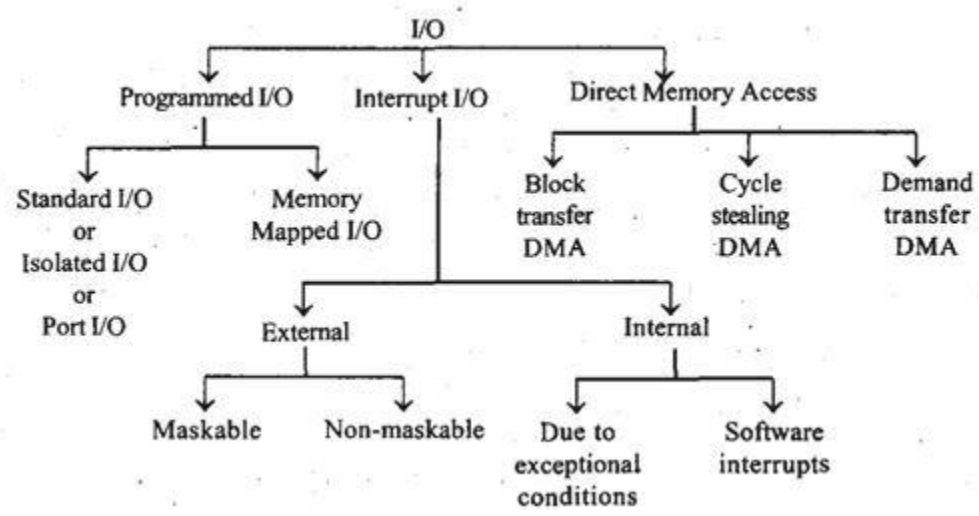
Fig.1.3 Storage Device Hierarchy

- Fig.1.3 shows that all the storage devices are arranged according to speed and cost.
- The higher levels are expensive, but they are fast. As we move down the hierarchy, the cost per bit generally decreases, where as the access time generally increases.
- The storage systems above the Electronic disk are Volatile, where as those below are Non-Volatile.
- An Electronic disk can be either designed to be either Volatile or Non-Volatile.

- During normal operation, the electronic disk stores data in a large DRAM array, which is Volatile.
- But many electronic disk devices contain a hidden magnetic hard disk and a battery for backup power.
- If external power is interrupted, the electronic disk controller copies the data from RAM to the magnetic disk. When external power is restored, the controller copies the data back into the RAM.
- The design of a complete memory system must balance all the factors. It must use only as much expensive memory as necessary while providing as much inexpensive, Non-Volatile memory as possible.
- Caches can be installed to improve performance where a large access-time or transfer-rate disparity exists between two components.

4. I/O Structure

- I/O Structure consists of Programmed I/O, Interrupt driven I/O, DMS, CPU, Memory, External devices, these are all connected with the help of Peripheral I/O Buses and General I/O buses.
- **I/O Types**



Programmed I/O

- In the programmed I/O when we write the input then the device should be ready to take the data otherwise the program should wait for some time so that the device or buffer will be free then it can take the input.

- Once the input is taken then it will be checked whether the output device or output buffer is free then it will be printed. This process is continued every time in transferring of the data.

I/O Interrupts

- To initiate any I / O operation, the CPU first loads the registers to the device controller. Then the device controller checks the contents of the registers to determine what operation to perform.
  - There are two possibilities if I / O operations want to be executed. These are as follows
    - **Synchronous I / O** – The control is returned to the user process after the I/O process is completed.
    - **Asynchronous I/O** – The control is returned to the user process without waiting for the I/O process to finish. Here, I/O process and the user process run simultaneously.

DMA Structure

- Direct Memory Access (DMA) is a method of handling I / O. Here the device controller directly communicates with memory without CPU involvement.
- After setting the resources of I/O devices like buffers, pointers, and counters, the device controller transfers blocks of data directly to storage without CPU intervention.
- DMA is generally used for high speed I / O devices.

2. Explain in detail about operating system objectives and functions.

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

Objectives

- **Convenience** – makes a computer more convenient to use.
- **Efficiency** - allows computer to use resources efficiently.
- **Ability to evolve** - constructed in a way to permit effective development, testing and introduction of new functions without interfering with service.

Functions

a) Operating System as a User/Computer Interface

b) Operating System as Resource Manager

c) Ease of Evolution of an Operating System

a) Operating System as a User/Computer Interface

- The hardware and software in a computer system can be viewed in a layered fashion as shown in the figure 1.4.

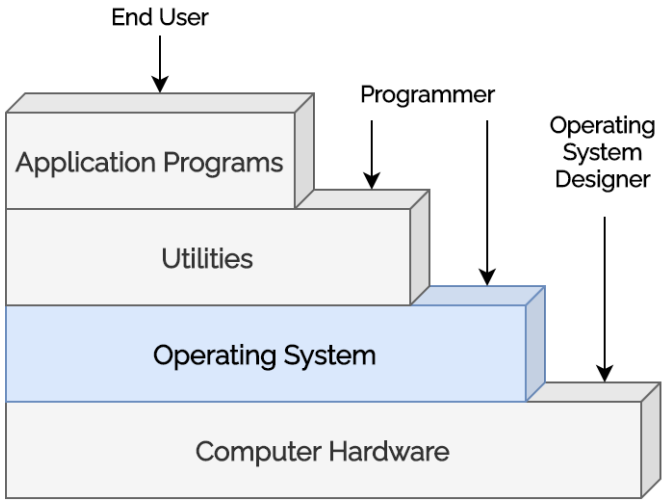


Fig.1.4 Computer Software and Hardware Structure

- The hardware and software used in providing applications to a user can be viewed in a layered or hierarchical fashion.
- The user of those applications, the end user, generally is not concerned with the details of computer hardware.
- Thus the end-user views the computer system as a set of applications.
- Application users are not concerned with the details of computer hardware.
- The operating system masks the details of the hardware from the programmer and acts as a mediator, making it easier to access the services.

a) Operating System as Resource Manager

- A computer is a set of resources for the movement, storage, and processing of data and for the control of these functions.
- The OS is responsible for managing these resources.
- The OS functions in the same way as ordinary computer software. That is a program or suite of programs executed by the processor.

- The OS frequently relinguishes control and must depend on the processor to allow it to regain control.
- The OS directs the processor in the use of the other system resources and in the timing of its execution of other programs.
- A portion of the OS is in main memory. This includes the kernel, or nucleus which contains.
- The most frequently used functions in the OS and, at a given time, other portions of the OS currently in use.
- The remainder of main memory contains user programs and data.
- The memory management hardware in the processor and the OS jointly control the allocation of main memory. Refer figure 1.5.

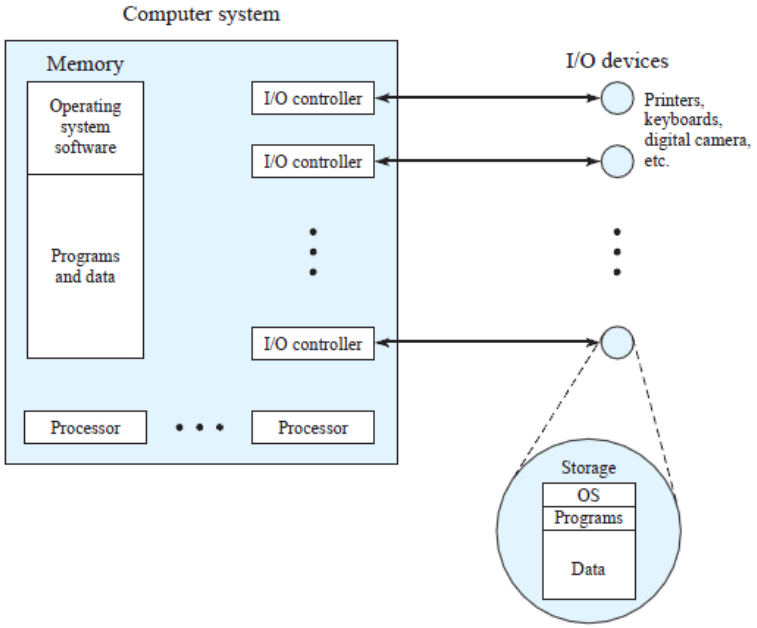


Fig.1.5 Operating System as a Resource Manager

3. Explain in detail about the evolution of the computer system.(or) Explain the evolution of operating system in detail. (April/May 2024) (or) Explain how the operating system has evolved from serial processing to multiprocessing system. (NOV/DEC 2024)

Key Points

1. Serial Processing
2. Simple batch systems
3. Multiprogrammed Batch Systems
4. Time-Sharing Systems
5. Mutiprocessor System
6. Distributed Systems
7. Client Server System
8. Clustered Systems
9. Real Time Systems
10. Hand Held Systems

1. Serial Processing:

- In serial processing, processor does only one process at a time (Requests, gets it, and fetches it).
- Therefore this kind of processing is slow; while in parallel processing, CPU does more than one process at a time, so it's faster & consumes less time.
- These computers were run from a console consisting of display lights, toggle switches, some form of input device, and a printer.
- If an error halted the program, the error condition was indicated by the lights.
- If the program proceeded to a normal completion, the output appeared on the printer.
- These early systems presented two main problems
  - Scheduling
  - Setup time

2. Simple batch systems:

- Early computers were expensive, and therefore it was important to maximize processor utilization.
- The wasted time due to scheduling and setup time was unacceptable.
- To improve utilization, the concept of a batch OS was accepted.
- The central idea behind the concept of a batch OS was developed.
- **Monitor point of view:** The monitor controls the sequence of events. For this to be so, much of the monitor must always be in main memory and available for execution. That portion is referred to as the **resident monitor**.
- **Processor point of view:** At a certain point, the processor is executing instructions from the portion of main memory, containing the monitor. These instructions cause the next job to be read into another portion of main memory.
- The user submits the job cards or tape to a computer operator, who batches the jobs together sequentially and places the entire batch on an input device, for use by the monitor.

3. Multiprogrammed Batch Systems:

- When one job needs to wait for I/O, the processor can switch to the other job, which is likely not waiting for I/O.
- Expand memory to hold three, four, or more programs and switch among all of them. The approach is known as multiprogramming, or multitasking. Refer figure 1.6.

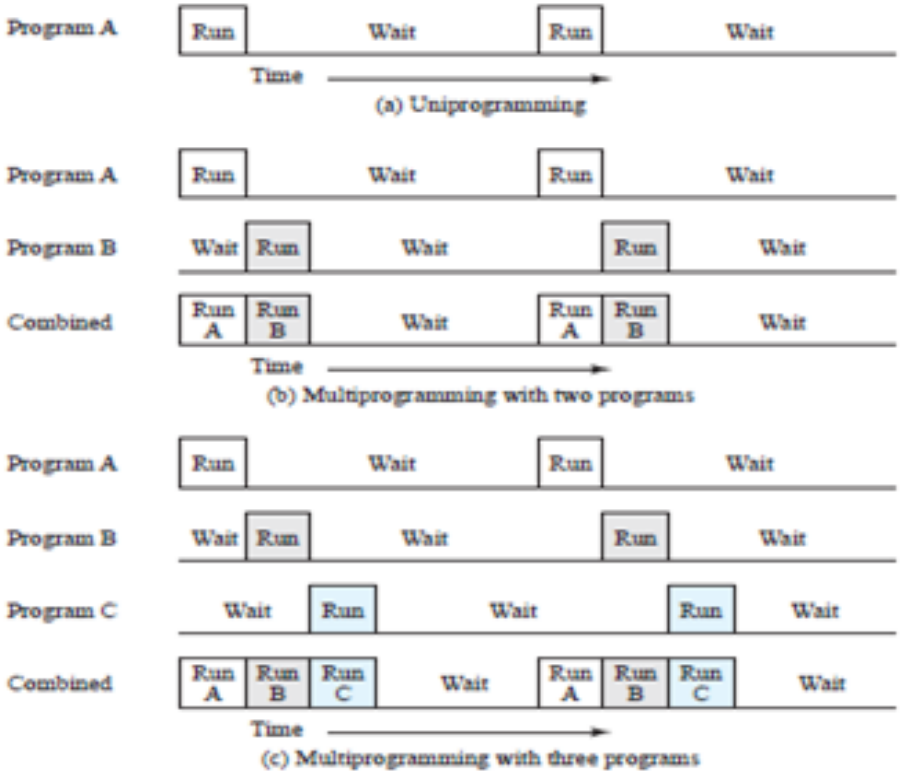


Fig.1.6 Multiprogramming Examples

4. Time-Sharing Systems:

- With the use of multiprogramming, batch processing can be quite efficient.
- Time sharing or multitasking is a logical extension of multiprogramming.
- The CPU executes multiple jobs by switching among them but switches occur so frequently that the user can interact with each program while it is running.
- However, for some jobs, such as transaction processing, an interactive mode is essential.
- A time shared OS allows many user to share computer simultaneously.
- Processor time is shared among multiple users.
- In a time-sharing system, multiple users simultaneously access the system through terminals.
- Both batch processing and time sharing use multiprogramming.

5. Multiprocessor Systems

- Multiprocessor system also known as parallel system or tightly coupled system have more than one processor in close communication sharing the computer bus, the clock and sometimes memory and peripheral devices.

Advantages of Multiprocessor systems:

- **Increased throughput**. By increasing the number of processors, we expect to get more work done in less time.
- **Economy of scale**. Multiprocessor systems can cost less than equivalent multiple single-processor systems, because they can share peripherals, mass storage, and power supplies.
- **Increased reliability**. If functions can be distributed properly among several processors, then the failure of one processor will not halt the system, only slow it down.

Types of Multiprocessor Systems:

Asymmetric multiprocessing:

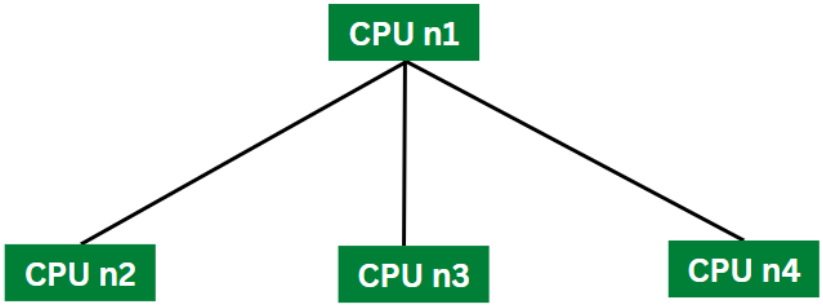


Figure 1.7 Asymmetrical Multiprocessing Operating System

- In which each processor is assigned a specific task.
- A master processor controls the system; the other processors either look to the slave for instruction or have predefined tasks.
- This scheme defines a master-slave relationship.
- The boss processor schedules and allocates work to the worker. Refer figure 1.7.

I. Advantages

- Asymmetrical multiprocessing operating system are cost-effective.
- They are easy to design and manage.

- They are more scalable.

II. Disadvantages

- There can be uneven distribution of workload among the processors.
- The processors do not share same memory.
- Entire system goes down if one process fails.

Symmetric multiprocessing:

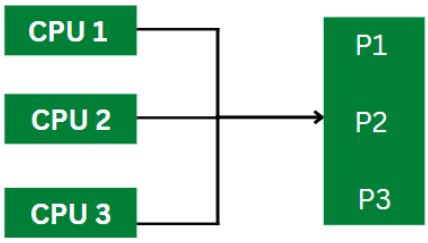


Figure 1.8 Symmetrical Multiprocessing Operating System

- In which each processor performs all tasks within the operating system.
- SMP means that all processors are peers; no master-slave relationship exists between processors. Refer figure 1.8.

III. Advantages

- Failure of one processor does not affect the functioning of other processors.
- It divides all the workload equally to the available processors.
- Makes use of available resources efficiently.

IV. Disadvantages

- Symmetrical multiprocessing OS are more complex.
- They are more costlier.
- Synchronization between multiple processors is difficult.

6. Distributed Operating Systems

- Each processor has its own local memory and clock
- The processor communicate with one another through various communication links such as high speed buses or telephone lines.
- These systems are referred to as loosely coupled systems.

- **Advantages**
  - Resource sharing
  - Computation speed up
  - Reliability

7. Client-Server Systems

- A computing system composed of two logical parts, a server which provides information or services and a client, which requests them is called client server system.
- The general structure of a client-server system is depicted in the figure below:
- Server Systems can be broadly categorized as compute servers and file servers.
  - **Compute-server systems** provide an interface to which clients can send requests to perform an action, in response to which they execute the action and send back results to the client.
  - **File-server systems** provide a file-system interface where clients can create, update, read, and delete files.

8. Clustered Systems

- Clustered systems gather together multiple CPUs to accomplish computational work.
- Clustered systems differ from parallel systems, however, in that they are composed of two or more individual systems coupled together.
- There are two types of clustering
- **Asymmetric Clustering**
  - In this, one machine is in hot standby mode while the other is running the applications.
  - The hot standby host (machine) does nothing but monitor the active server.
  - If that server fails, the hot standby host becomes the active server.
- **Symmetric Clustering**
  - In this, two or more hosts are running applications, and they are monitoring each other.
  - This mode is obviously more efficient, as it uses all of the available hardware.

9. Real-Time Operating System

- A real time system has well-defined fixed time constraints.
- Processing must be done within the defined constraints or the system will fail.
- Real time systems are of two types
  - Hard Real Time Systems
  - Soft Real Time Systems
- The Real-Time Operating system which guarantees the maximum time for critical operations and complete them on time are referred to as Hard Real-Time Operating Systems.
- While the real-time operating systems that can only guarantee a maximum of the time, i.e. the critical task will get priority over other tasks, but no assurity of completing it in a defined time. These systems are referred to as Soft Real-Time Operating Systems.

10. Handheld Systems

- Handheld systems include **Personal Digital Assistants**(PDAs), such as Palm-Pilots or Cellular Telephones with connectivity to a network such as the Internet.
- They are usually of **limited size** due to which most handheld devices have a small amount of memory, include slow processors, and feature small display screens.
- Many handheld devices have between 512 KB and 8 MB of memory. As a result, the operating system and applications must manage memory efficiently. This includes returning all allocated memory back to the memory manager once the memory is no longer being used.
- Currently, many handheld devices do not use virtual memory techniques, thus forcing program developers to work within the confines of limited physical memory.
- Processors for most handheld devices often run at a fraction of the speed of a processor in a PC. Faster processors require more power. To include a faster processor in a handheld device would require a larger battery that would have to be replaced more frequently.

- The last issue confronting program designers for handheld devices is the small display screens typically available.
- One approach for displaying the content in web pages is web clipping, where only a small subset of a web page is delivered and displayed on the handheld device.
- Some handheld devices may use wireless technology such as BlueTooth, allowing remote access to e-mail and web browsing.
- Cellular telephones with connectivity to the Internet fall into this category.
- Their use continues to expand as network connections become more available and other options such as cameras and MP3 players, expand their utility.

4. Explain in detail about operating system structure. (or) Explain the operating system structure in detail. What is the main advantage of the layered approach to system design? What are the disadvantages of the layered approach? (April/May 2024)

System Structure

- An operating system is a construct that allows the user application programs to interact with the system hardware.
- Since the operating system is such a complex structure, it should be created with utmost care so it can be used and modified easily.
- An easy way to do this is to create the operating system in parts.
- Each of these parts should be well defined with clear inputs, outputs and functions.

It is divided into 3 categories

1. Simple structure
2. Layered approach
3. Microkernal approach

1. Simple Structure

- Many commercial systems do not have a well-defined structure. Frequently, such operating systems started as small, simple, and limited systems, and then grew beyond their original scope.
    - MS-DOS structure
    - Unix structure

Example 1: MS-DOS structure

- It provides the most functionality in the least space
- In MS-DOS the interfaces and levels of functionality are not well separated.
- For instance, application programs are able to access the basic I/O routines to write as shown in figure 1.9.

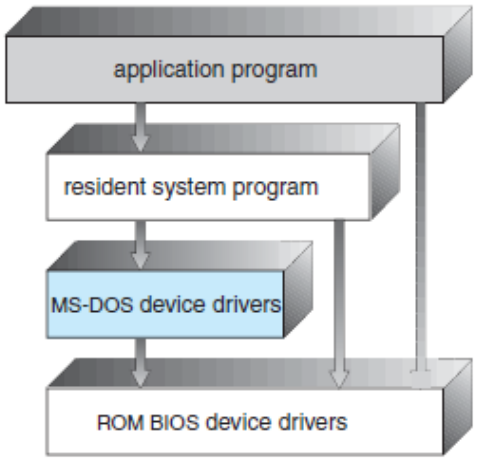


Fig.1.9. MS-DOS Layer Structure

Example 2: UNIX:

- UNIX is another system that was initially limited by hardware functionality.
- It consists of two separable parts:
  - Kernel program
  - System programs
- The kernel is further separated into a series of interfaces and device drivers, which were added and expanded over the years as UNIX evolved.
- The kernel provides the file system, CPU scheduling, memory management, and other operating system functions through system calls as shown in figure 1.10.

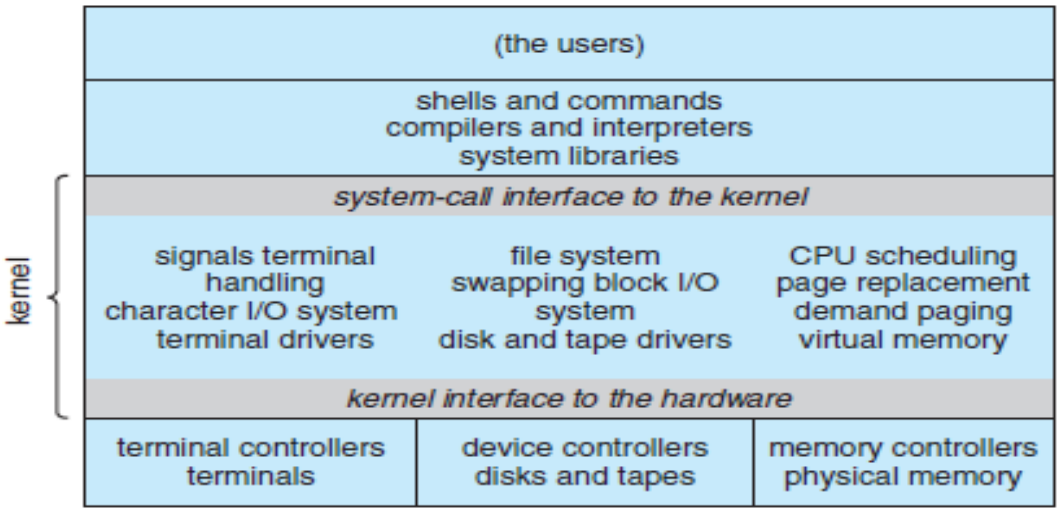


Fig.1.10. Unix System Structure

2. Layered approach

- In which the operating system is broken into a number of layers(levels). The bottom layer (layer 0) is the hardware; the highest (layer *N*) is the user interface.
- A typical operating-system layer — say, layer *M* — consists of data structures and a set of routines that can be invoked by higher-level layers.
- Layer *M,* in turn, can invoke operations on lower-level layers.
- Given proper hardware support, OS may be broken into smaller, more appropriate pieces.
- The modularization of a system can be done in may ways.
- One method is the layered approach, in which the operating system is broken up into a number of layers (or levels), each built on top of lower layers.
- The bottom layer (layer 0) is the hardware: the highest (layer N) is the user interface.

Advantage:

- The main advantage of the layered approach is modularity
- Modularity makes the debugging and system verification easy

Disadvantage:

- The major difficulty with the layered approach involves appropriately defining the various layers. Because a layer can use only lower-level layers, careful planning is necessary.
- A problem with layered implementations is that they tend to be less efficient than other types. Refer figure 1.11.

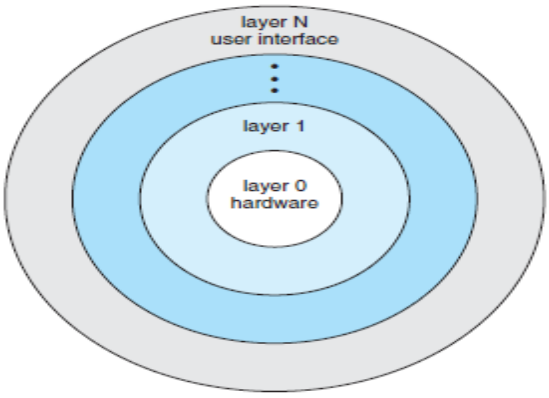


Fig.1.11 Layered Structure

- Layer 0 -Hardware
- Layer 1 - CPU Scheduling
- Layer 2 - Memory Management
- Layer 3 - Process Management
- Layer 4 - buffering for input and output
- Layer 5 - user programs

3. Microkernel approach

- Remove the nonessential components from the kernel into the user space.
- Moves as much from the kernel into user space
- Communication takes place between user modules using message passing as shown in figure 1.12.
- **Benefits**
  - Easier to extend a microkernel
  - Easier to port the operating system to new architectures
  - More reliable (less code is running in kernel mode)
  - More secure

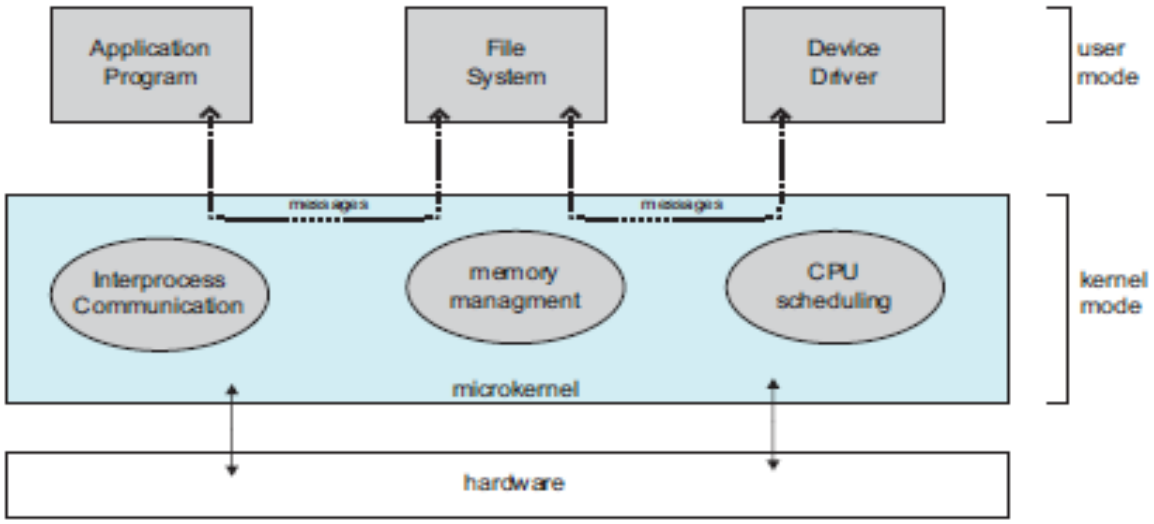


Fig.1.12 Microkernel

Disadvantage:

- The performance of microkernel can suffer due to increased system-function overhead.

5. Discuss about various services provided by operating system. (or) Discuss in detail about operating system services. (NOV/DEC 2024)

Services of OS:

User interface

- Almost all operating systems have a user interface (UI). This interface can take several forms.
- One is a command-line interface (CLI), which uses text commands and a method for entering them (say, a keyboard for typing in commands in a specific format with specific options).
- Another is a batch interface, in which commands and directives to control those commands are entered into files, and those files are executed. Most commonly, a graphical user interface (GUI) is used.
- Here, the interface is a window system with a pointing device to direct I/O, choose from menus, and make selections and a keyboard to enter text. Some systems provide two or all three of these variations.

Program execution:

- The system must be able to load a program into memory and to run that program.
- The program must be able to end its execution, either normally or abnormally (indicating error).

I/O Operations:

- A running program may require I/O, which may involve a file or an I/O device.
- For specific devices, special functions may be desired (such as recording to a CD or DVD drive or blanking a display screen).
- For efficiency and protection, users usually cannot control I/O devices directly. Therefore, the operating system must provide a means to do I/O.

File-system manipulation

- The file system is of particular interest. Obviously, programs need to read and write files and directories.
- They also need to create and delete them by name, search for a given file, and list file information.
- Finally, some operating systems include permissions management to allow or deny access to files or directories based on file ownership.
- Many operating systems provide a variety of file systems, sometimes to allow personal choice and sometimes to provide specific features or performance characteristics.

Communications

- There are many circumstances in which one process needs to exchange information with another process.
- Such communication may occur between processes that are executing on the same computer or between processes that are executing on different computer systems tied together by a computer network.
- Communications may be implemented via shared memory, in which two or more processes read and write to a shared section of memory, or message passing, in which packets of information in predefined formats are moved between processes by the operating system.

Error detection

- The operating system needs to be detecting and correcting errors constantly.
- Errors may occur in the CPU and memory hardware (such as a memory error or a power failure), in I/O devices (such as a parity error on disk, a connection failure on a network, or lack of paper in the printer), and in the user program (such as an arithmetic overflow, an attempt to access an illegal memory location, or a too-great use of CPU time).
- For each type of error, the operating system should take the appropriate action to ensure correct and consistent computing.
- Sometimes, it has no choice but to halt the system. At other times, it might terminate an error-causing process or return an error code to a process for the process to detect and possibly correct.

Resource allocation

- When there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them.
- The operating system manages many different types of resources. Some (such as CPU cycles, main memory, and file storage) may have special allocation code, whereas others (such as I/O devices) may have much more general request and release code.
- For instance, in determining how best to use the CPU, operating systems have CPU-scheduling routines that take into account the speed of the CPU, the jobs that must be executed, the number of registers available, and other factors.
- There may also be routines to allocate printers, USB storage drives, and other peripheral devices.

Accounting

- We want to keep track of which users use how much and what kinds of computer resources.
- This record keeping may be used for accounting (so that users can be billed) or simply for accumulating usage statistics.
- Usage statistics may be a valuable tool for researchers who wish to reconfigure the system to improve computing services.

Protection and security

- The owners of information stored in a multiuser or networked computer system may want to control use of that information.
- When several separate processes execute concurrently, it should not be possible for one process to interfere with the others or with the operating system itself. Refer figure 1.13.
- Protection involves ensuring that all access to system resources is controlled. Security of the system from outsiders is also important.
- Such security starts with requiring each user to authenticate himself or herself to the system, usually by means of a password, to gain access to system resources.

- If a system is to be protected and secure, precautions must be instituted throughout it. A chain is only as strong as its weakest link.

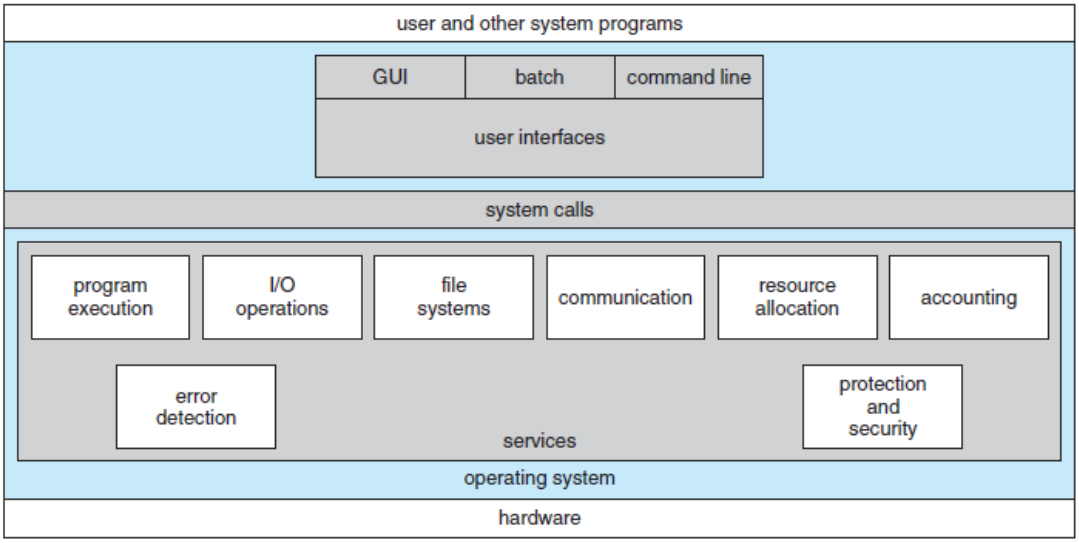


Fig.1.13 A view of Operating System services

4. Explain in detail about User and Operating System Interface.

- One provides a command line interface, or command interpreter, that allows users to directly enter commands to be performed by the operating system.
- The other allows users to interface with the operating system via a graphical user interface or GUI

Command Interpreters

1. Some operating systems include the command interpreter in the kernel.
2. Others, such as Windows and UNIX, treat the command interpreter as a special program that is running when a job is initiated or when a user first logs on (on interactive systems).
3. On systems with multiple command interpreters to choose from, the interpreters are known as shells.
4. The main function of the command interpreter is to get and execute the next user-specified command. Many of the commands given at this level manipulate files: create, delete, list, print, copy, execute, and so on.

The MS-DOS and UNIX shells operate in this way. These commands can be implemented in two general ways.

1. In one approach, the command interpreter itself contains the code to execute the command.
    - For example, a command to delete a file may cause the command interpreter to jump to a section of its code that sets up the parameters and makes the appropriate system call.
2. An alternative approach—used by UNIX, among other operating systems—implements most commands through system programs.
    - In this case, the command interpreter does not understand the command in any way; it merely uses the command to identify a file to be loaded into memory and executed.

Graphical User Interface

- A second strategy for interfacing with the operating system is through a user friendly graphical user interface, or GUI.
- Here, rather than entering commands directly via a command-line interface, users employ a mouse-based window and- menu system characterized by a desktop metaphor.
- The user moves the mouse to position its pointer on images, or **icons**, on the screen (the desktop) that represent programs, files, directories, and system functions.
- Depending on the mouse pointer's location, clicking a button on the mouse can invoke a program, select a file or directory—known as a **folder**—or pull down a menu that contains commands.

5. Explain the various types of system calls with an example for each.

Definition

- System calls provide an interface to the services made available by an operating system.
- These calls are generally available as assembly language instructions. Refer figure 1.14.

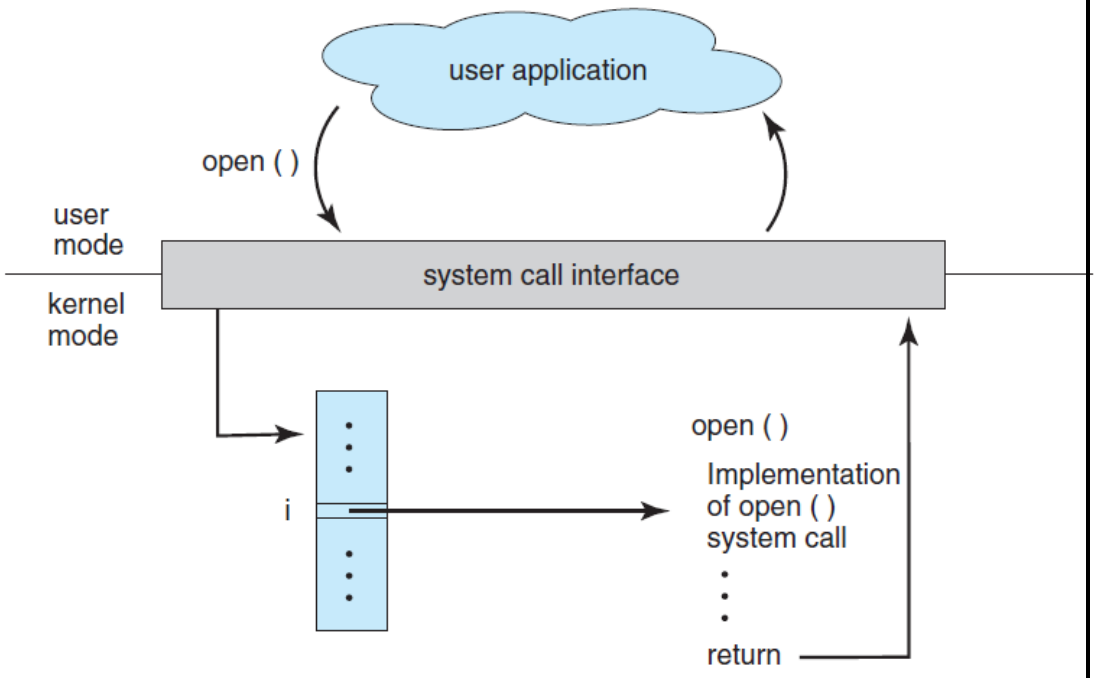


Fig.1.14 The handling of a user application invoking the open system call

- Three general methods are used to pass parameters to the operating system. The simplest approach is to pass the parameters in registers.
- In some cases, however, there may be more parameters than registers. In these cases, the parameters are generally stored in a block, or table, in memory, and the address of the block is passed as a parameter in a register. Refer figure 1.135.

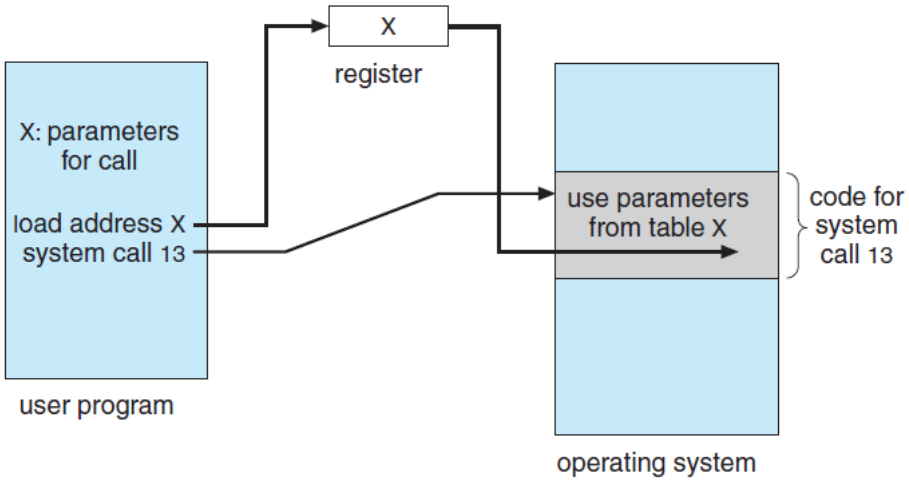


Fig.1.15 Passing Parameter as a Table

Types of System Calls:

System calls can be grouped roughly into six major categories:

- Process control
- File manipulation
- Device manipulation
- Information maintenance
- Communications
- Protection.

Process Control:

- A running program needs to be able to halt its execution either normally (end()) or abnormally (abort()).
- If a system call is made to terminate the currently running program abnormally, or if the program runs into a problem and causes an error trap, a dump of memory is sometimes taken and an error message generated.
- The dump is written to disk and may be examined by a debugger.
- A process is a program in execution. It is a active state of a program. A running program is called as process. It needs several system calls such as

Activities

- end, abort
- load, execute

- create process, terminate process
- get process attributes, set process attributes
- wait for time
- wait event, signal event
- allocate and free memory

File Management:

- Once the file is created, we need to open() it and to use it. We may also read(), write(), or reposition() (rewind or skip to the end of the file, for example).
- Finally, we need to close() the file, indicating that we are no longer using it.
- File attributes include the file name, file type, protection codes, accounting information, and so on.
- At least two system calls, get file attributes() and set file attributes(), are required for this function. Some operating systems provide many more calls, such as calls for file move() and copy().

Activities

- create file, delete file
- open, close
- read, write, reposition
- get file attributes, set file attributes

Device Management:

- A process may need several resources to execute—main memory, disk drives, access to files, and so on. If the resources are available, they can be granted, and control can be returned to the user process. Otherwise, the process will have to wait until sufficient resources are available.
- The various resources controlled by the operating system can be thought of as devices. Some of these devices are physical devices (for example, disk drives), while others can be thought of as abstract or virtual devices (for example, files)
- A system with multiple users may require us to first request() a device, to ensure exclusive use of it. After we are finished with the device, we

release() it. These functions are similar to the open() and close() system calls for files.

- A process may need several resources to execute — main memory, disk drives, access to files, and so on. If the resources are available, they can be granted, and control can be returned to the user process.

Activities

- request device, release device
- read, write, reposition
- get device attributes, set device attributes
- logically attach or detach devices

Information Maintenance:

- Many system calls exist simply for the purpose of transferring information between the user program and the operating system.
- For example, most systems have a system call to return the current time() and date(). Other system calls may return information about the system, such as the number of current users, the version number of the operating system, the amount of free memory or disk space, and so on.
- In addition, the operating system keeps information about all its processes, and system calls are used to access this information.

Activities

- get time or date, set time or date
- get system data, set system data
- get process, file, or device attributes
- set process, file, or device attributes

Communication:

- There are two common models of interprocess communication: the message-passing model and the shared-memory model.
  - In the **message-passing model**, the communicating processes exchange messages with one another to transfer information. Messages can be exchanged between the processes either directly or indirectly through a common mailbox.

- In the **shared-memory model**, processes use shared memory create() and shared memory attach() system calls to create and gain access to regions of memory owned by other processes. Recall that, normally, the operating system tries to prevent one process from accessing another process's memory.

Activities

- create, delete communication connection
- send, receive messages
- transfer status information
- attach or detach remote devices

Protection:

- Protection provides a mechanism for controlling access to the resources provided by a computer system.
- System calls providing protection include set permission() and get permission(), which manipulate the permission settings of resources such as files and disks.
- The allow user() and deny user() system calls specify whether particular users can — or cannot — be allowed access to certain resources.

6. **Describe system programs in detail with neat sketch.**

System Program:

- System programs, provide a convenient environment for program development and execution.

They can be divided into these categories

- **File management**. These programs create, delete, copy, rename, print, dump, list, and generally manipulate files and directories.
- **Status information**. Some programs simply ask the system for the date, time, amount of available memory or disk space, number of users, or similar status information. Others are more complex, providing detailed performance, logging, and debugging information, registry-which is used to store and retrieve configuration information.

- **File modification**. Several text editors may be available to create and modify the content of files stored on disk or other storage devices.
- **Programming-language support**. Compilers, assemblers, debuggers, and interpreters for common programming languages (such as C, C++, Java, and PERL) are often provided with the operating system.
- **Program loading and execution**. Once a program is assembled or compiled, it must be loaded into memory to be executed. The system may provide absolute loaders, relocatable loaders, linkage editors, and overlay loaders.
- **Communications**. These programs provide the mechanism for creating virtual connections among processes, users, and computer systems. They allow users to send messages to one another's screens, to browse Web pages, to send e-mail messages, to log in remotely, or to transfer files from one machine to another.
- **Background services**. All general-purpose systems have methods for launching certain system-program processes at boot time. Some of these processes terminate after completing their tasks, while others continue to run until the system is halted.

7. Discuss in detail about Operating-System Design and Implementation.

Design Goals

- The first problem in designing a system is to define goals and specifications.
- At the highest level, the design of the system will be affected by the choice of hardware and the type of system: batch, time sharing, single user, multiuser, distributed, real time, or general purpose.
- Beyond this highest design level, the requirements may be much harder to specify.
- The requirements can, however, be divided into two basic groups: user goals and system goals.
- **User Goals:** Operating system should be convenient to use, easy to learn, reliable, safe, and fast
- **System Goals:** operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient.

Mechanisms and Policies

- One important principle is the separation of policy from mechanism. Mechanisms determine *how* to do something; policies determine *what* will be done.
- The separation of policy and mechanism is important for flexibility
- If the mechanism is properly separated from policy, it can be used either to support a policy decision that I/O-intensive programs should have priority over CPU-intensive ones or to support the opposite policy.
- Policy decisions are important for all resource allocation. Whenever it is necessary to decide whether or not to allocate a resource, a policy decision must be made. Whenever the question is ***how*** rather than ***what***, it is a mechanism that must be determined.

Implementation

- Once an operating system is designed, it must be implemented. Because operating systems are collections of many programs, written by many people over a long period of time, it is difficult to make general statements about how they are implemented.
- Much variation
    - Early operating systems were written in assembly language
    - Then system programming languages like algol, PL/I
    - Now C, C++
- Actually usually a mix of languages
    - The lowest levels of the kernel might be assembly language.
    - Higher level routines might be in C
    - System programs in C, C++, scripting languages like PERL, Python, shell scripts
- More high level language easier to port to other hardware but slower
- Emulation can allow an OS to run on non-native hardware.
    - Emulators are programs that duplicate the functionality of one system on another system.

- The only possible disadvantages of implementing an operating system in a higher level language are reduced speed and increased storage requirements.
- The major performance improvements in operating systems are more likely to be the result of better data structures and algorithms than of excellent assembly language code. Although operating systems are large, only a small amount of the code is critical to high performance, the interrupt handler, I/O manager, memory manager, and CPU scheduler are probably the most critical routines.

UNIT II PROCESS MANAGEMENT

Processes - Process Concept - Process Scheduling - Operations on Processes - Inter-process Communication; CPU Scheduling - Scheduling criteria - Scheduling algorithms: Threads - Multithread Models – Threading issues; Process Synchronization - The Critical-Section problem - Synchronization hardware – Semaphores – Mutex - Classical problems of synchronization - Monitors; Deadlock - Methods for handling deadlocks, Deadlock prevention, Deadlock avoidance, Deadlock detection, Recovery from deadlock.

PART A

1. Give a programming example in which multithreading does not provide better performance than a single threaded solution.

- pAny kind of sequential program is not a good candidate to be threaded.
- An example of this is a program that calculates an individual tax return.
- Another example is a “shell” program such as the C-shell or Korn shell. Such a program must closely monitor its own working space such as open files, environment variables, and current working directory.

2. What is the meaning of the term busy waiting?

- Busy waiting means that a process is waiting for a condition to be satisfied in a tight loop without relinquishing the processor.
- Alternatively, a process could wait by relinquishing the processor, and block on a condition and wait to be awakened at some appropriate time in the future.
- Busy waiting can be avoided but incurs the overhead associated with putting a process to sleep and having to wake it up when the appropriate program state is reached.

3. Can a multithreaded solution using multiple level threads achieve better performance on a multiprocessor system than on a single processor system?

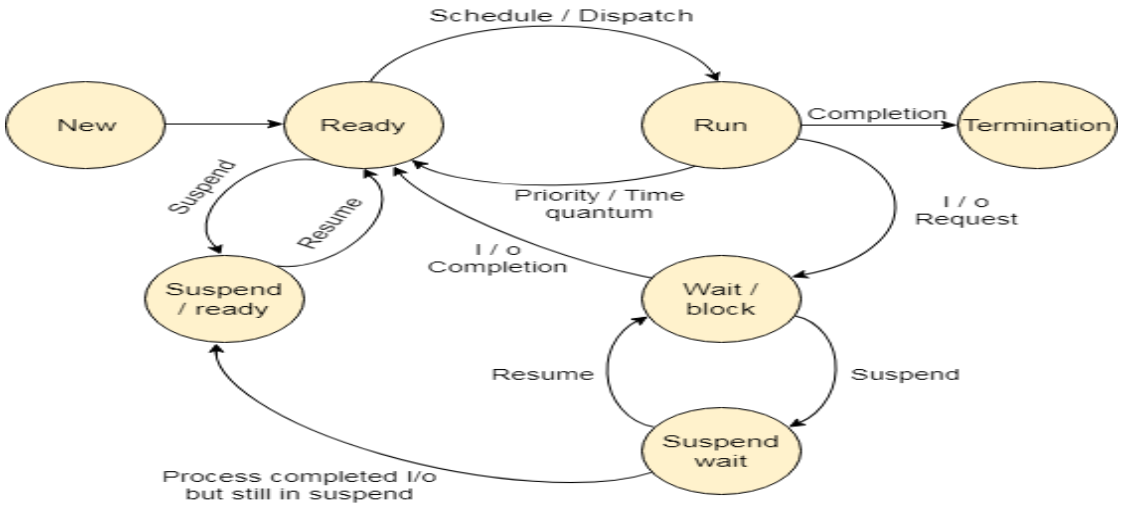
- A multithreaded system comprising of multiple user-level threads cannot make use of the different processors in a multiprocessor system simultaneously.

- The operating system sees only a single process and will not schedule the different threads of the process on separate processors.
- Consequently, there is no performance benefit associated with executing multiple user-level threads on a multiprocessor system.

4. Name and draw five different process states with proper definition. (or) Draw a diagram to show the different process states. (NOV/DEC 2024)

Process – A process is a program in execution

- **New**. The process is being created.
- **Running**. Instructions are being executed.
- **Waiting**. The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
- **Ready**. The process is waiting to be assigned to a processor.
- **Terminated**. The process has finished execution.



5. Elucidate mutex locks with its procedure.

- Operating-systems designers build software tools to solve the critical-section problem.
- The simplest of these tools is the **mutexlock**. (***mut***ual***ex***clusion.)
- We use the mutexlockto protect critical regions and thus prevent race conditions.
- That is, a process must acquire the lock before entering a critical section; it releases the lock when it exits the critical section.

- The acquire()function acquires the lock, and the release() function releases the lock.

6. "Priority inversion is a condition that occurs in real time systems where a lower priority access is starved because higher priority processes have gained hold of the CPU. ". Comment on this statement.(April/May 2024)

- Priority inversion is a problem that occurs in concurrent processes when low-priority threads hold shared resources required by some high-priority threads, causing the high priority-threads to block indefinitely.
- This problem is enlarged when the concurrent processes are in a real time system where high- priority threads must be served on time.
- When a high-priority thread needs a resource engaged by a low-priority thread, the low priority thread is preempted, the original resource is restored and the high-priority thread is allowed to use the original resource.

7. Differentiate single threaded and multithreaded processes.

User Level Threads or Single Thread

- User level threads are managed by a user level library
- In this case, the kernel may not favor a process that has many threads.
- User level threads are typically fast.
- They are a good choice for non blocking tasks otherwise the entire process will block if any of the threads blocks.

Kernel Level Threads or Multithread

- Kernel level threads are managed by the OS, therefore, thread operations (ex. Scheduling) are implemented in the kernel code.
- This means kernel level threads may favor thread heavy processes.
- If one thread blocks it does not cause the entire process to block.
- They are slower than user level threads due to the management overhead.

8. What is the difference between user level instruction and privileged instructions? Which of the following instructions should be privileged and only allowed to execute in kernel mode?

a) Load a value from a memory address to a general purpose register

b) Set a new value in a program counter register.

c) Turn off interrupts

Ans:

Load a value from a memory address to a general purpose register

User mode

- Regular instructions
- Access user memory

Kernel (privileged) mode

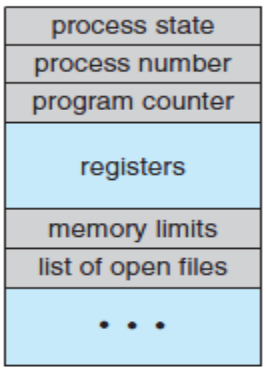
- Regular instructions
- Privileged instructions
- Access user memory
- Access kernel memory

9. Define a process.

A process is a program in execution. It is an active entity and it includes the process stack, containing temporary data and the data section contains global variables.

10. What is process control block?

Each process is represented in the OS by a process control block. It contains many pieces of information associated with a specific process.



11. What is zombie process?

A process that has terminated, but whose parent has not yet called wait(), isknown as a zombie process.

12. What are the benefits of threads?

- Responsiveness.
- Resource sharing.
- Economy.
- Scalability

13. What do you mean by multicore or multiprocessor system?

- System design is to place multiple computing cores on a single chip.
- Each core appears as a separate processor to the operating system.
- Whether the cores appear across CPU chips or within CPU chips, we call these systems multicore or multiprocessor systems.

14. What are the benefits of multithreaded programming?

The benefits of multithreaded programming can be broken down into four major categories:

- Responsiveness
- Resource sharing
- Economy
- Utilization of multiprocessor architectures

15. What is critical section problem?

- Consider a system consists of 'n' processes. Each process has segment of code called a critical section, in which the process may be changing common variables, updating a table, writing a file.
- When one process is executing in its critical section, no other process can allowed executing in its critical section.

16. Define busy waiting and spinlock.

- When a process is in its critical section, any other process that tries to enter its critical section must loop continuously in the entry code. This is called as busy waiting.
- Spinlock - the process spins while waiting for the lock.

17. What do you meant by semaphore?

A semaphoreS is an integer variable that, apart from initialization, isaccessed only through two standard atomic operations:

1. wait() - operation was originally termed P (from the Dutch ***proberen,*** "totest");
2. signal() - was originally called V (from ***verhogen,*** "to increment").

18. What are the four circumstances in CPU scheduling decisions?

1. When a process switches from the running state to the waiting state.
2. When a process switches from the running state to the ready state.
3. When a process switches from the waiting state to the ready state.
4. When a process terminates

19. Define race condition.

- When several process access and manipulate same data concurrently, then the outcome of the execution depends on particular order in which the access takes place is called race condition.
- To avoid race condition, only one process at a time can manipulate the shared variable.

20. Define deadlock.

- A process requests resources; if the resources are not available at that time, the process enters a wait state.
- Waiting processes may never again change state, because the resources they have requested are held by other waiting processes. This situation is called a deadlock.

21. What are a safe state and an unsafe state?

- A state is safe if the system can allocate resources to each process in some order and still avoid a deadlock. A system is in safe state only if there exists a safe sequence.
- If no such sequence exists, then the system state is said to be unsafe.

22. What is Amdahl's law?

Amdahl's Law is a formula that identifies potential performance gains from adding additional computing cores to an application that has both serial (nonparallel) and parallel components.

23. What are the benefits of multithreads?

1. **Responsiveness** - One thread may provide rapid response while other threads are

blocked or slowed down doing intensive calculations.

2. **Resource sharing** - By default threads share common code, data, and otherresources, which allows multiple tasks to be performed simultaneously in a single address space.

3. **Economy** - Creating and managing threads (and context switches between them)

is much faster than performing the same tasks for processes.

4. **Scalability,** i.e. Utilization of multiprocessor architectures - A single threaded process can only run on one CPU, no matter how many may beavailable, whereas the execution of a multi-threaded application may be split amongst available processors.

24. Give the necessary conditions for deadlock to occur?

Mutual exclusion:

- At least one resource must be held in a non sharable mode.
- That is only one process at a time can use the resource.
- If another process requests that resource, the requesting process must be delayed until the resource has been released.

Hold and wait:

- A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.

No preemption:

- Resources cannot be preempted.

Circular wait:

- P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P2...Pn-1.

25. Can multiple user level threads achieve better performance on a multiprocessor system than a single processor system? Justify your answer.

- A process has multiple tasks.
- When one of the tasks may block, and it is desired to allow the other tasks to proceed without blocking.
- For example in a word processor, a background thread may check spelling and grammar while a foreground thread processes user input (keystrokes), third thread loads images from the hard drive and a fourth does periodic automatic backups of the file being edited.

26. Write the four situations under which CPU scheduling decisions take place.

CPU scheduling decisions may take place under the following four circumstances:

1. When a process switches from the running state to the waiting state.
2. When a process switches from the running state to the ready state.
3. When a process switches from the waiting state to the ready state.
4. When a process terminates.

27. What is a critical region? How do they relate to controlling access to shared resources?

- A critical region is a section of code in which a shared resource is accessed.
- To control access to the shared resource, access is controlled to the critical region of code. By controlling the code that accessed the resource we can control access to the resource.

28. What is the producer consumer problem? Give an example of its occurrence in operating systems.

- The producer consumer problem is a classic concurrency problem. It arises when a process is producing some data, the producer, and another process is using that data, the consumer.

- An example in an operating system would be interfacing with a network device. The network device produces data at a certain rate and places it on a buffer, the operating system then consumes the data at another rate.

29. What are monitors and condition variables?

- Monitors are high level synchronization primitives that encapsulate data, variables and operations.
- A conditional variable is variable placed inside a monitor to allow for processes to wait inside a monitor until a special event has occurred.

30. What is a deadlock? What is starvation? How do they differ from each other? (April/May 2024)

- A deadlock is a situation where a number of processes cannot precede without a resource that another holds, but cannot release any resources that it is currently holding. A deadlock ensures that no processes can proceed.
- Where as a starvation is a situation where a signal process is never given a resource or event that it requires to proceed, this includes CPU time.
- They differ in that a deadlock ensures that no processes can proceed where as starvation only a single process fails to proceed.

31. Define Semaphore.

- A semaphore is an integer variable, shared among multiple processes. The main aim of using a semaphore is process synchronization and access control for a common resource in a concurrent environment.

32. Define MUTEX.

- Mutex lock is essentially a variable that is binary nature that provides code wise functionality for mutual exclusion. At times, there maybe multiple threads that may be trying to access same resource like memory or I/O etc.
- To make sure that there is no overriding. Mutex provides a locking mechanism.

- Only one thread at a time can take the ownership of a mutex and apply the lock.
- Once it done utilizing the resource and it may release the mutex lock.

33. How deadlocks can be avoided?

A deadlock occurs when the first process locks the first resource at the same time as the second process locks the second resource. The deadlock can be resolved by cancelling and restarting the first process.

34. List out the benefits and challenges of thread handling.

- Enhanced performance by decreased development time.
- Simplified and streamlined program coding.
- Improvised GUI responsiveness.
- Simultaneous and parallelized occurrence of tasks.
- Better use of cache storage by utilization of resources.
- Decreased cost of maintenance.

35. What is external fragmentation? (NOV/DEC 2024)

External fragmentation in computer memory management refers to a situation where there is enough free memory available in total, but it is scattered across numerous small, non-contiguous blocks, making it difficult to allocate a large contiguous block of memory to a new process, even if the total free memory is sufficient; essentially, the available memory is fragmented into unusable pieces despite there being enough space overall.

PART B

1. **Explain in detail about process concept. (or) Explain the seven-state model of a process and also explain the queuing diagram for the same. Is it possible to have more than one blocked queue, if so can a process reside in more than one queue? (April/May 2024)**

Process Concept

- Process is a program in execution. A process is the unit of work in a modern time-sharing system.
- Even on a single-user system, a user maybe able to run several programs at one time: a word processor, a Web browser, and an e-mail package.
- It also includes program counter, processor's registers, stack, data section, heap; Refer figure 1=2.1.

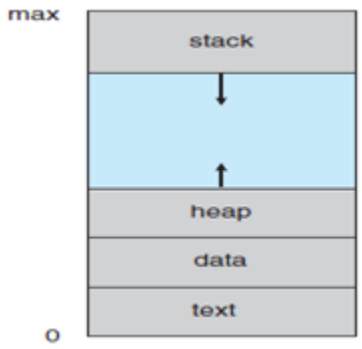


Fig.2.1 Process in memory

- A program is a ***passive*** entity, A process is an ***active*** entity,

Process States

- **New**: The process is being created.
- **Running**: Instructions are being executed.
- **Waiting or Blocked:** The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
- **Ready**: The process is waiting to be assigned to a processor.
- **Terminated**: The process has finished execution.
- **Suspended Ready:** The process is temporarily moved from the ready queue to secondary storage (like a hard disk) but is still considered ready to execute when needed.
- **Suspended Blocked:** A process is waiting for an event while being suspended in secondary storage. Refer figure 2.2.

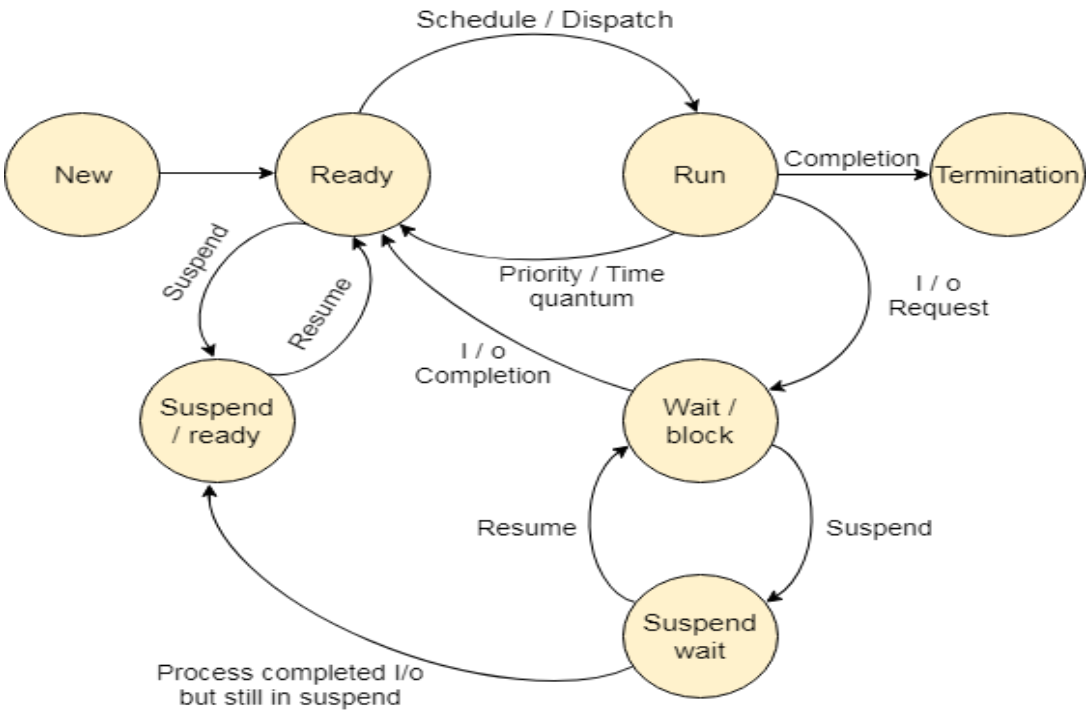


Fig.2.2 Process State

Process Control Block(or) Task control block.

- It contains many pieces of information associated with a specific process, including these as shown figure 2.3.

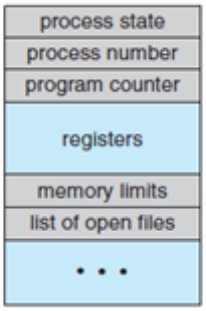


Fig.2.3 Process Control Block

- **Process state**. The state may be new, ready, running, and waiting, halted, and so on.
- **Program counter**. The counter indicates the address of the next instructionto be executed for this process.
- **CPU registers**. It include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code information.

- **CPU-scheduling information**. This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.
- **Memory-management information**. This information may include such items as the value of the base and limit registers and the page tables, or the segment tables, depending on the memory system used by the operating system.
- **Accounting information**. This information includes the amount of CPUand real time used, time limits, account numbers, job or process numbers, and so on.
- **I/O status information**. This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

2. **Describe the difference among short term, medium term and long term scheduling with suitable example.**

Process Scheduling

- The objective of **multiprogramming** is to have some process running at all times, to maximize CPU utilization.
- The objective of **time sharing** is to switch the CPU among processes so frequently that users can interact with each program while it is running.
- To meet these objectives, the process scheduler selects an available process (possibly from a set of several available processes) for program execution on the CPU.

Scheduling Queues

- As processes enter the system, they are put into a **job queue**, which consists of all processes in the system.
- The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the **ready queue**.
- The list of processes waiting for a particular I/O device is called a device queue. Each device has its own **device queue**. Refer figure 2.4.

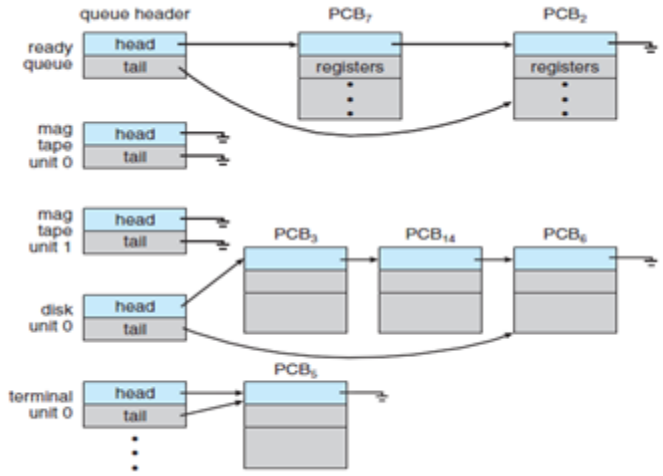


Fig.2.4 The ready queue and various I/O device queues

Schedulers

- The **long-term scheduler**, or **job scheduler**, selects processes from the job pool and loads them into memory for execution.
- The **short-term scheduler**, or **CPU scheduler**, selects from among the processes that are ready to execute and allocates the CPU to one of them. Refer figure 2.5.

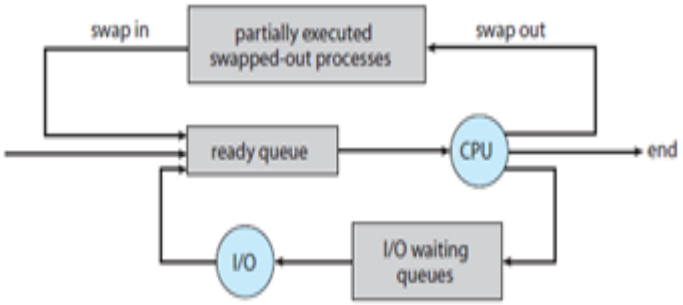


Fig.2.5 Medium term scheduling to the queuing diagram

3. **Explain in detail about operations of process.**

Operations

1. Process creation
2. Termination

1. Process Creation

- During the course of execution, a process may create several new processes.
- The creating process is called a parent process, and the new processes are called the children of that process.

When a process creates a new process, two possibilities for execution:

1. The parent continues to execute concurrently with its children.
2. The parent waits until some or all of its children have terminated.

There are also two address-space possibilities for the new process:

1. The child process is a duplicate of the parent process (it has the same program and data as the parent).
2. The child process has a new program loaded into it.

- A new process is created by the fork() system call. The new process consists of a copy of the address space of the original process.
- The exec () system call loads a binary file into memory and starts its execution as shown in figure 2.6.

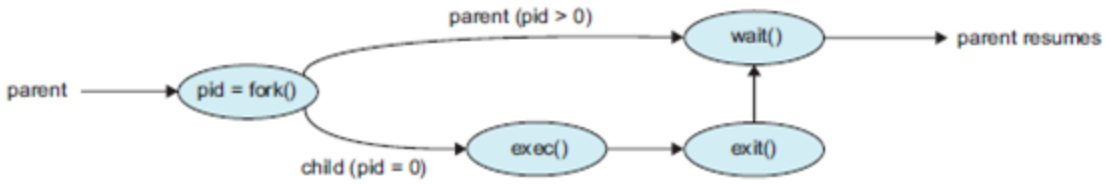


Fig.2.6 process creation using the fork() system call

Program to implement for Creating a separate process using the UNIX fork() system call.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
pid t pid;
/* fork a child process */
pid = fork();
if (pid< 0) { /* error occurred */
fprintf(stderr, "Fork Failed");
return 1;
}
else if (pid == 0) { /* child process */
execlp("/bin/ls","ls",NULL);
}
else{ /* parent process */
```

```
/* parent will wait for the child to complete */
wait(NULL);
printf("Child Complete");
}
return 0;
}
```

2. Process Termination

- A process terminates when it finishes executing its final statement and asks the operating system to delete it by using the exit() system call.

A parent may terminate the execution of one of its children for a variety of reasons, such as these:

1. The child has exceeded its usage of some of the resources that it has been allocated. (To determine whether this has occurred, the parent must have a mechanism to inspect the state of its children.)
2. The task assigned to the child is no longer required.
3. The parent is exiting, and the operating system does not allow a child to continue if its parent terminates.

Some systems do not allow a child to exist if its parent has terminated.

1. If a process terminates (either normally or abnormally), then all its children must also be terminated. This phenomenon, referred to as cascading termination, is normally initiated by the operating system.
2. A parent process may wait for the termination of a child process by using the wait() system call.
3. When a process terminates, its resources are de - allocated by the operating system.
4. A process that has terminated, but whose parent has not yet called wait(), is known as a **zombie** process.
5. If a parent did not invoke wait() and instead terminated, there by leaving its child processes as **orphans**.

4. Explain in detail about Inter Process Communication.

- Processes executing concurrently in the operating system may be either independent processes or cooperating processes.
- Several cooperating processes executing concurrently is Inter process Communication
- There are several reasons for providing an environment that allows process cooperation:
  - **Information sharing**
    Several users access the information concurrently
  - **Computation speedup**
    If we want a particular task to run faster, we must break it into subtasks, each of which will be executing in parallel with the others
  - **Modularity**
    Dividing the system functions into separate processes or threads
  - **Convenience**
    Even an individual user may work on many tasks at the same time

There are two fundamental models of Inter Process Communication:

1. Shared Memory
2. Message Passing

1. In the shared-memory model, a region of memory that is shared by cooperating processes is established. Processes can then exchange information by reading and writing data to the shared region.

2. In the message-passing model, communication takes place by means of messages exchanged between the cooperating processes. Refer figure 2.7.

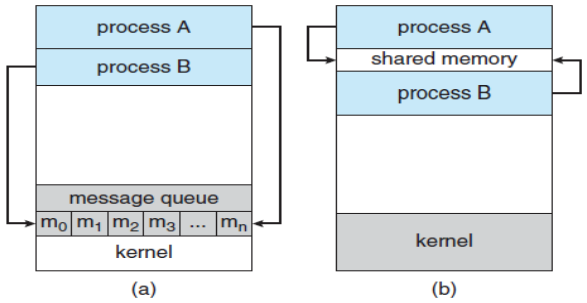


Fig. 2.7 Communications models. (a) Message passing. (b) Shared memory.

1. **Shared-Memory Systems**
   - Inter Process Communication using shared memory requires communicating processes to establish a region of shared memory.
   - A **producer** process produces information that is consumed by a **consumer** process.
   - One solution to the producer–consumer problem uses shared memory.
   - Two types of buffers can be used.
   - The unbounded buffer places no practical limit on the size of the buffer.
   - The bounded buffer assumes a fixed buffer size.
2. **Message-Passing Systems**
   - Message passing provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space.
   - A message-passing facility provides at least two operations:
     - send(message)
     - receive(message)

Here are several methods for send()/receive() operations:

- Naming
- Synchronization
- Buffering

1. **Naming:**
   - Direct or indirect communication
   - Synchronous or asynchronous communication
   - Automatic or explicit buffering

Direct and Indirect Communication

In Direct Communication, Each process that wants to communicate must explicitly name the recipient or sender of the communication.

In this scheme, the send() and receive() primitives are defined as:

- send(P, message)—Send a message to process P.
- receive(Q, message)—Receive a message from process Q.

A direct communication link in this scheme has the following properties:

- *Symmetry communication* - both the sender process and the receiver process must name the other to communicate.

- *Asymmetry communication* - only the sender names the recipient; the recipient is not required to name the sender.
  - send(P, message)—Send a message to process P.
  - receive(id, message)—Receive a message from any process.

The disadvantage in both of these schemes symmetric and asymmetric is the limited modularity of the resulting process definitions.

With *indirect communication*, the messages are sent to and received from *mailboxes*, or *ports*.

- A mail box can be viewed abstractly as an object into which messages can be placed by processes and from which messages can be removed.
- Each mailbox has a unique identification.

The send() and receive()primitives are defined as follows:

- send(A, message)—Send a message to mailbox A.
- receive(A, message)—Receive a message from mailbox A.

In this scheme, a communication link has the following properties:

- A link is established between a pair of processes only if both members' of the pair has a shared mailbox.
- A link may be associated with more than two processes.
- Between each pair of communicating processes, a number of different links may exist, with each link corresponding to one mailbox.

2. Synchronization

- Communication between processes takes place through calls to send() and receive() primitives.
- **Blocking send.** The sending process is blocked until the message is received by the receiving process or by the mailbox.
- **Nonblocking send**. The sending process sends the message and resumes operation.
- **Blocking receive.** The receiver blocks until a message is available.
- **Nonblocking receive.** The receiver retrieves either a valid message or a null.

3. Buffering

- Whether communication is direct or indirect, messages exchanged by communicating processes reside in a temporary queue.

Such queues can be implemented in three ways:

- **Zero capacity**. The queue has a maximum length of zero;
- **Bounded capacity**. The queue has finite length *n;*
- **Unbounded capacity**. The queue's length is potentially infinite;

5. Explain in detail about threads.

Thread Overview:

- A thread is a basic unit of CPU utilization.
- It comprises a thread ID, a program counter, a register set, and a stack.
- Figure 2.8 shows the difference between single threaded and multithreaded processes.

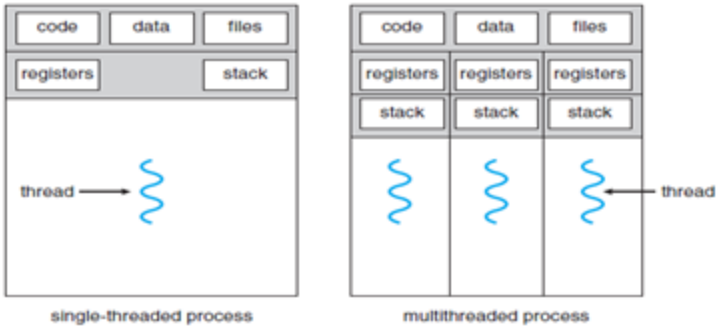


Fig. 2.8 Single threaded and multithreaded processes

Motivation

- Most software applications that run on modern computers are multithreaded.
- A web browser might have one thread display images or text while another thread retrieves data from the network.
- For example: A word processor may have a thread for displaying graphics, another thread for responding to keystrokes from the user, and a third thread for performing spelling and grammar checking in the background.

Benefits

1. Responsiveness.

- Multithreading an interactive application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user.

2. Resource sharing.

- Processes can only share resources through techniques such as shared memory and message passing.

- The benefit of sharing code and data is that it allows an application to have several different threads of activity within the same address space.

3. Economy.

- Allocating memory and resources for process creation is costly.
- Because threads share the resources of the process to which they belong, it is more economical to create and context-switch threads.

4. Scalability.

- The benefits of multithreading can be even greater in a multiprocessor architecture, where threads may be running in parallel on different processing cores.
- A single-threaded process can run on only one processor, regardless how many are available.

6. Explain in detail about multicore programming.

Multicore Programming

- Computer systems need for more computing performance, single-CPU systems evolved into multi-CPU systems.
- System design is to place multiple computing cores on a single chip.
- Each core appears as a separate processor to the operating system. Refer figure 2.9.
- Whether the cores appear across CPU chips or within CPU chips, we call these systems **multicore** or **multiprocessor** systems. Refer figure 2.10.

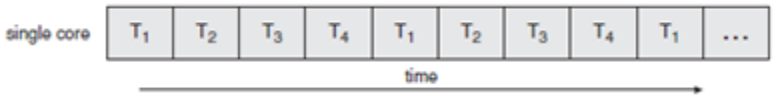


Fig. 2.9 concurrent execution on a single-core system

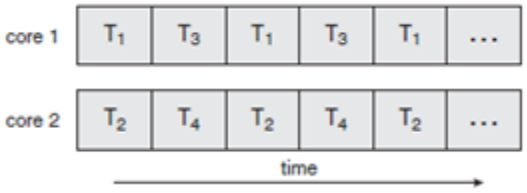


Fig.2.10. parallel execution on a multicore system

- A system is parallel if it can perform more than one task simultaneously.

AMDAHL'S LAW

- Amdahl's Law is a formula that identifies potential performance gains from adding additional computing cores to an application that has both serial (nonparallel) and parallel components.
- If S is the portion of the application that must be performed serially on a system with N processing cores, the formula appears as follows:

$$speedup \leq \frac{1}{S + (1-S)N}$$

Programming Challenges

- Designers of operating systems must write scheduling algorithms that use multiple processing cores to allow the parallel execution shown in Figure.

In general, **five areas** present challenges in programming for multicore systems:

- **Identifying tasks**
  - This involves examining applications to find are as that can be divided into separate, concurrent tasks.
  - Ideally, tasks are independent of one another and thus can run in parallel on individual cores.
- **Balance**
  - While identifying tasks that can run in parallel, programmers must also ensure that the tasks perform equal work of equal value.
- **Data splitting**
  - Just as applications are divided into separate tasks, the data accessed and manipulated by the tasks must be divided to run on separate cores.
- **Data dependency**
  - The data accessed by the tasks must be examined for dependencies between two or more tasks.
  - When one task depends on data from another, programmers must ensure that the execution of the tasks is synchronized to accommodate the data dependency.
- **Testing and debugging**
  - When a program is running in parallel on multiple cores, many different execution paths are possible.
  - Testing and debugging such concurrent programs is inherently more difficult than testing and debugging single-threaded applications.

Types of Parallelism

- In general, there are two types of parallelism:
  1. Data parallelism
  2. Task parallelism.
- **Data parallelism -** The two threads would be running in parallel on separate computing cores.
- **Task parallelism -** involves distributing not data but tasks (threads) across multiple computing cores. Each thread is performing a unique operation.

7. Explain in detail about multithreading models.

Multithreading Models

- Support for threads may be provided either at the user level, for **user threads**, or by the kernel, for **kernel threads**.
- User threads are supported above the kernel and are managed without kernel support, whereas kernel threads are supported and managed directly by the operating system.
- Three common ways of establishing such are Relationship:
  - many-to-one model
  - one-to-one model
  - many-to-many model

Many-to-One Model

- The many-to-one model maps many user-level threads to one kernel thread.
- Thread management is done by the thread library in user space, so it is efficient.
- However, the entire process will block if a thread makes a blocking system call.
- Many-to-one model allows the developer to create as many user threads, it does not result in true concurrency, because the kernel can schedule only one thread at a time. Refer figure 2.11.

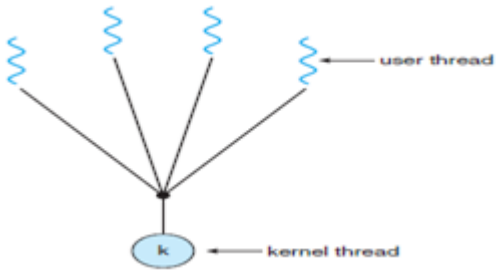


Fig.2.11 Many to One Model

One-to-One Model

- The one-to-one model maps each user thread to a kernel thread.
- It also allows multiple threads to run in parallel on multiprocessors.
- The only drawback to this model is that creating a user thread requires creating the corresponding kernel thread.
- The one-to-one model allows greater concurrency, but the developer has to be careful not to create too many threads within an application. Refer 2.12.

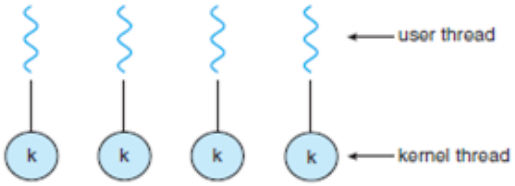


Fig.2.12. One to One Model

Many-to-Many Model

- The many-to-many model multiplexes many user-level threads to a smaller or equal number of kernel threads as shown in figure 2.13.

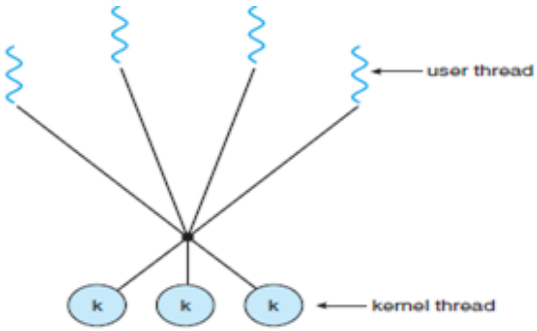


Fig.2.13 Many to Many Model

Two level model

- One variation on the many-to-many model still multiplexes many user level threads to a smaller or equal number of kernel threads but also allows a user-level thread to be bound to a kernel thread as shown in figure 2.14.
- This variation is sometimes referred to as the **two-level model.**

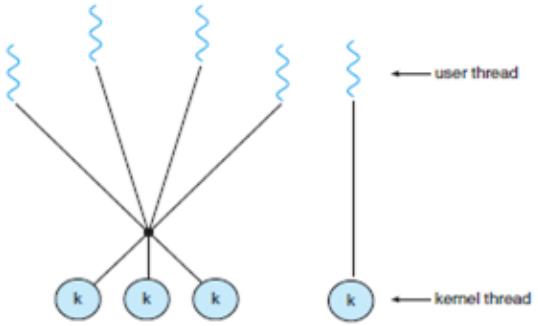


Fig.2.14 Two level model

Threading issues:

- fork() and exec() system calls
- Thread cancellation
- Signal handling
- Thread pools
- Thread specific data

fork() and exec() system calls:

- A fork() system call may duplicate all threads or duplicate only the thread that invoked fork().
- If a thread invoke exec() system call ,the program specified in the parameter to exec will replace the entire process.

Thread cancellation:

- It is the task of terminating a thread before it has completed.
- A thread that is to be cancelled is called a target thread.
- There are two types of cancellation namely

1. **Asynchronous Cancellation** – One thread immediately terminates the target thread.
2. **Deferred Cancellation** – The target thread can periodically check if it should terminate, and does so in an orderly fashion.

Signal handling:

1. A signal is a used to notify a process that a particular event has occurred.
2. A generated signal is delivered to the process.
    - Deliver the signal to the thread to which the signal applies.
    - Deliver the signal to every thread in the process.

- Deliver the signal to certain threads in the process.
- Assign a specific thread to receive all signals for the process.

3. Once delivered the signal must be handled.

Signal is handled by

i. A default signal handler

ii. A user defined signal handler

Thread pools:

- Creation of unlimited threads exhausts system resources such as CPU time or memory. Hence we use a thread pool.
- In a thread pool, a number of threads are created at process startup and placed in the pool.
- Then there is a need for a thread the process will pick a thread from the pool and assign it a task.
- After completion of the task, the thread is returned to the pool.

Thread specific data

- Threads belonging to a process share the data of the process. However each thread might need its own copy of certain data known as thread-specific data.

8. Explain in detail about SMP (Symmetric **M**ulti **P**rocessor**) management.**

- One approach to CPU scheduling in a multiprocessor system has all scheduling decisions, I/O processing, and other system activities handled by a single processor—the master server.
- The other processors execute only user code.
- This asymmetric multiprocessing is simple because only one processor accesses the system data structures, reducing the need for data sharing.
- A second approach uses Symmetric Multi Processing (SMP), where each processor is self-scheduling.
- All processes may be in a common ready queue, or each processor may have its own private queue of ready processes.
- Regardless, scheduling proceeds by having the scheduler for each processor examine the ready queue and select a process to execute.

- If we have multiple processors trying to access and update a common data structure, the scheduler must be programmed carefully.
- We must ensure that two separate processors do not choose to schedule the same process and that processes are not lost from the queue.

9. Explain in detail about process synchronization and explain critical section problem.

Process Synchronization

- Concurrent access to shared data may result in data inconsistency.
- Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes.
- Shared memory solution to bounded buffer problem allows at most n-1 items in buffer at the same time. A solution, where all N buffers are used is not simple.
- Suppose that we modify the producer-consumer code by adding a variable counter, initialized to 0 and increment it each time a new item is added to the buffer.
- **Race condition:** the situation where several processes access and manipulate shared data concurrently. The final value of the shared data depends upon which process finishes last.
- To prevent race conditions, concurrent processes must be synchronized.

Critical-Section Problem

- Consider a system consisting of n processes$\{P0, P1, ...,Pn-1\}$.
- Each process has a segment of code, called a **critical section**, in which the process may be changing common variables, updating a table, writing a file, and so on.
- The important feature of the system is that, when one process is executing in its critical section, no other process is allowed to execute in its critical section.
- That is, no two processes are executing in their critical sections at the same time.

- The ***critical-section problem*** is to design a protocol that the processes can use to cooperate. Each process must request permission to enter its critical section.
- The section of code implementing this request is the **entry section**.
- The critical section may be followed by an **exit section**.
- The remaining code is the **remainder section**.

The general structure of a typical process *Pi*

```
do
{
entry section
critical section
exit section
remainder section
} while (true);
```

A solution to the critical-section problem must satisfy the following three requirements:

1. Mutual exclusion.

- If process *Pi* is executing in its critical section, then no other processes can be executing in their critical sections.

2. Progress.

- If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder sections can participate in deciding which will enter its critical section next, and this selection cannot be postponed indefinitely.

3. Bounded waiting.

- There exists a bound, or limit, on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

10. Define Critical Section, critical resource. Explain the need for enforcing mutual exclusion using echo function as an example in both uniprocessor and multiprocessor environment. (April/May 2024)

Critical Section: A critical section is a part of a program that accesses shared resources, such as memory, files, or variables, which must not be accessed by more than one process or thread at a time to prevent race conditions.

Critical Resource: A critical resource is a resource that multiple processes or threads need to access, but only one can use at a time to maintain data consistency and avoid conflicts.

Need for Enforcing Mutual Exclusion

Mutual exclusion ensures that when one process is executing in its critical section, no other process can enter its critical section simultaneously. This is necessary to prevent data corruption, inconsistent states, and race conditions.

Example: echo Function in Uniprocessor and Multiprocessor Environments

The echo command in Unix-based systems prints text to the terminal. If multiple processes execute echo simultaneously, they may try to write to the same output stream (e.g., the terminal), causing interleaved or corrupted output.

Uniprocessor Environment

- Since only one process executes at a time due to time-sharing, mutual exclusion is enforced using software mechanisms like semaphores or locks.
- Example issue: If two processes execute echo "Hello" concurrently, without mutual exclusion, their outputs may mix (HHeelllooo instead of Hello).
- Solution: Implement a lock (e.g., a semaphore) that ensures only one process executes echo at a time.

Multiprocessor Environment

- Multiple processors can execute different processes simultaneously, increasing the risk of race conditions.
- Example issue: If two processes running on separate CPUs execute echo "Hello" simultaneously, both may try to write to the terminal at the same time, leading to mixed output.

- Solution: Hardware-based locking mechanisms (e.g., test-and-set locks, spinlocks) or OS-level synchronization (mutexes, semaphores) enforce mutual exclusion to prevent data corruption.

Thus, enforcing **mutual exclusion** ensures that shared resources like output buffers are accessed in a controlled manner, preserving correct execution order and preventing unexpected behavior.

11. Explain in detail about mutex locks.

Mutex Locks(***mut***ual ***ex***clusion.)

- We use the mutex lock to protect critical regions and thus prevent race conditions.
- That is, a process must acquire the lock before entering a critical section; it releases the lock when it exits the critical section.
- The acquire() function acquires the lock, and the release() function releases the lock,

Solution to the critical-section problem using mutex locks

The definition of acquire() is as follows:

```
acquire()
{
while (!available)
; /* busy wait */
available = false;;
}
do
{
acquire lock
critical section
release lock
remainder section
} while (true);
```

- A mutex lock has a boolean variable available whose value indicates if the lock is available or not.
- If the lock is available, a call to acquire() succeeds, and the lock is then considered unavailable.

- A process that attempts to acquire an unavailable lock is blocked until the lock is released.

The definition of release() is as follows:

```
release()
{
available = true;
}
```

- Calls to either acquire() or release() must be performed atomically.
- The main disadvantage of the implementation given here is that it requires **busy waiting**.
- While a process is in its critical section, any other process that tries to enter its critical section must loop continuously in the call to acquire().
- In fact, this type of mutex lock is also called a **spin lock** because the process "spins" while waiting for the lock to become available.
- In multiprocessor systems, one thread can "spin" on one processor while another thread performs its critical section on another processor.

12. Explain in detail about semaphores. Write the algorithm using test and set instruction that satisfy all the critical section requirements.

A semaphore S is an integer variable that, apart from initialization, is accessed only through two standard atomic operations:

wait() - operation was originally termed P (from the Dutch ***proberen,*** "totest");

signal() - was originally called V (from ***verhogen,*** "to increment").

The definition of ***wait()*** is as follows:

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

The definition of signal() is as follows:

```
signal(S)
{
```

```
S++;
}
```

- All modifications to the integer value of the semaphore in the wait() and signal() operations must be executed indivisibly.
- That is, when one process modifies the semaphore value, no other process can simultaneously modify that same semaphore value.

Test and Set() instruction

```
booleanTestAndSet (boolean&target)
{
booleanrv = target;
target = true;
returnrv;
}
```

Swap

```
void Swap (boolean&a, boolean&b)
{
boolean temp = a;
a = b;
b = temp;
}
```

Semaphore Usage

- Operating systems often distinguish between counting and binary semaphores.
- The value of a counting semaphore can range over an unrestricted domain.
- The value of a binary semaphore can range only between 0 and 1.
- Thus, binary semaphores behave similarly to mutex locks.
- Counting semaphores can be used to control access to a given resource consisting of a finite number of instances.

Semaphore Implementation

To implement semaphores under this definition, we define a semaphore as follows:

```
type def struct
{
int value;
struct process *list;
} semaphore;
```

- When a process must wait on a semaphore, it is added to the list of processes.
- A signal() operation removes one process from the list of waiting processes and awakens that process.

The wait() semaphore operation can be defined as

```
wait(semaphore *S)
{
S->value--;
if (S->value < 0) {
add this process to S->list;
block();
}
}
```

The signal() semaphore operation can be defined as

```
signal(semaphore *S)
{
S->value++;
if (S->value <= 0) {
remove a process P from S->list;
wakeup(P);
}
}
```

- The block() operation suspends the process that invokes it.
- The wakeup(P) operation resumes the execution of a blocked process P.

13. Explain in detail about Classical Problems of Synchronization.

The following problems of synchronization are considered as classical problems:

- Bounded-buffer (or Producer-Consumer) Problem
- Dining-Philosophers Problem
- Readers and Writers Problem
- Sleeping Barber Problem

1. Bounded-buffer (or Producer-Consumer) Problem:

- Bounded Buffer problem is also called producer consumer problem.
- This problem is generalized in terms of the Producer-Consumer problem.
- Solution to this problem is, creating two counting semaphores "full" and "empty" to keep track of the current number of full and empty buffers respectively. Producers produce a product and consumers consume the product, but both use of one of the containers each time.

Consumer process using shared memory

```
while (true) {
/* produce an item in next produced */
while (((in + 1) % BUFFER SIZE) == out)
; /* do nothing */
buffer[in] = next produced;
in = (in + 1) % BUFFER SIZE;
        }
```

The structure of the consumer process.

```
do {
wait(full);
wait(mutex);
. . .
/* remove an item from buffer to next consumed */
. . .
signal(mutex);
signal(empty);
. . .
```

```
/* consume the item in next consumed */
. . .
} while (true);
```

2. Dining-Philosophers Problem:

- The Dining Philosopher Problem states that K philosophers seated around a circular table with one chopstick between each pair of philosophers as shown figure 2.15.
- There is one chopstick between each philosopher. A philosopher may eat if he can pickup the two chopsticks adjacent to him.
- One chopstick may be picked up by any one of its adjacent followers but not both. This problem involves the allocation of limited resources to a group of processes in a deadlock-free and starvation-free manner.

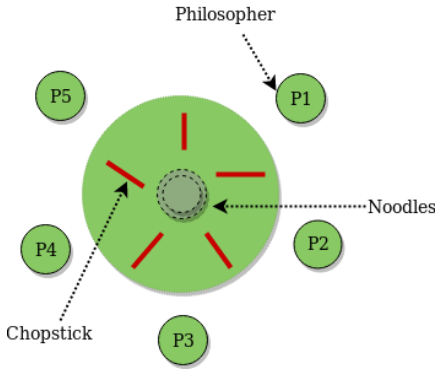


Fig.2.15 Dining philosophers

The structure of philosopher

```
do {
wait(chopstick[i]);
wait(chopstick[(i+1) % 5]);
. . .
/* eat for awhile */
. . .
signal(chopstick[i]);
signal(chopstick[(i+1) % 5]);
. . .
/* think for awhile */
. . .
```

```
} while (true);
```

2. Readers and Writers Problem:

- Suppose that a database is to be shared among several concurrent processes. Some of these processes may want only to read the database, whereas others may want to update (that is, to read and write) the database.
- We distinguish between these two types of processes by referring to the former as readers and to the latter as writers.
- Precisely in OS we call this situation as the readers-writers problem.
- Problem parameters:
  - One set of data is shared among a number of processes.
  - Once a writer is ready, it performs its write. Only one writer may write at a time.
  - If a process is writing, no other process can read it.
  - If at least one reader is reading, no other process can write.
  - Readers may not write and only read.

The structure of writer process

```
do {
wait(rw mutex);
. . .
/* writing is performed */
. . .
signal(rw mutex);
} while (true);
```

The structure of reader process

```
do {
wait(mutex);
read count++;
if (read count == 1)
wait(rw mutex);
signal(mutex);
. . .
/* reading is performed */
```

```
. . .
wait(mutex);
read count--;
if (read count == 0)
signal(rw mutex);
signal(mutex);
} while (true);
```

4. Sleeping Barber Problem:

- Barber shop with one barber, one barber chair and N chairs to wait in.
- When no customers the barber goes to sleep in barber chair and must be woken when a customer comes in.
    - When barber is cutting hair new customers take empty seats to wait, or leave if no vacancy.

14. Explain in detail about monitors. Explain the dining philosopher critical section problem solution using monitor.

Monitors

- A high-level abstraction that provides a convenient and effective mechanism for process synchronization. Refer 2.17.
- Only one process may be active within the monitor at a time.

```
monitor monitor-name
{
// shared variable declarations
procedure body P1 (…) { …. }
…
procedure body Pn (…) {……}
{
initialization code
}
}
```

- To allow a process to wait within the monitor, a condition variable must be declared as condition x, y;
- Two operations on a condition variable:
    - x.wait () –a process that invokes the operation is suspended.

- x.signal () –resumes one of the suspended processes (if any)

Schematic view of a monitor

- Refer figure 2.16 for the schematic view of a monitor.

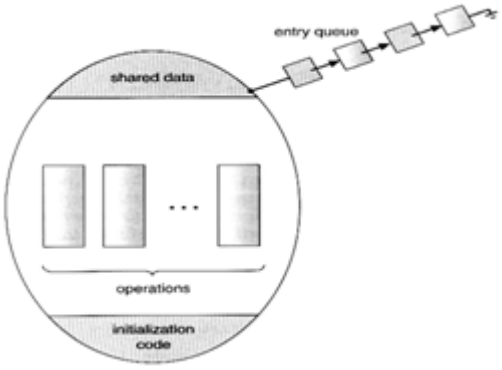


Fig.2.16 schematic view of a monitor

Monitor with condition variables

- Refer figure 2.16 for the monitor with condition variables.

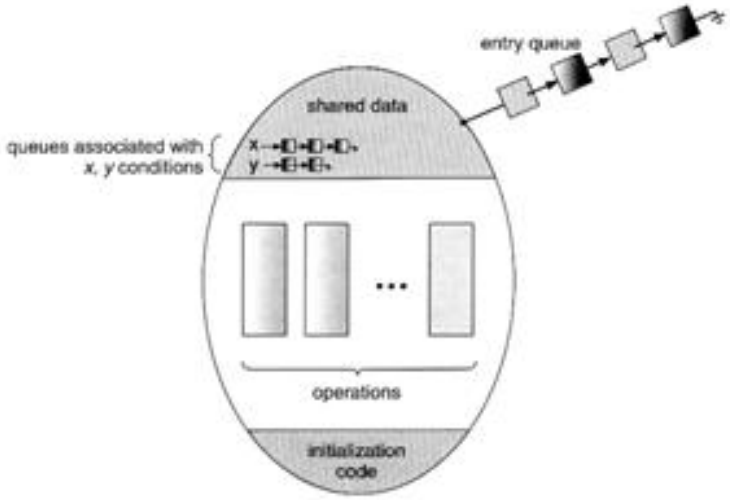


Fig.2.17 monitor with condition variables

Solution to Dining Philosophers Problem using monitor

```
monitor DP
{
enum { THINKING; HUNGRY, EATING) state [5] ;
condition self [5];
void pickup (int i) {
state[i] = HUNGRY;
test(i);
```

```
if (state[i]!= EATING) self [i].wait;
}
void putdown (int i) {
state[i] = THINKING;
// test left and right neighbors
test((i + 4) % 5);
test((i + 1) % 5);
}
void test (int i) {
if ( (state[(i + 4) % 5] != EATING) &&
(state[i] == HUNGRY) &&
(state[(i + 1) % 5] != EATING) ) {
state[i] = EATING;
self[i].signal () ;
}
}
initialization_code() {
for (int i = 0; i < 5; i++)
state[i] = THINKING;
}
}
```

15. Explain in detail about CPU scheduling algorithms with suitable examples.

Basic Concepts

- In a single-processor system, only one process can run at a time.
- The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization.
- A process is executed until it must wait, for the completion of some I/O request.
- Several processes are kept in memory at one time.
- When one process has to wait, the operating system takes the CPU away from that process and gives the CPU to another process.

- **CPU–I/O Burst Cycle**
  - Process execution consists of a cycle of CPU execution and I/O wait.
  - Processes alternate between these two states. Process execution begins with a CPU burst as shown in figure 2.18.

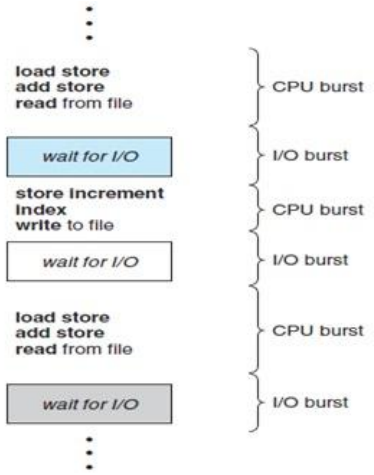


Fig. 2.18. Alternative sequence of CPU and I/O burst

- **CPU Scheduler**

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

- **Preemptive Scheduling**

CPU-scheduling decisions may take place under the following four circumstances:

- When a process switches from the running state to the waiting state (for example, as the result of an I/O request or an invocation of wait() for the termination of a child process)
- When a process switches from the running state to the ready state (for example, when an interrupt occurs)
- When a process switches from the waiting state to the ready state (for example, at completion of I/O)
- When a process terminates

- When scheduling takes place only under circumstances 1 and 4, we say that the scheduling scheme is **Non preemptive** or **cooperative**. Otherwise, it is **preemptive**.
- **Non preemptive scheduling**, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.
- **Preemptive scheduling** can result in race conditions when data are shared among several processes.

Scheduling Criteria

1. **CPU utilization**- We want to keep the CPU as busy as possible. Conceptually, CPU utilization can range from 0 to 100 percent.
2. **Throughput**- One measure of work is the number of processes that are completed per time unit, called throughput.
3. **Turnaround time**- The interval from the time of submission of a process to the time of completion is the turnaround time. Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O.
4. **Waiting time**- Waiting time is the sum of the periods spent waiting in the ready queue.
5. **Response time**- The measure is the time from the submission of a request until the first response is produced. This measure, called response time, is the time it takes to start responding, not the time it takes to output the response.

16. Explain Briefly CPU scheduling algorithms. Explain in detail FCFS, shortest job first, Priority and round robin (time slice =2) scheduling algorithms with Gantt chart.

There are many different CPU-scheduling algorithms.

- **First-Come, First-Served Scheduling**
  - With this scheme, the process that requests the CPU first is allocated the CPU first.
  - When a process enters the ready queue, its PCB is linked onto the tail of the queue.

- When the CPU is free, it is allocated to the process at the head of the queue.
- The running process is then removed from the queue.

Advantage: The code for FCFS scheduling is simple to write and understand.

Disadvantage: The average waiting time under the FCFS policy is often quite long.

Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds:

Process	Burst Time
*P*1	24
*P*2	3
*P*3	3

If the processes arrive in the order *P*1, *P*2, *P*3, and are served in FCFS order,

Gantt chart

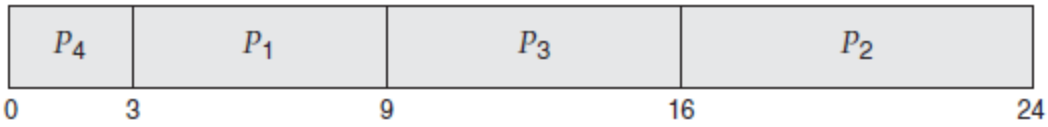
- The waiting time is 0 milliseconds for process *P*1, 24 milliseconds for process *P*2, and 27 milliseconds for process *P*3.
- Thus, the average waiting time is (0+ 24 + 27)/3 = 17 milliseconds.

- **Shortest-Job-First Scheduling**
  - This algorithm associates with each process the length of the process's next CPU burst.
  - When the CPU is available, it is assigned to the process that has the smallest next CPU burst.
  - If the next CPU bursts of two processes are the same, FCFS scheduling is used to break the tie.
  - ***It is shortest-next-CPU-burst*** algorithm, because scheduling depends on the length of the next

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

Process	Burst Time
*P*1	6
*P*2	8
*P*3	7
*P*4	3

Gantt chart:



The waiting time is 3 milliseconds for process *P*1, 16 milliseconds for process *P*2, 9 milliseconds for process *P*3, and 0 milliseconds for process *P*4.

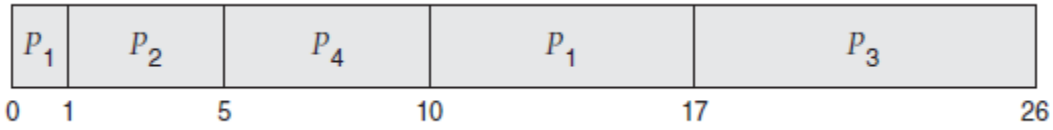
Thus, the average waiting time is (3 + 16 + 9 + 0)/4 = 7 milliseconds.

- With short-term scheduling, there is no way to know the length of the next CPU burst.
- A preemptive SJF algorithm will preempt the currently executing process, whereas a nonpreemptive SJF algorithm will allow the currently running process to finish its CPU burst.
- Preemptive SJF scheduling is sometimes called shortest-remaining-time-first scheduling.

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

Gantt chart:

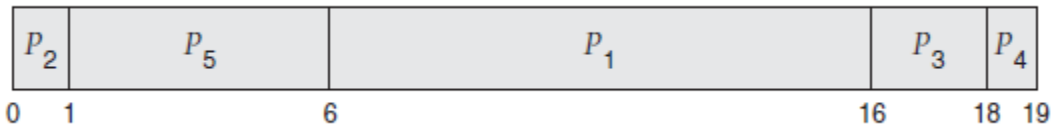


- **Priority Scheduling**
  - Apriority is associated with each process, and the CPU is allocated to the process with the highest priority.
  - Equal-priority processes are scheduled in FCFS order.
  - An SJF algorithm is simply a priority algorithm where the priority (*p*) is the inverse of the (predicted) next CPU burst.

  Consider the following set of processes, assumed to have arrived at time 0 in the order *P*1, *P*2, · · ·, *P*5, with the length of the CPU burst given in milliseconds:

Process	Burst Time	Priority
*P*1	10	3
*P*2	1	1
*P*3	2	4
*P*4	1	5
*P*5	5	2

Gantt chart:



The average waiting time is 8.2 milliseconds.

- Priority scheduling can be either preemptive or nonpreemptive.
- When a process arrives at the ready queue, its priority is compared with the priority of the currently running process.
- A preemptive priority scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the priority of the currently running process.
- A nonpreemptive priority scheduling algorithm will simply put the new process at the head of the ready queue.

- **Round-Robin Scheduling**
  - The **round-robin (RR)** scheduling algorithm is designed especially for time sharing systems.

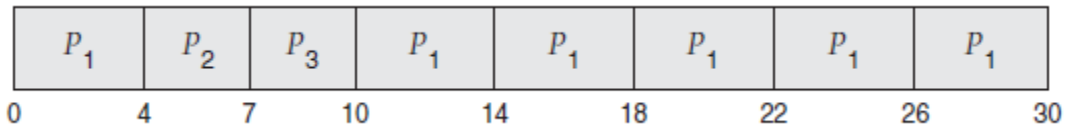
- It is similar to FCFS scheduling, but preemption is added to enable the system to switch between processes. A small unit of time, called a **time quantum** or **time slice**, is defined.
- A time quantum is generally from 10 to 100 milliseconds in length.
- The ready queue is treated as a circular queue.
- The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum.

Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds:

Process	Burst Time
*P*1	24
*P*2	3
*P*3	3

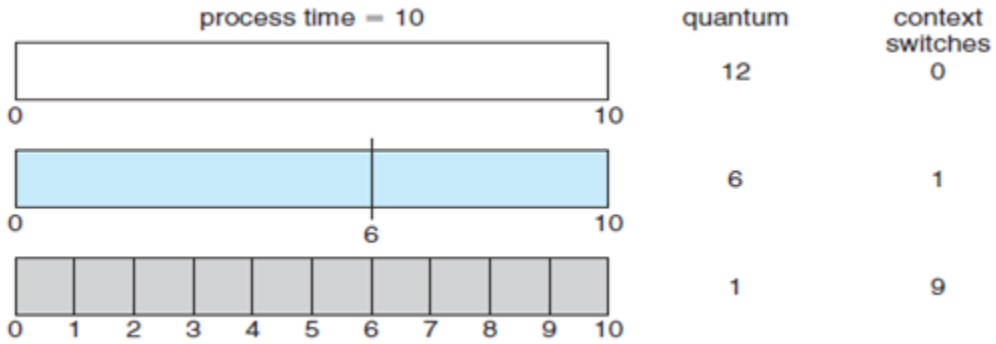
- If we use a time quantum of 4 milliseconds, then process *P*1 gets the first 4 milliseconds.

The resulting RR schedule is as follows:



Let's calculate the average waiting time for this schedule. *P*1 waits for 6 milliseconds (10 - 4), *P*2 waits for 4 milliseconds, and *P*3 waits for 7 milliseconds.

Thus, the average waiting time is 17/3 = 5.66 milliseconds.



- **Multilevel Queue Scheduling**
  - Scheduling algorithms has been created for situations in which processes are easily classified into different groups. For example, a

common division is made between **foreground** (interactive) processes and **background** (batch) processes.

- These two types of processes have different response-time requirements and so may have different scheduling needs. In addition, foreground processes may have priority (externally defined) over background processes.
- A **multilevel queue** scheduling algorithm partitions the ready queue into several separate queues as shown in figure 2.19.

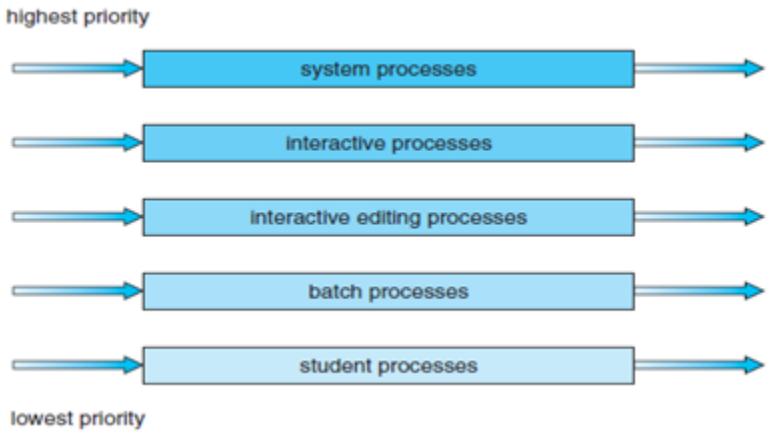


Fig.2.19 Multilevel Queue Scheduling

- Multilevel queue scheduling algorithm with five queues
  1. System processes
  2. Interactive processes
  3. Interactive editing processes
  4. Batch processes
  5. Student processes

6. Multilevel Feedback Queue Scheduling

- The **multilevel feedback queue** scheduling algorithm, allows a process to move between queues.
- If a process uses too much CPU time, it will be moved to a lower-priority queue.
- A process that waits too long in a lower-priority queue may be moved to a higher-priority queue as shown in figure 2.20.

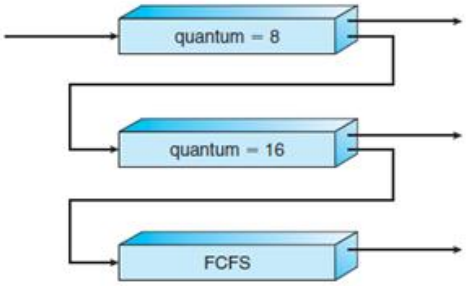


Fig.2.20 Multilevel feedback Queue

17. Explain in detail about multiple processor scheduling.

Multiple Processor Scheduling

- If multiple CPUs are available, the scheduling problem is correspondingly more complex.
- If several identical processors are available, then load-sharing can occur.
- It is possible to provide a separate queue for each processor.
- In this case however, one processor could be idle, with an empty queue, while another processor was very busy.
- To prevent this situation, we use a common ready queue.

1. Self Scheduling- Each processor is self-scheduling. Each processor examines the common ready queue and selects a process to execute. We must ensure that two processors do not choose the same process, and that processes are not lost from the queue.

2. Master – Slave Structure - This avoids the problem by appointing one processor as scheduler for the other processors, thus creating a master-slave structure.

18. Explain in detail about real-time scheduling.

Real-Time Scheduling

- Real-time computing is divided into two types.
  - Hard real-time systems
  - Soft real-time systems
- **Hard RTS** are required to complete a critical task within a guaranteed amount of time.

- Generally, a process is submitted along with a statement of the amount of time in which it needs to complete or perform I/O.
- The scheduler then either admits the process, guaranteeing that the process will complete on time, or rejects the request as impossible. This is known as resource reservation.
- **Soft real-time computing** is less restrictive. It requires that critical processes receive priority over less fortunate ones.
- The system must have priority scheduling, and real-time processes must have the highest priority.
- The priority of real-time processes must not degrade over time, even though the priority of non-real-time processes may.
- Dispatch latency must be small. The smaller the latency, the faster a real-time process can start executing.
- The high-priority process would be waiting for a lower-priority one to finish. This situation is known as priority inversion.

19. What is a deadlock? What are the necessary conditions for a deadlock to occur? (or) Explain deadlock detection with examples. (NOV/DEC 2024)

Deadlock Definition

- A process requests resources.
- If the resources are not available at that time, the process enters a wait state.
- Waiting processes may never change state again because the resources they have requested are held by other waiting processes. This situation is called a **deadlock.**

Resources

- Request: If the request cannot be granted immediately then the requesting process must wait until it can acquire the resource.
- Use: The process can operate on the resource
- Release: The process releases the resource.

Four Necessary conditions for a deadlock

Mutual exclusion:

- At least one resource must be held in a non sharable mode.

- That is only one process at a time can use the resource.
- If another process requests that resource, the requesting process must be delayed until the resource has been released.

Hold and wait:

- A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.

No preemption:

- Resources cannot be preempted.

Circular wait:

- P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P2...Pn-1.

20. Explain in detail about deadlock characterization.

Resource-Allocation Graph

- It is a Directed Graph with a set of vertices V and set of edges E.
- V is partitioned into two types:
- Nodes P = {p1, p2,..pn}
- Resource type R ={R1,R2,...Rm}
- Pi -->Rj - request => request edge
- Rj-->Pi - allocated => assignment edge.
- Pi is denoted as a circle and Rj as a square.
- Rj may have more than one instance represented as a dot with in the square.

  P = { P1,P2,P3}

  R = {R1,R2,R3,R4}

  E= {P1->R1, P2->R3, R1->P2, R2->P1, R3->P3 }

Resource instances

- One instance of resource type R1, Two instance of resource type R2, One instance of resource type R3, Three instances of resource type R4 as shown in figure 2.21.

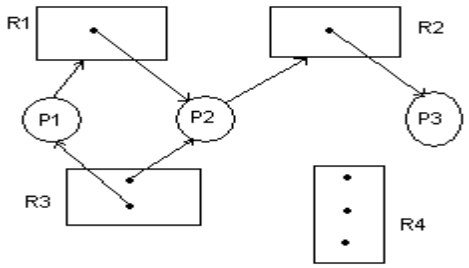


Fig.2.21 Resource Allocation graph

Process states

- Process P1 is holding an instance of resource type R2, and is waiting for an instance of resource type R1 as shown in figure 2.22.

Resource Allocation Graph with a deadlock

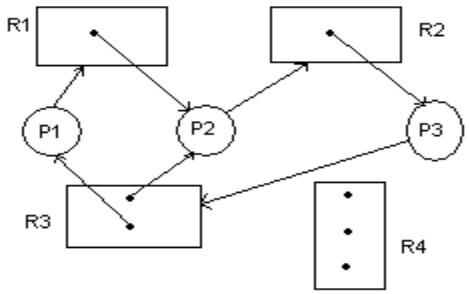


Fig.2.22 Resource allocation graph with deadlock

- Process P2 is holding an instance of R1 and R2 and is waiting for an instance of resource type R3. Process P3 is holding an instance of R3.
- P1->R1->P2->R3->P3->R2->P1
- P2->R3->P3->R2->P2

21. Explain about the methods used to prevent deadlocks

Deadlock Prevention:

- This ensures that the system never enters the deadlock state.
- Deadlock prevention is a set of methods for ensuring that at least one of the necessary conditions cannot hold.
- By ensuring that at least one of these conditions cannot hold, we can prevent the occurrence of a deadlock.

1. Denying Mutual exclusion

- Mutual exclusion condition must hold for non-sharable resources.
- Printer cannot be shared simultaneously shared by prevent processes.
- Sharable resource - example Read-only files.
- If several processes attempt to open a read-only file at the same time, they can be granted simultaneous access to the file.
- A process never needs to wait for a sharable resource.

2. Denying Hold and wait

- Whenever a process requests a resource, it does not hold any other resource.
- One technique that can be used requires each process to request and be allocated all its resources before it begins execution.
- Another technique is before it can request any additional resources, it must release all the resources that it is currently allocated.
- These techniques have two main disadvantages:
    - First, resource utilization may be low, since many of the resources may be allocated but unused for a long time.
    - We must request all resources at the beginning for both protocols.

3. Denying No preemption

- If a Process is holding some resources and requests another resource that cannot be immediately allocated to it. (i.e. the process must wait), then all resources currently being held are preempted.
- These resources are implicitly released.
- The process will be restarted only when it can regain its old resources.

4. Denying Circular wait

- Impose a total ordering of all resource types and allow each process to request for resources in an increasing order of enumeration.
- Let R = {R1,R2,...Rm} be the set of resource types.
- Assign to each resource type a unique integer number.
- If the set of resource types R includes tape drives, disk drives and printers.

F(tapedrive)=1,

F(diskdrive)=5,

F(Printer)=12.

- Each process can request resources only in an increasing order of enumeration.

22. Explain in detail about Banker's deadlock avoidance algorithm with an illustration.

Deadlock Avoidance:

- Deadlock avoidance request that the OS be given in advance additional information concerning which resources a process will request and use during its life time.
- To decide whether the current request can be satisfied or must be delayed, a system must consider the resources currently available, the resources currently allocated to each process and future requests and releases of each process.

Safe State

A state is safe if the system can allocate resources to each process in some order and still avoid a dead lock as shown in figure 2.23.

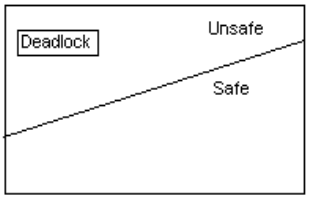


Fig.2.23 Deadlock Safe/Unsafe

- A deadlock is an unsafe state.
- Not all unsafe states are dead locks
- An unsafe state may lead to a dead lock

- Two algorithms are used for deadlock avoidance namely;

  1. Resource Allocation Graph Algorithm - single instance of a resource type.
  2. Banker's Algorithm – several instances of a resource type.

Resource allocation graph algorithm

- Claim edge - Claim edge Pi--->Rj indicates that process Pi may request resource Rj at some time, represented by a dashed directed edge.

- When process Pi request resource Rj, the claim edge Pi ->Rj is converted to a request edge.
- Similarly, when a resource Rj is released by Pi the assignment edge Rj -> Pi is reconverted to a claim edge Pi ->Rj
- The request can be granted only if converting the request edge Pi ->Rj to an assignment edge Rj -> Pi does not form a cycle.
- If no cycle exists, then the allocation of the resource will leave the system in a safe state.
- If a cycle is found, then the allocation will put the system in an unsafe state.

Banker's algorithm

- Safety Algorithm
- Resource request algorithm

Data structures used for bankers algorithm

- **Available:** indicates the number of available resources of each type.
- **Max:** Max[i, j]=k then process Pi may request at most k instances of resource type Rj
- **Allocation:** Allocation[i. j]=k, then process Pi is currently allocated K instances of resource type Rj
- **Need:** if Need[i, j]=k then process Pi may need K more instances of resource type Rj

  Need [i, j]=Max[i, j]-Allocation[i, j]

1. Safety algorithm

1. Initialize work := available and Finish [i]:=false for i=1,2,3 .. n
2. Find an i such that both

   Finish[i]=false

   a. Needi<= Work

      i. if no such i exists, goto step 4
3. work :=work+ allocationi;

   a. Finish[i]:=true

   goto step 2
4. If finish[i]=true for all i, then the system is in a safe state

2. Resource Request Algorithm

Let Requesti be the request from process Pi for resources.

1. If Requesti<= Needi goto step2, otherwise raise an error condition, since the process has exceeded its maximum claim.
2. If Requesti<= Available, goto step3, otherwise Pi must wait, since the resources are not available.
3. Available := Availabe-Requesti;
   Allocationi :=Allocationi + Requesti
   Needi :=Needi - Requesti;

- Now apply the safety algorithm to check whether this new state is safe or not. If it is safe then the request from process Pi can be granted.

23. Explain the two solutions of recovery from deadlock.

Deadlock Recovery

1. Process Termination

1. Abort all deadlocked processes.

2. Abort one deadlocked process at a time until the deadlock cycle is eliminated.

- After each process is aborted, a deadlock detection algorithm must be invoked to determine where any process is still dead locked.

2. Resource Preemption

- Preemptive some resources from process and give these resources to other processes until the deadlock cycle is broken.

Factors considered for resource preemption

- **Selecting a victim:** which resources and which process are to be preempted.
- **Rollback:** if we preempt a resource from a process it cannot continue with its normal execution. It is missing some needed resource. we must rollback the process to some safe state, and restart it from that state.
- **Starvation:** How can we guarantee that resources will not always be preempted from the same process?

24. How can deadlock be detected?

Two methods to detect a deadlock

- Single instance of resource type
- Several instances of a resource type

Single Instance of Each Resource Type

- If all resources have only a single instance, then we can define a deadlock detection algorithm that use a variant of resource-allocation graph called a **wait for graph** as shown in figure 2.24.

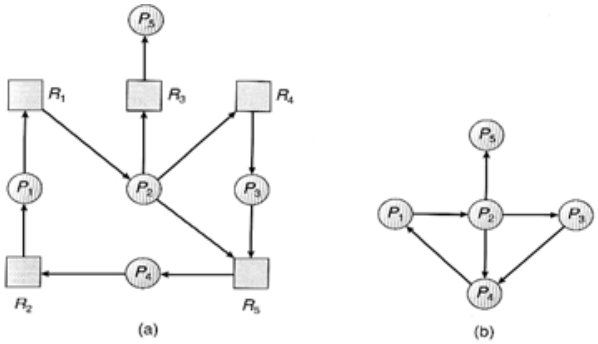


Fig.2.24 a) Resource Allocation graph b) wait for graph

Several Instances of a Resource Type

Available: Number of available resources of each type

Allocation: number of resources of each type currently allocated to each process

Request: Current request of each process

If Request [i,j]=k, then process Pi is requesting K more instances of resource type Rj.

1. Initialize work := available

   Finish[i]=false, otherwise finish [i]:=true

2. Find an index i such that both

   a. Finish[i]=false

   b. Requesti<=work

   if no such i exists go to step4.

3. Work:=work+allocationi

   Finish[i]:=true

   goto step2

- If finish[i]=false then process Pi is deadlocked

25. Consider the table given below for a system, find the need matrix and the safety sequence, is the request from process P1(0, 1, 2) can be granted immediately.

Resource – 3 types

A – (10 instances)

B – (5 instances)

C – (7 instances)

Solution: Banker's Algorithm

Step 1:

Safety for process P_0

$need_0$ = (7, 4, 3)

If $need_0 \leq$ Available

if [(7, 4, 3) ≤ (3, 3, 2)] (false)

Process P_0 must wait.

Step 2:

Safety for process P_0

$need_1$ = (1, 2, 2)

if$need_i \leq$ Available

if [(1, 2, 2) ≤ (3, 3, 2)]

P_i will execute.

Available = Available + Allocation

= (3, 3, 2) + (2, 0, 0)

= (5, 3, 2)

Step 3:

Safety for process P_2

$need_2$ = (6, 0, 0)

if $need_2 \leq$ Available

if [(6, 0, 0) ≤(5, 3, 2)] (false)

P_3 will execute.

Available = Available + Allocation

= (5, 3, 2) + (2, 1, 1)

= (7, 4, 3)

Step 5:

Safety for process P_4

$need_4$ = (4, 3, 1)

If $need_4 \leq$ Available

If [(4, 3, 1) ≤ (6, 4, 3)]

P_4 will execute.

Available = Available + Allocation

= (7, 4, 3) + (0, 0, 2)

= (7, 4, 5)

Step 6:

Safety for process P_0

$need_0$ = (7, 4, 3)

if $need_0 \leq$ Available

if [(7, 4, 3) ≤ (7, 4, 5)]

P_0 will execute.

Available = Available + Allocation

= (7, 4, 5) + (0, 1, 0)

= (7, 5, 5)

Step 7:

Safety for process P_2

$need_2$ = (6, 0, 0)

if $need_2 \leq$ Available

if [(6, 0, 0) ≤ (7, 5, 5)]

P_2 will execute.

Available = Available + Allocation

= (7, 5, 5) + (3, 0, 2)

= (10, 5, 7)

Safety Sequence = <P1, P3, P4, P0, P2>

26. Compare and contrast preemptive and non-preemptive scheduling.

- In preemptive scheduling, the CPU is allocated to the processes for a limited time whereas, in Non-preemptive scheduling, the CPU is allocated to the process till it terminates or switches to the waiting state.
- The executing process in preemptive scheduling is interrupted in the middle of execution when higher priority one comes whereas, the executing process in non-preemptive scheduling is not interrupted in the middle of execution and waits till its execution.
- In Preemptive Scheduling, there is the overhead of switching the process from the ready state to running state, vise-verse and maintaining the ready queue. Whereas in the case of non-preemptive scheduling has no overhead of switching the process from running state to ready state.
- In preemptive scheduling, if a high-priority process frequently arrives in the ready queue then the process with low priority has to wait for a long, and it may have to starve. , in the non-preemptive scheduling, if CPU is allocated to the process having a larger burst time then the processes with small burst time may have to starve.
- Preemptive scheduling attains flexibility by allowing the critical processes to access the CPU as they arrive into the ready queue, no matter what process is executing currently. Non-preemptive scheduling is called rigid as even if a critical process enters the ready queue the process running CPU is not disturbed.
- Preemptive Scheduling has to maintain the integrity of shared data that's why it is cost associative which is not the case with Non-preemptive Scheduling.

27. Describe why the interrupts are not appropriate for implementing synchronous preemptive in multiprocessing systems. (8) (NOV/DEC 2024)

In a multiprocessing system, synchronous preemption refers to preempting a process or thread in a controlled and predictable manner to ensure fair CPU scheduling, resource allocation, and responsiveness. While interrupts are commonly used in OS scheduling, they are not ideal for implementing synchronous preemption in multiprocessing systems due to several reasons:

1. Interrupts Are Asynchronous by Nature

- Interrupts occur unpredictably based on hardware events (e.g., I/O completion, timer expiration, or external signals).
- Synchronous preemption, on the other hand, requires predictable and controlled preemption at specific execution points (e.g., after a quantum expires or at well-defined system calls).
- Since interrupts can occur at any time, they may disrupt critical operations and lead to race conditions or inconsistent states.

2. Interrupt Handling Overhead in Multiprocessing

- In a multiprocessor system, interrupts must be handled carefully to avoid conflicts.
- If multiple CPUs receive interrupts simultaneously, coordination and context switching overhead increase.
- Excessive interrupt handling degrades system performance due to frequent context switches, cache invalidations, and inter-processor communication (IPI) overhead.

3. Race Conditions and Synchronization Issues

- When multiple processors handle interrupts, race conditions may occur if interrupt handlers modify shared data structures (e.g., process queues, resource tables).
- Without proper locking mechanisms, processors may experience inconsistent states, leading to deadlocks or priority inversion.

4. Increased Latency and Unpredictability

- Interrupts introduce latency because the CPU must stop executing the current process, save its state, and switch to the interrupt handler.
- In real-time systems, strict timing guarantees are needed, and interrupts can introduce jitter, making timing unpredictable.
- Synchronous preemption, however, requires low-latency, predictable preemption that ensures fair CPU usage without unnecessary interruptions.

5. Alternative Approaches for Synchronous Preemption

Instead of relying on interrupts, multiprocessing systems use:

- Timer-based preemption: The OS sets periodic timers to trigger preemption at fixed intervals, ensuring fairness.
- Scheduler-controlled preemption: The OS checks process states at known synchronization points (e.g., system calls, kernel mode transitions).
- Polling mechanisms: Some real-time systems use polling instead of interrupts to ensure deterministic scheduling.

While interrupts are essential for handling asynchronous events, they are not ideal for synchronous preemption in multiprocessing due to unpredictability, race conditions, and overhead. Instead, timer-based and scheduler-driven preemption methods ensure fair and efficient process scheduling in a controlled manner.

28. Summarize the difference between user thread and kernel thread. (NOV/DEC 2024)

Feature	**User Thread**	**Kernel Thread**
Definition	Managed by user-level libraries, without direct OS involvement.	Managed and scheduled by the operating system kernel.
Performance	Faster, as thread management (creation, switching) is done in user space without kernel intervention.	Slower, due to kernel involvement in scheduling and context switching.
Scheduling	Handled by the user-level thread library; the OS is unaware of user threads.	Handled by the OS scheduler, ensuring system-wide fairness.
Blocking	If one thread blocks (e.g., on I/O), all threads in that process may block.	If a thread blocks, other threads in the same process can still execute.
Portability	More portable, as they do not depend on the OS kernel.	Less portable, as they rely on specific OS implementations.
Multiprocessing Support	Cannot take full	Can run on multiple CPUs

	advantage of multiple CPUs, as the OS sees only a single process.	simultaneously, improving parallelism.
Example APIs	POSIX Pthreads (user-level), Java threads	Windows threads, Linux kernel threads (k thread)

29. Consider the following snapshot: (NOV/DEC 2024)

Process	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	0	1	2	0	0	1	2	1	5	2	0
P_1	1	0	0	0	1	7	5	0				
P_2	1	3	5	4	2	3	5	6				
P_3	0	6	3	2	0	6	5	2				
P_4	0	0	1	4	0	6	5	6				

Answer the following questions using the Banker's algorithm

(1) What is the content of the matrix needed? (2)

(2) Is the system in a safe state? Justify your answer. (2)

(3) If a request from process P1 arrives for (0, 4, 2, 0), can the request be granted immediately? (6)

Step 1…Calculate the Need matrix.

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

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

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

Check if the system is in a safe state.

Step 3 …Check if the request from process P1can be granted immediately.

The request from $P1$ is (0,4,2,0). The available resources are (3,3,2).

. The request cannot be granted immediately because there are not enough resources available.

Solution

The Need matrix is $\begin{bmatrix} 7 & 3 & 3 \\ -1 & 2 & 2 \\ 3 & 0 & -2 \\ -2 & 0 & 0 \\ 4 & 3 & -1 \end{bmatrix}$. The system is in a safe state. The request from process P_1 cannot be granted immediately.

30. Explain the methods how deadlock can be avoided? Assume that there are three resources, A,B, and C. There are 4 processes P0 to P3. At T0, the state of the system is given below.

	Allocation			Maximum			Need		
	A	B	C	A	B	C	A	B	C
P0	1	0	1	2	1	1	2	1	1
P1	2	1	2	5	4	4			
P2	3	0	0	3	1	1			
P3	1	0	1	1	1	1			

(i) Create a need matrix. (7)

(ii) Use Banker's algorithm to check if the system in a safe state? Why or why not? (8)

1. Create the Need Matrix:

- The "Need" matrix represents the maximum amount of each resource a process still needs to complete its execution.

Process	Need (A)	Need (B)	Need (C)
P0	1	1	1
P1	2	3	2
P2	0	1	1
P3	1	0	0

How to calculate Need Matrix:

- For each process, subtract the allocated resources from the maximum needed resources:
  - Need(A) = Maximum(A) - Allocation(A)
  - Need(B) = Maximum(B) - Allocation(B)
  - Need(C) = Maximum(C) - Allocation(C)

2. Apply Banker's Algorithm:

- **Available Resources:** Let's assume the available resources at T0 are A=1, B=1, C=1.
- **Check for Safe State:**
  - **Step 1:** Find a process that can be completely satisfied with the available resources.
    - P2 can be satisfied with its need (A=0, B=1, C=1) as it is fully covered by the available resources.
    - Update available resources: A=1, B=0, C=0.
  - **Step 2:**
    - P3 can be satisfied with its need (A=1, B=0, C=0) using the updated available resources.
    - Update available resources: A=0, B=0, C=0.
  - **Step 3:**
    - P0 can be satisfied with its need (A=1, B=1, C=1) using the updated available resources.
    - Update available resources: A=0, B=0, C=0.
  - **Step 4:**
    - P1 cannot be satisfied with its need (A=2, B=3, C=2) as there are no available resources left.
- **Conclusion:** The system is in a safe state because a sequence of processes (P2, P3, P0) can be executed without causing a deadlock, even if each process requests its maximum need.

Deadlock Avoidance with Banker's Algorithm:

- **Before allocating a resource to a process, check if the system would remain in a safe state after allocation.**
- If allocating a resource would result in an unsafe state, deny the request to prevent potential deadlock.

31. Consider the execution of two processes P1 and P2 with the following CPU and I/O burst times.

P1	P2
CPU-3	CPU-4
Net-4	Disk-3
CPU-2	CPU-3
Disk-3	Net-3

Each row shows the required resource for the process and the time that the process needs that resource. For example "Net 3" in fourth row says that P2 needs network card for 3 time units.

(i) If P2 arrives 2 time units after P1 and the scheduling policy is non-preemptive SJF then calculate the finish time for each process and the CPU idle time in that duration. (7)

(ii) If P2 arrives 2 time units before P1 and the scheduling policy is preemptive SJF then calculate the finish time for each process and the CPU idle time in that duration. (8) (April/May 2024)

(i) Non-preemptive SJF with P2 arriving 2 time units after P1:

- Process execution order: P1 (CPU 3) -> P1 (Net 4) -> P1 (Disk 3) -> P1 (CPU 2) -> P2 (CPU 4) -> P2 (Net 4) -> P2 (Disk 3) -> P2 (CPU 3) -> P2 (Net 3)
- Finish times:
  - P1: 12 time units (3 CPU + 4 Net + 3 Disk + 2 CPU)
  - P2: 18 time units (from the time P2 arrives at time unit 2: 4 CPU + 4 Net + 3 Disk + 3 CPU + 3 Net)
- CPU idle time: 2 time units (between when P1 finishes its first CPU burst and P2 arrives)

(ii) Preemptive SJF with P2 arriving 2 time units before P1:

- Process execution order:
  P2 (CPU 4) -> P1 (CPU 3) -> P2 (Net 4) -> P1 (Net 4) -> P2 (Disk 3) -> P1 (Disk 3) -> P2 (CPU 3) -> P1 (CPU 2) -> P2 (Net 3)
- Finish times:
  - P1: 13 time units (3 CPU + 4 Net + 3 Disk + 2 CPU)

- P2: 16 time units (4 CPU + 4 Net + 3 Disk + 3 CPU + 3 Net)

- **CPU idle time:**
  0 time units (since preemptive scheduling allows for immediate switching between processes when a shorter burst time becomes available).
  Explanation:
- **Non-preemptive SJF:**
  - P1 starts first as it has the shortest initial CPU burst.
  - P2 arrives after 2 time units and is added to the queue.
  - Since P1 has a shorter next burst (Net 4) than P2's CPU burst, P1 continues execution until it finishes its entire sequence.
- **Preemptive SJF:**
  - P2 starts first as it arrives earlier and has the shortest initial CPU burst.
  - Whenever a new process with a shorter remaining burst time arrives, the currently executing process is preempted and the new process takes over the CPU.
- In non-preemptive scheduling, a process will run completely before another process can start executing even if a shorter burst arrives later.
- Preemptive scheduling allows for more efficient CPU utilization by switching to a shorter burst whenever available.

32. Consider the following resource-allocation policy. Requests and releases for resources are allowed at any time. If a request for resources cannot be satisfied because the resources are not available, then we check any processes that are blocked, waiting for resources. If they have the desired resources, then these resources are taken away from them and are given to the requesting process. The vector of resources for which the waiting process is waiting is increased to include the resources that were taken away. For example, consider a system with three resource types and the vector Available initialized to (4,2,2). If process Po asks for (2,2,1) it gets them. If P1 asks for (1,0,1), it gets them. Then, if Po asks for (0,0,1), it is blocked (resource not available). If P2 now asks for (2,0,0), it gets the available one (1,0,0) and one that was allocated to Po (since Po is blocked). Po's Allocation vector goes down to (1, 2, 1), and its Need vector goes up to (1, 0, 1). Answer the followings:

(i) Predict whether deadlock occurs or not. If it occurs, give an example. If not, which necessary condition cannot occur?

(ii) Predict whether indefinite blocking occurs or not. (Nov/Dec 2024)

Example scenario to illustrate indefinite blocking:

- Consider three processes P0, P1, and P2 with resource needs:
  - P0 needs (1, 0, 0)
  - P1 needs (0, 1, 0)
  - P2 needs (0, 0, 1)

- If P0 is currently allocated (1, 0, 0) and P1 requests (0, 1, 0), P1 will be granted the resource immediately.
- Now, if P2 requests (0, 0, 1), it will be blocked because the resource is not available.
- If P0 then requests (0, 1, 0), it will be granted the resource that was previously held by P1 since P1 is no longer waiting for it.
- This scenario can repeat continuously, causing P2 to remain blocked indefinitely waiting for the resource currently held by either P0 or P1.

(i) Predict whether deadlock occurs or not. If it occurs, give an example. If not, which necessary condition cannot occur?

Deadlock Prediction

- Deadlock occurs when the system enters a state where a set of processes are waiting for resources held by each other in a circular dependency, with no process able to proceed.
- To analyze this, let's check the four necessary conditions for deadlock:
  1. Mutual Exclusion: Resources are allocated to one process at a time—this condition holds.
  2. Hold and Wait: Processes may hold some resources while requesting more—this condition holds.
  3. No Preemption: The system policy allows preemption, meaning resources can be taken away from a blocked process and given to another process. This violates the no preemption condition, preventing deadlock from occurring.
  4. Circular Wait: Deadlock occurs when a circular chain of processes exists, where each process is waiting for a resource held by the next in the chain. However, since resources can be taken away, the chain is broken before deadlock can occur.

Hence Deadlock does not occur because the no preemption condition is violated.

(ii) Predict whether indefinite blocking occurs or not.

- Indefinite blocking (starvation) occurs when a process waits indefinitely because resources are continuously taken away and allocated to other processes.
- In this system, a blocked process can **lose its allocated resources** to another process. If a process repeatedly gets resources taken away before it can complete execution, it may remain in a **perpetual wait state**, leading to starvation.
- **Conclusion:** Indefinite blocking **can occur** because a process that keeps getting its resources preempted may never finish execution.

UNIT III MEMORY MANAGEMENT

Main Memory - Swapping - Contiguous Memory Allocation – Paging - Structure of the Page Table - Segmentation, Segmentation with paging; Virtual Memory - Demand Paging – Copy on Write - Page Replacement - Allocation of Frames –Thrashing.

PART-A

1. Define logical address and physical address.

An address generated by the CPU is referred as logical address. An address seen by the memory unit that is the one loaded into the memory address register of the memory is commonly referred to as physical address.

2. What is logical address space and physical address space?

The set of all logical addresses generated by a program is called a logical address space; the set of all physical addresses corresponding to these logical addresses is a physical address space.

3. What is the main function of the memory-management unit?

The runtime mapping from virtual to physical addresses is done by a hardware device called a memory management unit (MMU).

4. Define dynamic loading.

To obtain better memory-space utilization dynamic loading is used. With dynamic loading, a routine is not loaded until it is called. All routines are kept on disk in a relocatable load format. The main program is loaded into memory and executed. If the routine needs another routine, the calling routine checks whether the routine has been loaded. If not, the relocatable linking loader is called to load the desired program into memory.

5. What are overlays? [Nov/Dec2012]

To enable a process to be larger than the amount of memory allocated to it, overlays are used. The idea of overlays is to keep in memory only those instructions and data that are needed at a given time**.** When other instructions are needed, they are loaded into space occupied previously by instructions that are no longer needed.

6. Define swapping. [April/May-2021]

A process needs to be in memory to be executed. However, a process can be swapped temporarily out of memory to a backing store and then brought back into memory for continued execution. This process is called swapping.

7. How is memory protected in a paged environment?

Protection bits that are associated with each frame accomplish memory protection in a paged environment. The protection bits can be checked to verify that no writes are being made to a read-only page.

8. What do you mean by compaction?

Compaction is a solution to external fragmentation. The memory contents are shuffled to place all free memory together in one large block. It is possible only if relocation is dynamic, and is done at execution time.

9. What are pages and frames?

Paging is a memory management scheme that permits the physical-address space of a process to be non contiguous. In the case of paging, physical memory is broken into fixed-sized blocks called frames and logical memory is broken into blocks of the same size called pages.

10. What is the use of valid-invalid bits in paging?

When the bit is set to valid, this value indicates that the associated page is in the process's logical address space, and is thus a legal page. If the bit is said to invalid, this value indicates that the page is not in the process's logical address space. Using the valid-invalid bit traps illegal addresses.

11. What is the basic method of segmentation?

Segmentation is a memory management scheme that supports the user view of memory. A logical address space is a collection of segments. The logical address consists of segment number and offset. If the offset is legal, it is added to the segment base to produce the address in physical memory of the desired byte.

The logical-address space is a collection of segments. The process of mapping the logical address space to the physical address space using a segment table is known as segmentation.

12. What is virtual memory?

Virtual memory is a technique that allows the execution of processes that may not be completely in memory. It is the separation of user logical memory from physical memory. This separation provides an extremely large virtual memory, when only a smaller physical memory is available.

13. What is Demand paging? Write its advantages. [Nov/Dec-2021]

Virtual memory is commonly implemented by demand paging. In demand paging, the pager brings only those necessary pages into memory instead of swapping in a whole process. Thus, it avoids reading into memory pages that will not be used anyway, decreasing the swap time and the amount of physical memory needed.

Advantages of demand paging.

- Only loads pages that are demanded by the executing process.
- As there is more space in main memory, more processes can be loaded reducing context switching time which utilizes large amounts of resources.
- Less loading latency occurs at program startup, as less information is accessed from secondary storage and less information is brought into main memory.

14. Define lazy swapper.

Rather than swapping the entire process into main memory, a lazy swapper is used. A lazy swapper never swaps a page into memory unless that page will be needed.

15. What is a pure demand paging?

When starting execution of a process with no pages in memory, the operating system sets the instruction pointer to the first instruction of the process, which is on a non- memory resident page, the process immediately faults for the page. After this page is brought

into memory, the process continues to execute, faulting as necessary until every page that it needs is in memory. At that point, it can execute with no more faults. This schema is pure demand paging.

16. Define effective access time.

Let p be the probability of a page fault (0<=p<=1). The value of p is expected to be close to 0; that is, there will be only a few page faults. The effective access time is

Effective access time = (1-p) * ma + p * page fault time. (ma: memory-access time).

17. Define secondary memory.

This memory holds those pages that are not present in main memory. The secondary memory is usually a high-speed disk. It is known as the swap device, and the section of the disk used for this purpose is known as swap space.

18. What is the various page replacement algorithms used for page replacement?

- FIFO page replacement
- Optimal page replacement
- LRU page replacement
- LRU approximation page replacement
- Counting based page replacement
- Page buffering algorithm

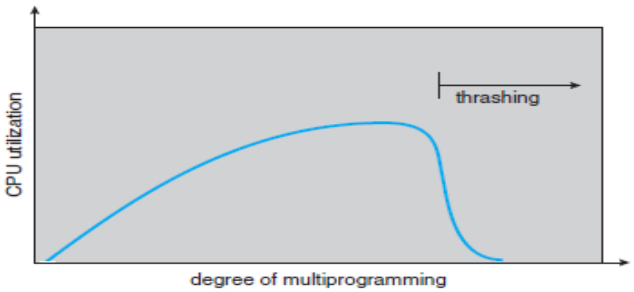
19. What are the major problems to implement demand paging? (Nov/Dec 2015)

The two major problems to implement demand paging is developing

- Frame allocation algorithm and Page replacement algorithm

20. Define Thrashing and how to limit the effect of thrashing? (Apr/May 2015) (April/May-2019)

The page is brought in and taken out of the memory and is not allowed to execute. This technique is known as thrashing**.** We can limit the effects of thrashing by using a local replacement algorithm.



To prevent thrashing, we must provide a process as many frames as it needs. The working-set strategy starts by looking at how many frames a process is actually using. This approach defines the **locality model** of process execution.

21. Define roll out and roll in.

If a higher-priority process arrived and wants service, the memory manager can swap out the lower-priority process so that it can load and execute the higher-priority process.

When the higher-priority process finishes, the lower-priority process can be swapped back in and continued. This variant of swapping is sometimes called roll out, Rollin.

22. Write notes on contiguous storage allocation.

The memory is usually divided into two partitions: one for resident operating system, and one for the user processes. The operating system may be placed in either low or high memory.

The major factor affecting this decision is the location of interrupt vector. Since the interrupt vector is often in low memory, programmers usually place the operating system in low memory as well. In this contiguous memory allocation, each process is contained in a single contiguous section of memory.

23. What is the difference between Internal and External Fragmentation? (APRIL/MAY 2024)

Internal Fragmentation	External fragmentation
Internal Fragmentation occurs when a fixed size memory allocation technique is used	External fragmentation occurs when a dynamic memory allocation technique is used
Internal fragmentation occurs when a fixed size partition is assigned to a program/file with less size than the partition making the rest of the space in that partition unusable	External fragmentation is due to the lack of enough adjacent space after loading and unloading of programs or files for some time because then all free space is distributed here and there
Internal fragmentation can be maimed by having partitions of several sizes and assigning a program based on the best fit. However, still internal fragmentation is not fully eliminated	External fragmentation can be prevented by mechanisms such as segmentation and paging.
When the allocated memory may be slightly larger than the requested memory, the difference between these two numbers is internal fragmentation.	It exists when enough total memory space exists to satisfy a request, but it is not contiguous; storage is fragmented into a large number of small holes.

24. Define. Translation look–aside buffer.

The TLB is associative, high-speed memory. Each entry in the TLB consists of two parts: a key (or tag) and a value. When the associative memory presented with an item, it is compared with all keys simultaneously. If the item is found, the corresponding value field is returned. The search is fast; the hardware is expensive. Typically, the number of entries in a TLB is small, often numbering between 64, and 1024.

25. What is a hit ratio?

The percentage of times that a particular page number is found in the TLB is called the hit ratio.

26. Define page replacement approach.

Page replacement uses the following approach:

a. Find the location of the desired page on the disk.

b. Find a free frame:

i. If there is a free frame, use it.

ii. If there is no free frame, use a page replacement algorithm to Select a victim frame.

iii. Write the victim page to the disk; change the page and frame tables accordingly.

c. Read the desired page into the (newly) free frame; change the page and frame tables.

d. Restart the user process**.**

27. Define Slab Allocation and its states.

A second strategy for allocating kernel memory is known as slab allocation. A slab is made up of one or more physically contiguous pages.

In Linux, a slab may be in one of three possible states:

1. Full. All objects in the slab are marked as used.

2. Empty. All objects in the slab are marked as free.

3. Partial. The slab consists of both used and free objects.

28. Name two differences between logical and physical addresses.(May/June 2016)(Nov/Dec-2019)

Logical Address	Physical Address
Logical address does not refer to an actual existing address; rather, it refers to an abstract address in an abstract address space.	Physical address that refers to an actual physical address in memory
A logical address is generated by the CPU and is translated in to a physical address by the memory management unit (MMU) (when address binding occurs at execution time)	Physical addresses are generated by the MMU.

29. How does the system detect thrashing? (May/June 2016)

Thrashing is caused by under allocation of the minimum number of pages required by a process, forcing it to continuously page fault. The system can detect thrashing by evaluating the level of CPU utilization as compared to the level of multiprogramming. It can be eliminated by reducing the level of multiprogramming.

30. What is thrashing? How to resolve it? (April/May-2019, 2021)

With a computer, **thrashing** or **disk thrashing** describes when a hard drive is being overworked by moving information between the system memory and virtual memory excessively. Thrashing occurs when the system does not have enough memory, the system swap file is not properly configured, too much is running at the same time, or has low system resources. When thrashing occurs, you will notice the computer hard drive always working, and a decrease in system performance. Thrashing is serious because of the amount of work the hard drive has to do, and if left unfixed can cause an early hard drive failure.

Ways to eliminate thrashing:

To resolve hard drive thrashing, you can do any of the suggestions below.

1. Increase the amount of RAM in the computer.
2. Decrease the number of programs being run on the computer.
3. Adjust the size of the swap file.

31.Under what circumstances do page faults occur? State the actions taken by the operating system when a page fault occurs. (Nov/Dec-2019)

A page fault occurs when an access to a page that has not been brought into main memory takes place. The operating system verifies the memory access, aborting the program if it is invalid. If it is valid, a free frame is located and I/O is requested to read the needed page into the free frame. Upon completion of I/O, the process table and page table are updated and the instruction is restarted.

32. When trashing is used? (Nov/Dec-2021)

Thrashing is a condition or a situation when the system is spending a major portion of its time in servicing the page faults, but the actual processing done is very negligible. The basic concept involved is that if a process is allocated too few frames, then there will be too many and too frequent page faults.

33. What is page fault? How is page fault frequency related to degree of multiprogramming of a computer? (APRIL/MAY 2024)

- A "page fault" occurs when a process tries to access a memory page that is not currently loaded in the main memory, causing the operating system to retrieve that page from secondary storage (like a hard disk) and load it into RAM before the process can continue accessing it;
- The frequency of page faults is directly related to the degree of multiprogramming because as more processes are running concurrently, the higher the likelihood of competing for limited RAM, leading to more page faults happening frequently.

34. What is External fragmentation? (NOV/DEC 2024)

- External fragmentation is a memory management issue in an operating system (OS) that occurs when memory is divided into small fragments that are difficult to use. This happens when memory allocation methods can't manage memory effectively.
- Both the first-fit and best-fit strategies for memory allocation suffer from external fragmentation.

35. Mention the steps to handle page fault in demand paging. (NOV/DEC 2024)

When a page fault occurs in demand paging, the operating system performs the following steps:

- Identify the page fault:
- Save the process state:
- Check the page table:
- Find a free frame:
- Read the page from disk:
- Update page table:
- Restart the instruction:

PART-B

1. **Explain about Swapping in detail. (Nov/Dec-19)**

A process must be in memory to be executed. A process, however, can be swapped temporarily out of memory to a backing store and then brought back into memory for continued execution.

Standard Swapping:

- Standard swapping involves **moving processes between main memory and a backing store**.

Backing store:

- The backing store is commonly a fast disk.
- It must be large enough to accommodate copies of all memory images for all users, and it must provide direct access to these memory images.
- The system maintains a ready queue consisting of all processes is on the backing store or in memory and are ready to run is shown in figure 3.1.

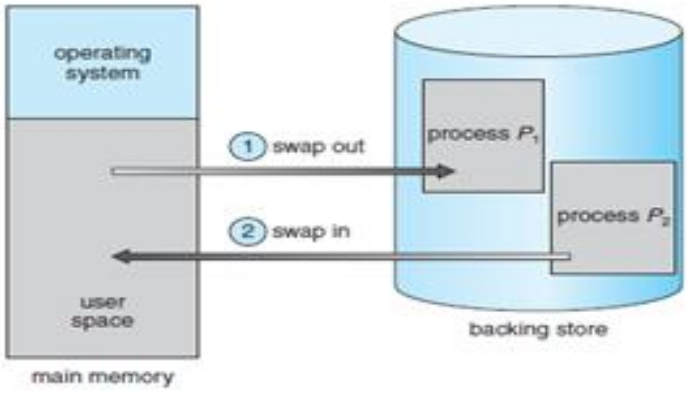


Fig 3.1 Swapping of two processes using a disk as a backing store

Dispatcher:

Whenever the CPU scheduler decides to execute a process, it calls the dispatcher. The dispatcher checks to see whether the next process in the queue is in memory. If it is not, and if there is no free memory region, the dispatcher swaps out a process currently in memory and swaps in the desired process.

- It then reloads registers and transfers control to the selected process
- The context-switch time in such a swapping system is fairly high.

2. Briefly Explain about Contiguous memory allocation with neat diagram. [April/May-2021]

The main memory must accommodate both the operating system and the various user processes. Contiguous memory allocation is used to allocate different parts of the main memory in the most efficient way possible.

The memory is divided into two partitions:

- Resident operating system,
- User processes.
- The operating system is placed in either low memory or high memory.
- The major factor affecting this decision is the location of the interrupt vector.

Interrupt vector:

- The interrupt vector is often in low memory; programmers usually place the operating system in low memory as well.
- Several user processes to reside in memory at the same time.
- In this contiguous memory allocation, each process is contained in a single contiguous section of memory.

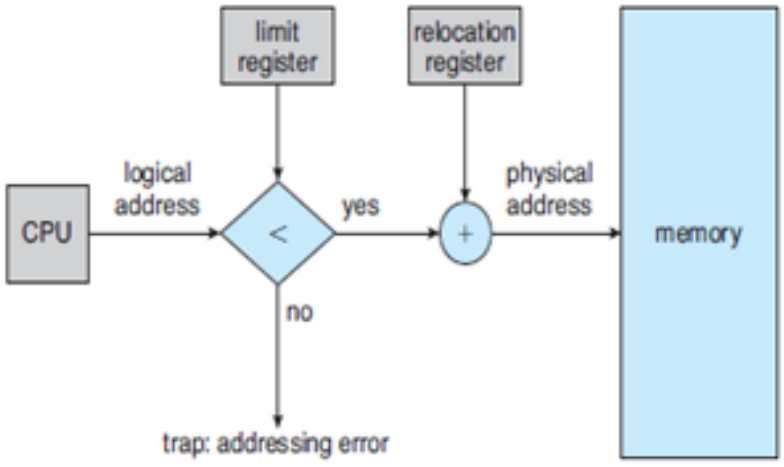


Fig 3.2 Hardware Support for relocation and limit register

Memory Protection:

Relocation Register:

The relocation register contains the value of the smallest physical address as shown in figure 3.3. (Example, relocation = 100040)

Limit Register:

- The limit register contains the range of logical addresses**.** (Example, limit = 74600).
- With relocation and limit registers, each logical address must be less than the limit register is shown in fig 3.2.
- The MMU maps the logical address dynamically by adding the value in the relocation register. This mapped address is sent to memory is shown in fig 3.3.

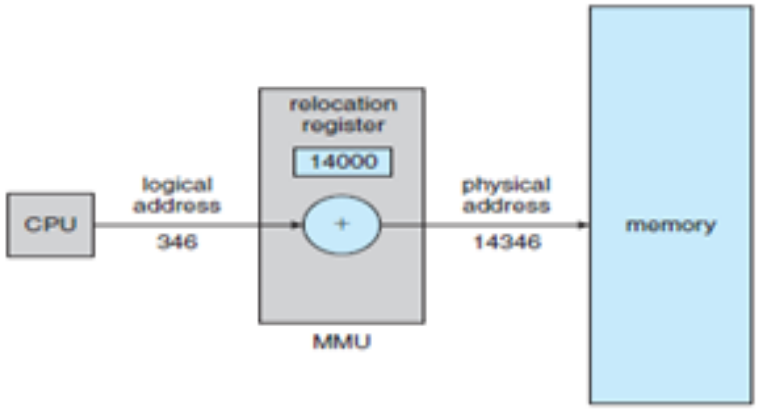


Fig 3.3 Dynamic relocation using a relocation register

Memory Allocation:

Fixed-partition scheme (called MVT):

- Fixed-partition scheme (called MVT) is used primarily in a batch environment.
- Many of the ideas presented here are also applicable to a time-sharing environment in which pure segmentation is used for memory management.

Variable-partition scheme:

- The operating system keeps a table indicating which parts of memory are available and which are occupied.
- When a process is allocated space, it is loaded into memory.
- When a process terminates, it releases its memory, which the operating system may then fill with another process from the input queue.

Dynamic storage allocation problem:

Dynamic storage allocation problem, concerns how to satisfy a request of size *n* from a list of free holes. **Solutions for Dynamic storage allocation problem:**

- **First fit:** Allocate the first hole that is big enough.
- **Best fit:** Allocate the smallest hole that is big enough.
- **Worst fit:** Allocate the largest hole.

Fragmentation:

✓**External fragmentation:**

- Both the first-fit and best-fit strategies for memory allocation suffer from external fragmentation.
- External fragmentation exists when there is enough total memory space to satisfy a request but the available spaces are not **contiguous:** storage is fragmented into a large number of small holes.

✓**Internal fragmentation:** The memory allocated to a process may be slightly larger than the requested memory. The difference between these two numbers is internal fragmentation—unused memory that is internal to a partition.

Compaction:

- One solution to the problem of external fragmentation is **compaction**.
- The goal is to shuffle the Memory contents so as to place all free memory together in one large block.

- Compaction is not always possible, if relocation is static and is done at assembly or load time, compaction cannot be done.

3.Explain the concept of Paging and Translation Look- aside Buffer. [April/May 2017, 2021] [Nov/Dec -2021] (or) With a neat sketch, explain how the logical address is translated into physical address using paging mechanism. (Nov/Dec 2024)

Paging:

- Segmentation permits the physical address space of a process to be noncontiguous. **Paging** is another memory-management scheme that offers this advantage.
- The difference between paging and segmentation is Paging avoids external fragmentation and the need for compaction, whereas segmentation does not.
- Paging solves the problem of memory chunks of varying sizes onto the backing store.

Advantages:

- Paging in its various forms is used in most operating systems.
- Paging is implemented through cooperation between the operating system and the computer hardware.

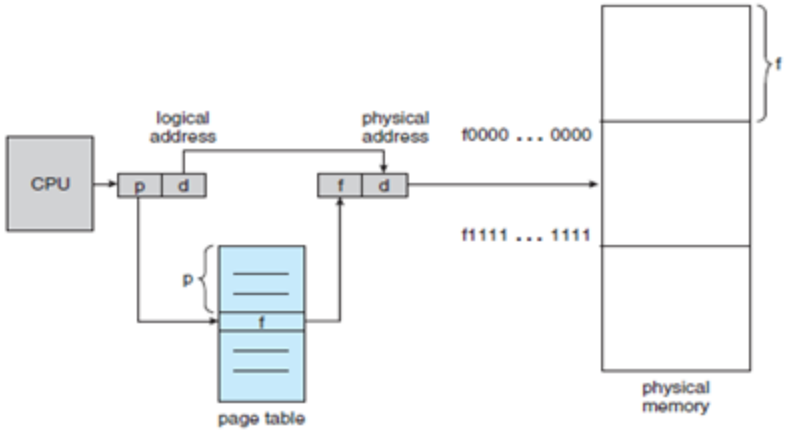


Fig 3.4 Paging Hardware

Basic method:

- Physical memory into fixed-sized blocks called **frames** and breaking logical memory into blocks of the same size called **pages**.
- When a process is to be executed, its pages are loaded into any available memory frames from their source.
- The backing store is divided into fixed-sized blocks that are of the same size memory frames or clusters of multiple frames.

The above diagram 3.4 shows, The hardware support for paging. Every address generated by the CPU is divided into two parts:

P**age number (p):**

- The page number is used as an index into a **page table.**
- The page table contains the base address of each page in physical memory.

Page offset (d):

- The base address is combined with the page offset to define the physical memory address that is sent to the memory unit.

Paging model of Logical and physical memory:

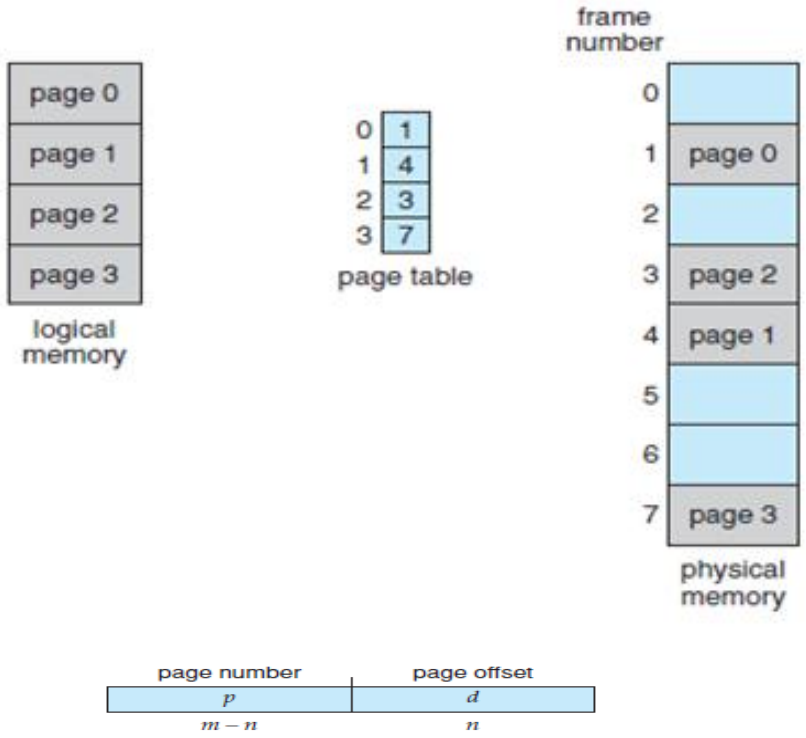


Fig 3.5 Paging model of logical and physical memory

The paging model of memory is shown in Figure 3.5. The page size (like the frame size) is defined by the hardware. For example, page 0 is stored in frame 1 it is showed in page table and physical memory.

The size of a page is typically a power of 2, varying between 512 bytes and 16 MB per page, depending on the computer architecture.

- The Figure 3.6. Shows that the page size is 4 bytes and the physical memory contains 32 bytes (8 pages).
- Logical address 0 is page 0, offset 0. Indexing into the page table, find that page 0 is in frame 5.
- Thus, logical address 0 maps to physical address 20 (= (5 x 4) + 0).
- Logical address 3 (page 0, offset 3) maps to physical address 23 (= (5 x 4) + 3).

Some CPUs and kernels even support multiple page sizes. Solaris uses 8 KB and 4 MB page sizes, depending on the data stored by the pages. Researchers are now developing variable on-the-fly page-size.

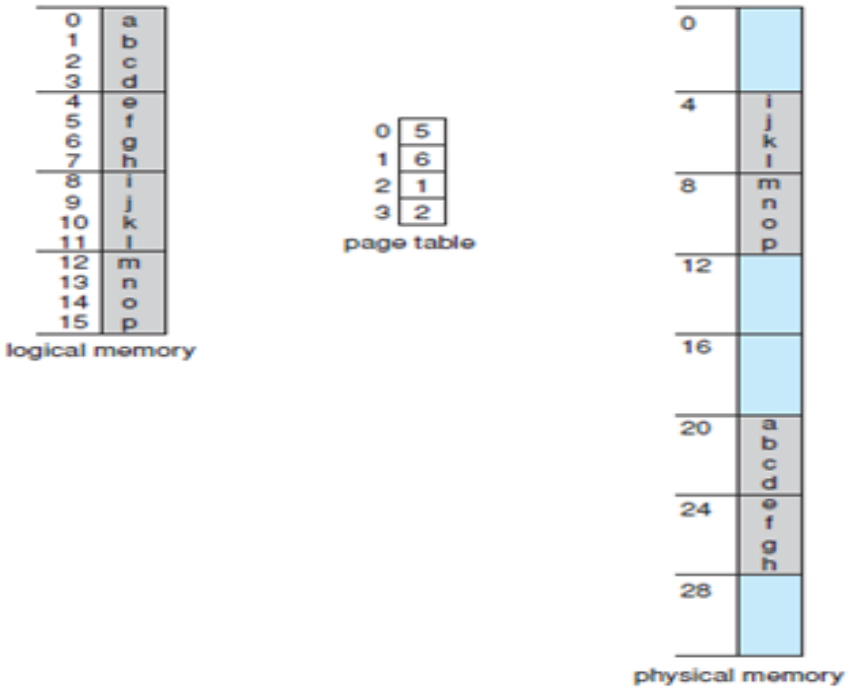


Fig. 3.6 Paging example for a 32 –byte memory with 4-byte page

- For example, which frames are allocated, which frames are available, how many total frames there are, and so on. This information is generally kept in a data structure called a frame table and free frame list is shown in fig 3.7.

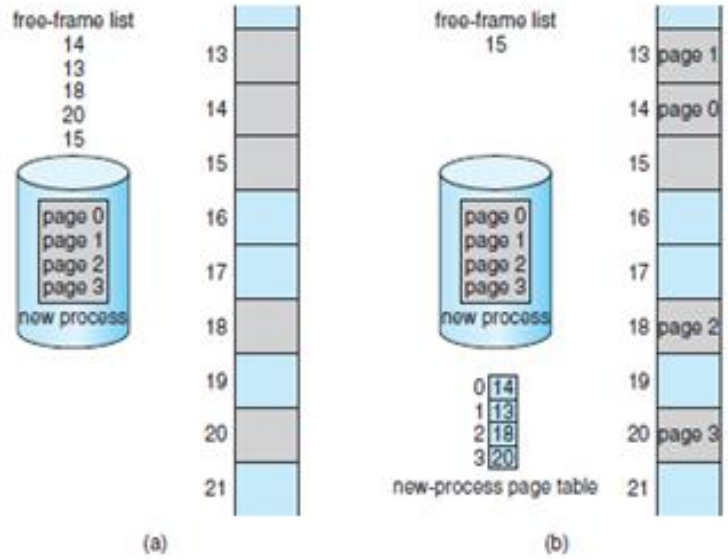


Fig 3.7 Free frames a) Before allocation and b)After allocation

Page-table base register (PTBR):

The page table is kept in main memory, and a **page-table base register (PTBR)** points to the page table.

Translation look-aside buffer (TLB):

The standard solution to the time required to access a user memory location problem is to use a special, small, fast look up hardware cache, called **translation look-aside buffer (TLB)**. The TLB is associative, high-speed memory.

Each entry in the TLB consists of two parts:

- Key (or tag)
- Value.

TLB miss:

If the page number is not in the TLB (known as a **TLB miss**), a memory reference to the page table must be made.

The below diagram 3.8 shows,

- The page number and frame number is presented in the TLB.
- If the TLB is already full of entries, the operating system must select one for replacement.
- Replacement policies range from least recently used (LRU) to random.

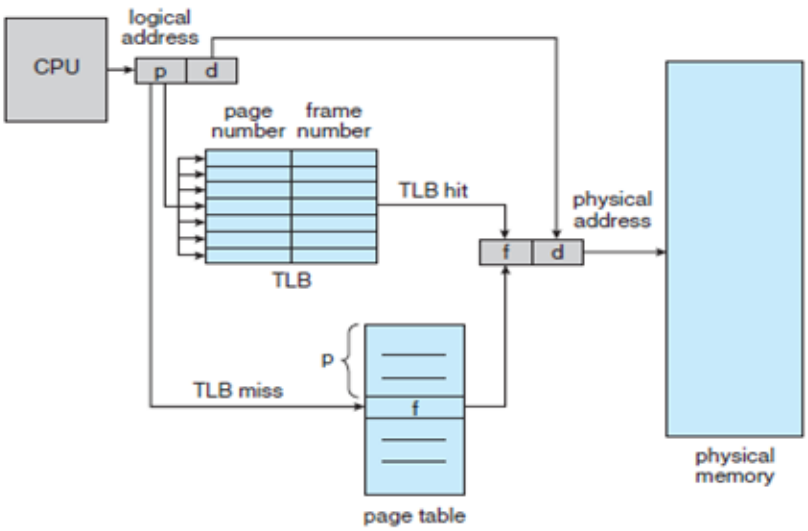


Fig 3.8 Paging Hardware with TLB

Wired down:

Some TLBs allow entries to be **wired down**, meaning that they cannot be removed from the TLB. Typically, TLB entries for kernel code are often wired down.

Address-space identifiers (ASIDs):

- The TLBs store **address-space identifiers (ASIDs)** in each entry of the TLB.
- An ASID uniquely identifies each process and is used to provide address space protection for that process.
- Every time a new page table is selected (for instance, each context switch), the TLB must be flushed (or erased) to ensure that the next executing process does not use the wrong translation information.

Hit ratio: The percentage of times that a particular page number is found in the TLB is called the **hit ratio**.

Protection:

- Memory protection in a paged environment is accomplished by **protection bits** 11 that are associated with each frame. Normally, these bits are kept in the page table.
- One bit can define **a page to be read-write or read-only.**

Valid bit: When this bit is set to "valid," this value indicates that the associated page is in the process' logical

address space, and is thus a legal (or valid) page.

Invalid bit: If the bit is set to "invalid," this value indicates that the page is not in the process' logical-address space. Illegal addresses are trapped by using the valid-invalid bit.

Page-table length register (PTLR): Some systems provide hardware, in the form of a page-table length register (PTLR), to indicate the size of the page table.

- This value is checked against every logical address to verify that the address is in the valid range for the process.

Reentrant code (or **pure code):**

- Reentrant code (or pure code) is non-self-modifying code. If the code is reentrant, then it never changes during execution.
- If the code is reentrant code (or pure code), however, it can be shared, as shown in Figure 3.9. Here three-page editor-each page of size 50 KB;
- The large page size is used to simplify the figure-being shared among three processes.

- Each process has its own data page.

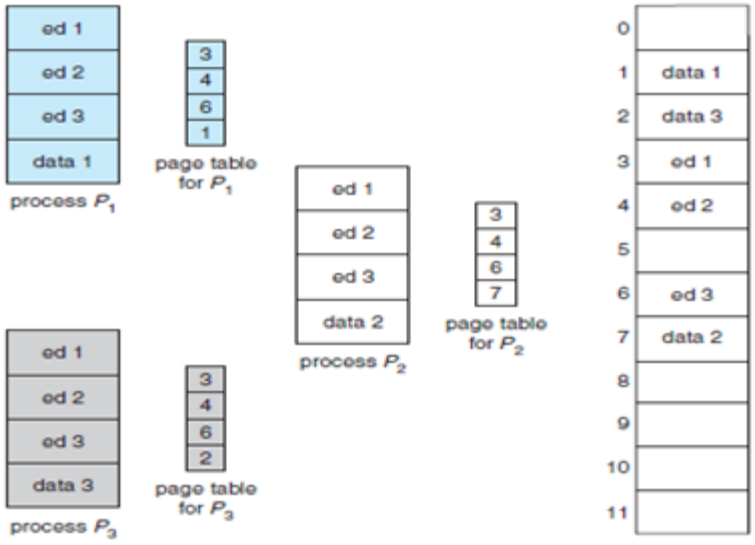


Fig 3.9 Sharing of code in a paging environment

- Only one copy of the editor needs to be kept in physical memory. Each user's page table maps onto the same physical copy of the editor, but data pages are mapped onto different frames.

4. Write about the techniques for structuring the page table.

The most common techniques used for the structuring the page table.

Hierarchical Paging:

Most modern computer systems support a large logical-address space (2^{32} to 2^{64}). In such an environment, the page table itself becomes excessively large.

Example:

- Consider a system with a 32-bit logical-address space. If the page size in such a system is 4 KB (2^{12}), then a page table may consist of up to 1 million entries ($2^{32}/2^{12}$)
- Assuming that each entry consists of 4 bytes, each process may need up to 4 MB of physical-address space for the page table alone.
- Clearly, we would not want to allocate the page table contiguously in main memory.

- One simple solution to this problem is to **divide the page table into smaller pieces**. There are several ways to accomplish this division.
- One way is to use a two-level paging algorithm, in which the page table itself is also paged is shown in below fig 3.10.

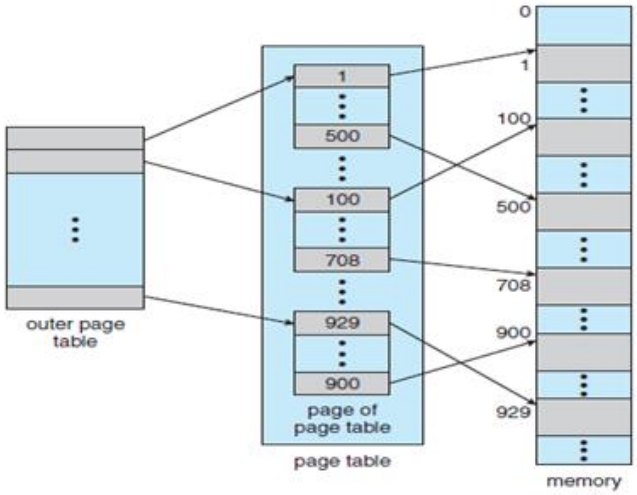


Fig 3.10 A two level page table scheme

Remember our example to our 32-bit machine with a page size of 4 KB.

A logical address is divided into a page number consisting of 20 bits, and a page offset consisting of 12 bits.

Because we page the page table, the page number is further divided into a 10-bit page number and a 10-bit page offset.

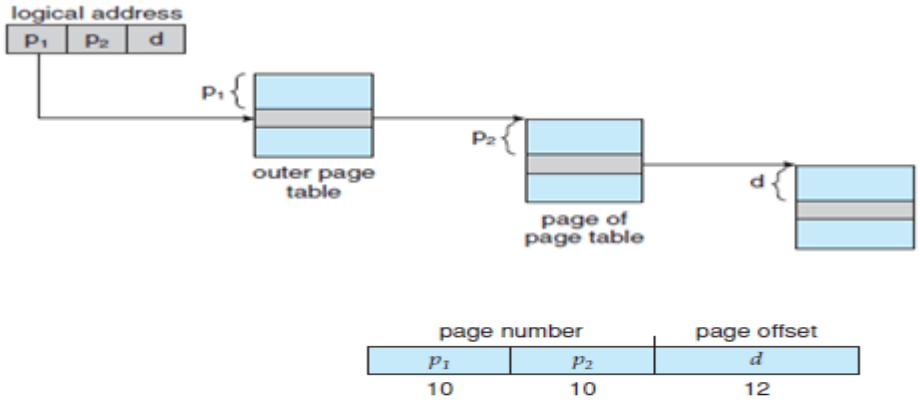


Fig 3.11 Address translation for a two level 32 bit paging Architecture

Where pl is an index into the outer page table and p2 is the displacement within the page of the outer page table. The address-translation method for this architecture is shown in Figure 3.11.

- **Forward-mapped page table**:

Address translation works from the outer page table inwards; this scheme is also known as a **forward-mapped page table**. The **Pentium-II** uses this architecture.

VAX Architecture:

- Consider the memory management of one of the classic systems, the **VAX** minicomputer from **Digital Equipment Corporation (DEC)**.
- The **VAX** architecture also supports a variation of two-level paging.
- The **VAX** is a 32-bit machine with page size of 512 bytes is shown in fig 3.12.
- The logical-address space of a process is divided into four equal sections, each of which consists of 2^{30} bytes.

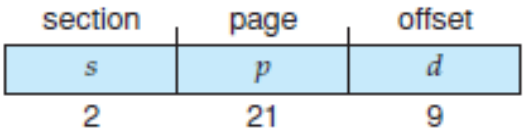


Fig 3.12 -32-bit VAX architecture

The above figure 3.12 shows that, where s designates the section number, p is an index into the page table, and d is the displacement within the page.

Section:

Each section represents a different part of the logical-address space of a process. The first 2 high-order bits of the logical address designate the appropriate section.

Page: The next **21 bits represent the logical page number** of that section.

Offset: The final **9 bits represent an offset** in the desired page.

The **below diagram** 3.13 shows that, the size of a one-level page table for a **VAX** process using one section still is 2^{21} bits * 4 bytes per entry = 8 MB.

Inner page table: The inner page tables could conveniently be one page long, or contain 2^{10} 4-byte entries.

Outer page table: The outer page table will consist of 2^{42} entries, or 2^{44} bytes.

- The addresses would look like:

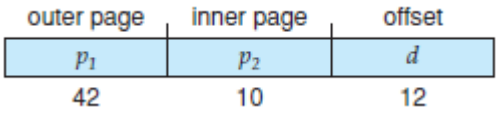


Fig 3.13 64-bit page address

- The obvious method to avoid such a large table is to divide the outer page table into smaller pieces.
- This approach is also used on some 32-bit processors for added flexibility and efficiency.

The below diagram 3.14 shows that, the outer page table can also be divided into various ways, it also giving a three-level paging scheme

- Suppose that the outer page table is made up of standard-size pages (2^{10}entries, or 2^{12} bytes); a 64-bit address space is still daunting: The outer page table is still 2^{34} bytes large.

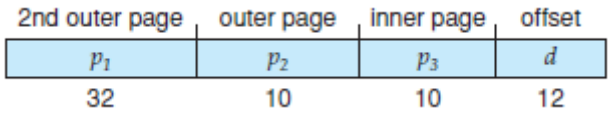


Fig 3.14 -64 bit page address

- SPARC architecture-The SPARC architecture (with 32-bit addressing) supports a three-level paging scheme,
- Motorola 68030 architecture-The 32-bit Motorola 68030 architecture supports a four-level paging scheme.
- UltraSPARC architecture -64-bit UltraSPARC would require seven levels of paging scheme.

Hashed Page Tables:

A common approach for handling address spaces larger than 32 bits is to use a hashed page table, with the hash value being the virtual-page number.

- Each entry in the hash table contains a linked list of elements that hash to the same location (to handle collisions).
- Each element consists of three fields:
  - **(a) The virtual page number,**
  - **(b) The value of the mapped page frame, and**
  - **(c) Pointer to the next element in the linked list.**

If there is no match, subsequent entries in the linked list are searched for a matching virtual page number. This scheme is shown in Figure 3.15.

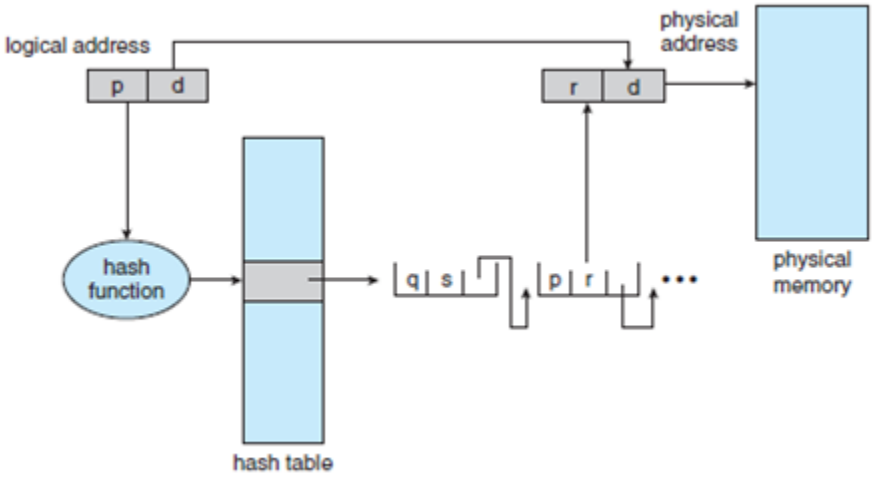


Fig 3.15 Hashed page table

Clustered page tables:

Clustered page tables are similar to hashed page tables except that each entry in the hash table refers to several pages (such as 16) rather than a single page.

Sparse:

- Clustered page tables are particularly useful for **sparse** address spaces where memory references are noncontiguous and scattered throughout the address space.

Inverted page table:

- Inverted page table is **used to overcome the problem of Page table**.

- An inverted page table has **one entry for each real page (or frame) of memory**. Each entry consists of the virtual address of the page stored in that real memory location; with information about the process that owns that page.

The below diagram 3.16 shows,

- The operation of an inverted page table.
- Inverted page tables often require an address-space identifier stored in each entry of the page table.
- Storing the address-space identifier ensures the mapping of a logical page for a particular process to the corresponding physical page frame

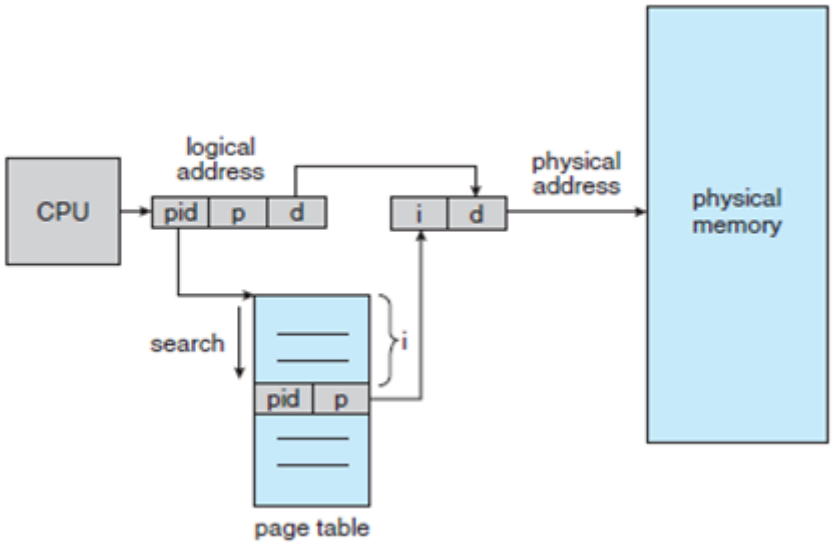


Fig 3.16 Inverted page table

Each virtual address in the system consists of a triple **<process-id, page-number, offset>.**

5. <u>Draw the diagram of segmentation memory management scheme and explain its principle.</u> [April/May2010] [Nov/Dec 2017]

Segmentation is a memory-management scheme that supports this user view of memory. A logical address space is a collection of segments. Each segment has a name and a length. The addresses specify both the segment name and the offset within the segment.

Basic Method:

- When writing a program, a programmer thinks of it as a main program with a set of methods, procedures, or functions.
- It may also include various data structures: objects, arrays, stacks, variables, and so on. Each of these modules or data elements is referred to by name.
- The programmer talks about "the stack," "the math library," and "the main program" without caring what addresses in memory these elements occupy.
- **Segments vary in length, and the length of each is intrinsically defined by its purpose in the program**. Elements within a segment are identified by their offset from the beginning of the segment: the first statement of the program, the seventh stack frame entry in the stack, the fifth instruction of the Sqrt() is shown in fig 3.17.

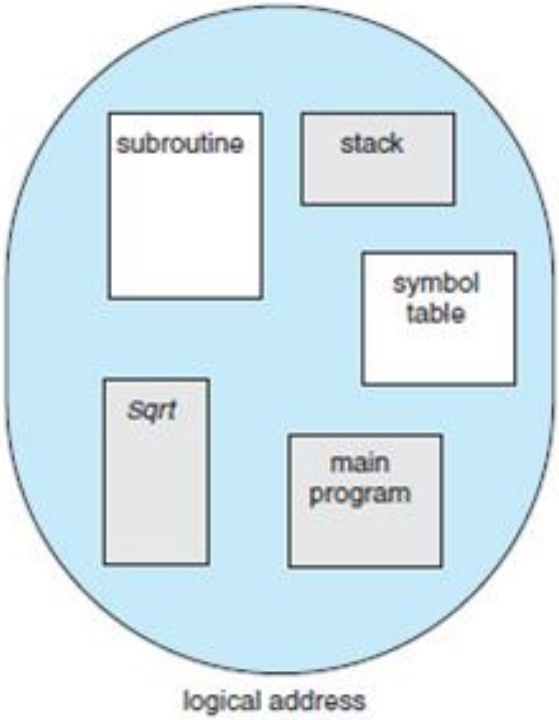


Fig 3.17 Programmer's view of a program

The user therefore specifies each address by two quantities:

- Segment name
- Offset.

- Thus, a logical address consists of a ***two*** tuple:

<segment – number, offset>

- Normally, the user program is compiled, and the compiler automatically constructs segments reflecting the input program. A Pascal compiler might create separate segments for the following:

1. The code

2. Global variables

3. The heap, from which memory is allocated

4. The stacks used by each thread

5. The standard C library

Segmentation Hardware:

Segment table:

- Each entry of the segment table has a segment base and a segment limit.
- The segment base contains the starting physical address where the segment resides in memory, whereas the segment limit specifies the length of the segment.
- Define an implementation to map two-dimensional user-defined addresses into one-dimensional physical addresses. This mapping is affected by a segment table.

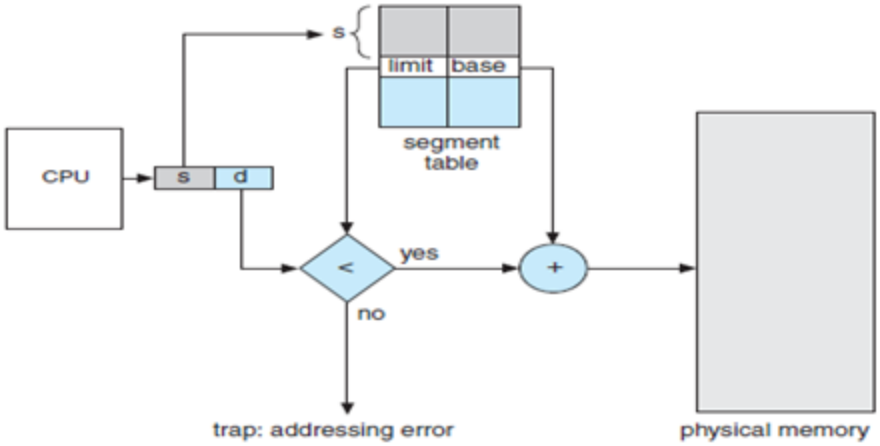


Fig.3.18 Segmentation Hardware

The above diagram 3.18 Shows,

- A logical address consists of two parts:
- Segment number, s,
- Offset into that segment, d.
- The segment number is used as an index into the segment table.

- The offset d of the logical address must be between 0 and the segment limit.

6. Explain in detail about Virtual Memory.

(or)

What do you mean by virtual memory? Consider a system where the virtual memory page size is 2K (2048 bytes), and main memory consists of 4 page frames. Now consider a process which requires 8 pages of storage. At some point during its execution, the page table is as shown below:

Virtual page	Presence bit	Physical Page
0	0	
1	0	
2	1	1
3	0	
4	1	3
5	0	
6	1	0
7	1	2

(i) List the virtual address ranges for each virtual page. (3)

(ii) List the virtual address ranges that will result in a page fault. (4)

(iii) Write the main memory (physical) addresses for each of the following virtual addresses (all numbers decimal): (1) 8500, (2) 14000, (3) 5000, (4) 2100. (6) (April/May 2024)

Virtual memory is a technique **that allows the execution of processes that may not be completely in memory.**

Advantages:

- Programs can be larger than physical memory. Further, virtual memory abstracts main memory into an extremely large, uniform array of storage, separating logical memory as viewed by the user from physical memory.
- This technique frees programmers from the concerns of memory-storage limitations.
- It allows processes to easily share files and address spaces, and it provides an efficient mechanism for process creation.

Disadvantages:

- It is not easy to implement.
- Decrease performance if it is used carelessly.

Basic requirement for the memory-management algorithms:

- The instructions being executed must be in physical memory.
- The first approach to meeting this requirement is to place the entire logical address space in physical memory.
- Overlays and dynamic loading can help to ease this restriction, but they generally require special precautions and extra work by the programmer.
- This restriction seems both necessary and reasonable, but it is also unfortunate, since it limits the size of a program to the size of physical memory.

Virtual memory involves the separation of user logical memory from physical memory. This separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available.

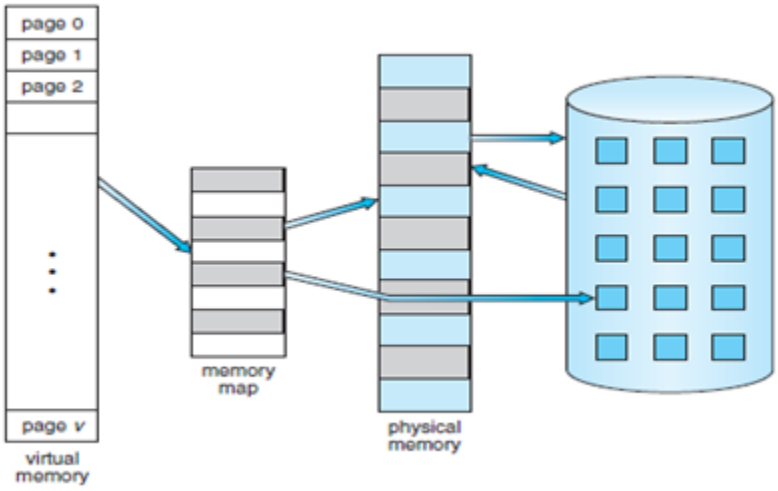


Fig 3.19. Diagram showing virtual memory that is larger than physical memory

- The above diagram 3.19 Shows,
- The **virtual address space** of a process refers to the logical (or virtual) view of how a process is stored in memory.
- Typically, this view is that a process begins at a certain logical address—say, address 0—and exists in contiguous memory physical memory may be organized in page frames and that the physical page frames assigned to a process may not be contiguous.

- It is up to the memory management unit (MMU) to map logical pages to physical page frames in memory.
- Allow heap to grow upward in memory as it is used for dynamic memory allocation.
- Similarly, allow for the stack to grow downward in memory through successive function calls.
- The large blank space (or hole) between the heap and the stack is part of the virtual address space but will require actual physical pages only if the heap or stack grows is shown in fig 3.20.

Sparse:

- Virtual address spaces that include holes are known as **sparse** address spaces. Using a sparse address space is beneficial because the holes can be filled as the stack or heap segments grow or if we wish to dynamically link libraries (or possibly other shared objects) during program execution.

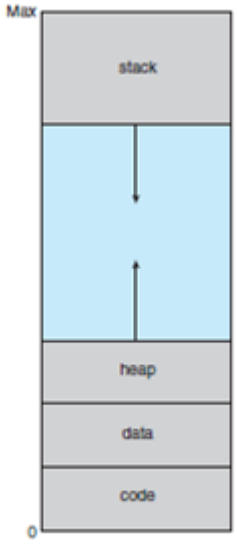


Fig 3.20 Virtual address space

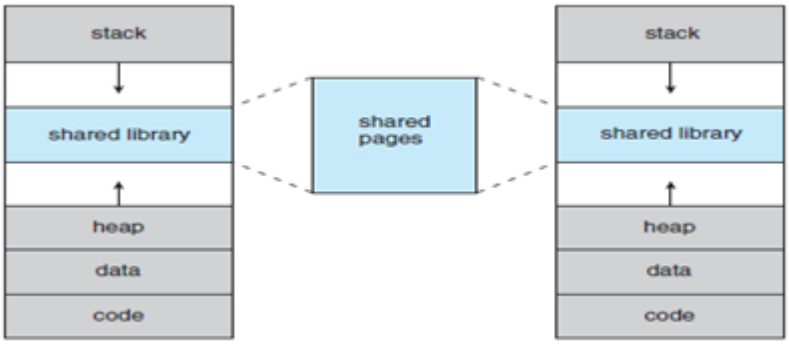


Fig 3.21 Shared library using virtual memory

- System libraries can be shared by several processes through mapping of the shared object into a virtual address space. Although each process considers the libraries to be part of its virtual address space, the actual pages where the libraries reside in physical memory are shared by all the processes is shown in fig 3.21. Typically, a library is mapped read-only into the space of each process that is linked with it.
- Similarly, processes can share memory. Virtual memory allows one process to create a region of memory that it can share with another process. Processes sharing this region consider it part of their virtual address space, yet the actual physical pages of memory are shared.

Virtual page	Presence bit	Physical Page
0	0	
1	0	
2	1	1
3	0	
4	1	3
5	0	
6	1	0
7	1	2

- Assuming a page size of 2000 bytes, the virtual address ranges for each virtual page would be:
  - **Page 0:** 0 - 1999
  - **Page 1:** 2000 - 3999
  - **Page 2:** 4000 - 5999

- **Page 3:** 6000 - 7999

To find the physical addresses for the given virtual addresses:

- **8500:**
  - Calculate the page number: 8500 / 2000 = 4.25 (round down to 4)
  - Since 8500 falls within the range of page 4 (6000 - 7999), the physical address would be calculated based on the page table entry for page 4.
- **14000:**
  - Calculate the page number: 14000 / 2000 = 7
  - The physical address would be based on the page table entry for page 7.
- **5000:**
  - Calculate the page number: 5000 / 2000 = 2.5 (round down to 2)
  - The physical address would be based on the page table entry for page 2.
- **2100:**
  - Calculate the page number: 2100 / 2000 = 1.05 (round down to 1)
  - The physical address would be based on the page table entry for page 1.
  - To determine the exact physical addresses, you need to know the corresponding physical frame number assigned to each virtual page in the page table, which is not provided in this question.

7.Write about the concepts of Demand Paging (or) Swapping (Nov/Dec-2019)[April/May-2021]

Demand Paging (or) Swapping:

- A demand-paging system is similar to a paging system with swapping. Processes reside on secondary memory (which is usually a disk).
- When we want to execute a process, we swap it into memory. Rather than swapping the entire process into memory, however, we use a **lazy swapper.**

Lazy swapper:

- A lazy swapper never swaps a page into memory unless that page will be needed.
- Since we are now viewing a process as a sequence of pages, rather than as one large contiguous address space, use of *swap* is technically incorrect.

Swapper and Pager:

- A **swapper** manipulates entire processes, whereas a **pager** is concerned with the individual pages of a process. So use *pager,* rather than *swapper,* in connection with demand paging is shown in fig 3.22.

Basic Concepts:

- Swapping in a whole process, the pager brings only those necessary pages into memory.
- It avoids reading into memory pages that will not be used anyway.
- It decreases the swap time and the amount of physical memory needed.

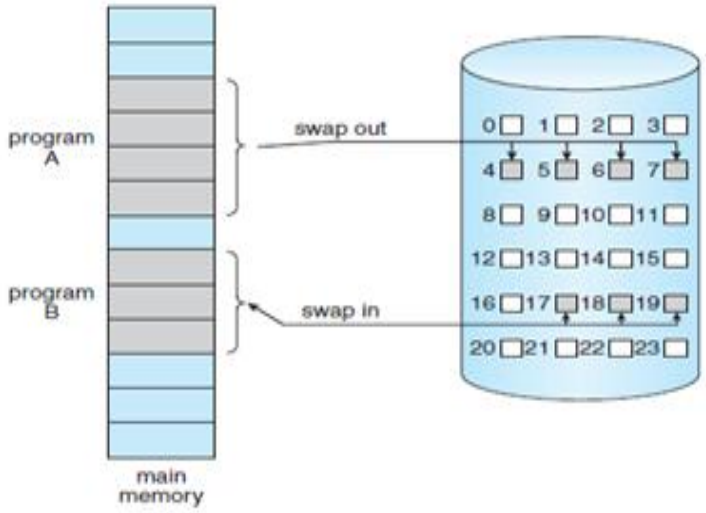


Fig 3.22. Transfer of a paged memory to contiguous disk space

The **valid-invalid bit** scheme can be used for this purpose.

- **Valid**-If the bit is set to "**valid,**" this value indicates that the associated page is both legal and in memory.
- **Invalid**-If the bit is set to "invalid," this value indicates that the page either is not valid (that is, not in the logical address space of the process), or is valid but is currently on the disk.

- While the process executes and accesses pages that are **memory resident,** execution proceeds normally.

Page-fault trap:

- Access to a page marked invalid causes a **page-fault trap.**
- The paging hardware, in translating the address through the page table, If the invalid bit is set, causing a trap to the operating system is shown in fig 3.23

- This trap is the result of the operating system's failure to bring the desired page into memory rather than an invalid address error as a result of an attempt to use an illegal memory address.

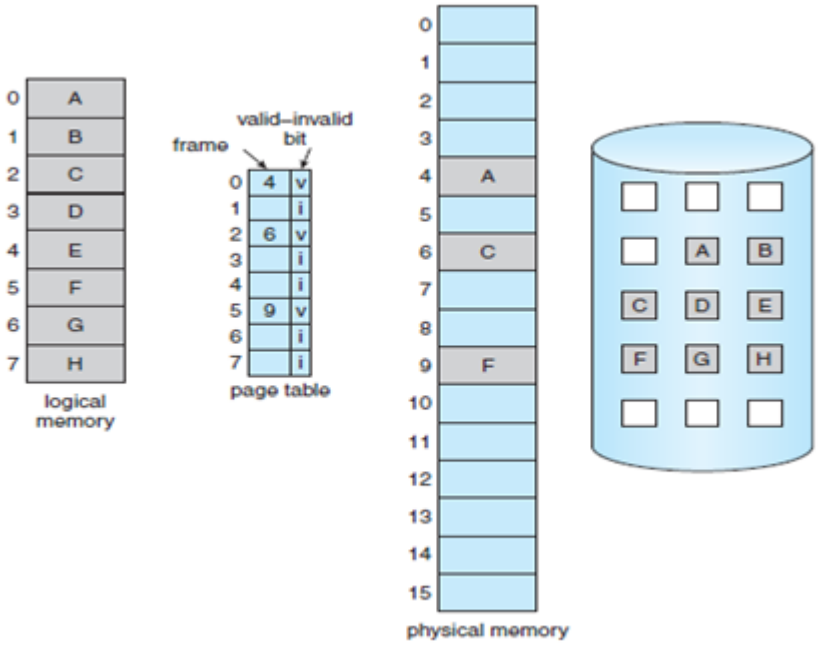


Fig 3.23. Page table when some pages are not in memory

The procedure for handling this page fault is straightforward

- We check an internal table (usually kept with the process control block) for this process, to determine whether the reference was a valid or invalid memory access.
- If the reference was invalid, we terminate the process. If it was valid, but we have not yet brought in that page, we now page it in.
- We find a free frame (by taking one from the free-frame list, for example).

Pure demand paging:

After this page is brought into memory, the process continues to execute, faulting as necessary until every page that it needs is in memory. At that point, it can execute with no more faults. This scheme is **pure demand paging:** Never bring a page into memory until it is required.

Locality of reference:

- Analysis of running processes shows that this behavior is exceedingly unlikely.
- Programs tend to have **locality of reference,** which results in reasonable performance from demand paging.

The hardware to support demand paging is the same as the hardware for paging and swapping:

Page table: This table has the ability to mark an entry invalid through a valid-invalid bit or special value of protection bits.

Secondary memory: This memory holds those pages that are not presenting main memory. The secondary memory is usually a high-speed disk. It is known as the swap device, and the section of disk used for this purpose is known as **swap space.**

Performance of Demand Paging:

Effective access time:

Demand paging can have a significant effect on the performance of a computer system. To compute the **effective access time** for a demand paged memory.

$$\text{effective access time} = (1 - p) \times ma + p \times \text{page fault time}.$$

8.Explain in detail about Page Replacement.

Page replacement:

If no frame is free, we find one that is not currently being used and free it. We can free a frame by writing its contents to swap space, and changing the page table (and all other tables) to indicate that the page is no longer in memory. Use the freed frame to hold the page for which the process faulted is shown in fig 3.24.

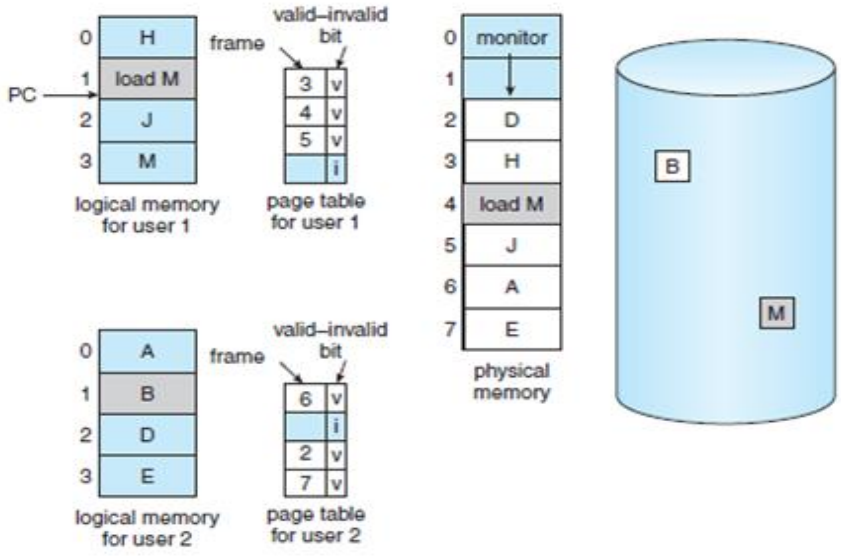


Fig 3.24 Need for page replacement

Basic Scheme:

Modify the page-fault service routine to include page replacement:

1. Find the location of the desired page on the disk.
2. Find a free frame:

   a. If there is a free frame, use it

   b. If there is no free frame, use a page-replacement algorithm to select a victim frame.

   c. Write the victim page to the disk; change the page and frame tables accordingly.

3. Read the desired page into the (newly) free frame; change the page and frame tables.
4. Restart the user process.

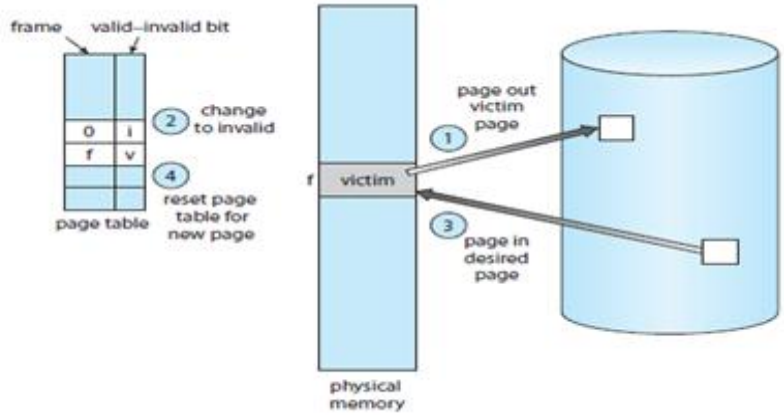
- If no frames are free, *two-page* transfers (one out and one in) are required. This situation effectively doubles the page-fault service time and increases the effective access time accordingly.

Modify bit (or dirty bit):

Each page or frame may have a **modify bit associated with it in the hardware**. The **modify bit for a page is set by the hardware** whenever any word or byte in the page is written into, indicating that the page has been modified.

- If the modify bit is set, we know that the page has been modified since it was read in from the disk.

- If the modify bit is not set, however, the page has not been modified since it was read into memory. Therefore, if the copy of the page on the disk has not been overwritten (by some other page, for example), then we can avoid writing the memory page to the disk: it is already there.
- This technique also applies to read-only pages. Such pages cannot be modified; thus, they may be discarded when desired is shown in fig 3.25.



.

Fig 3.25 Page replacement

Solve two major problems to implement demand paging:

- Frame-allocation algorithm
- Page-replacement algorithm.

Reference string:

- An algorithm by running it on a particular string of memory references and computing the number of page faults. The string of memory references is called a **reference string.**
- Generate reference strings artificially (by a random-number generator, for example) or we can trace a given system and record the address of each memory reference
- The latter choice produces a large number of data (on the order of 1 million addresses per second). To reduce the number of data, we use two facts.
- First, for a given page size (and the page size is generally fixed by the hardware or system), we need to consider only the page number, rather than the entire address.

- Second, if we have a reference to a page p, then any immediately following references to page p will never cause a page fault.
- Page p will be in memory after the first reference; the immediately following references will not fault is shown in fig 3.26

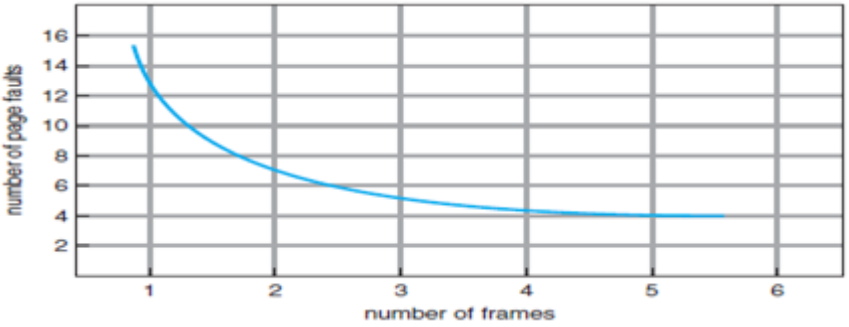


Fig 3.26 Graph of page fault vs number of frames

9.With neat diagram explain the different page replacement algorithms.[April/May-2021][Nov/Dec-2021]

(i) FIFO Page Replacement:

- A simple and obvious page replacement strategy is ***FIFO***, i.e., **first-in-first-out**.
- As new pages are brought in, they are added to the tail of a queue, and the page at the head of the queue is the next victim. In the following example, 20-page requests result in 15-page faults is shown in fig 3.27.

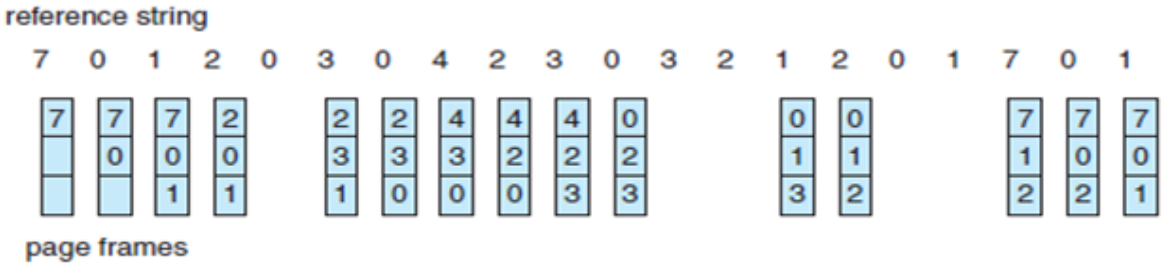


Fig 3.27 FIFO page replacement Algorithms

- Although FIFO is simple and easy, it is not always optimal, or even efficient.
- An interesting effect that can occur with FIFO is Belady's anomaly, in which increasing the number of frames available can actually ***increase*** the number of page faults that occur is shown in fig 3.28.
- Consider, for example, the following chart based on the page sequence (1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5) and a varying number of available frames. Obviously, the maximum number of faults is 12 (every request generates a fault), and the minimum

number is 5 (each page loaded only once), but in between there are some interesting results:

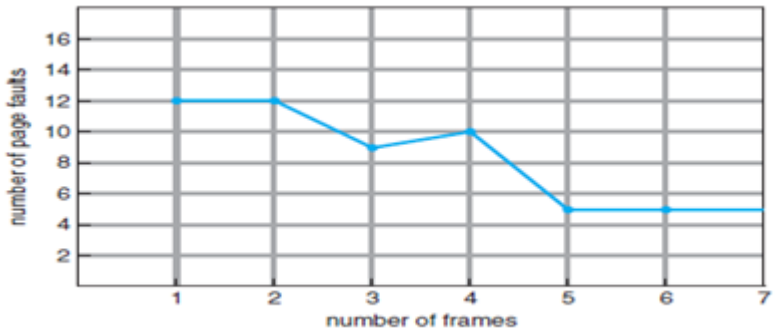


Fig 3.28 Page fault curve for FIFO replacement on a reference string

(ii) Optimal Page Replacement

- The discovery of Belady's anomaly led to the search for an ***optimal page-replacement algorithm***, which is simply that which yields the lowest of all possible page-faults, and which does not suffer from Belady's anomaly.
- Such an algorithm does exist, and is called ***OPT or MIN.*** This algorithm is simply "Replace the page that will not be used for the longest time in the future."
- For example, Figure 3.29 shows that by applying OPT to the same reference string used for the FIFO example, the minimum number of possible page faults is 9. Since 6 of the page-faults are unavoidable (the first reference to each new page), FIFO can be shown to require 3 times as many (extra) page faults as the optimal algorithm.

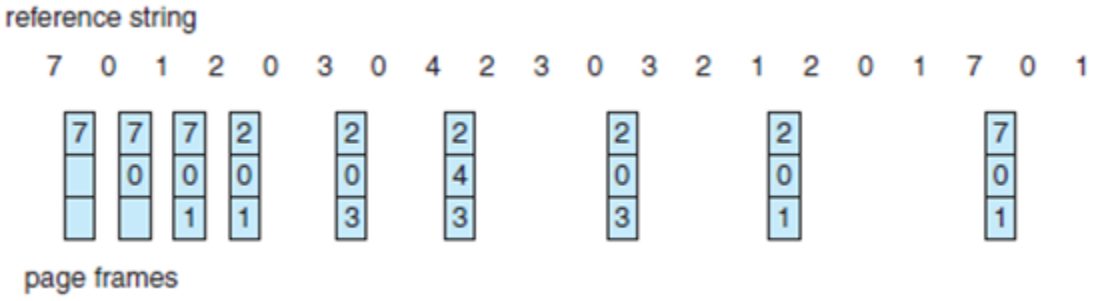


Fig 3.29 Optimal page replacement algorithm

(iii) LRU Page Replacement

- The prediction behind ***LRU,*** the ***Least Recently Used,*** algorithm is that the page that has not been used in the longest time is the one that will not be used again in the near future
- Figure 3.30 illustrates LRU for our sample string, yielding 12 page faults, (as compared to 15 for FIFO and 9 for OPT.)

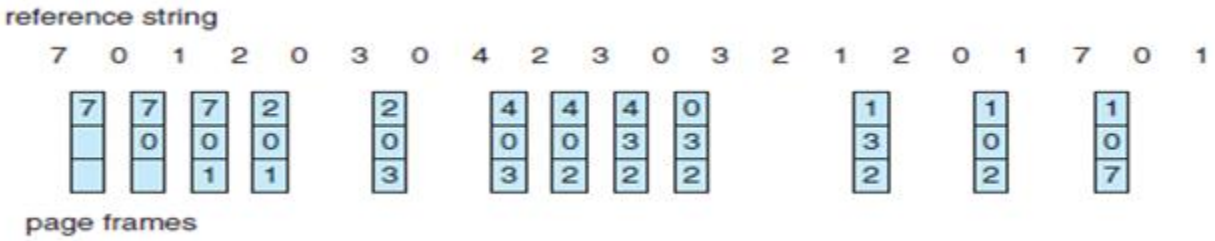


Fig 3.30 Optimal page replacement algorithm

10. Write notes about the Thrashing and its effects.(Nov/Dec 2015)

Thrashing:

- Thrashing is the coincidence of high page traffic and low CPU efficiency.
- The high paging activity is called thrashing.
- If the process does not have number of frames it needs to support pages in active use, it will quickly page fault.A process is thrashing if it is spending more time paging than executing.

Cause of Thrashing:

- Thrashing results in **severe performance problems**.
- Consider the following scenario, which is based on the actual behavior of early paging systems.
- The operating system monitors CPU utilization. If CPU utilization is too low, we increase the degree of multiprogramming by introducing a new process to the system is shown in fig 3.31.
- A global page-replacement algorithm is used; it replaces pages with no regard to the process to which they belong.
- The CPU scheduler sees the decreasing CPU utilization, and *increases* the degree of multiprogramming as a result.
- If the degree of multiprogramming is increased further, thrashing sets in and CPU utilization drops sharply

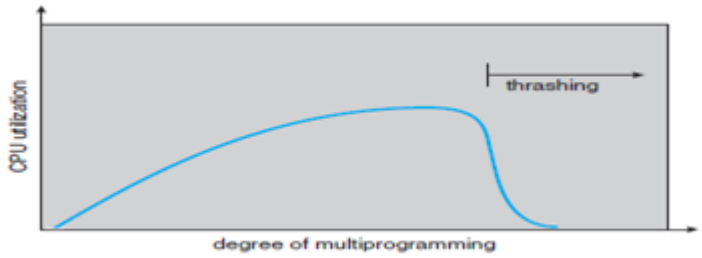


Fig 3.31 - Thrashing

Locality model:

The working-set strategy starts by looking at how many frames a process is actually using. This approach defines the **locality model** of process execution.

Working-Set Model:

- The **working-set model** is based on the assumption of locality.
- This model uses a parameter, A, to define the **working-set window.**
- The idea is to examine the most recent **A** page references.
- The set of pages in the most recent A page references is the **working set**.
- If a page is in active use, it will be in the working set.
- If it is no longer being used, it will drop from the working set **A** time units after its last reference.
- Thus, the working set is an approximation of the program's locality.

For example, given the sequence of memory references shown in Figure 3.32, if **A** = 10 memory references, then the working set at time tl is (1, 2, 5, 6,7).

- By time *t2,* the working set has changed to {3,4). The accuracy of the working set depends on the selection of **A.**

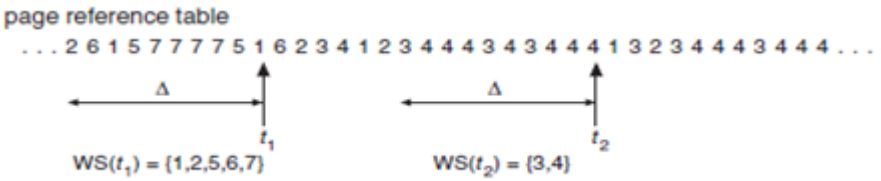


Fig 3.32 Working set model

- If A is too small, it will not encompass the entire locality; if A is too large, it may overlap several localities.

- In the extreme, if **A** is infinite, the working set is the set of pages touched during the process execution. The most important property of the working set is its size.
- If we compute the working-set size, *WSSi,* for each process in the system, we can then consider

$$D = \sum WSS_i,$$

- Where D is the total demand for frames. Each process is actively using the pages in its working set. Thus, process i needs *WSSi* frames.
- If the total demand is greater than the total number of available frames (D > *m),* thrashing will occur, because some processes will not have enough frames.

Page-Fault Frequency:

- A strategy that uses the page-fault frequency **(PFF)** takes a more direct approach.

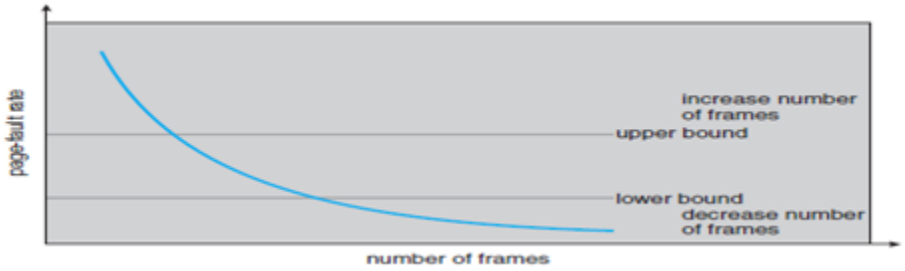


Fig 3.33 -Page fault frequency

- Thrashing has a high page-fault rate. It Control the page-fault rate.
- When it is too high, we know that the process needs more frames.
- Similarly, if the page-fault rate is too low, then the process may have too many frames.
- Establish upper and lower bounds on the desired page-fault rate is shown in above fig 3.33.
- If the actual page-fault rate exceeds the upper limit, we allocate that process another frame; if the page-fault rate falls below the lower limit, we remove a frame from that process.

11.Explain paging with Segmentation.[[Nov/Dec-2021]

Both paging and segmentation have advantages and disadvantages. In fact, of the two most popular microprocessors now being used, the Motorola is designed based on a flat-address space, whereas the Intel 80x86 and Pentium family are based on segmentation.

- Both are merging memory models toward a mixture of paging and segmentation. We can combine these two methods to improve on each.
- This combination is best illustrated by the architecture of the Intel 386. The IBM OS/2 32-bit version is operating system running on top of the Intel 386 (and later) architecture.
- The Intel 386 uses segmentation with paging for memory management.
- The maximum number of segments per process is 16 KB, and each segment can be as large as 4 gigabytes. The page size is 4 KB.

The logical-address space of a process is divided into two partitions:

- The first partition consists of up to 8 KB segments that are private to that process.
- The second partition consists of up to 8 KB segments that are shared among all the processesis shown in fig 3.34.

- **Local descriptor table (LDT):**
  - Information about the first partition is kept in the **local descriptor table (LDT).**
- **Global descriptor table (GDT):**
  - Information about the second partition is kept in the **global descriptor table (GDT).**

Each entry in the LDT and GDT consists of 8 bytes, with detailed information about a particular segment including the base location and length of that segment.

Fig 3.34 Logical address

- The logical address is a pair (selector, offset), where the selector is a 16-bit number: in which s designates the segment number, g indicates whether the segment is in the GDT or LDT, and ***p*** deals with protection.

Linear address:

The base and limit information about the segment in question are used to generate a **linear address.**

- First, the limit is used to check for address validity. If the address is not valid, a memory fault is generated, resulting in a trap to the operating system.
- If it is valid, then the value of the offset is added to the value of the base, resulting in a 32-bit linear address. This address is then translated into a physical address.
- The linear address is divided into a page number consisting of 20 bits, and a page offset consisting of 12 bits is shown in fig 3.35.

The page table, the page number is further divided into a 10-bit page directory pointer and a 10-bit page table pointer.

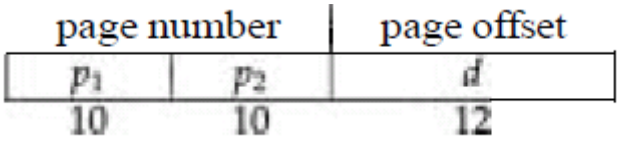


Fig 3.35 Linear Address

- The Intel address translation is shown in Figure 3.36. To improve the efficiency of physical-memory use, Intel 386 page tables can be swapped to disk.
- In this case, an invalid bit is used in the page directory entry to indicate whether the table to which the entry is pointing is Page table.

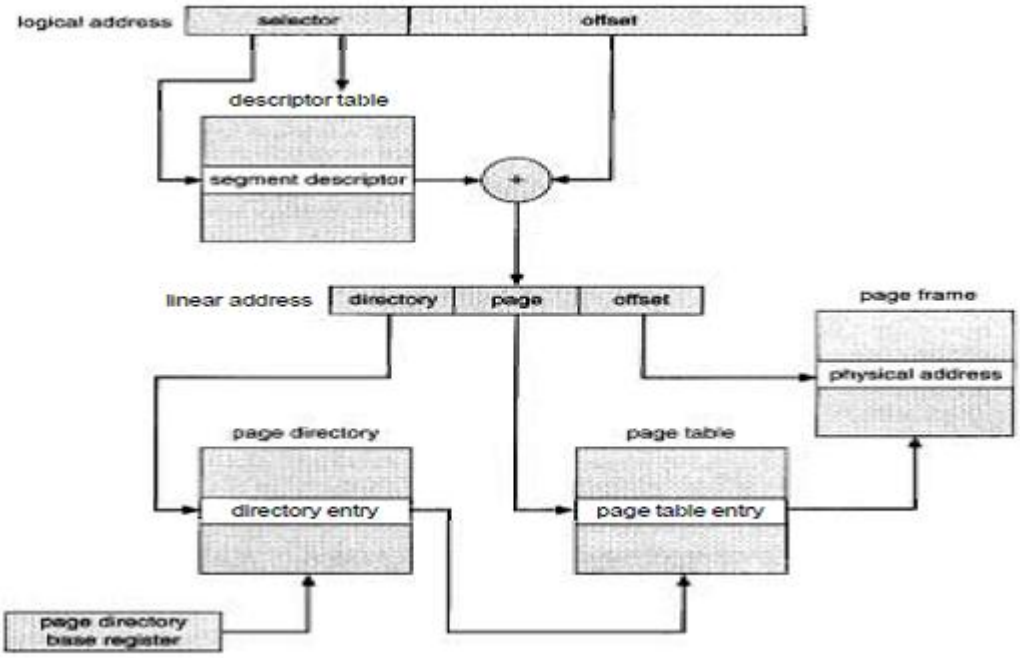


Fig 3.36 . Intel 80386 The address-translation scheme

12. Write short notes Copy on write.

1.Copy on Write:

This works by allowing the parent and child processes to initially share the same pages. These shared pages are marked as Copy on Write pages.

- Traditionally, fork() worked by creating a copy of the parent's address space for the child, duplicating the pages belonging to the parent.
- However, considering that many child processes invoke the exec() system call immediately after creation, the copying of the parent's address space may be unnecessary. Instead, we can use a technique known as **copy-on-write**, which works by allowing the parent and child processes initially to share the same pages. These shared pages are marked as copy-on-write pages, meaning that if either process writes to a shared page, a copy of the shared page is created.

Copy-on-write is illustrated in Figures 3.37 and 3.38, which show the contents of the physical memory before and after process 1 modifies page C.

For example, assume that the child process attempts to modify a page containing portions of the stack, with the pages set to be copy-on-write.

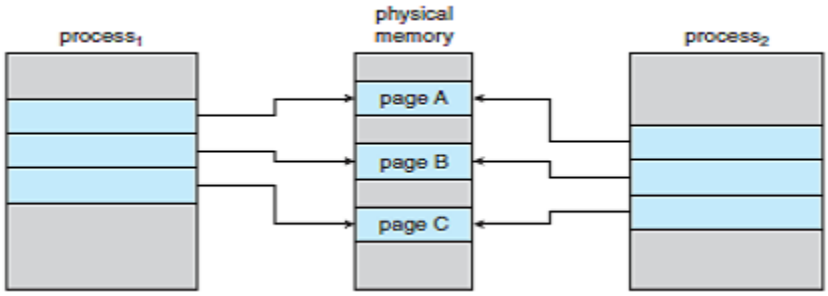


Fig 3.37 Before process1 modifies process C

The operating system will obtain a frame from the free frame list and create a copy of this page, mapping it to the address space of the child process.

- The child process will then modify its copied page and not the page belonging to the parent process. Obviously, when the copy-on-write technique is used, only the pages that are modified by either process are copied; all unmodified pages can be shared by the parent and child processes.
- Several versions of UNIX (including Linux, macOS, and BSD UNIX) provide a variation of the fork() system call—vfork() (for **virtual memory fork**)— that operates differently from fork() with copy-on-write. With vfork(), theparent process is suspended, and the child process uses the address space of the parent.
- Because vfork() does not use copy-on-write, if the child process changes any pages of the parent's address space, the altered pages will be visible to the parent once it resumes.
- Therefore, vfork() must be used with caution to ensure that the child process does not modify the address space of the parent.
- vfork() is intended to be used when the child process calls exec() immediately after creation. Because no copying of pages takes place, vfork()

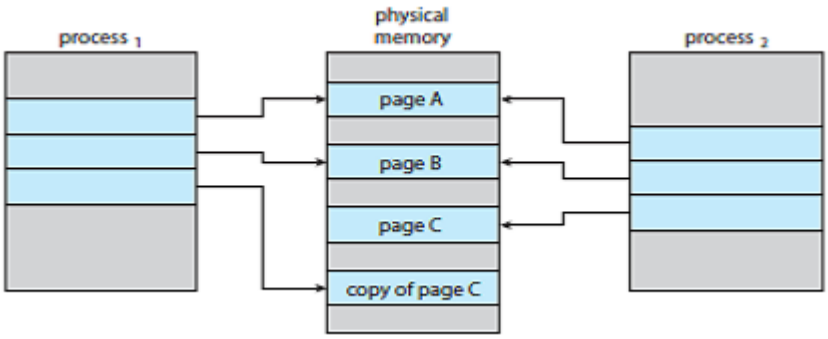


Fig 3.38 After process1 modifies process C

13. Explain About Allocation of Frames.

Consider a simple case of a system with 128 frames. The operating system may take 35, leaving 93 frames for the user process. Under pure demand paging, all 93 frames would initially be put on the free-frame list. When a user process started execution, it would generate a sequence of page faults.

- The first 93-page faults would all get free frames from the free-frame list. When the free-frame list was exhausted, a page-replacement algorithm would be used to select one of the 93 in-memory pages to be replaced with the 94th, and so on. When the process terminated, the 93 frames would once again be placed on the free-frame list.

- There are many variations on this simple strategy. We can require that the operating system allocate all its buffer and table space from the free-frame list. When this space is not in use by the operating system, it can be used to support user paging.

We can try to keep three free frames reserved on the free-frame list at all times. Thus, when a page fault occurs, there is a free frame available to page into. While the page swap is taking place, a replacement can be selected, which is then written to the storage device as the user process continues to execute. Other variants are also possible, but the basic strategy is clear: the user process is allocated any free frame.

Minimum Number of Frames:

Our strategies for the allocation of frames are constrained in various ways. We cannot, for example, allocate more than the total number of available frames (unless there is page sharing.

- We must also allocate at least a minimum number of frames. Here, we look more closely at the latter requirement.
- One reason for allocating at least a minimum number of frames involves performance. Obviously, as the number of frames allocated to each process decreases, the page-fault rate increases, slowing process execution.
- In addition, remember that, when a page fault occurs before an executing instruction is complete, the instruction must be restarted. Consequently, we must have enough frames to hold all the different pages that any single instruction can reference.

For example, consider a machine in which all memory-reference instructions may reference only one memory address. In this case, we need at least one frame for the instruction and one frame for the memory reference.

In addition, if one-level indirect addressing is allowed (for example, a load instruction on frame 16 can refer to an address on frame 0, which is an indirect reference to frame 23), then paging requires at least three frames per process. The minimum number of frames is defined by the computer architecture.

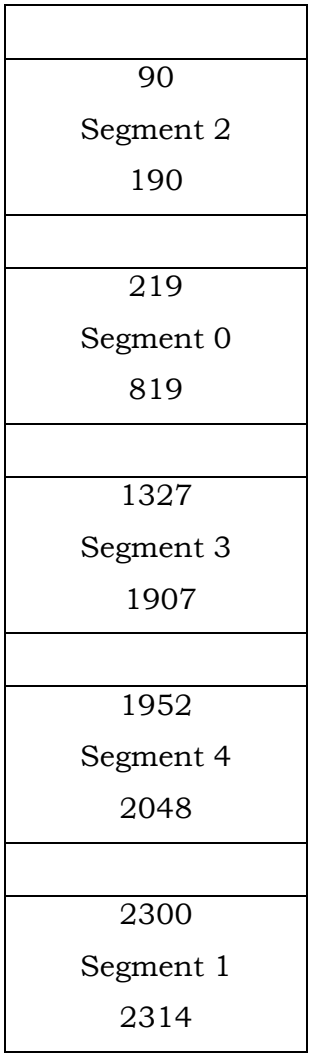
14. Consider the following segment table:[May/Jun '12](April/May-2019)

Segment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

What are the physical addresses for the following logical addresses?

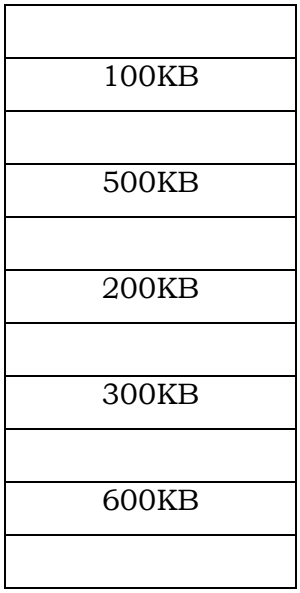
a. 0, 430 b. 1, 10 c. 2, 500 d. 3, 400 e. 4, 112

What are the physical addresses for the logical addresses 3400 and 0110?



Logical address	**Physical address**	
a.0,430	219+430=649	valid
b. 1,10	2300+10=2310	valid
c. 2,500	90+500=290	invalid
d. 3,400	1327+400=1727	valid
e. 4,112	1952+112=2065	invalid

15. Given memory partitions of 100KB, 500KB, 200KB, 300KB and 600KB (in order), how would each of the first fit, best fit and worst fit algorithms place processes of 212KB, 417KB, 112Kband 426KB(in order)? Which algorithm makes the most efficient use of memory? (Nov/Dec 2015)



Solution:

FIRST FIT:

212KB is put in 500KB partition

417KB is put in 600KB partition

112KB is put in 288KB partition (new partition 288KB=500KB-212KB)

426KB must wait

BEST FIT:

212KB is put in 300KB partition

417KB is put in 500KB partition

112KB is put in 200KB partition

426KB is put in 600KB partition

WORST FIT:

212KB is put in 600KB partition

417KB is put in 500KB partition

112KB is put in 388KB partition

426KB must wait (In this example, Best fit turns out to be the best.)

CASE STUDY

16. Consider the following page-reference string:[Apr/May-15] [Nov/Dec2015]{April/May-2019)(Nov/Dec-19)

1,2,3,4,2,1,5,6,2,1,2,3,7,6,3,2,1,2,3,6.

How many page faults would occur for the following replacement algorithms, assuming one, two, three, four, five, six, or seven frames? Remember that all frames are initially empty, so your first unique pages will all cost one fault each.

- **LRU replacement**
- **FIFO replacement**
- **Optimal replacement**

(or)

When do the page faults occurs? Consider the reference string:
1, 2, 3, 4, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6
Explain the occurrence of page faults and page fault rate for FIFO, LRU and optimal replacement algorithms, assuming one, two, three, four-page frames. (Nov/Dec 2024)

(or)

Explain the concept of Paged Memory Management and solve the following problem using virtual memory concept.
Assume a process contains 6 virtual pages on disk and is assigned a fixed allocation of page frame in main memory. The following page trace occurs: 1,2,4,1,6,5,3,4,5,6,4,6,4,3,2,3,2,5.
Show the successive pages residing in the three and four frames using (i) OPT (ii) FIFO (iii) LRU. (1+4+4+4) (April/May 2024)- Refer Class notes
Sol:

a) One frames:

As there is no page referenced more than once all the algorithms viz LRU, FIFO and Optimal will result in 20 page faults.

b) Two frames:

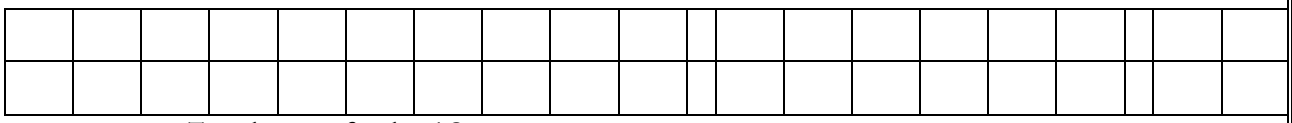
LRU:

	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6

Total page fault: 18

FIFO:

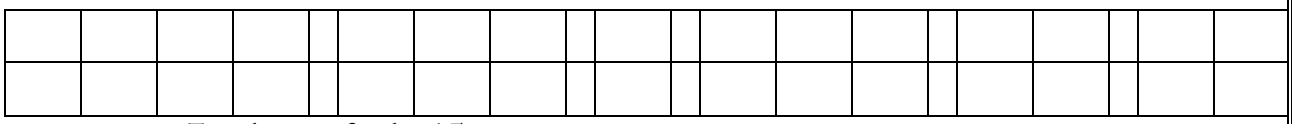
1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6



Total page fault: 18

Optimal:

1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6

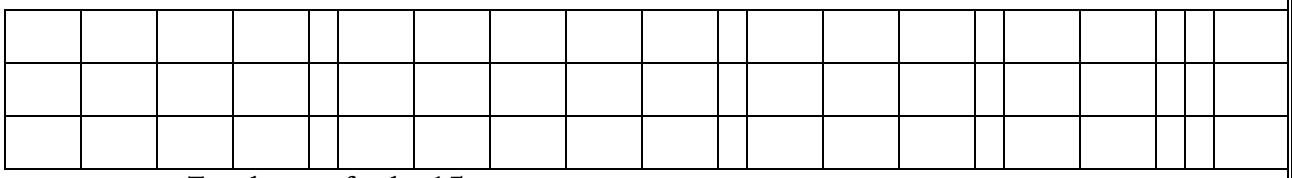


Total page fault: 15

c) **Three frames**:

LRU:

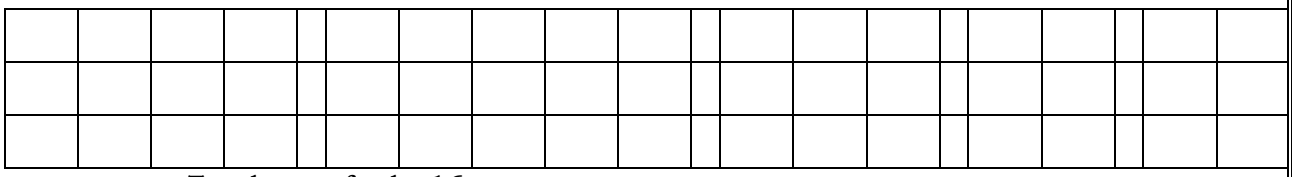
1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6



Total page fault: 15

FIFO:

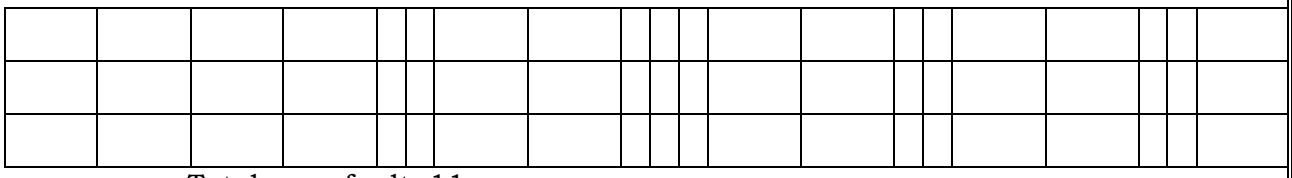
1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6



Total page fault: 16

Optimal:

1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6

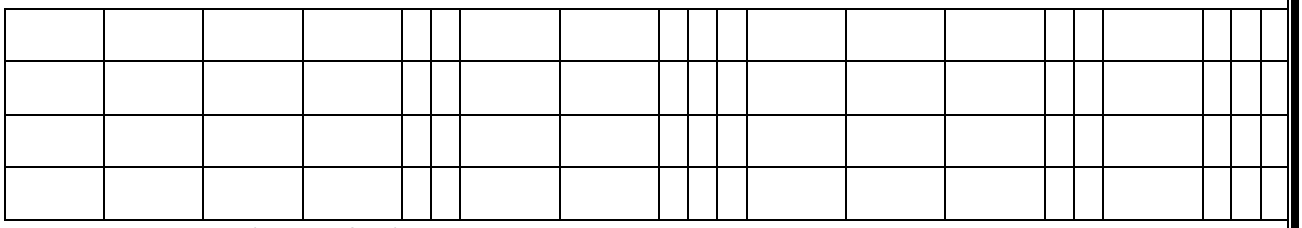


Total page fault: 11

d) Four frames:

LRU:

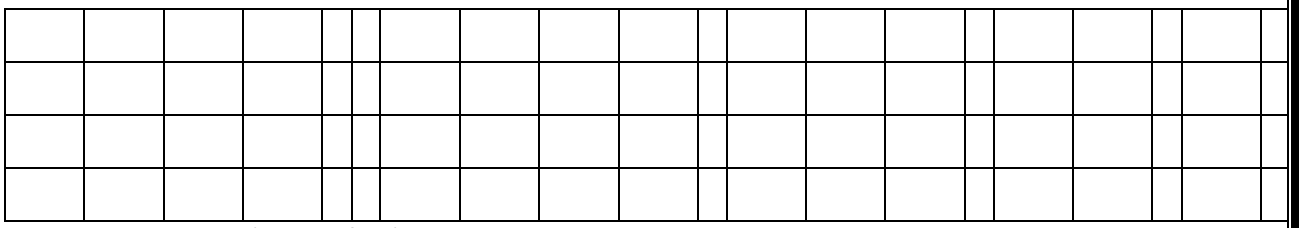
1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6



Total page fault: 10

FIFO:

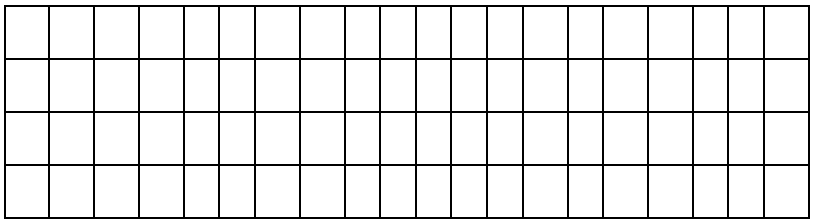
1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6



Total page fault: 14

Optimal:

1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6

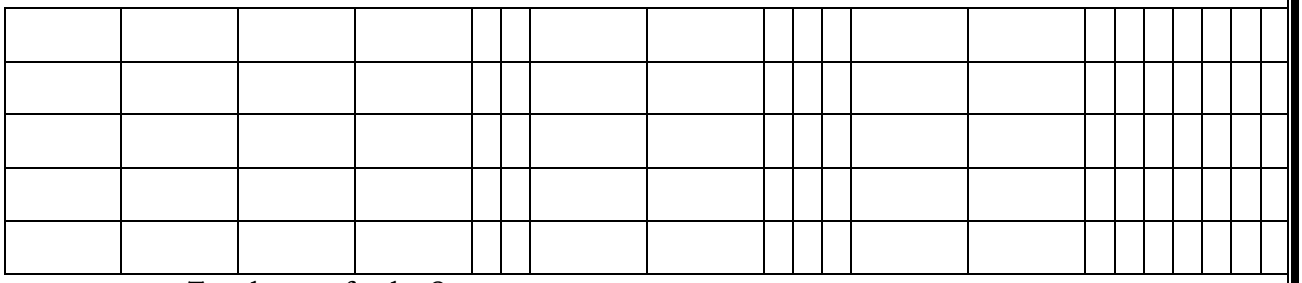


Total page fault: 8

e) Five frames:

LRU:

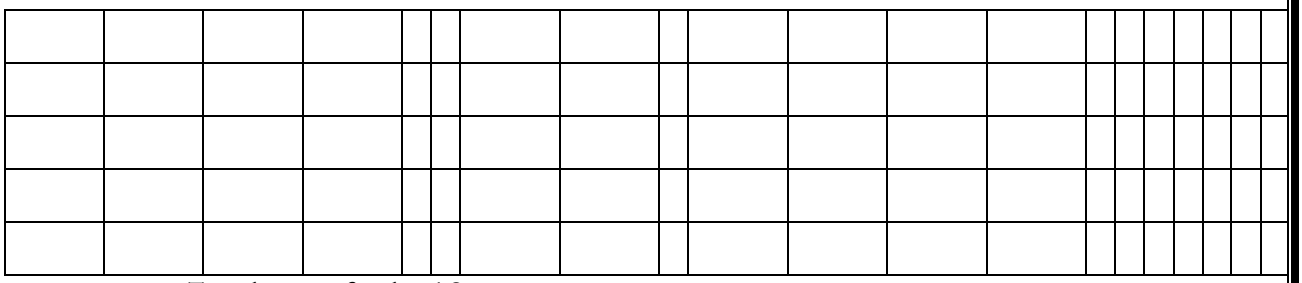
1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6



Total page fault: 8

FIFO:

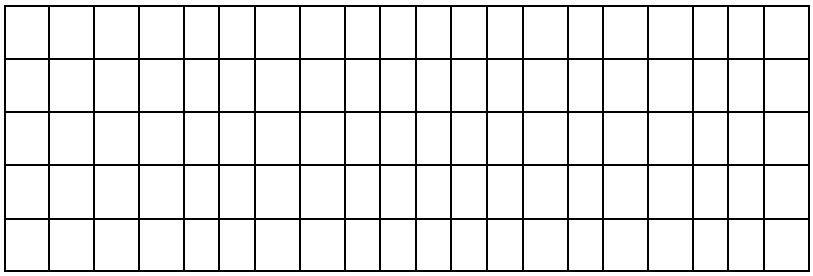
1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6



Total page fault: 10

Optimal:

1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6

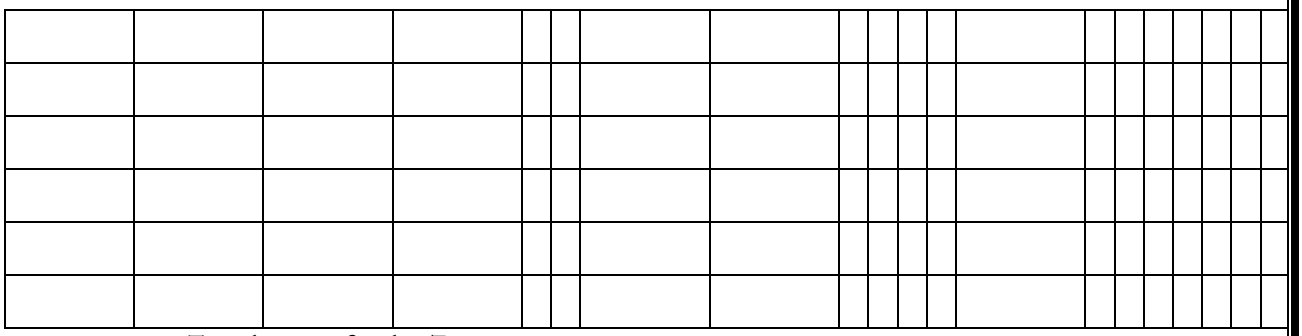


Total page fault: 7

f) Six frames:

LRU:

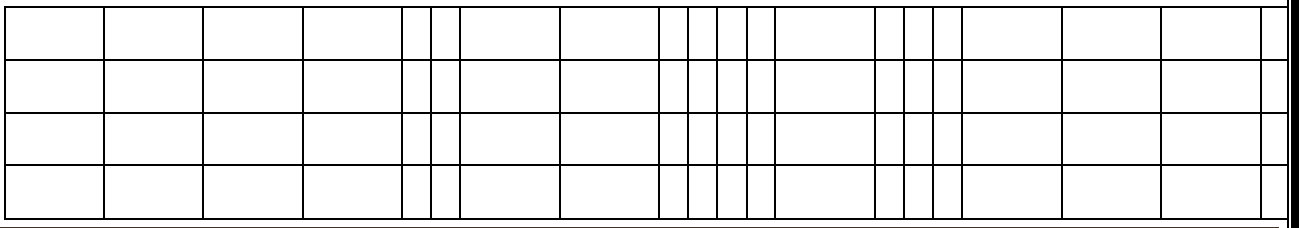
1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6

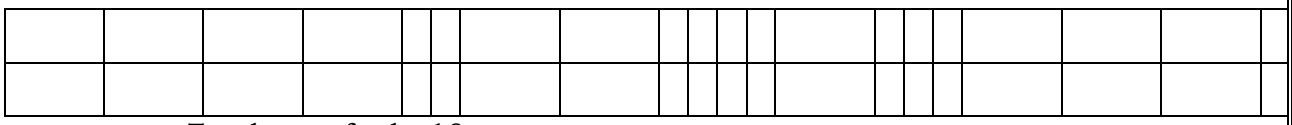


Total page fault: 7

FIFO:

1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6

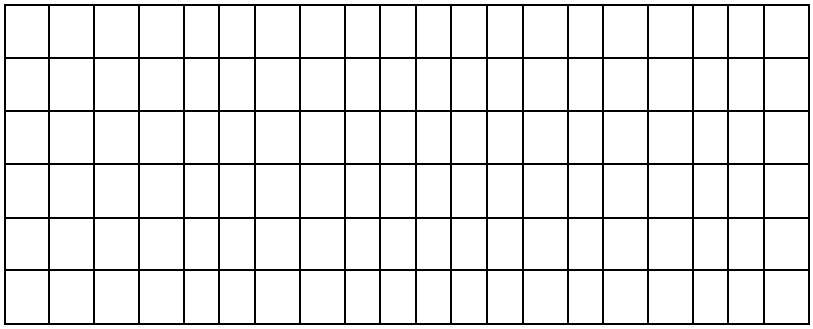




Total page fault: 10

Optimal:

1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6

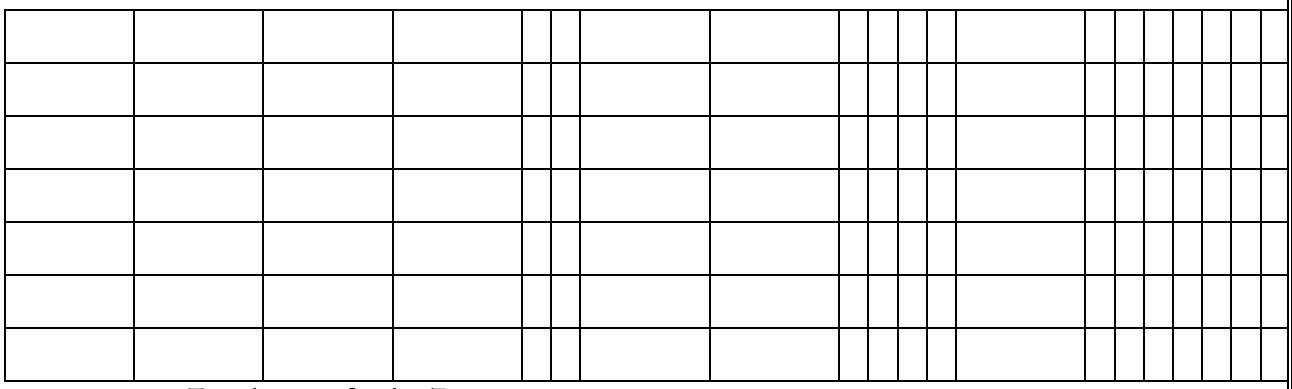


Total page fault: 7

g) Seven frames:

LRU:

1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6



Total page fault: 7

FIFO and Optimal will also be same as above.

UNIT IV STORAGE MANAGEMENT

Mass Storage system – Disk Structure - Disk Scheduling and Management; File-System Interface - File concept - Access methods - Directory Structure - Directory organization - File system mounting - File Sharing and Protection; File System Implementation - File System Structure - Directory implementation - Allocation Methods - Free Space Management; I/O Systems – I/O Hardware, Application I/O interface, Kernel I/O subsystem.

PARTA

1. State the typical bad-sector transactions.

- Bad sector in computing refers to a disk sector on a disk storage unit that is permanently damaged.
- Upon taking damage, all information stored on that sector is lost. When a bad sector is found and marked, the operating system skips it in the future.

2. What is the advantage of bit vector approach in free space management?

Bitmap or Bit vector

- A Bitmap or Bit Vector is series or collection of bits where each bit corresponds to a disk block. The bit can take two values: 0 and 1: 0 indicates that the block is allocated and 1 indicates a free block.
- The given instance of disk blocks on the disk in Figure 1 (where green blocks are allocated) can be represented by a bitmap of 16 bits as: 0000111000000110.

Advantages

- Simple to understand.
- Finding the first free block is efficient. It requires scanning the words (a group of 8 bits) in a bitmap for a non-zero word. (A 0-valued word has all bits 0). The first free block is then found by scanning for the first 1 bit in the non-zero word.

3. **Mention the significance of LDT and GDT in segmentation?**
    - LDT (Local Descriptor Table) Register **and contains the linear base address and limit for the LDT**. However, LDT takes a selector as input (which must point into the GDT), not the plain base and limit.
    - Multiple LDTs can be defined in the GDT, but only one is current at any one time: usually associated with the current Task. While the LDT contains memory segments which are private to a specific program, the GDT contains global segments.

4. **List the major attributes and operations of a file systems?**
    - A file is a collection of related information.
    - The attributes of a file are Name, identifier, type, location, size, protection, time, date and user identification

    Operations
    - Creating a file
    - Writing a file
    - Reading a file
    - Repositioning within a file
    - Deleting a file
    - Truncating a file

5. **Suppose that the disk rotates at 7200rpm. What is the average rotational latency of the disk drive. Identify seek distance can be covered in the time?**
    - 7200 rpm gives 120 rotations per second. Thus, a full rotation takes 8.33 ms, and the average rotational latency (a half rotation) takes 4.167 ms.

6. **Enlist various different types of Directory Structure. (April/May 2024)**
    1. Single-level directory
    2. Two-level directory
    3. Tree-Structured directory
    4. Acyclic Graph directory
    5. General Graph directory

7. Differentiate between File and Directory.

- A file is any kind of computer document and a directory is a computer document folder or filing cabinet.
- All the data on your hard drive consists of files and folders.
- The basic difference between the two is that files store data, while folders store files and other folders.
- The folders, often referred to as directories, are used to organize files on your computer.

8. Define C-SCAN scheduling.

The elevator algorithm (also SCAN) is a disk scheduling algorithm to determine the motion of the disk's arm and head in servicing read and write requests.

9. Why is it important to scale up system bus and device speeds as CPU speed increases? (April/May 2024)

- Consider a system which performs 50% I/O and 50% computes. Doubling the CPU performance on this system would increase total system performance by only 50%.
- Doubling both system aspects would increase performance by 100%.
- Generally, it is important to remove the current system bottleneck, and to increase overall system performance.

10. Why rotational latency is usually not considered in disk scheduling? How would you modify SSTF, SCAN, and C-SCAN to include latency optimization?

- Most disks do not export their rotational position information to the host.
- Even if they did, the time for this information to reach the scheduler would be subject to imprecision and the time consumed by the scheduler is variable, so the rotational position information would become incorrect.
- Further, the disk requests are usually given in terms of logical block numbers, and the mapping between logical blocks and physical locations is very complex.

11. How does DMA increase system concurrency?

- DMA increases system concurrency by allowing the CPU to perform tasks while the DMA system transfers data via the system and memory buses.
- Hardware design is complicated because the DMA controller must be integrated into the system and the system must allow the DMA controller to be a bus master.

12. What is a file management system?

- A file is a named collection of related information that is recorded on secondary storage.
- A file contains either programs or data.
- A file has certain "structure" based on its type.

13. What are the responsibilities of a file manager?

- Track where each file is stored
- Allocate each file when a user has been cleared for access to it, and then record its use
- Deallocate the file when it is returned to storage.

14. Define rotational latency.

The rotational latency is the additional time waiting for the disk to rotate the desired sector to the disk head.

15. What is meant by free space management?

- Disk space is limited; we need to reuse the space from deleted files for new files, if possible.
- To keep track of free disk space, the system maintains a free-space list.
- The free-space list records all free disk blocks – those not allocated to some file or directory.

16. What is need for disk scheduling?

- For a multiprogramming system with many processes, the disk queue may often have several pending requests.
- When one request is completed the operating system chooses

which pending request to service next.

- To reduce seek time and increase disk bandwidth disk scheduling is required.

17. What is meant by polling?

- Polling is a communications technique that determines when a terminal is ready to send data.
- If a terminal has data to send, it sends back an acknowledgment and the transmission begins.

18. State any three advantages & disadvantages of placing functionality in a device controller rather than in the kernel.

Advantages

1. Bugs are less likely to cause an operating system crash
2. Performance can be improved by utilizing dedicated hardware and hard coded algorithms
3. The kernel is simplified by moving algorithms out of it

Disadvantages

1. Bugs are harder to fix a new firmware version or new hardware is needed
2. Improving algorithms likewise require a hardware update rather than just a kernel or device-driver update
3. Embedded algorithms could conflict with application's use of the device, causing decreased performance.

19. What file allocation strategy is most appropriate for random access files?

Contiguous allocation as you do not have to follow a linked list to data, it can be randomly accessed.

20. What is rotational latency?

Once the head is at right track, it must wait until the desired block rotates under the read- write head. This delay is latency time.

21. Define seek time.

The time taken by the head to move to the appropriate cylinder or track is called seek time.

22. What is sector sparing?

- Low-level formatting also sets aside spare sectors not visible to the operating system.
- The controller can be told to replace each bad sector logically with one of the spare sectors. This scheme is known as sector sparing.

23. Differentiate absolute path from relative path.

- An absolute path begins at the root and follows a path down to the specified file, giving the directory names on the path.
- A relative path defines a path from the current directory.

Eg. Absolute path: root\spell\mail\prt\first Relative path: prt\first

24. Define UFD and MFD.

- In the two-level directory structure, each user has her own user file directory (UFD).
- Each UFD has a similar structure, but lists only the files of a single user.
- When a job starts the system's master file directory (MFD) is searched.
- The MFD is indexed by the user name or account number, and each entry points to the UFD for that user.

25. What is a path name?

- A pathname is the path from the root through all subdirectories to a specified file.
- In a two-level directory structure a user name and a file name define a path name.

26. What file allocation strategy is most appropriate for random access files?

- Contiguous allocation as you do not have to follow a linked list to data, it can be randomly accessed.

27. Compare bitmap-based allocation of blocks on disk with a free block list.

- Bit-map-based allocation maintains the order of the free blocks, easily supporting contiguous block allocation, but sacrifices time required to find free space and to release space, they are no longer order 1 as the bit-map must be traversed.
- A free block-list after a period of time during which numerous allocations and deallocations take place will result in a random list of free space. This does not support contiguous allocation as the free list will force the FS to place data randomly.

28. Enlist different types of file directory structure.

- Single-Level Directory.
- Two-Level Directory.
- Tree-Structured Directory.
- Acyclic Graph Directory.
- General-Graph Directory.

29. Is FAT file system advantageous? Justify.

FAT32 is best for cross-compatibility with other platforms. There are NTFS file system drivers for Linux, but not really for Windows Me. FAT32, however, can be read more or less transparently by both operating systems.

30. Name any three file attributes.

The attributes of a file are Name, identifier, type, location, size, protection, time, date and user identification

31. What is file mounting?

- Mounting refers to making a group of files in a file system structure accessible to user or group of users. It is done by

attaching a root directory from one file system to that of another. This ensures that the other directory or device appears as a directory or subdirectory of that system.

32. Write short notes on file system mounting?

- Mounting a file system attaches that file system to a directory (mount point) and makes it available to the system. The root (/) file system is always mounted. Any other file system can be connected or disconnected from the root (/) file system.

33. What is SSD?

(Solid State Drive)SSDs **use flash-based memory**, which is much faster than a traditional mechanical hard disk. Upgrading to an SSD is one of the best ways to speed up your computer. Learn how SSDs work and how to keep them optimized with a specialized performance-boosting tool.

34. What is the reason that all I/O operations are done in privileged mode? (April/May 2024)

All I/O operations are performed in privileged mode because it protects the system from potential errors or malicious activity by ensuring that only the operating system (kernel) can directly access hardware devices, preventing user applications from manipulating them without proper authorization, thereby maintaining system stability and security.

35. List and define the various file access methods. (April/May 2024)

The primary file access methods are:

- Sequential Access,
- Direct Access (Random Access), and
- Indexed Sequential Access (ISAM),

where sequential access reads data in order from the beginning of a file, direct access allows random access to any record within a file, and indexed sequential combines features of both by using an index to quickly locate specific records within a sequential file.

PART B

1. Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 37, 122, 14, 124, 65, 67. If the disk head is start at 53, then find out the total head movement with respect to FCFS, SSTF, SCAN, C-SCAN and LOOK scheduling. State and explain the FCFS, SSTF and SCAN disk scheduling with examples. (or) State and describe the FCFS, SSTF, and SCAN disk scheduling with examples. (Nov/Dec 2024)

Disk scheduling:

One of the responsibilities of the operating system is to use the hardware efficiently. For the disk drives,

1. A fast access time and
2. High disk bandwidth.

1. The **access time** has two major components;
   - The seek time is the time for the disk arm to move the heads to the cylinder containing the desired sector.
   - The rotational latency is the additional time waiting for the disk to rotate the desired sector to the disk head.
2. The **disk bandwidth** is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.
   - We can improve both the access time and the bandwidth by disk scheduling.
   - Disk scheduling: Servicing of disk I/O requests in a good order.

FCFS Scheduling: The simplest & fastest form of disk scheduling as shown in fig 4.1

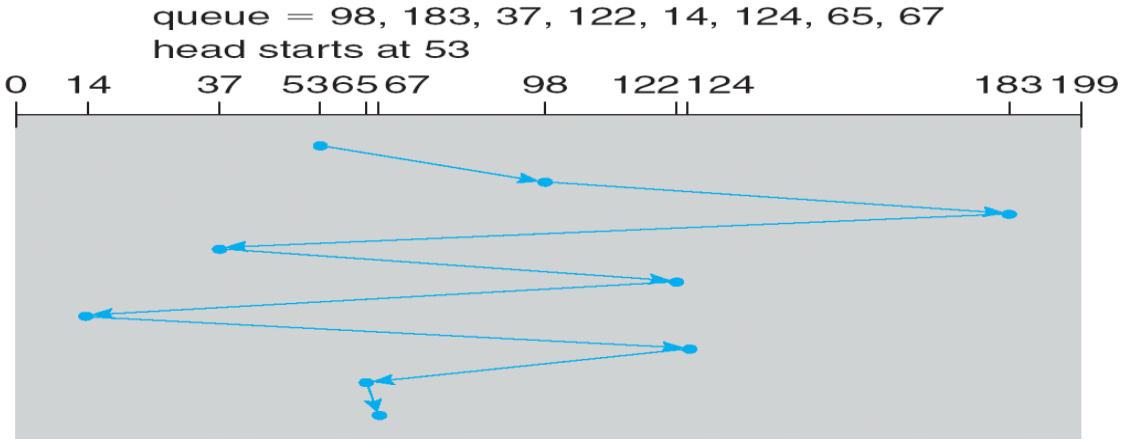


Fig.4.1 FCFS Scheduling

Total head movement = 640 cylinders

SSTF (shortest-seek-time-first) Scheduling

- Service all the requests close to the current head position, before moving the head far away to service other requests.
- That is selects the request with the minimum seek time from the current head position as shown in fig 4.2.

Total head movement =236 cylinders

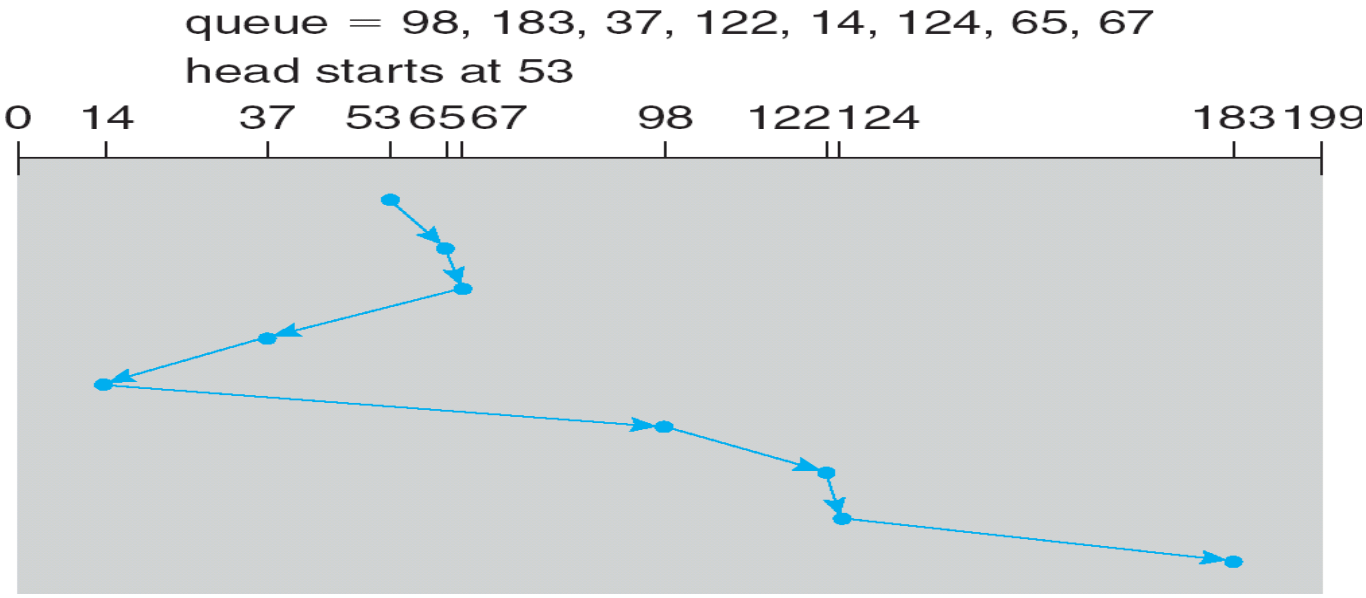


Fig.4.2. SSTF scheduling

SCAN Scheduling

The disk head starts at one end of the disk, and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk as shown in figure 4.3.

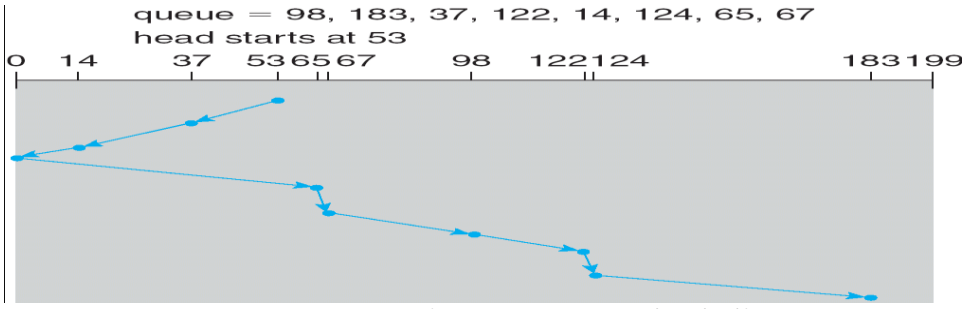


Fig.4.3 Scan Scheduling

Total head movement = 236 cylinders

Elevator algorithm: Sometimes the SCAN algorithm is called as the elevator algorithm, since the disk arm behaves just like an elevator in a building, first servicing all the requests going up, and then reversing to service requests the other way.

C-SCAN Scheduling

Variant of SCAN designed to provide a more uniform wait time. It moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip as shown in figure 4.4.

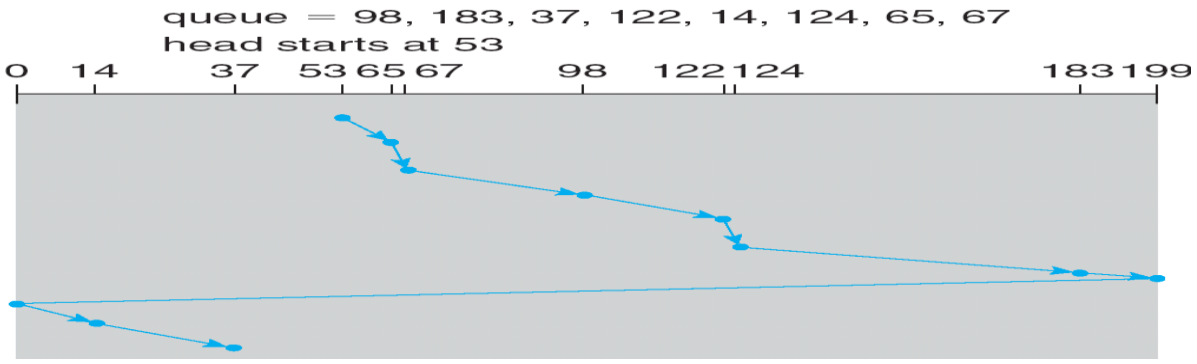


Fig.4.4 C-SCAN Scheduling

Total head movement =230 cylinders

C-LOOK Scheduling

Both SCAN and C-SCAN move the disk arm across the full width of the disk. In this, the arm goes only as far as the final request in each direction. Then, it reverses direction immediately, without going all the way to the end of the disk as shown 4.5.

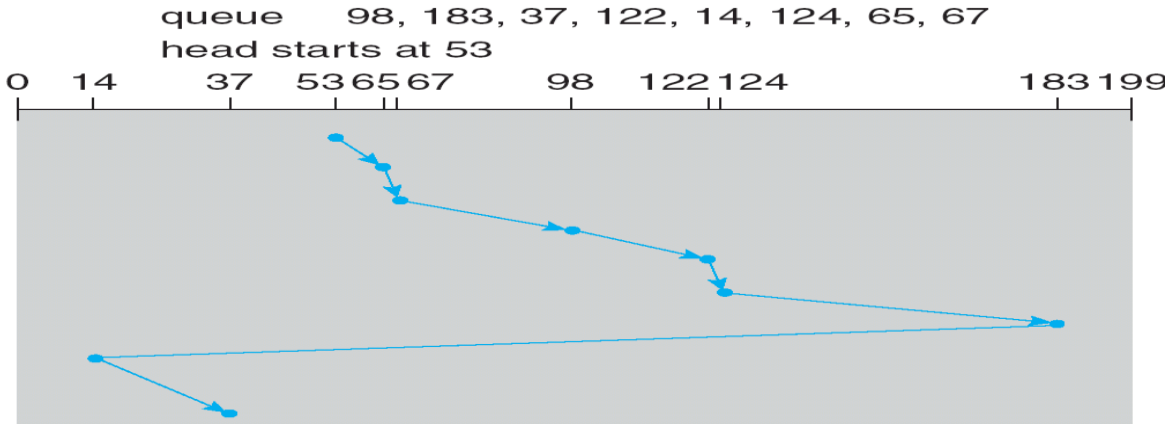


Fig.4.5 CLOOK Scheduling

Total head movement =230 cylinders

2. Explain in detail about disk management

Disk Management has two main sections- a top and a bottom:

- The bottom section of Disk Management contains a graphical representation of the physical drives installed in the computer.
- The top section of Disk Management contains a list of all the partitions, formatted or not, that Windows recognizes.

Disk formatting:

- A new magnetic disk is a blank slate: It is just a platter of a magnetic recording material.
- Before a disk can store data, it must be divided into sectors that the disk controller can read and write. This process is called low-level formatting, or physical formatting.
- Low-level formatting fills the disk with a special data structure for each sector.
- The data structure for a sector typically consists of a header, a data area (usually 512bytes in size), and a trailer.
- The header and trailer contain information used by the disk controller, such as a sector number and an error-correcting code (ECC).

To use a disk to hold files, the operating system still needs to record its own data structures on the disk.

- The first step is **Partition** the disk into one or more groups of cylinders. Among the partitions, one partition can hold a copy of the OS‘s executable code, while another holds user files.
- The second step is **logical formatting**. The operating system stores the initial file-system data structures onto the disk. These data structures may include maps of free and allocated space and an initial empty directory

The bootstrap is stored in read-only memory **(ROM).**

Advantages:

- ROM needs no initialization.
- It is at a fixed location that the processor can start executing when powered up or reset.
- It cannot be infected by a computer virus. Since, ROM is read only.

- The full bootstrap program is stored in a partition called the boot blocks, at a fixed location on the disk. A disk that has a boot partition is called a boot disk or system disk.

Bootstrap loader:

Load the entire operating system from a non-fixed location on disk, and to start the operating system running.

Bad Blocks:

The disk with defected sector is called as bad block.

Bad blocks handled by the following ways

1. Handled manually

- If blocks go bad during normal operation, a **special program** must be run manually to search for the bad blocks and to lock them away as before.
- Data that resided on the bad blocks usually are lost.

2. Sector sparing or forwarding

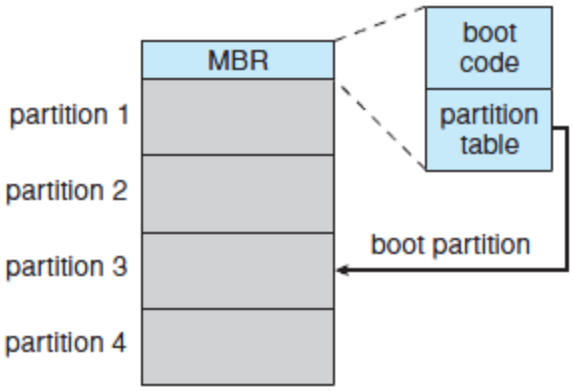


Fig.4.6 Booting from disk in Windows

- The controller maintains a list of bad blocks on the disk.
- Then the controller can be told to replace each bad sector logically with one of the spare sectors as shown in figure 4.6.
- This scheme is known as sector sparing or forwarding.

3. Sector slipping

- Suppose that logical block 17 becomes defective, and the first available spare follows sector 202.
- Then, sector slipping would remap all the sectors from 17 to 202, moving them all down one spot.

- That is, sector 202 would be copied into the spare, then sector 201 into 202, and then 200 into 201, and so on, until sector 18 is copied into sector 19.
- Slipping the sectors in this way frees up the space of sector 18, so sector 17 can be mapped to it.

3. Write briefly about file attributes, operations, types and structure.

File Concept

- A file is a named collection of related information that is recorded on secondary storage.
- From a user's perspective, a file is the smallest allotment of logical secondary storage; that is, data cannot be written to secondary storage unless they are within a file.

Examples of files:

- A text file is a sequence of characters organized into lines (and possibly pages).
- A source file is a sequence of subroutines and functions, each of which is further organized as declarations followed by executable statements.
- An object file is a sequence of bytes organized into blocks understandable by the system's linker.
- An executable file is a series of code sections that the loader can bring into memory and execute.

File Attributes

- **Name:** The symbolic file name is the only information kept in human readable form.
- **Identifier:** This unique tag, usually a number identifies the file within the file system. It is the non-human readable name for the file.
- **Type:** This information is needed for those systems that support different types.
- **Location:** This information is a pointer to a device and to the location of the file on that device.
- **Size:** The current size of the file (in bytes, words or blocks) and possibly the maximum allowed size are included in this

attribute.

- **Protection:** Access-control information determines who can do reading, writing, executing and soon.
- **Time, date and user identification:** This information may be kept for creation, last modification and last use.

File Operations

- Creating a file
- Writing a file
- Reading a file
- Repositioning within a file
- Deleting a file
- Truncating a file

File types

File type	**Usual extension**	**Function**
Executable	e co, bin, Or x	Read to run machine language program
Object	obj,o	Compiled, machine language,
Source code	C, cc, java, pas	Source code in various languages
Batch	bat, sh	Commands to the command interpreter
Text	txt, doc	Textual data, documents
Word processor	wp, tex, rrf, doc	Various word-processor formats
Library	lib, a, so, dll, mpeg, mov, rm	Libraries of routines for programmers
print or view	arc, zip,tar	ASCII or binary file in a format for printing or viewing
Archive	arc, zip, tar	Related files grouped into one f sometimes compressed, storage
Multimedia	mpeg, mov, rm	Binary file audio or containing A/V

4. Explain the different file access methods in detail. (or) Explain about various types of file access methods.

Access Methods

1. Sequential Access

The simplest access method is sequential access. Information in the file is processed in order, one record after the other as shown in fig 4.7.

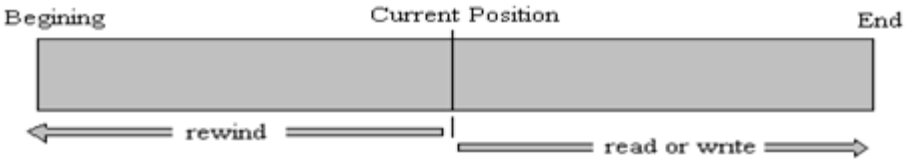


Fig.4.7 Sequential Access File

- The bulk of the operations on a file is reads and writes.
- A read operation reads the next portion of the file and automatically advances a file pointer, which tracks the I/O location.
- Similarly, a write appends to the end of the file and advances to the end of the newly written material (the new end of file).

2. Direct Access

- Direct access (or relative access). A file is made up of fixed length logical records that allow programs to read and write records rapidly in no particular order.
- The direct- access methods is based on a disk model of a file, since disks allow random access to any file block.
- For direct access, the file is viewed as a numbered sequence of blocks or records. A direct- access file allows arbitrary blocks to be read or written.
- Thus, we may read block 14, then read block 53, and then write block7. There are no restrictions on the order of reading or writing for a direct-access file.

3. Other Access methods

- Other access methods can be built on top of a direct access method these methods generally involve the construction of an index for the file.
- An index in the back of a book contains pointers to the various blocks in find a record in the file.

- We first search the index, and then use the pointer to access the file directly and the find the desired record as shown in fig 4.8.

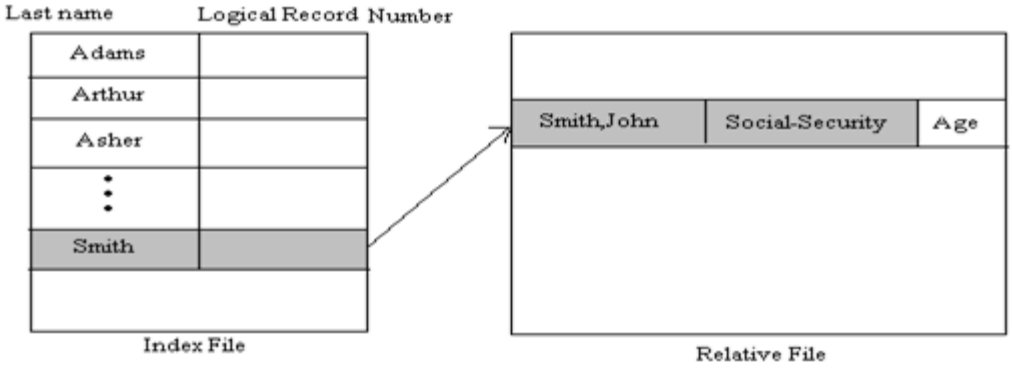


Fig.4.8 Example of Index and Relative File

5. Briefly discuss about the various directory structures.

With a diagram discuss about the tree structured directory structure.

What do you mean by directory structure? Also discuss Tree-Structured Directories and Acyclic-Graph Directories.

Directory Overview

The directory can be viewed as a symbol table that translates file names into their directory entries.

Directory Structure

There are five directory structures.

- Single-level directory
- Two-level directory
- Tree-Structured directory
- Acyclic Graph directory
- General Graph directory

1. Single – Level Directory

- The simplest directory structure is the single- level directory as shown in figure 4.9.
- All files are contained in the same directory.
- Disadvantage:
  - When the number of files increases or when the system has more than one user, since all files are in the same directory, they must have unique names.

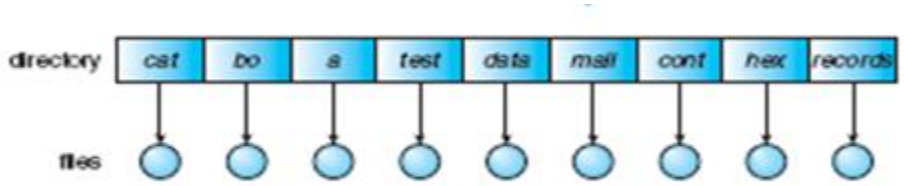


Fig.4.9 Single Level Directory

2. Two – Level Directory

- In the two level directory structures, each user has own user file directory (UFD) as shown in fig 4.10.
- When a user job starts or a user log in, the system's master file directory (MFD) is searched.
- The MFD is indexed by user name or account number, and each entry points to the UFD for that user.
- When a user refers to a particular file, only his own UFD is searched. Thus, different users may have files with the same name.
- Although the two – level directory structure solves the name-collision problem

Disadvantage: Users cannot create their own sub-directories.

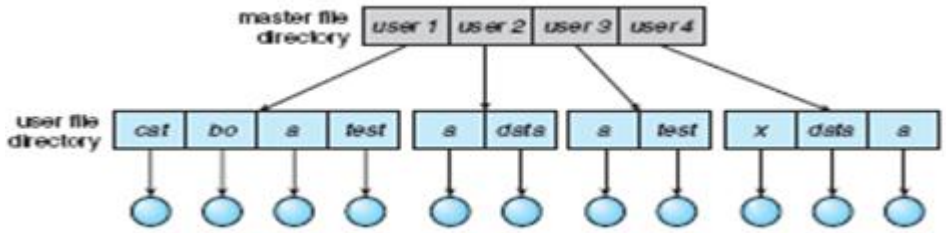


Fig.4.10 Two Level Directory

3. Tree–Structured Directory

- The tree has a root directory. Every file in the system has a unique path name.
- A path name is the path from the root, through all the subdirectories to a specified file.
- A directory (or sub directory) contains a set of files or sub directories.
- Path names can be of two types:
    - absolute path names
    - relative path names

- An absolute path name begins at the root and follows a path down to the specified file, giving the directory names on the path as shown in figure 4.11.
- A relative path name defines a path from the current directory.

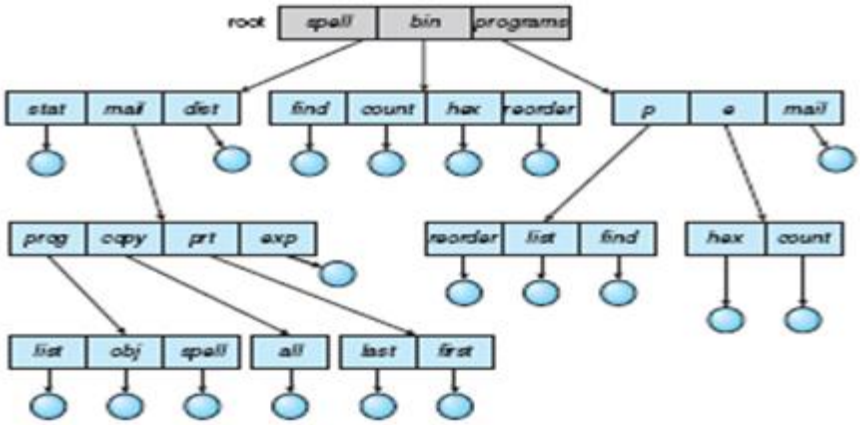


Fig.4.11 Tree Structured Directory Structure

4. Acyclic Graph Directory.

- An acyclic graph is a graph with no cycles as show in figure 4.12.
- To implement shared files and subdirectories this directory structure is used.
- An acyclic – graph directory structure is more flexible than is a simple tree structure, but it is also more complex.
- Advantage of an acyclic graph is the relative simplicity of the algorithms to traverse the graph and to determine when there are no more references to a file.

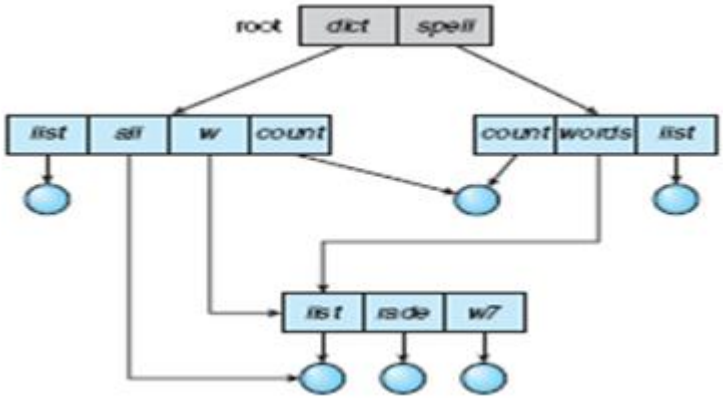


Fig.4.12 Acyclic Graph Directory

5. General Graph Directory

- If cycles are allowed to exist in the directory as shown in figure 4.13.
- A poorly designed algorithm might result in an infinite loop continually searching through the cycle and never terminating.

  One solution is to limit arbitrarily the number of directories that will be accessed during a search.
- Garbage collection (Collection of unwanted files) involves traversing the entire file system, marking everything that can be accessed.

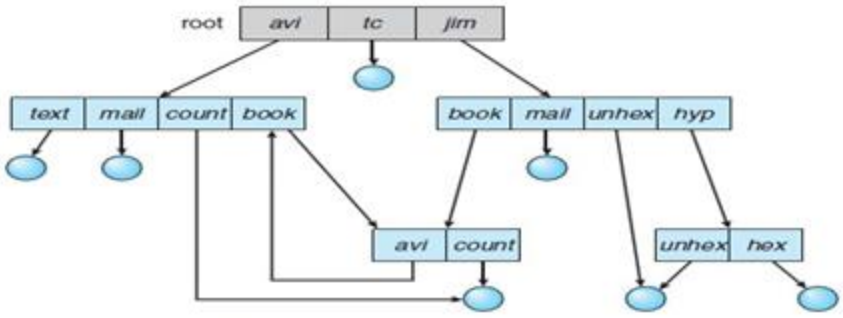


Fig.4.13 General Graph Directory

6. Explain in detail about file mounting.

File System Mounting

- Just as a file must be opened before it is used, a file system must be mounted before it can be available to processes on the system.
- The mount procedure is straightforward. The operating system is given the name of the device, and the location within the file structure at which to attach the File system (or mount point).
- A mount point is an empty directory at which the mounted file system will be attached.

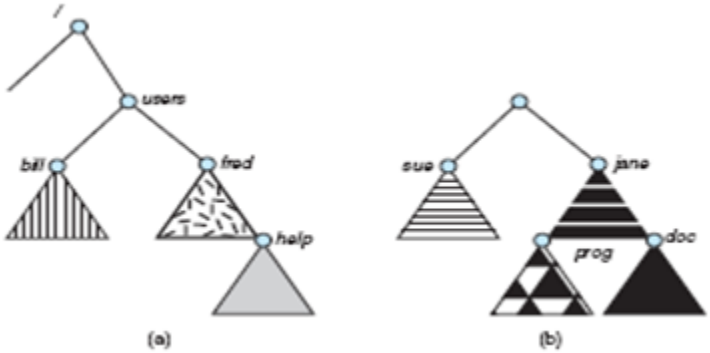


Fig.4.14. File System a) Existing System b) Unmounted Volume

- Where the triangles represent sub trees of directories that are of interest.

- Figure 4.14(a) shows an existing file system,
- While Figure 4.14(b) shows an un mounted volume residing on /device/dsk.
- At this point, only the files on the existing file system can be accessed.
- Figure 4.15 shows the effects of mounting the volume residing on /device/dsk over /users.

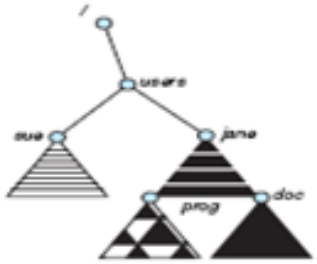


Fig.4.15 Mount Point

- Systems impose semantics to clarify functionality.
- For example, a system may allow the same file system to be mounted repeatedly, at different mount points; or it may only allow one mount per file system.

7. Explain in detail about file sharing.

File Sharing

1. Multiple Users:

- To implementing sharing and protection, the system must maintain more file and directory attributes than on a single-user system.
- The owner is the user who may change attributes, grand access, and has the most control over the file or directory.
- The group attribute of a file is used to define a subset of users who may share access to the file.
- Most systems implement owner attributes by managing a list of user names and associated user identifiers.
- Every user can be in one or more groups, depending upon operating system design decisions. The user's group Ids is also included in every associated process and thread.

2. Remote File System:

- Networks allowed communications between remote computers.
- User manually transfer files between machines via programs like **ftp**.
- A **distributed file system** (DFS) in which remote directories is visible from the local machine.
- The **World Wide Web**: A browser is needed to gain access to the remote file and separate operations are used to transfer files.

It has four types.

a) The client-server Model:

- Remote file systems allow a computer to mount one or more file systems from one or more remote machines.
- A server can serve multiple clients, and a client can use multiple servers, depending on the implementation details of a given client server facility.
- Client identification is more difficult. Clients can be specified by their network name or other identifier, such as IP address.

b) Distributed Information systems:

- To provide a unified access to the information needed for remote computing.
- Domain name system (DNS) provides host-name-to-network address translations for their entire Internet (including the World Wide Web).

c) Failure Modes:

- **Redundant arrays of inexpensive disks** (**RAID**) can prevent the loss of a disk from resulting in the loss of data.
- Remote file system has more failure modes. By nature of the complexity of networking system and the required interactions between remote machines, many more problems can interfere with the proper operation of remote file systems.

d) Consistency Semantics:

- These semantics should specify when modifications of data by one user are observable by other users.
- The semantics are typically implemented as code with the file system.

It has three types

1. **UNIX Semantics:**
   - Writes to an open file by a user are visible immediately to other users that have this file open at the same time.
   - One mode of sharing allows users to share the pointer of current location into the file.
2. **Session Semantics:**
   - Writes to an open file by a user are not visible immediately to other users that have the same file open simultaneously.
   - Once a file is closed, the changes made to it are visible only in sessions starting later.
3. **Immutable shared File Semantics:**
   - Once a file is declared as shared by its creator, it cannot be modified.

8. Explain in detail about file protection.

File Protection

(i) Need for file protection.

When information is kept in a computer system, we want to keep it safe from physical damage (reliability) and improper access (protection).

1. Reliability is generally provided by duplicate copies of files. Many computers have system programs that automatically copy disk files to tape at regular intervals to maintain a copy should a file system be accidentally destroyed.
2. File systems can be damaged by hardware problems (such as errors in reading or writing), power surges or failures, head crashes, dirt, temperature extremes, and vandalism. Files may be deleted accidentally. Bugs in the file system software can also cause file contents to be lost.
3. Protection can be provided in many ways. For a small single user system, we might provide protection by physically removing the floppy disks and locking them in a desk drawer or file cabinet.

Several different types of operations may be controlled:

1. Read: Read from the file.
2. Write: Write or rewrite the file.
3. Execute: Load the file into memory and execute it.

4. Append: Write new information at the end of the file.
5. Delete: Delete the file and free its space for possible reuse.
6. List: List the name and attributes of the file.

Access Control

- Associate with each file and directory an Access-Control List (ACL) specifying the user name and the types of access allowed for each user.
- When a user requests access to a particular file, the operating system checks the access list associated with that file.
- If that user is listed for the requested access, the access is allowed.
- Otherwise, a protection violation occurs and the user job is denied access to the file.

To condense the length of the access control list, many systems recognize three classifications of users in connection with each file.

Owner: The user who created the file is the owner.

Group: A set of users who are sharing the file and need similar access is a group, or work group.

Universe: All other users in the system constitute the universe.

9. Explain in detail about file system structure.

File System Structure

- Disk provides the bulk of secondary storage on which a file system is maintained.
- They can be rewritten in place; it is possible to read a block from the disk, to modify the block and to write it back into the same place.
- To produce an efficient and convenient access to the disk, the operating system imposes one or more file system to allow the data to be stored, located and retrieved easily. Refer figure 4.16.
- The file system itself is generally composed of many different levels. Each level in the design uses the features of lower level to create new features for use by higher levels.

Layered File System

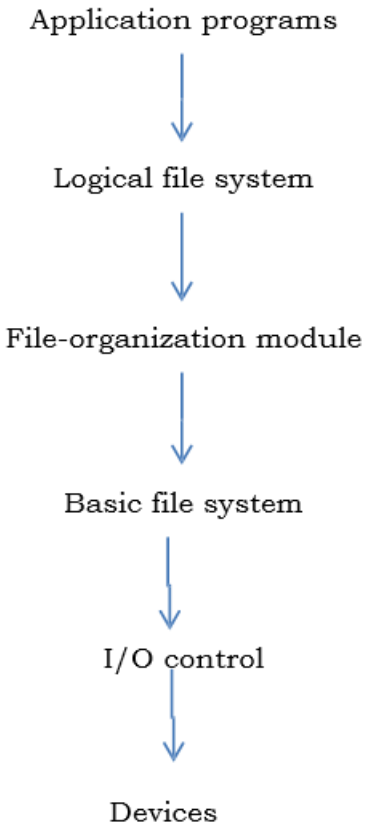


Fig.4.16 Layered File System

Disks provide most of the secondary storage on which file systems are maintained. Two characteristics make them convenient for this purpose:

1. A disk can be rewritten in place; it is possible to read a block from the disk, modify the block, and write it back into the same place.

2. A disk can access directly any block of information it contains. Thus, it is simple to access any file either sequentially or randomly, and switching from one file to another requires only moving the read write heads and waiting for the disk to rotate.

The **logical file system** manages metadata information.

- Metadata includes all of the file-system structure, excluding the actual data (or contents of the files).
- The logical file system manages the directory structure to provide the file organization module.
- It maintains file structure, via file control blocks.

- A **file control block** (FCB) contains information about the file, including ownership, permissions, and location of the file contents. The logical file system is also responsible for protection and security.

The **file-organization module** knows about file and their logical blocks, as well as physical blocks.

- The file organization module can translate logical block address to physical block addresses for the basic fie system to transfer.
- The file-organization module also includes the free-space manager, which tracks unallocated blocks.
- Each physical block is identified by its numeric disk address.
- The I/O control consists of device drivers and interrupts handler to transfer information between the main memory and the disk system.

10. Explain in detail about file system implementation.

File System Implementation

Several-on-disk and in-memory structures are used to implement a file system.

On-disk structures include:

1. A **boot control block** can contain information needed by the system to boot an operating from that partition. If the disk does not contain an operating System, this block can be empty. It is typically the first block of a partition. In **UFS**, this is called the **boot block**; In **NTFS**, it is **partition boot sector**.
2. A **partition control block** contains partition details such as the number of blocks in the partition, size of the blocks, free-block count and free block pointers and free FCB count and FCB pointers. In **UFS** this is called a **super block**; in **NTFS**, it is the **Master File Table** A **directory structure** is used to organize the files.
3. An **FCB** contains many of the files details, including file permissions, ownership, size and location of the data blocks. IN **UFS** this called the **inode**. In NTFS, this information's actually stored within the Master File Table, which uses a relational database structure, with a row per file

In-memory structures include:

1. An **in-memory partition table** containing, information about each mounted partition.

2. An **in-memory directory structures** that holds the directory information of recently accessed directories.
3. The system-wide open-file table contains a copy of the FCB of each open files, as well as other information.
4. The **per-process open-file table** contains a pointer to the appropriate entry in the systems wide open file table, as well as other information. Refer 4.17.

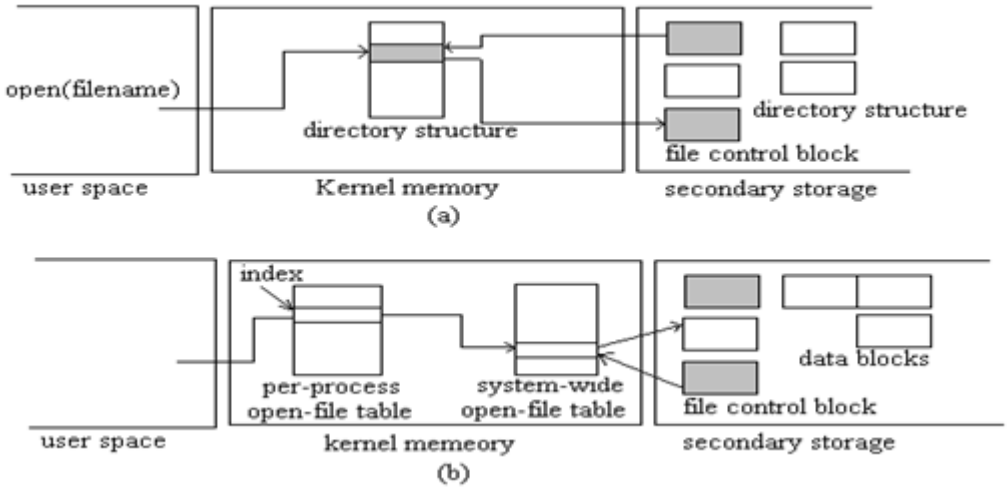


Fig.4.17. In memory File System Structure a) File Open b) File Read

File Control Block

A File Control Block (FCB) is a file system structure that stores information about open files. It's managed by the operating system (OS) but resides in the memory of the program that uses the file. Refer 4.18.

File permissions
File dates (create, access, write)
File owner, group, Act
File size
File data blocks

Fig.4.18 File Control Block

Partitions and Mounting

- Each partition can be either "raw," containing no file system, or "cooked," containing a file system. **Raw disk** is used where no file system is appropriate.
- UNIX swap space can use a raw partition, for example, since it uses its own format on disk and does not use a file system.
- The **root partition**, which contains the operating-system kernel and sometimes other system files, is mounted at boot time.
- **Mounting** is implemented by setting a flag in the in-memory copy of the inode for that directory. The flag indicates that the directory is a mount point. A field then points to an entry in the mount table, indicating which device is mounted there.
- The mount table entry contains a pointer to the superblock of the file system on that device.

Virtual File Systems

The second layer is called the **virtual file system (VFS)** layer. The VFS layer serves two important functions:

1. It separates file-system-generic operations from their implementation by defining a clean VFS interface. Several implementations for the VFS interface may coexist on the same machine, allowing transparent access to different types of file systems mounted locally.
2. It provides a mechanism for uniquely representing a file throughout a network. The VFS is based on a file-representation structure, called a **vnode**. The kernel maintains one vnode structure for each active node (file or directory). Refer figure 4.19

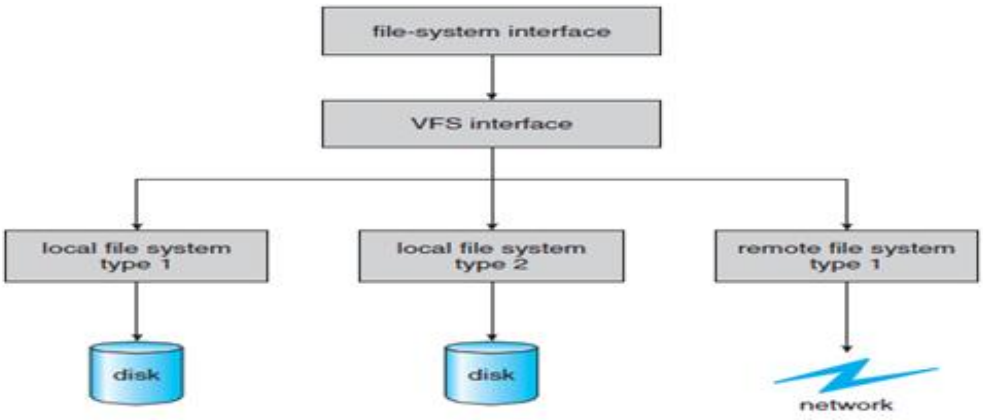


Fig.4.19 Schematic view of Virtual File System

11. Explain in detail about directory implementation.

Directory Implementation

1. Linear List

- A linear list of directory entries requires a linear search to find a particular entry.
- Time- consuming to execute.
- The real disadvantage of a linear list of directory entries is the linear search to find a file.

2. Hash Table

- In this method, a linear list stores the directory entries, but a hash data structure is also used.
- The hash table takes a value computed from the file name and returns a pointer to the file name in the linear list.
- The major difficulties with a hash table are its generally fixed size and the dependence of the hash function on that size.

12. Discuss in detail about file allocation methods. (or) Explain various file allocation method with suitable illustrations. Also discuss in detail various file organization methods. (April/May 2024)

The allocation methods define how the files are stored in the disk blocks. There are three main disk space or file allocation methods.

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

The main idea behind these methods is to provide:

- Efficient disk space utilization.
- Fast access to the file blocks.

All the three methods have their own advantages and disadvantages as discussed below:

1. Contiguous Allocation

- In this scheme, each file occupies a contiguous set of blocks on the disk. For example, if a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be: b, b+1, b+2,……b+n-1.

- This means that given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file.
- The directory entry for a file with contiguous allocation contains
- Address of starting block
- Length of the allocated portion. Refer figure 4.20.
- The file 'mail' in the following figure starts from the block 19 with length = 6 blocks. Therefore, it occupies 19, 20, 21, 22, 23, 24 blocks.

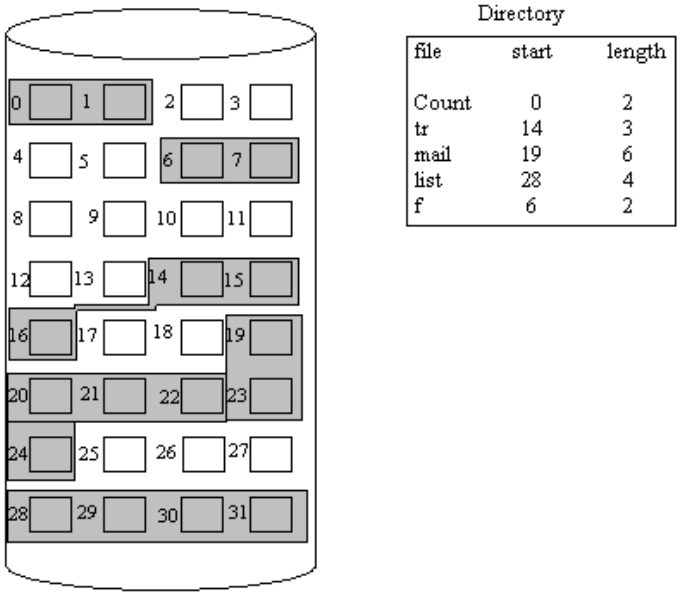


Fig.4.20 Contiguous Allocation

Advantages:

- Both the Sequential and Direct Accesses are supported by this. For direct access, the address of the kth block of the file which starts at block b can easily be obtained as (b+k).
- This is extremely fast since the number of seeks are minimal because of contiguous allocation of file blocks.

Disadvantages:

- Finding space for a new file.
- Determining how much space is needed for a file.
- This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.

- Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

2. Linked Allocation

- In this scheme, each file is a linked list of disk blocks which need not be contiguous. The disk blocks can be scattered anywhere on the disk.
- The directory entry contains a pointer to the starting and the ending file block. Each block contains a pointer to the next block occupied by the file.
    - For example, a file of five blocks might start at block 9, continue at block 16, then block 1, block 10, and finally bock25.
- The file 'jeep' in following image shows how the blocks are randomly distributed. The last block (25) contains -1 indicating a null pointer and does not point to any other block. Refer figure 4.21.

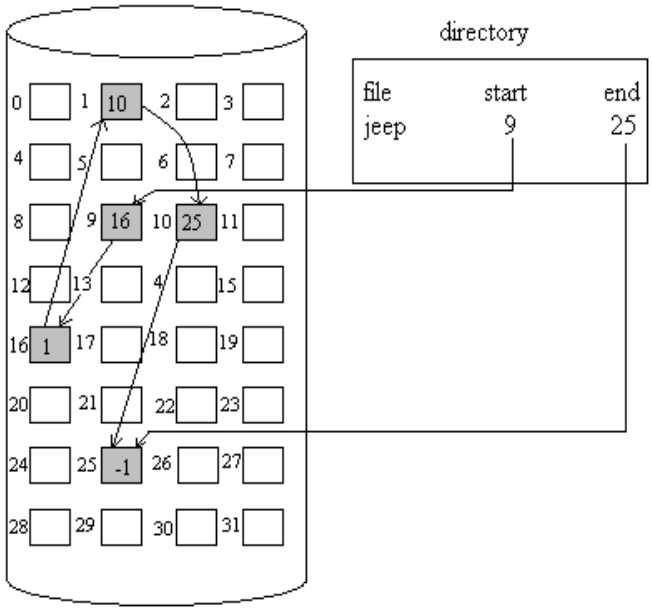


Fig.4.21 Linked Allocation

Advantage:

- This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for a contiguous chunk of memory.
- This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.
- There is no external fragmentation with linked allocation, and any free block on the free space list can be used to satisfy a request.

- The size of a file does not need to the declared when that file is created.
- A file can continue to grow as long as free blocks are available consequently, it is never necessary to compacts disk space.

Disadvantages

- Because the file blocks are distributed randomly on the disk, a large number of seeks are needed to access every block individually. This makes linked allocation slower.
- It does not support random or direct access. We can not directly access the blocks of a file. A block k of a file can be accessed by traversing k blocks sequentially (sequential access) from the starting block of the file via block pointers.
- Pointers required in the linked allocation incur some extra overhead.

Indexed Allocation

- In this scheme, a special block known as the Index block contains the pointers to all the blocks occupied by a file.
- Each file has its own index block. The ith entry in the index block contains the disk address of the ith file block.
- Indexed allocation bringing all the pointers together into one location i.e. the index block.
- The directory contains the address of the index block.
- To read the ith block, we use the pointer in the ith index block entry to find and read the desired block as shown as figure 4.22.

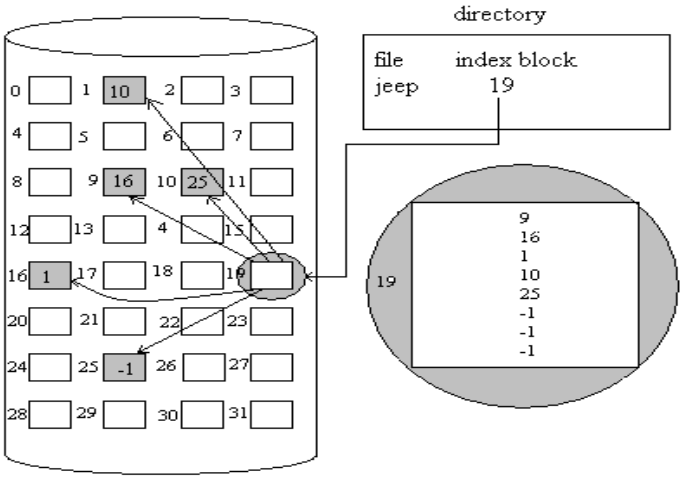


Fig.4.22 Indexed Allocation

Advantages

- Indexed allocation supports direct access, without suffering from external fragmentation, because any free block on the disk may satisfy a request for more space.
- It overcomes the problem of external fragmentation

Disadvantages

1. Pointer Overhead

- Indexed allocation does suffer from wasted space. The pointer over head of the index block is generally greater than the pointer over head of linked allocation.

2. Size of Index block

If the index block is too small, however, it will not be able to hold enough pointers for a large file.

- **Linked Scheme:** To allow for large files, we may link together several index blocks.
- **Multilevel index:** A variant of the linked representation is to use a first level index block to point to a set of second level index blocks.

13. Discuss how free space is managed by operating system.

Free-space Management

- Disk space is limited; we need to reuse the space from deleted files for new files, if possible.
- To keep track of free disk space, the system maintains a free-space list.
- To create a file, we search the free-space list for the required amount of space, and allocate that space to the new file.
- This space is then removed from the free-space list.
- When a file is deleted, its disk space is added to the free-space list.

1. Bit Vector

- The free-space list is implemented as a bit map or bit vector as shown figure 4.23.
- Each block is represented by 1 bit. If the block is free, the bit is 1; if the block is allocated, the bit is 0.
- For example, consider a disk where block 2,3,4,5,8,9,10,11,12,13,17,18,25,26 and 27 are free, and the rest of the block are allocated. The free space bit map would be

001111001111110001100000011100000…

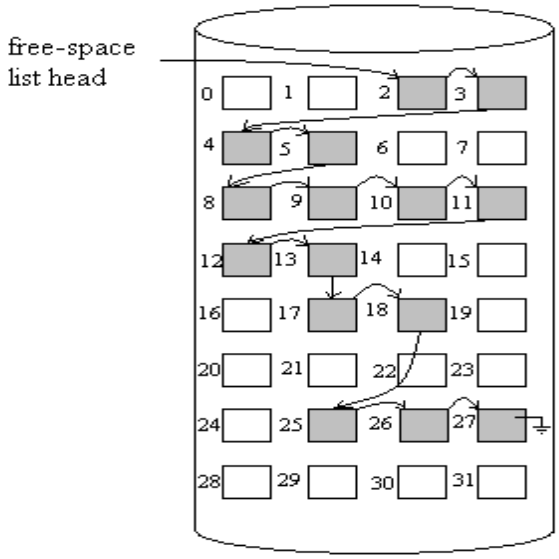


Fig.4.23 Linked List

- The main **advantage** of this approach is it's relatively simplicity and efficiency in finding the first free block, or n consecutive free blocks on the disk.

2. Linked List

- Another approach to free-space management is to link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory.
- This first block contains a pointer to the next free disk block, and so on.

3. Grouping

- A modification of the free-list approach is to store the addresses of n free blocks in the first free block.
- The first n-1 of these blocks is actually free.
- The last block contains the addresses of another n free block, and so on.

4. Counting

- We can keep the address of the first free block and the number of free contiguous blocks that follow the first block. Each entry in the free-space list then consists of a disk address and a count.

14. Explain in Detail about IO Hardware.

I/O Hardware

- The role of the operating system in computer I/O is to manage and control I/O operations and I/O devices.
- A device communicates with a computer system by sending signals over a cable or even through the air.
- Port: The device communicates with the machine via a connection point (or port), for example, a serial port.
- Bus: If one or more devices use a common set of wires, the connection is called a bus.

Daisy chain:

- Device 'A' has a cable that plugs into device 'B', and device 'B' has a cable that plugs into device 'C', and device 'C' plugs into a port on the computer, this arrangement is called a daisy chain.
- A daisy chain usually operates as a bus.

PC bus structure:

- A PCI bus that connects the processor-memory subsystem to the fast devices, and an expansion bus that connects relatively slow devices such as the keyboard and serial and parallel ports.
- Four disks are connected together on a SCSI bus plugged into a SCSI controller.
- A controller or host adapter is a collection of electronics that can operate a port, a bus, or device.
- A serial-port controller is a simple device controller. It is a single chip in the computer that controls the signals on the wires of a serial port.
- SCSI bus controller is not simple. Because the SCSI protocol is complex, the SCSI bus controller is often implemented as a separate circuit board It typically contains a processor, microcode, and some private memory. Some devices have their own built-in controllers.

Control register:

- Written by the host to start a command or to change the mode of a device.

data-in register:

- Read by the host to get input

data-out register:

- Written by the host to send output

Polling

- Interaction between the host and a controller
  - The controller sets the busy bit when it is busy working, and clears the busy bit when it is ready to accept the next command.
  - The host sets the command ready bit when a command is available for the controller to execute.

Coordination between the host & the controller is done by handshaking as follows:

1. The host repeatedly reads the busy bit until that bit becomes clear.
2. The host sets the write bit in the command register and writes a byte into the data-out register.
3. The host sets the command-ready bit.
4. When the controller notices that the command-ready bit is set, it sets the busy bit.

15. Write a brief note on Interrupts.

Interrupts

The CPU hardware has a wire called the interrupt request line.

The basic interrupt mechanism works as follows;

- Device controller raises an interrupt by asserting a signal on the interrupt request line.
- The CPU catches the interrupt and dispatches to the interrupt handler and the handler clears the interrupt by servicing the device. Refer figure 4.24.

Two interrupt request lines:

- Non maskable interrupt
- Maskable Interrupt

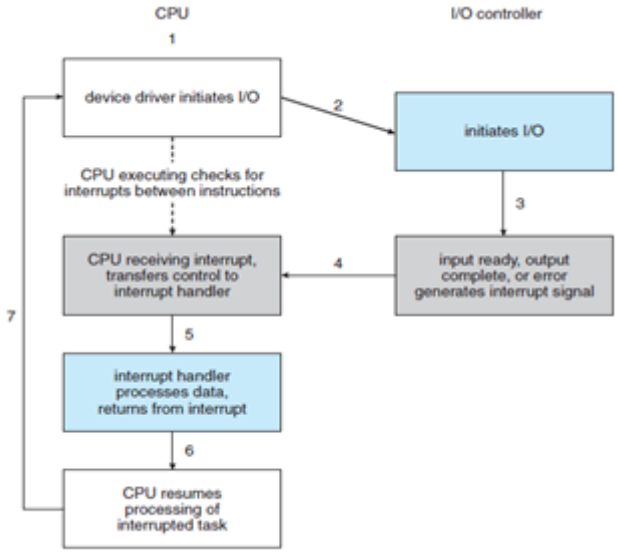


Fig.4.24 Interrupt Driven I/O cycle

14. Write a brief note on the steps involved in DMA transfer. [June2014]

- Handshaking between the DMA controller and the device controller is performed via a pair of wires called DMA-request and DMA-acknowledge.
- The device controller places a signal on the DMA-request wire when a word of data is available for transfer.
- This signal causes the DMA controller to seize the memory bus, place the desired address on the memory-address wires, and place a signal on the DMA-acknowledge wire.
- When the device controller receives the DMA-acknowledge signal, it transfers the word of data to memory and removes the DMA-request signal.
- When the entire transfer is finished, the DMA controller interrupts the CPU.

16. Explain in detail about Application I/O Interface.

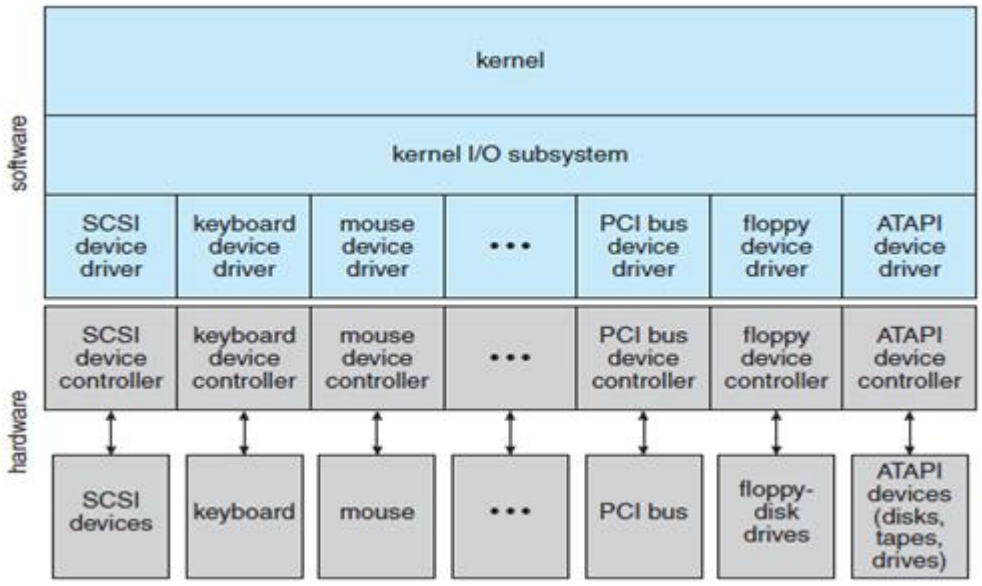


Fig.4.25 Kernel I/O structure

Application I/O Interface

Devices vary in many dimensions as shown figure 4.25.

1. Character-stream or block - A character-stream device transfers bytes one by one, whereas a block device transfers a block of bytes as a unit
2. Sequential or random-access - A sequential device transfers data in a fixed order determined by the device, whereas the user of a random-access device can instruct the device to seek to any of the available data storage locations.
3. Sharable or dedicated - A sharable device can be used concurrently by several processes or threads; a dedicated device cannot.
4. Speed of operation - Device speeds range from a few bytes per second to a few gigabytes per second.
5. Read-write, read only, or writes only - Some devices perform both input and output, but others support only one data transfer direction.

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read–write	CD-ROM graphics controller disk

Fig.4.26 Characteristics of I/O devices

Block and Character Devices:

Block-device:

- The block-device interface captures all the aspects necessary for accessing disk drives and other block-oriented devices.
- The device should understand the commands such as read () & write (), and if it is a random access device, it has a seek() command to specify which block to transfer next.

Character Devices:

- A keyboard is an example of a device that is accessed through a character stream interface.

Network Devices

- Windows NT provides one interface to the network interface card, and a second interface to the network protocols.

Clocks and Timers

Most computers have hardware clocks and timers that provide three basic functions:

1. Give the current time
2. Give the elapsed time
3. Set a timer to trigger operation X at time T

18. Explain the special services provided by kernel I/O subsystem. (or) Explain about kernel I/O subsystems and transforming I/O to hardware operations. (Nov/Dec 2024)

Kernels provide many services related to I/O.

- One way that the I/O subsystem improves the efficiency of the computer is by scheduling I/O operations. Another way is by using storage space in main memory or on disk, via techniques called buffering, caching, and spooling.

Services include

1. I/O Scheduling:

To determine a good order in which to execute the set of I/O requests

Uses:

- It can improve overall system performance,
- It can share device access fairly among processes, and
- It can reduce the average waiting time for 1/0 to complete.

2. Buffering:

A memory area that stores data while they are transferred between two devices or between a device and an application

Reasons for buffering:

- To cope with a speed mismatch between the producer and consumer of a data stream.
- To adapt between devices that have different data-transfer sizes.
- To support copy semantics for application I/O.

3. Copy semantics:

- With copy semantics, the version of the data written to disk is guaranteed to be the version at the time of the application system call, independent of any subsequent changes in the application's buffer.

4. Caching

- A cache is a region of fast memory that holds copies of data.
- A buffer may hold the only existing copy of a data item, whereas a cache just holds a copy on faster storage of an item that resides elsewhere.

5. Spooling and Device Reservation:

Spool

- It is a buffer that holds output for a device, such as a printer, that

cannot accept interleaved data streams.

- A printer can serve only one job at a time, several applications may wish to print their output concurrently, without having their output mixed together.

Device reservation

- It provides exclusive access to a device
- System calls for allocation and de-allocation

6. Error Handling

- An operating system that uses protected memory can guard against many kinds of hardware and application errors.
- OS can recover from disk read, device unavailable, transient write failures
- Most return an error number or code when I/O request fails
- System error logs hold problem reports

7. I/O Protection

- To prevent users from performing illegal I/O, we define all I/O instructions to be privileged instructions. Thus, users cannot issue I/O instructions directly;

19. Explain how I/O related portions of the kernel are structured in software layers.

Kernel I/O Subsystem

The I/O subsystem supervises these procedures:

- Management of the name space for files and devices
- Access control to files and devices
- Operation control (for example, a modem cannot seek())
- File-system space allocation
- Device allocation
- Buffering, caching, and spooling
- I/O scheduling
- Device-status monitoring, error handling, and failure recovery
- Device-driver configuration and initialization

20. Explain in detail about the lifecycle of an I/O request.

1. A process issues a blocking read() system call to a file descriptor of a file that has been opened previously.
2. The system-call code in the kernel checks the parameters for correctness.
3. The device driver allocates kernel buffer space to receive the data and schedules the I/O.
4. The device controller operates the device hardware to perform the data transfer.
5. The driver may poll for status and data, or it may have set up a DMA transfer into kernel memory.
6. The correct interrupt handler receives the interrupt via the interrupt vector table, stores any necessary data, signals the device driver, and returns from the interrupt.

7. The device driver receives the signal, determines which I/O request has completed, determines the request's status, and signals the kernel I/O subsystem that the request has been completed. Refer 4.27.

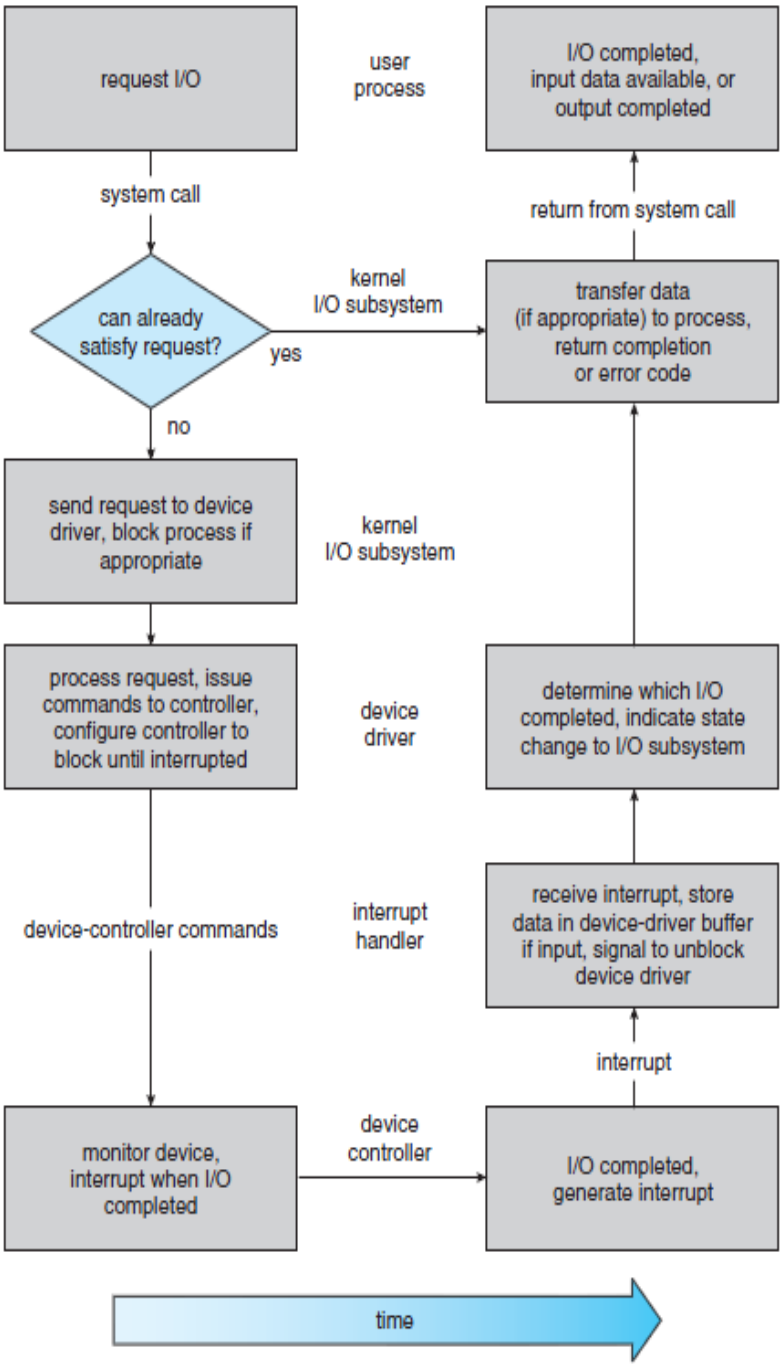
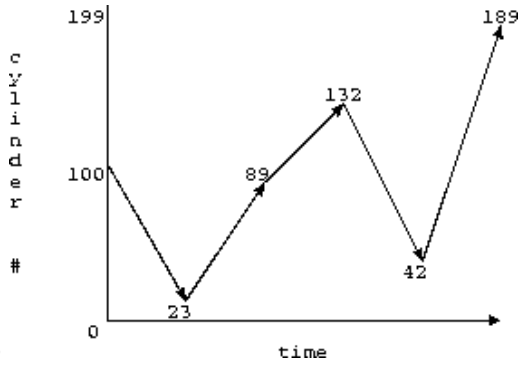


Figure 4.27 The life cycle of an I/O request.

21. Examples of Disk Scheduling Algorithms. Work Queue: 23, 89, 132, 42, 187

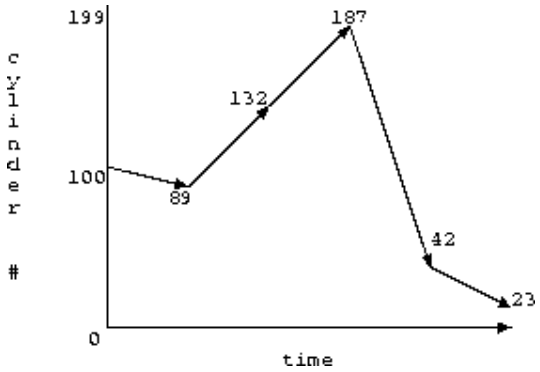
- there are 200 cylinders numbered from 0 - 199
- the disk head stars at number 100



1. **FCFS**

- total time is estimated by total arm motion

$$|100-23|+|23-89|+|89-132|+|23-132|+|132-42|+|42-187|=77+66+43+90$$

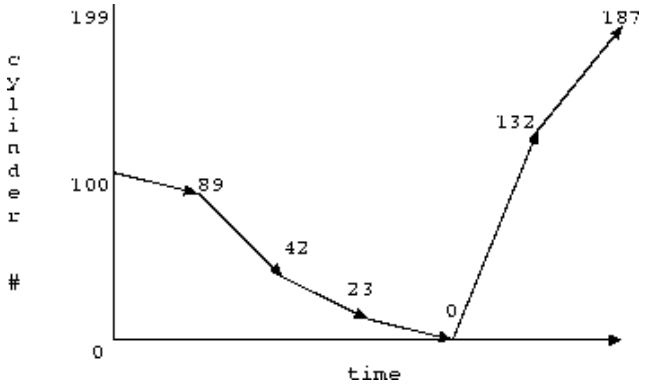


2. **SSTF**

$$|100-89|+|89-132|+132-187|+|187-42|+|42-23|=11+43+55+145+19=273$$

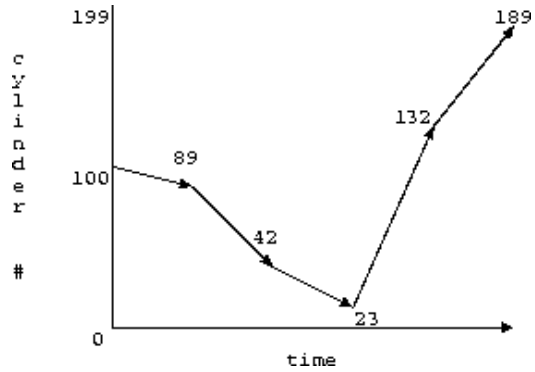
3. **SCAN**

- assume we are going inwards (i.e., towards 0)



$$|100-89|+|89-42|+|42-23|+|23-0|+|0-132|+|132-187|=11+47+19+23+1$$

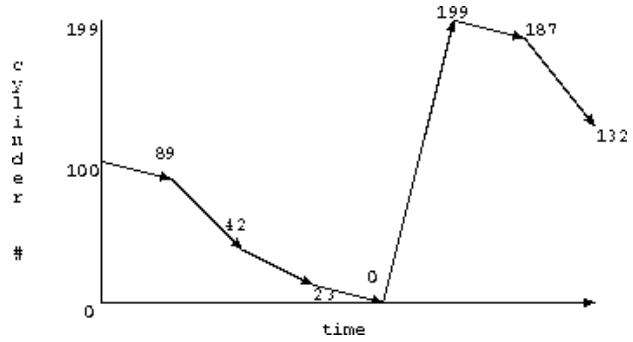
4. **LOOK**



$$|100-89|+|89-42|+|42-23|+|23-132|+|132-187|=11+47+19+109+55=241$$

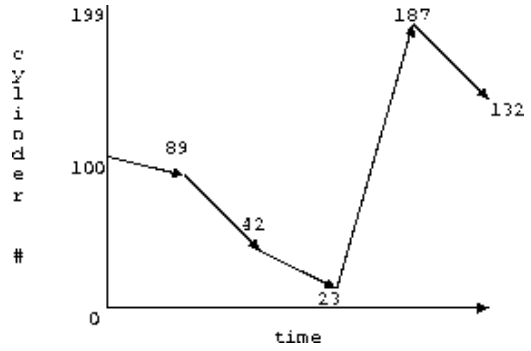
- reduce variance compared to SCAN

5. **C-SCAN**



$$|100-89|+|89-42|+|42-23|+|23-0|+|0-199|+|199-187|+|187-132|=11+4'$$

6. **C-LOOK**



$$|100-89|+|89-42|+|42-23|+|23-187|+|187-132|=11+47+19+164+55=296$$

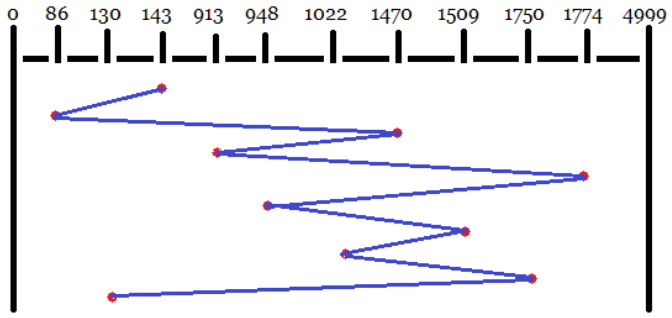
Suppose that a disk drive has 5000 cylinders numbered 0 to 4999. The drive is currently serving a request at cylinder 143. The queue of pending requests in FIFO order is- 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130. Starting from the current head position, what is the total distance that the disk arm moves to satisfy

all the pending requests for each of the following disk scheduling algorithms?

(i) FCFS
(ii) SSTF
(iii) SCAN
(iv) LOOK
(v) C-SCAN
(vi) C-LOOK

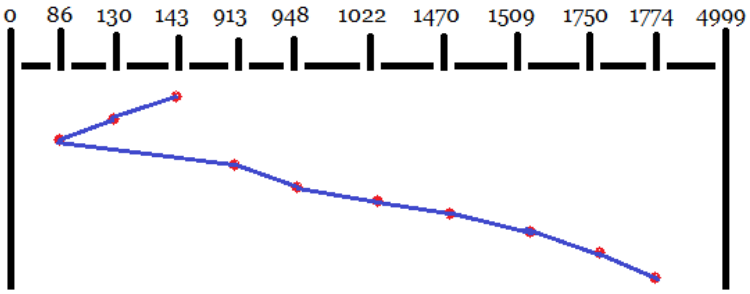
Sol.

(i). FCFS: The movement of disk head from 143 to others is as shown in diagram.



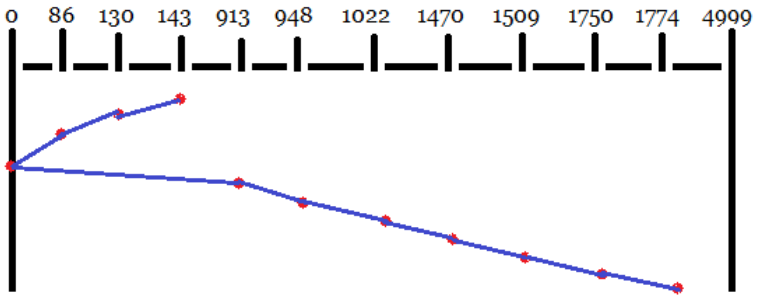
So, total head movement = 57+1384+557+861+826+561+487+728+1620=7081 cylinders.

(ii). SSTF: The movement of disk head from 143 to others is as shown in diagram.



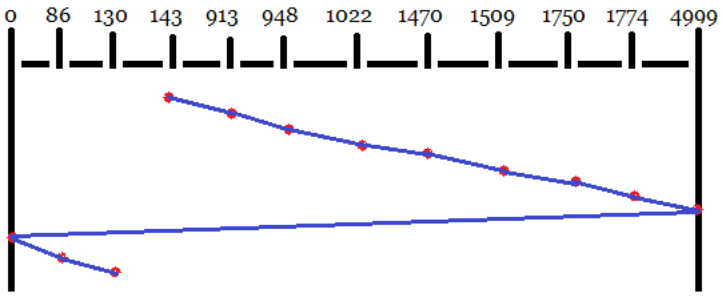
So, total head movement = 13+44+827+35+74+448+39+241+24=1745 cylinders.

(iii). SCAN: The movement of disk head from 143 to others is as shown in diagram.

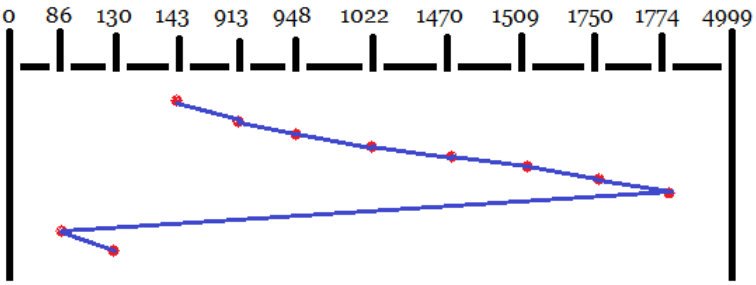


So, total head movement = 13+44+86+913+35+74+448+39+241+24=1917 cylinders.

(iv). C-SCAN: The movement of disk head from 143 to others is as shown in diagram.



(V). C-LOOK/LOOK: The movement of disk head from 143 to others is as shown in diagram.



22. Disk requests come into the disk driver for cylinders in the following order: 10, 22, 20, 2, 40, 6 and 38. Assume that the disk has 100 cylinders. A seek takes 6 milli-second per cylinder moved. Compute the average seek time for the request sequence given above www.Enggtree.com using the following scheduling algorithm. (April/May 2024)

(i) First come first served. (4)

(ii) Shortest Seek Time First (SSTF). (3)

(iii) LOOK (Assume disk arm moves towards higher number cylinders from lower number cylinder). (3)

(iv) C-SCAN. (3)

To calculate the average seek time for the given request sequence using different scheduling algorithms, first list the request sequence and the current head position, then calculate the total seek time for each algorithm and divide by the number of requests to get the average seek time.

Given:

- Seek time per cylinder = 6 milliseconds
- Request sequence: (Assume the current head position is cylinder 20)
    - 10
    - 22
    - 2
    - 40
    - 6
    - 38

Calculations:

1. First Come First Served (FCFS):

- Seek sequence: 20 -> 10 -> 22 -> 2 -> 40 -> 6 -> 38
- Total seek time: (20-10) + (22-10) + (22-2) + (40-2) + (40-6) + (38-6) = 146 cylinders
- Average seek time: (146 cylinders * 6 milliseconds/cylinder) / 6 requests = 146 milliseconds

2. Shortest Seek Time First (SSTF):

- Seek sequence: 20 -> 22 -> 20 -> 2 -> 6 -> 10 -> 38

- Total seek time: (20-22) + (22-20) + (20-2) + (2-6) + (6-10) + (10-38) = 60 cylinders
- Average seek time: (60 cylinders * 6 milliseconds/cylinder) / 6 requests = 60 milliseconds

3. LOOK:

- Seek sequence: 20 -> 22 -> 38 -> 40 -> 38 -> 38 -> 38
- Total seek time: (20-22) + (22-38) + (38-40) + (40-38) + (38-38) + (38-38) = 40 cylinders
- Average seek time: (40 cylinders * 6 milliseconds/cylinder) / 6 requests = 40 milliseconds

4. C-SCAN:

- Seek sequence: 20 -> 22 -> 38 -> 40 -> 38 -> 38 -> 38
- Total seek time: (20-22) + (22-38) + (38-40) + (40-38) + (38-38) + (38-38) = 40 cylinders
- Average seek time: (40 cylinders * 6 milliseconds/cylinder) / 6 requests = 40 milliseconds.

23. Discuss the advantages and disadvantages of support links to files that cross mount points. (Nov/Dec 2024)

- In many operating systems (like Linux and Unix), **symbolic links (soft links)** can be used to reference files across different file systems and mount points. This allows flexible file organization but also introduces certain complexities.

Advantages of Supporting Links Across Mount Points

1. **Flexible File Organization**
   - Users can create links across different mounted file systems, making it easier to manage files spread across multiple storage devices.
   - Example: A symbolic link in /home/user/docs pointing to a file on /mnt/usb_drive.
2. **Efficient Storage Utilization**
   - Instead of duplicating files, symbolic links save space by referencing existing files elsewhere.
   - Useful for **backup systems**, where multiple references to the same file may be needed.
3. **Transparent Access**
   - Users and applications can access linked files **as if they were in the same**

directory, improving ease of use.

- Example: A link in /usr/bin can point to an executable stored on another partition.

4. **Simplifies Software Management**
   - Software installations can use links to reference libraries or dependencies stored on separate volumes, preventing redundancy.
   - Useful in environments where files need to be shared across multiple locations.
5. **Supports Network File Systems (NFS)**
   - Allows linking files between local and network-mounted file systems, useful for **distributed computing** and **remote access**.

Disadvantages of Supporting Links Across Mount Points

1. **Broken Links Due to Unavailable Mounts**
   - If the mounted file system is **unmounted or fails**, symbolic links pointing to it become **dangling (broken)**.
   - Example: A link to /mnt/server/file becomes useless if the server is disconnected.
2. **Performance Overhead**
   - Accessing files across different mount points can introduce **latency**, especially if they reside on **slow devices or network-mounted storage**.
   - Additional system calls are required to resolve links.
3. **Complexity in Backup & Restore**
   - Backup tools may not properly handle **cross-mount links**, leading to missing files if only the local mount is backed up.
   - Restoring symbolic links may require ensuring the target filesystem is properly mounted.
4. **Security & Access Control Issues**
   - If a symbolic link crosses into a **restricted mount point**, users might unintentionally gain access to sensitive files.
   - Example: A user in /home/user1 linking to /mnt/secure_data/secret.txt.
5. **Hard Links Not Allowed Across Mount Points**
   - Unlike **soft links**, **hard links** cannot be created across different file systems, limiting their use.

UNIT V VIRTUAL MACHINES AND MOBILE OS

Virtual Machines – History, Benefits and Features, Building Blocks, Types of Virtual Machines and their Implementations, Virtualization and Operating-System Components; Mobile OS - iOS and Android.

PART-A

1. What is virtual machine?

A Virtual Machine (VM) is a compute resource that uses software instead of a physical computer to run programs and deploy apps. One or more virtual "guest" machines run on a physical "host" machine. Each virtual machine runs its own operating system and functions separately from the other VMs, even when they are all running on the same host. This means that, for example, a virtual MacOS virtual machine can run on a physical PC.

2. What are the applications of virtual machines?

- Building and deploying apps to the cloud.
- Trying out a new operating system (OS), including beta releases.
- Spinning up a new environment to make it simpler and quicker for developers to run dev-test scenarios.
- Backing up your existing OS.
- Accessing virus-infected data or running an old application by installing an older OS.

3. What are the benefits of virtual machine?

a. Security benefits
b. Scalability
c. Lowered downtime
d. Agility and speed
e. Cost savings

4. What are building blocks of e-virtual machine?

(i)Trap-and-Emulate
(ii)Binary Translation and
(iii)Hardware Assistance

5. What is trap-and-emulate in virtual machine?

Trap-and-emulate is a technique used by the virtual machine to emulate privileged instructions and registers and pretend to the OS that it's still in kernel mode. An operation system is designed to have full control of the system.

6. Define binary translation.

The VMM scans the instruction stream and identifies the privileged, control- and behavior-sensitive instructions. When these instructions are identified, they are trapped into the VMM, which emulates the behavior of these instructions. The method used in this emulation is called binary translation.

7. What is hardware assistance in VM?

- When using this assistance, the guest can use a separate mode of execution called guest mode.
- The guest code, whether application code or privileged code, runs in the guest mode.

8. What are the types of virtual machines?

- Process virtual machines
- System virtual machines

9. Define Para virtualization.

Para virtualization is the category of CPU virtualization which uses hyper calls for operations to handle instructions at compile time. In para virtualization, guest OS is not completely isolated but it is partially isolated by the virtual machine from the virtualization layer and hardware. VMware and Xen are some examples of para virtualization.

10. What is Programming-Environment Virtualization?

- The virtualization of programming environments is a separate type of virtualization that is based on a different execution paradigm.
- A programming language is meant to run in a custom-built virtualized environment in this case.

11. Define emulation.

Emulation, as name suggests, is a technique in which Virtual machines simulates complete hardware in software. There are many virtualization techniques that were developed in or inherited from emulation technique. It is very useful when designing software for various systems. It simply allows us to use current platform to access an older application, data, or operating system.

12. Define CPU scheduling in VM.

A system with virtualization, even a single-CPU system, frequently acts like a multiprocessor system. The virtualization software presents one or more virtual CPUs to each of the virtual machines running on the system and then schedules the use of the physical CPUs among the virtual machines.

13. What is virtual-to physical procedure?

Virtual to physical (V2P) is the process of converting or porting a virtual machine (VM) onto and/or as a standard physical machine. V2P allows a VM to transform into a physical machine without losing its state, data and overall operations.

14. What are the types of virtualizations?

- Network Virtualization
- Storage Virtualization
- Server Virtualization
- Application Virtualization
- Desktop Virtualization

15. Write some of the key features of UIKit.

- Application lifecycle management
- Application event handling (e.g. touch screen user interaction)
- Multitasking
- Wireless Printing
- Data protection via encryption
- Cut, copy, and paste functionality
- Web and text content presentation and management
- Data handling

16. Write some SDK framework.

- UIKit Framework (UIKit.framework)
- Map Kit Framework (MapKit.framework)
- Message UI Framework (MessageUI.framework)
- Game Kit Framework (GameKit.framework)

17. What are the iOS Media Layer?

- Core Video Framework (CoreVideo.framework)
- Core Text Framework (CoreText.framework)
- Image I/O Framework (ImageIO.framework)
- Assets Library Framework (AssetsLibrary.framework)
- Core Graphics Framework (CoreGraphics.framework)

18. List the iOS Audio support files.

- Foundation framework (AVFoundation.framework)
- Core Audio Frameworks
- Open Audio Library (OpenAL)
- Media Player Framework (MediaPlayer.framework)
- Core Midi Framework (CoreMIDI.framework)

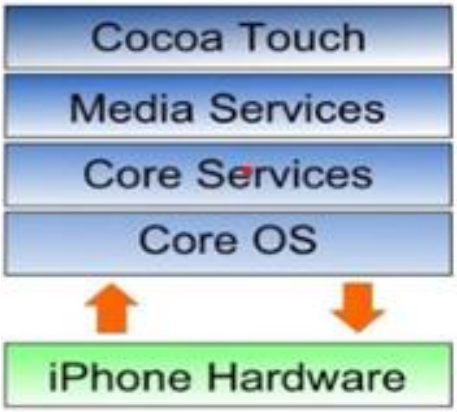
19. What is iOS Core OS Layer?

- The Core OS Layer occupies the bottom position of the iOS stack and, as such, sits directly on top of the device hardware.
- The layer provides a variety of services including low level networking, access to external accessories and the usual fundamental operating system services such as memory management, file system handling and threads.

20. List the iOS Core OS Layer.

- Accelerate Framework (Accelerate.framework)
- External Accessory Framework (ExternalAccessory.framework)
- Security Framework (Security.framework)

21. Draw an iOS 6 Architecture.



22. What are the advantages and disadvantages of writing an operating System in a high-level language, such as C ? (Nov/Dec-2019)

Advantages: The code can be written faster, is more compact, and is easier to understand and debug. In addition, improvements in compiler technology will improve the generated code for the entire operating system by simple recompilation.

An operating system is far easier to port — to move to some other hardware — if it is written in a higher-level language.

Disadvantages: Using high **level language** for **implementing an operating system** leads to a loss in speed and increase in storage requirements. However in modern **systems** only a small amount of code is needed for high performance, such as the CPU scheduler and memory manager.

23. What are the Basic features of Linux?

1.Portable
2.Open Source
3.Multi user
4.Multi programming
5.Hierarchical File system.

24. What is the significance of using virtual machines? (April/May 2024)

Virtual machines are significant because they allow users to run multiple operating systems on a single physical computer, providing flexibility, scalability, cost savings, and isolation by efficiently utilizing hardware resources, making it easier to test applications, manage different environments, and implement disaster recovery strategies, all while minimizing the need for dedicated physical servers.

25. Compare iOS and Android OS? (April/May 2024) (or) Compare and contrast Android OS and iOS. (Nov/Dec 2024)

S.No.	IOS	ANDROID OS
1.	It was developed and is owned by **Apple Incorporation**.	It was developed by Google and Open Handset Alliance and is owned by **Google LLC**.
2.	IOS was initially released on July 29, 2007	Google was initially released on 23 September 2008.
3.	when IOS was released its first version is iPhone OS 1 before named IOS.	When Google released its first version of Android 1.0, Alpha.
4.	It was launched in 2007.	It was launched in 2008.
5.	Latest stable release version: iOS 15.3.1 and iPadOS 15.3.1	Latest stable release version: Android 14
6.	Its target system types are smartphones,	Its target system types are

S.No.	IOS	ANDROID OS
	music players, and tablet computers.	smartphones and tablets.
7.	It is specially designed for Apple iphones and ipads.	It is designed for smartphones of all companies.
8.	Its kernel type is Hybrid.	Its kernel type is Linux-based.
9.	It has preferred license is Proprietary, APSL, and GNU GPL.	It has the preferred license of Apache 2.0 and GNU GPLv2.
10.	It is mainly written in C, C++, Objective-C, assembly language, and Swift.	It is written using C, C++, Java, and other languages.
11.	Its update management is Software Update.	Its update management is Systems Software Update.
12.	Swift is majorly used for iOS application development.	Java and Kotlin are majorly used for Android application development.
13.	IOS has a Commercial Based Source model with open source components.	Android is an Open Source based Source model.
14.	IOS-based Devices have Safari as the	Android devices have google chrome but one can install any

S.No.	IOS	ANDROID OS
	default Internet Browser.	Internet Browser.
15.	IOS has Siri as Voice Assistant.	Google has Google Assistance.
16.	IOS-based devices have the feature of blocking 3rd party app stores.	But Google doesn't block 3rd party app stores.
17.	IOS devices are available in 40 languages.	Android Devices are available in 100+ languages.
18.	In IOS customizability is limited unless jailbroken.	In Android, we can change almost anything.
19.	File transfer in android is easier than in IOS.	File transfer in IOS is more difficult than in android.
20.	None on the iPhone 7 and later; lighting 3.5mm no longer comes with the phone after the iPhone XS.	have a headphone jack, but some current Android smartphones lack a headphone jack.

26. Define Virtualization. (Nov/Dec 2024)

Virtualization refers to the technology that allows multiple virtual machines (VMs), each running its own operating system, to operate simultaneously on a single physical computer by creating a software layer that simulates hardware, effectively dividing the physical hardware resources into multiple virtual environments, enabling

efficient use of computing power and allowing different OSes to run on the same machine.

27. Write the importance of the kernel in the Linux operating system. (Nov/Dec 2024)

The Linux kernel is the central component of the Linux operating system, acting as the primary interface between the computer hardware and software processes, managing system resources like memory, CPU, and peripherals, allowing applications to utilize hardware without directly interacting with it, and ensuring smooth operation by handling tasks like process management, memory allocation, and device control - essentially making it the core foundation for the entire Linux system to function effectively.

PART-B

1. Explain in Detail about History of Virtual machines.

Virtual machines first appeared commercially on **IBM mainframes in 1972**. Virtualization was provided by the IBM VM operating system. This system has evolved and is still available. In addition, many of its original concepts are found in other systems, making it worth exploring**.**

IBM VM370 divided a mainframe into multiple virtual machines, each running its own operating system. A major difficulty with the VM approach involved disk systems. Suppose that the physical machine had three disk drives but wanted to support seven virtual machines.

It could not allocate a disk drive to each virtual machine. The solution was to provide virtual disks— termed minidisks in IBM's VM operating system.

- The minidisks Virtual Machines to the system's hard **disks in all respects except size**. The system implemented each minidisk by allocating as many tracks on the physical disks as the minidisk needed.
- Once the virtual machines were created, users could run any of the operating systems or software packages that were available on the underlying machine. For the IBM VM system, a user normally ran CMS— **a single-user interactive operating system.**

- For many years after **IBM introduced this technology, virtualization remained in its domain**. Most systems could not support virtualization. However, a formal definition of virtualization helped to establish system requirements and a target for functionality.

The virtualization requirements stated that:

1. A VMM provides an environment for programs that is essentially identical to the original machine.

2. Programs running within that environment show only minor performance decreases.

3. The VMM is in complete control of system resources. These requirements of fidelity, performance, and safety still guide virtualization efforts today. **By the late 1990s, Intel 80x86 CPUs had become common, fast, and rich in features.** Accordingly, developers launched multiple efforts to implement virtualization on that platform.

- Both Xen and **VMware created technologies, still used today**, to allow guest operating systems to run on the 80x86. Since that time, virtualization has expanded to include all common CPUs, many commercial and open-source tools, and many operating systems.
- For example, the open-source **Virtual Box project (http://www.virtualbox.org) provides a program than runs on Intel x86 and AMD64** CPUs and on Windows, Linux, Mac OS X, and Solaris host operating systems. Possible guest operating systems include many versions of Windows, Linux, Solaris, and BSD, including even MS-DOS and IBM OS/2.

2. Discuss in Detail about the Benefits and Features of Virtual machines. (or) Explain the features of virtual machine and also discuss the architecture in detail. (April/May 2024)

Several advantages make virtualization is more attractive. Most of them are fundamentally related to the ability to share the same hardware yet run several different execution environments (that is, different operating systems) concurrently. One important advantage of virtualization is that the host system is protected from the virtual machines, just as the virtual machines are protected from each other.

- A virus inside a guest **operating system might damage that operating system** but is unlikely to affect the host or the other guests. Because each virtual machine is almost completely isolated from all other virtual machines, there are almost no protection problems.
- A potential disadvantage of isolation is that it can **prevent sharing of resources**. Two approaches to provide sharing have been implemented.
- First, it is possible to share a file-system volume and thus to share files. Second, it is possible to define a network of virtual machines, each of which can send information over the virtual communications network.
- The network is modeled after physical communication networks but is implemented in software. Of course, the VMM is free to allow any number of its guests to use physical resources, such as a physical network connection (with sharing provided by the VMM), in which case the allowed guests could communicate with each other via the physical network.
- One feature common to most **virtualization implementations is the ability to freeze**, or suspend, a running virtual machine. Many operating systems provide that basic feature for processes, but VMMs go one step further and allow copies and snapshots to be made of the guest.
- The copy can be used to create a **new VM or to move a VM from one machine to another** with its current state intact. The guest can then resume where it was, as if on its original machine, creating a clone. The snapshot records a point in time, and the guest can be reset to that point if necessary (for example, if a change was made but is no longer wanted).
- Often, VMMs allow many **snapshots to be taken**. For example, snapshots might record a guest's state every day for a month, making restoration to any of those snapshot states possible. These abilities are used to good advantage in virtual environments. A virtual machine system is a perfect vehicle for operating-system research and development.
- Normally, changing an **operating system is a difficult task**. Operating systems are large and complex programs, and a change in one part may cause obscure bugs to appear in some other part. The power of the operating system makes changing it particularly dangerous. Because the operating system executes in kernel mode, a wrong change in a pointer

could cause an error that would destroy the entire file system. Thus, it is necessary to test all changes to the operating system carefully.

- Furthermore, the operating **system runs on and controls the entire machine**, meaning that the system must be stopped and taken out of use while changes are made and tested. This period is commonly called system-development time. Since it makes the system unavailable to users, system-development time on shared systems is often scheduled late at night or on weekends, when system load is low.
- A virtual-machine system can **eliminate much of this latter problem**. System programmers are given their own virtual machine, and system development is done on the virtual machine instead of on a physical machine. Normal system operation is disrupted only when a completed and tested change is ready to be put into production.
- Another advantage of **virtual machines for developers is that multiple operating systems** can run concurrently on the developer's workstation. This virtualized workstation allows for rapid porting and testing of programs in varying environments.
- In addition, **multiple versions of a program can run**, each in its own isolated operating system, within one system. Similarly, quality assurance engineers can test their applications in multiple environments without buying, powering, and maintaining a computer for each environment.
- A major advantage of **virtual machines in production data-center** use is system consolidation, which involves taking two or more separate systems and running them in virtual machines on one system. Such physical-to-virtual conversions result in resource optimization, since many lightly used systems can be combined to create one more heavily used system.
- Virtual Machines Consider, too, that **management tools that are part of the VMM allow system** administrators to manage many more systems than they otherwise could. A virtual environment might include 100 physical servers, each running 20 virtual servers.
- Without virtualization, 2,000 servers would require several system administrators. With virtualization and its tools, the same work can be managed by one or two administrators. One of the tools that make this

possible is templating, in which one standard virtual machine image, including an installed and configured guest operating system and applications, is saved and used as a source for multiple running VMs.

- Other features include **managing the patching of all guests**, backing up and restoring the guests, and monitoring their resource use. Virtualization can improve not only resource utilization but also resource management. Some VMMs include a live migration feature that moves a running guest from one physical server to another without interrupting its operation or active network connections.
- If a server is overloaded, live **migration can thus free resources** on the source host while not disrupting the guest. Similarly, when host hardware must be repaired or upgraded, guests can be migrated to other servers, the evacuated host can be maintained, and then the guests can be migrated back.
- This operation occurs without downtime and without interruption to users. Think about the possible effects of virtualization on how applications are deployed. If a system can easily add, remove, and move a virtual **machine, then why install applications on that system** directly? Instead, the application could be preinstalled on a tuned and customized operating system in a virtual machine.
- Thus, they need not be expensive, high-performance components. Other uses of virtualization are sure to follow as it becomes more prevalent and hardware support continues to improve.

3. Explain in detail about Building Blocks of virtual machine.

Although the virtual machine concept is useful, it is difficult to implement. Much work is required to provide an exact duplicate of the underlying machine. This is especially a challenge on dual-mode systems, where the underlying machine has only user mode and kernel mode.

- **The building blocks that are needed for efficient virtualization**. The ability to virtualize depends on the features provided by the CPU. If the features are sufficient, then it is possible to write a VMM that provides a guest environment. Otherwise, virtualization is impossible.

- VMMs use several techniques to **implement virtualization, including trap-and-emulate** and binary translation. We discuss each of these techniques in this section, along with the hardware support needed to support virtualization.
- One important concept found in **most virtualization options is the implementation** of a virtual CPU (VCPU). The VCPU does not execute code. Rather, it represents the state of the CPU as the guest machine believes it to be.
- For each guest, the **VMM maintains a VCPU representing that guest's current CPU state**. When the guest is context-switched onto a CPU by the VMM, information from the VCPU is used to load the right context, much as a general-purpose operating system would use the PCB.
- The kernel, of course, runs in kernel mode, and it is not safe to allow user-level code to run in kernel mode. Just as the physical machine has two modes, however, so must the virtual machine. Consequently, we must have a virtual user mode and a virtual kernel mode, both of which run in physical user mode.
- Those actions that cause a **transfer from user mode to kernel mode** on a real machine (such as a system call, an interrupt, or an attempt to execute a privileged instruction) must also cause a transfer from virtual user mode to virtual kernel mode in the virtual machine. How can such a transfer be accomplished?
- **The procedure is as follows: When the kernel in the guest attempts to execute** a privileged instruction, that is an error (because the system is in user mode) and causes a trap to the VMM in the real machine.
- The VMM gains control and executes (or "emulates") the action that was attempted by the guest kernel on the part of the guest. It then returns control to the virtual machine. This is called the trap-and-emulate method.
- Most virtualization products use this method to one extent or other. With privileged instructions, time becomes an issue. All non privileged instructions run natively on the hardware, providing the same performance

- Privileged instructions create **extra overhead, however, causing the guest to run more slowly than** it would natively. In addition, the CPU is being multiprogrammed among many virtual machines, which can further slow down the virtual machines in unpredictable ways.
- This problem has been approached **in various ways. IBM VM, for example**, allows normal instructions for the virtual machines to execute directly on the hardware. Only the privileged instructions (needed mainly for I/O) must be emulated and hence execute more slowly.
- In general, with the evolution of **hardware, the performance of trap-and-emulate functionality** has been improved, and cases in which it is needed have been reduced. For example, many CPUs now have extra modes added to their standard dual-mode operation.
- The VCPU need not keep track of what **mode the guest operating system is** in, because the physical CPU performs that function. In fact, some CPUs provide guest CPU state management in hardware, so the VMM need not supply that functionality, removing the extra overhead.

(i)Trap-and-Emulate

On a typical dual-mode system, the virtual machine guest can execute only in user mode (unless extra hardware support is provided). The kernel, of course, runs in kernel mode, and it is not safe to allow user-level code to run in kernel mode.

- Just as the physical machine has two modes, however, so must the virtual machine. Consequently, we must have a virtual user mode and a virtual kernel mode, both of which run in physical user mode.
- Those actions that cause a transfer from user mode to kernel mode on a real machine (such as a system call, an interrupt, or an attempt to execute a privileged instruction) must also cause a transfer from virtual user mode to virtual kernel mode in the virtual machine.
- How can such a transfer be accomplished? The procedure is as follows: When the kernel in the guest attempts to execute a privileged instruction, that is an error (because the system is in user mode) and causes a trap to the VMM in the real machine.

- The VMM gains control and executes (or "emulates") the action that was attempted by the guest kernel on the part of the guest. It then returns control to the virtual machine. This is called the trap-and-emulate method and is **shown in Fig 5.1.**

Most virtualization products use this method to one extent or other. With privileged instructions, time becomes an issue. All nonprivileged instructions run natively on the hardware, providing the same performance

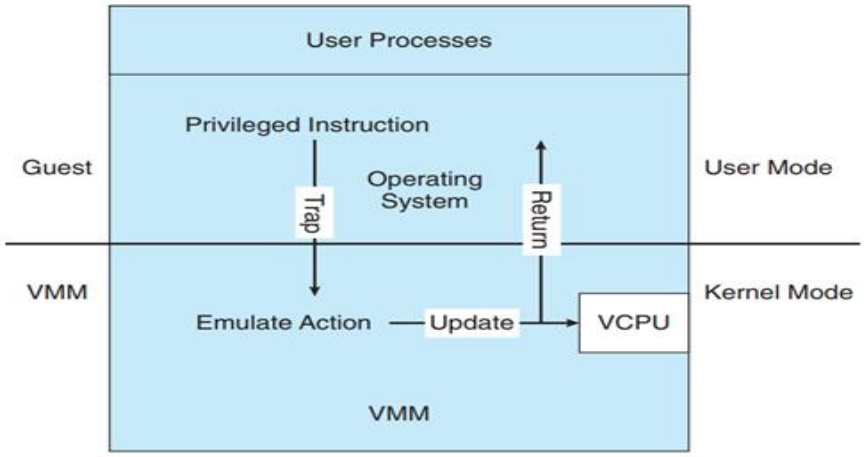


Fig 5.1: **Trap-and-Emulate**

- Privileged instructions create extra overhead, however, causing the guest to run more slowly than it would natively. In addition, the CPU is being multiprogrammed among many virtual machines, which can further slow down the virtual machines in unpredictable ways.
- This problem has been approached in various ways. IBM VM, for example, allows normal instructions for the virtual machines to execute directly on the hardware. Only the privileged instructions (needed mainly for I/O) must be emulated and hence execute more slowly.
- In general, with the evolution of hardware, the performance of trap-and-emulate functionality has been improved, and cases in which it is needed have been reduced. For example, many CPUs now have extra modes added to their standard dual-mode operation.

The VCPU need not keep track of what mode the guest operating system is in, because the physical CPU performs that function. In fact, some CPUs provide guest CPU state management in hardware, so the VMM need not supply that functionality, removing the extra overhead.

(ii)Binary Translation

Some CPUs do not have a clean separation of privileged and non privileged instructions. Unfortunately for virtualization implementers, the Intel x86 CPU line is one of them.

- No thought was given to running virtualization on the x86 when it was designed. (In fact, the first CPU in the family—the Intel 4004, released in 1971—was designed to be the core of a calculator.) The chip has maintained backward compatibility throughout its lifetime, preventing changes that would have made virtualization easier through many generations.
- Let's consider an example of the problem. The command popf loads the flag register from the contents of the stack. If the CPU is in privileged mode, all of the flags are replaced from the stack. If the CPU is in user mode, then only some flags are replaced, and others are ignored.

Binary translation is fairly simple in concept but complex in implementation. The basic steps are as follows:

1. If the guest VCPU is in user mode, the guest can run its instructions natively on a physical CPU.

2. If the guest VCPU is in kernel mode, then the guest believes that it is running in kernel mode. The VMM examines every instruction the guest executes in virtual kernel mode by reading the next few instructions that the guest is going to execute, based on the guest's program counter. Instructions other than special instructions are run natively. Special instructions are translated into a new set of instructions that perform the equivalent task—for example changing the flags in the VCPU. Binary translation is **shown in Fig 5.2.**

3. It is implemented by translation code within the VMM. The code reads native binary instructions dynamically from the guest, on demand, and generates native binary code that executes in place of the original code.

The basic method of binary translation just described would execute correctly but perform poorly. Fortunately, the vast majority of instructions would execute natively.

- VMware tested the performance impact of binary translation by booting one such system, Windows XP, and immediately shutting it down while

monitoring the elapsed time and the number of translations produced by the binary translation method.

- The result was 950,000 translations, taking 3 microseconds each, for a total increase of 3 seconds (about 5%) over native execution of Windows XP. To achieve that result, developers used many performance improvements that we do not discuss here. For more information, consult the bibliographical notes at the end of this chapter.

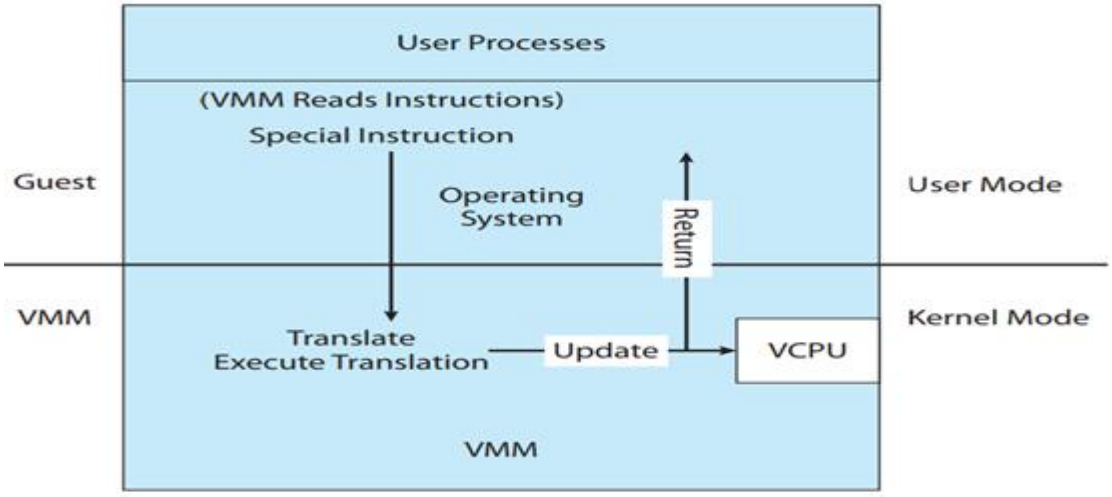


Fig 5.2: **Binary Translation**

(iii)Hardware Assistance

Without some level of hardware support, virtualization would be impossible. The more hardware support available within a system, the more feature-rich and stable the virtual machines can be and the better they can perform. In the Intel x86 CPU family, Intel added new virtualization support in successive generations (the VT-x instructions) beginning in 2005.

- Now, binary translation is no longer needed. In fact, all major general-purpose CPUs are providing extended amounts of hardware support for virtualization. For example, AMD virtualization technology (AMD-V) has appeared in several AMD processors starting in 2006.
- It defines two new modes of operation—host **and guest—thus moving from a dual-mode** to a multimode processor. The VMM can enable host mode, define the characteristics of each guest virtual machine, and then

switch the system to guest mode, passing control of the system to a guest operating system that is running in the virtual machine.

- In guest mode, the **virtualized operating system thinks it is running on native hardware** and sees whatever devices are included in the host's definition of the guest.
- The functionality in Intel VT-X is **similar, providing root and non root modes, equivalent** to host and guest modes. Both provide guest VCPU state data structures to load and save guest CPU state automatically during guest context switches.
- **In addition, virtual machine control structures (VMCSs) are provided to manage** guest and host state, as well as the various guest execution controls, exit controls, and information about why guests exit back to the host.
- In the latter case, for example, a nested **page-table violation caused by an attempt to access** unavailable memory can result in the guest's exit. AMD and Intel have also addressed memory management in the virtual environment. With AMD's RVI and Intel's EPT memory management enhancements, VMMs no longer need to implement software NPTs.
- In essence, these CPUs implement nested **page tables in hardware to allow the VMM** to fully control paging while the CPUs accelerate the translation from virtual to physical addresses. The NPTs add a new layer, one representing the guest's view of logical-to-physical address translation.
- The CPU page-table walking **function includes this new layer as necessary,** walking through the guest table to the VMM table to find the physical address desired. A TLB miss results in a performance penalty, because more tables must be traversed (the guest and host page tables) to complete the lookup.
- Shows the extra translation work **performed by the hardware to translate** from a guest virtual address to a final physical address. I/O is another area improved by hardware assistance. Consider that the standard direct-memory-access (DMA) controller accepts a target memory

address and a source I/O device and transfers data between the two without operating-system action.

- Without hardware assistance, a guest might try to set up a DMA transfer that affects the memory of the VMM or other guests. In CPUs that provide hardware-assisted DMA (such as Intel CPUs with VT-d), even DMA has a level of indirection.
- First, the VMM sets up protection domains to tell **the CPU which physical memory belongs** to each guest. Next, it assigns the I/O devices to the protection domains, allowing them direct access to those memory regions and only those regions.
- The hardware then transforms the address **in a DMA request issued by an I/O device** to the host physical memory address associated with the I/O. In this manner DMA transfers are passed through between a guest and a device without VMM interference. Similarly, interrupts must be delivered to the appropriate guest and must not be visible to other guests.
- By providing an interrupt remapping **feature, CPUs with virtualization hardware** assistance automatically deliver an interrupt destined for a guest to a core that is currently running a thread of that guest.
- That way, the guest receives **interrupts without the VMM's needing to intercede in their** delivery. Without interrupt remapping, malicious guests can generate interrupts that can be used to gain control of the host system. (See the bibliographical notes at the end of this chapter for more details.)

4. Explain in detail about Types of Virtual Machines and Their Implementations. (or) Describe the different types of virtual machines and their implementation. (Nov/Dec 2024)

Without some level of hardware support, virtualization would be impossible. The more hardware support available within a system, the more feature-rich and stable the virtual machines can be and the better they can perform.

- In the Intel x86 CPU family, Intel **added new virtualization support in successive generations (the VT-x instructions) beginning in 2005. Now, binary translation is no** longer needed. In fact, all major general-purpose CPUs are providing extended amounts of hardware support for virtualization.
- The NPTs add a new layer, one representing the guest's view of logical-to-physical address translation. The CPU page-table walking function includes this new layer as necessary, walking through the guest table to the VMM table to find the physical address desired.
- A TLB miss results in a **performance penalty, because more tables must** be traversed (the guest and host page tables) to complete the lookup. Translation work performed by the hardware to translate from a guest virtual address to a final physical address.
- I/O is another area improved by hardware assistance. Consider that the standard direct-memory-access (DMA) controller **accepts a target memory address and a source I/O device and transfers data between the two without operating-system action. Without hardware assistance**, a guest might try to set up a DMA transfer that affects the memory of the VMM or other guests.
- In CPUs that provide hardware-assisted DMA. First, the VMM sets up protection domains to tell the CPU which physical memory belongs to each guest. Next, it assigns the I/O devices to the protection domains, allowing them direct access to those memory regions and only those regions.
- The hardware then transforms the address in a DMA request issued by an I/O device to the host physical memory address **associated with the I/O. In this manner DMA transfers are passed through between a guest and a device without VMM interference. Similarly**, interrupts must be delivered to the appropriate guest and must not be visible to other guests.
- By providing an interrupt remapping feature, CPUs with virtualization hardware assistance automatically deliver an interrupt destined for a guest to a core that is currently running a thread of that guest.

- That way, the guest receives interrupts without the VMM's needing to intercede in their delivery. Without interrupt remapping, malicious guests can generate interrupts that can be used to gain control of the host system.

Types of Virtual Machines and Their Implementations

We've now looked at some of the techniques used to implement virtualization. Next, we consider the major types of virtual machines, their implementation, their functionality, and how they use the building blocks just described to create a virtual environment.

Of course, the hardware on which the virtual machines are running can cause great variation in implementation methods. Here, we discuss the implementations in general, with the understanding that VMMs take advantage of hardware assistance where it is available Shown in Fig 5.3.

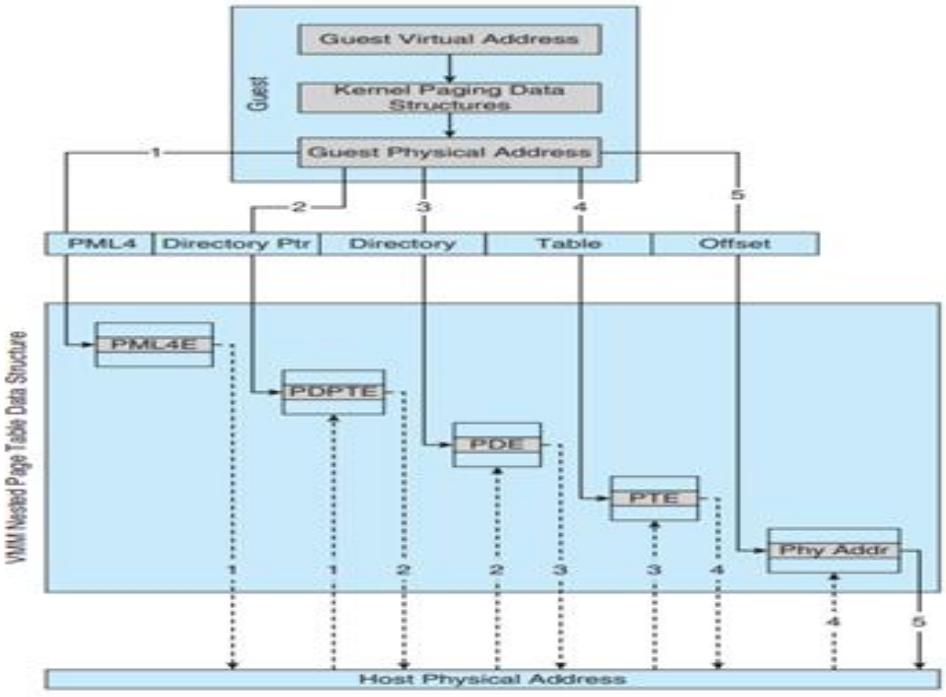


Fig 5.3: VMM Machine

(i) The Virtual Machine Life Cycle

Whatever the hypervisor type, at the time a virtual machine is created, its creator gives the VMM certain parameters. These parameters usually include the

number of CPUs, amount of memory, networking details, and storage details that the VMM will take into account when creating the guest.

For example, a user might want to create a new guest with two virtual CPUs, 4 GB of memory, 10 GB of disk space, one network interface that gets its IP address via DHCP, and access to the DVD drive.

- The VMM then creates the virtual machine with those parameters. In the case of a type 0 hypervisor, the resources are usually dedicated. In this situation, if there are not two virtual CPUs available and unallocated, the creation request in our example will fail. For other hypervisor types, the resources are dedicated or virtualized, depending on the type.
- Certainly, an IP address cannot be shared, but the virtual CPUs are usually multiplexed on the physical CPUs. Similarly, memory management usually involves allocating more memory to guests than actually exists in physical memory.
- Finally, when the virtual machine is no longer needed, it can be deleted. When this happens, the VMM first frees up any used disk space and then removes the configuration associated with the virtual machine, essentially forgetting the virtual machine.
- These steps are quite simple compared with building, configuring, running, and removing physical machines. Creating a virtual machine from an existing one can be as easy as clicking the “clone” button and providing a new name and IP address.
- This ease of creation can lead to virtual machine sprawl, which occurs when there are so many virtual machines on a system that their use, history, and state become confusing and difficult to track.

(ii) Type 0 Hypervisor

Type 0 hypervisors have existed for many years under many names, including “partitions” and “domains”. They are a hardware feature, and that brings its own positives and negatives. Operating systems need do nothing special to take advantage of their features. The VMM itself is encoded in the

firmware and loaded at boot time. In turn, it loads the guest images to run in each partition.

The feature set of a type 0 hypervisor tends to be smaller than those of the other types because it is implemented in hardware. For example, a system might be split into four virtual systems, each with dedicated CPUs, memory, and I/O devices.

- Each guest believes that it has dedicated hardware because it does, simplifying many implementation details. I/O presents some difficulty, because it is not easy to dedicate I/O devices to guests if there are not enough.
- What if a system has two Ethernet ports and more than two guests? For example, either all guests must get their own I/O devices, or the system must provide I/O device sharing. In these cases, the hypervisor manages shared access or grants all devices to a control partition.
- In the control partition, a guest operating system provides services (such as networking) via daemons to other guests, and the hypervisor routes I/O requests appropriately.
- Some type 0 hypervisors are even more sophisticated and can move physical CPUs and memory between running guests. In these cases, the guests are para virtualized, aware of the virtualization and assisting in its execution is **Shown in Fig 5.4.**

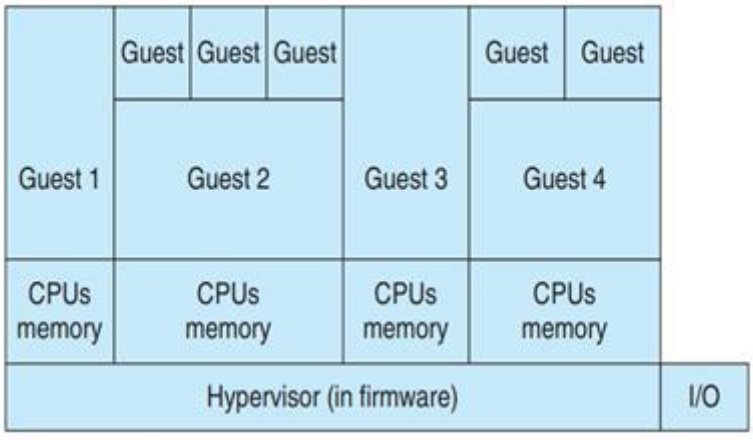


Fig 5.4: Type 0 Hypervisor

(iii) Type 1 Hypervisor

Type 1 hypervisors are commonly found in company data centers and are in a sense becoming "the data-center operating system." They are special-purpose operating systems that run natively on the hardware, but rather than providing system calls and other interfaces for running programs, they create, run, and manage guest operating systems.

- In addition to running on standard hardware, they can run on type 0 hypervisors, but not on other type 1 hypervisors. Whatever the platform, guests generally do not know they are running on anything but the native hardware.
- Type 1 hypervisors run in kernel mode, taking advantage of hardware protection. Where the host CPU allows, they use multiple modes to give guest operating systems their own control and improved performance. They implement device drivers for the hardware they run on, because no other component could do so.
- Because they are operating systems, they must also provide CPU scheduling, memory management, I/O management, protection, and even security. Frequently, they provide APIs, but those APIs support applications in guests or external applications that supply features like backups, monitoring, and security.
- The price of this increased manageability is the cost of the VMM (if it is a commercial product), the need to learn new management tools and methods, and the increased complexity. Another type of type 1 hypervisor includes various general-purpose operating systems with VMM functionality.

(iv) Type 2 Hypervisor

Type 2 hypervisors are less interesting to us as operating-system explorers, because there is very little operating-system involvement in these application level virtual machine managers.

- This type of VMM is simply another process run and managed by the host, and even the host does not know virtualization is happening within the VMM. Type 2 hypervisors have limits not associated with some of the other types.

- For example, a user needs administrative privileges to access many of the hardware assistance features of modern CPUs. If the VMM is being run by a standard user without additional privileges, the VMM cannot take advantage of these features.
- Due to this limitation, as well as the extra overhead of running a general-purpose operating system as well as guest operating systems, type 2 hypervisors tend to have poorer overall performance than type 0 or 1.

- As is often the case, the limitations of type 2 hypervisors also provide some benefits. They run on a variety of general-purpose operating systems, and running them requires no changes to the host operating system. A student can use a type 2 hypervisor, for example, to test a non-native operating system without replacing the native operating system.
- In fact, on an Apple laptop, a student could have versions of Windows, Linux, Unix, and less common operating systems all available for learning and experimentation.

(v) Para virtualization:

As we've seen, Para virtualization takes a different tack than the other types of virtualizations. Rather than try to trick a guest operating system into believing it has a system to itself, Para virtualization presents the guest with a system that is similar but not identical to the guest's preferred system.

- The guest must be modified to run on the Para virtualized virtual hardware. The gain for this extra work is more efficient use of resources and a smaller virtualization layer. The Xen VMM, which is the leader in Para virtualization, has implemented several techniques to optimize the performance of guests as well as of the host system.
- For example, as we have seen, some VMMs present virtual devices to guests that appear to be real devices. Instead of taking that approach, the Xen VMM presents clean and simple device abstractions that allow efficient I/O, as well as good communication between the guest and the VMM about device I/O. For each device used by each guest, there is a

circular buffer shared by the guest and the VMM via shared memory. Read and write data are placed in this buffer

- For memory management, Xen does not implement nested page tables. Rather, each guest has its own set of page tables, set to read-only. Xen requires the guest to use a specific mechanism, a hyper call from the guest to the hypervisor VMM, when a page-table change is needed.

- This means that the guest operating system's kernel code must be changed from the default code to these Xen-specific methods. To optimize performance, Xen allows the guest to queue up multiple page-table changes asynchronously via hyper calls and then check to ensure that the changes are complete before continuing operation.
- Xen allowed virtualization of x86 CPUs without the use of binary translation, instead requiring modifications in the guest operating systems like the one described above. Over time, Xen has taken advantage of hardware features supporting virtualization.
- As a result, it no longer requires modified guests and essentially does not need the Para virtualization method. Para virtualization is still used in other solutions, however, such as type 0 hypervisors **Shown in Fig 5.5.**

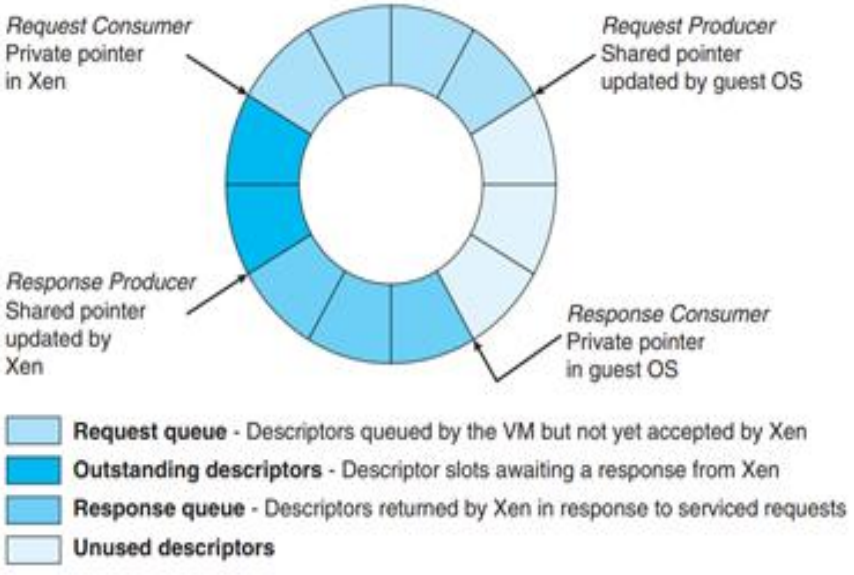


Fig 5.5: **Para virtualization**

(vi) Programming-Environment Virtualization:

Another kind of virtualization, based on a different execution model, is the virtualization of programming environments. Here, a programming language is designed to run within a custom-built virtualized environment.

- For example, Oracle's Java has many features that depend on its running in the Java virtual machine (JVM), including specific methods for security and memory management. If we define virtualization as including only duplication of hardware, this is not really virtualization at all.
- But we need not limit ourselves to that definition. Instead, we can define a virtual environment, based on APIs, that provides a set of features that we want to have available for a particular language and programs written in that language. Java programs run within the JVM environment, and the JVM is compiled to be a native program on systems on which it runs.
- This arrangement means that Java programs are written once and then can run on any system (including all of the major operating systems) on which a JVM is available. The same can be said for interpreted languages, which run inside programs that read each instruction and interpret it into native operations.

(vii) Emulation

Virtualization is probably the most common method for running applications designed for one operating system on a different operating system, but on the same CPU. This method works relatively efficiently because the applications were compiled for the same instruction set as the target system uses.

- But what if an application or operating system needs to run on a different CPU? Here, it is necessary to translate the entire source CPU's instructions so that they are turned into the equivalent instructions of the target CPU.

- Such an environment is no longer virtualized but rather is fully emulated. Emulation is useful when the host system has one system architecture and the guest system was compiled for a different architecture.

- For example, suppose a company has replaced its outdated computer system with a new system but would like to continue to run certain important programs that were compiled for the old system. The programs could be run in an emulator that translates each of the outdated system's instructions into the native instruction set of the new system.

- Emulation can increase the life of programs and allow us to explore old architectures without having an actual old machine. As may be expected, the major challenge of emulation is performance. Instruction-set emulation can run an order of magnitude slower than native instructions.

(viii) Application Containment

The goal of virtualization in some instances is to provide a method to segregate applications, manage their performance and resource use, and create an easy way to start, stop, move, and manage them.

- In such cases, perhaps full-fledged virtualization is not needed. If the applications are all compiled for the same operating system, then we do not need complete virtualization to provide these features. We can instead use application containment is Shown in Fig 5.6.

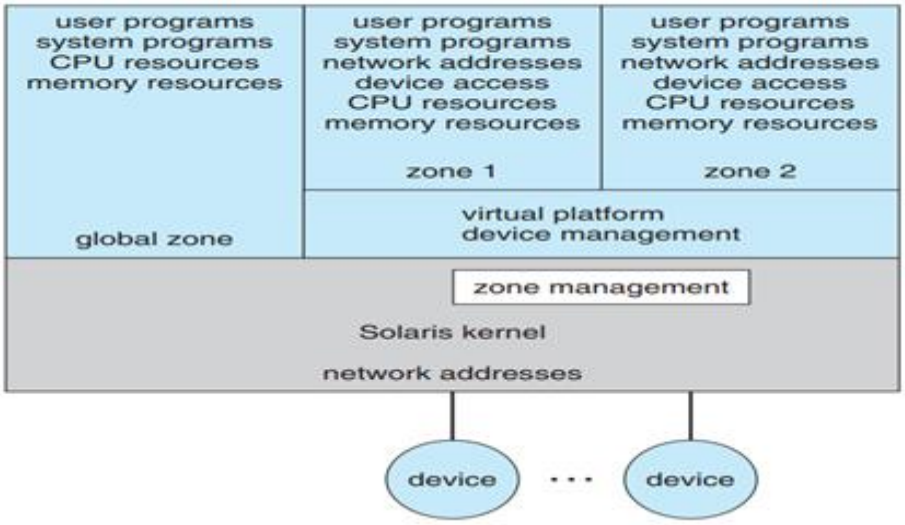


Fig 5.6: **Application Containment**

- Consider one example of application containment. Starting with version 10, Oracle Solaris has included containers, or zones, that create a virtual layer between the operating system and the applications.
- In this system, only one kernel is installed, and the hardware is not virtualized. Rather, the operating system and its devices are virtualized, providing processes within a zone with the impression that they are the only processes on the system.
- One or more containers can be created, and each can have its own applications, network stacks, network address and ports, user accounts, and so on. CPU and memory resources can be divided among the zones and the system-wide processes.
- Each zone in fact can run its own scheduler to optimize the performance of its applications on the allotted resources.

5. Explain in detail about Virtualization and Operating-System Components.

Thus far, we have explored the building blocks of virtualization and the various types of virtualizations. In this section, we take a deeper dive into the operating system aspects of virtualization, including how the VMM provides core operating-system functions like scheduling, I/O, and memory management.

Here, we answer questions such as these: How do VMMs schedule CPU use when guest operating systems believe they have dedicated CPUs? How can memory management work when many guests require large amounts of memory?

i) **CPU Scheduling**

A system with virtualization, even a single-CPU system, frequently acts like a multiprocessor system. The virtualization software presents one or more virtual CPUs to each of the virtual machines running on the system and then schedules the use of the physical CPUs among the virtual machines.

- The significant variations among virtualization technologies make it difficult to summarize the effect of virtualization on scheduling. First, let's consider the general case of VMM scheduling. The VMM has a number of physical CPUs available and a number of threads to run on those CPUs.

- In this situation, the guests act much like native operating systems running on native CPUs. Of course, in other situations, there may not be enough CPUs to go around. The VMM itself needs some CPU cycles for guest management and I/O management and can steal cycles from the guests by scheduling its threads across all of the system CPUs, but the impact of this action is relatively minor.
- More difficult is the case of over commitment, in which the guests are configured for more CPUs than exist in the system. Here, a VMM can use standard scheduling algorithms to make progress on each thread but can also add a fairness aspect to those algorithms.
- For example, if there are six hardware CPUs and 12 guest-allocated CPUs, the VMM could allocate CPU resources proportionally, giving each guest half of the CPU resources it believes it has.
- The VMM can still present all 12 virtual CPUs to the guests, but in mapping them onto physical CPUs, the VMM can use its scheduler to share them appropriately. Even given a scheduler that provides fairness, any guest operating-system scheduling algorithm that assumes a certain

amount of progress in a given amount of time will be negatively affected by virtualization.

- Consider a timesharing operating system that tries to allot 100 milliseconds to each time slice to give users a reasonable response time. Within a virtual machine, this operating system is at the mercy of the virtualization system as to what CPU resources it actually receives.

(ii) Memory Management

Efficient memory use in general-purpose operating systems is one of the major keys to performance. In virtualized environments, there are more users of memory (the guests and their applications, as well as the VMM), leading to more pressure on memory use.

- Further adding to this pressure is that VMMs typically overcommit memory, so that the total memory with which guests are configured exceeds the amount of memory that physically exists in the system. The extra need for efficient memory use is not lost on the implementers of VMMs, who take great measures to ensure the optimal use of memory.
- For example, VMware ESX uses at least three methods of memory management. Before memory optimization can occur, the VMM must establish how much real memory each guest should use. To do that, the VMM first evaluates the maximum memory size of each guest as dictated when it is configured.
- General-purpose operating systems do not expect the amount of memory in the system to change, so VMMs must maintain the illusion that the guest has that amount of memory. Next, the VMM computes a target real memory allocation for each guest based on the configured memory for that guest and other factor, such as over commitment and system load.
- It then uses the three low-level mechanisms below to reclaim memory from the guests. The overall effect is to enable guests to behave and perform as if they had the full amount of memory requested although in reality they have less.
- Recall that a guest believes it controls memory allocation via its page table management, whereas in reality the VMM maintains a nested page table

that re-translates the guest page table to the real page table. The VMM can use this extra level of indirection to optimize the guest's use of memory without the guest's knowledge or help.

- One approach is to provide double paging, in which the VMM has its own page-replacement algorithms and pages to backing-store pages that the guest believes are in physical memory. Of course, the VMM has knows less about the guest's memory access patterns than the guest does, so its paging is less efficient, creating performance problems.
- VMMs do use this method when other methods are not available or are not providing enough free memory. However, it is not the preferred approach.
- At the same time, the guest is using its own memory management and paging algorithms to manage the available memory, which is the most efficient option. If memory pressure within the entire system decreases, the VMM will tell the balloon process within the guest to unpin and free some or all of the memory, allowing the guest more pages for its use.

3. Another common method for reducing memory pressure is for the VMM to determine if the same page has been loaded more than once. If this is the case, to the VMM reduces the number of copies of the page to one and maps the other users of the page to that one copy.

- for example, randomly samples guest memory and creates a hash for each page sampled. That hash value is a "thumbprint" of the page. The hash of every page examined is compared with other hashes already stored in a hash table. If there is a match, the pages are compared byte by byte to see if they really are identical.
- If they are, one page is freed, and its logical address is mapped to the other's physical address. This technique might seem at first to be ineffective, but consider that guests run operating systems. If multiple guests run the same operating system, then only one copy of the active operating-system pages need be in memory.

(iii) Input / Output:

In the area of I/O, hypervisors have some leeway and can be less concerned with exactly representing the underlying hardware to their guests. Because of all the variation in I/O devices, operating systems are used to dealing with varying and flexible I/O mechanisms.

- For example, operating systems have a device-driver mechanism that provides a uniform interface to the operating system whatever the I/O device. Device-driver interfaces are designed to allow third-party hardware manufacturers to provide device drivers connecting their devices to the operating system.
- In the area of networking, VMMs also have work to do. General-purpose operating systems typically have one Internet protocol (IP) address, although they sometimes have more than one—for example, to connect to a management network, backup network, and production network.
- With virtualization, each guest needs at least one IP address, because that is the guest's main mode of communication. Therefore, a server running a VMM may have dozens of addresses, and the VMM acts as a virtual switch to route the network packets to the addressed guest. The guests can be "directly" connected to the network by an IP address that is seen by the broader network (this is known as bridging).
- Alternatively, the VMM can provide a network address translation (NAT) address. The NAT address is local to the server on which the guest is running, and the VMM provides routing between the broader network and the guest. The VMM also provides firewalling, moderating connections between guests within the system and between guests and external systems.

(iv) Storage Management

An important question in determining how virtualization works is this: If multiple operating systems have been installed, what and where is the boot disk? Clearly, virtualized environments need to approach the area of storage management differently from native operating systems.

- Even the standard multi boot method of slicing the root disk into partitions, installing a boot manager in one partition, and installing each other operating system in another partition is not sufficient, because partitioning has limits that would prevent it from working for tens or hundreds of virtual machines.
- Once again, the solution to this problem depends on the type of hypervisor. Type 0 hypervisors do tend to allow root disk partitioning, partly because these systems tend to run fewer guests than other systems.
- Alternatively, they may have a disk manager as part of the control partition, and that disk manager provides disk space (including boot disks) to the other partitions.
- The guest then executes as usual, with the VMM translating the disk I/O requests coming from the guest into file I/O commands to the correct files. Frequently, VMMs provide a mechanism to capture a physical system as it is currently configured and convert it to a guest that the VMM can manage and run.
- Based on the discussion above, it should be clear that this physicalto-virtual (P-to-V) conversion reads the disk blocks of the physical system's disks and stores them within files on the VMM's system or on shared storage that the VMM can access.

(v) Live Migration:

One feature not found in general-purpose operating systems but found in type 0 and type 1 hypervisors is the live migration of a running guest from one system to another. We mentioned this capability earlier.

1. The source VMM establishes a connection with the target VMM and confirms that it is allowed to send a guest.
2. The target creates a new guest by creating a new VCPU, new nested page table, and other state storage.
3. The source sends all read-only memory pages to the target.
4. The source sends all read-write pages to the target, marking them as clean.

5. The source repeats step 4, as during that step some pages were probably modified by the guest and are now dirty. These pages need to be sent again and marked again as clean.

6. When the cycle of steps 4 and 5 becomes very short, the source VMM freezes the guest, sends the VCPU's final state, sends other state details, sends the final dirty pages, and tells the target to start running the guest. Once the target acknowledges that the guest is running, the source terminates the guest is shown in fig 5.7.

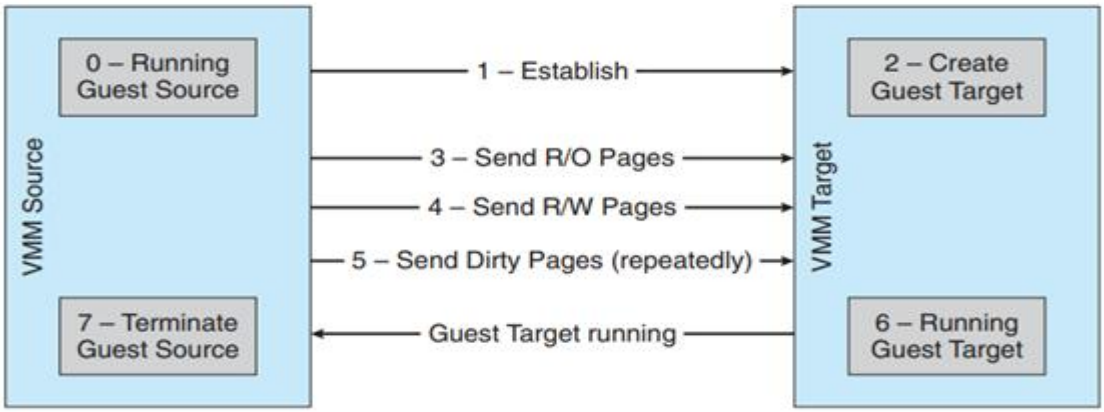


Fig 5.7: VMM Source

- Before virtualization, this did not happen, as the MAC address was tied to physical hardware. With virtualization, the MAC must be movable for existing networking connections to continue without resetting. Modern network switches understand this and route traffic wherever the MAC address is, even accommodating a move. A limitation of live migration is that no disk state is transferred.

6. Write a note on Mobile OS – iOS and Android of Architecture and SDK Framework.(April/May-2021) (or) Discuss in detail the system components of mobile OS. Also analyze the features of iOS and Android OS.(April/May 2024)

iPhone OS becomes iOS

❖ Prior to the release of the iPad in 2010, the operating system running on the iPhone was generally referred to as iPhone OS. Unfortunately, iOS is also the name used by Cisco for the operating system on its routers. When performing an

internet search for iOS, therefore, be prepared to see large numbers of results for Cisco "iOS which have absolutely nothing to do with Apple "iOS.

An Overview of the iOS 6 Architecture

✓ iOS consists of a number of different software layers, each of which provides programming frameworks for the development of applications that run on top of the underlying hardware.

✓ Some diagrams designed to graphically depict the iOS software stack show an additional box positioned above the Cocoa Touch layer to indicate the applications running on the device.

✓ In the above diagram we have not done so since this would suggest that the only interface available to the app is Cocoa Touch. In practice, an app can directly call down any of the layers of the stack to perform tasks on the physical device.

✓ That said, however, each operating system layer provides an increasing level of abstraction away from the complexity of working with the hardware.

✓ As an iOS developer you should, therefore, always look for solutions to your programming goals in the frameworks located in the higher-level iOS layers before resorting to writing code that reaches down to the lower-level layers.

✓ In general, the higher level of layer you program to, the less effort and fewer lines of code you will have to write to achieve your objective. The layers of IOS are given below in fig 5.8.

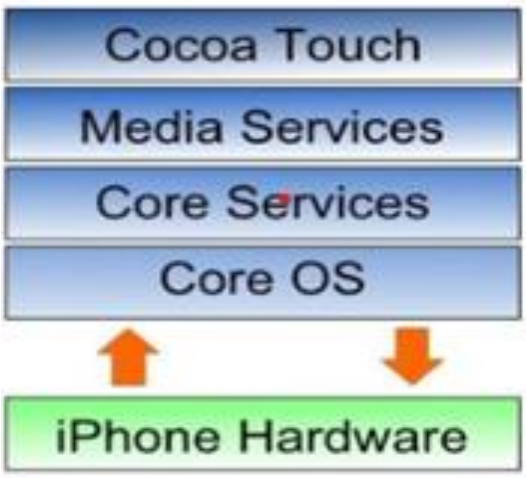


Fig 5.8: **iOS Architecture**

The Cocoa Touch Layer

❖ The Cocoa Touch layer sits at the top of the iOS stack and contains the frameworks that are most commonly used by iPhone application developers. Cocoa Touch is primarily written in Objective-C, is based on the standard Mac OS X Cocoa API (as found on Apple desktop and laptop computers) and has been extended and modified to meet the needs of the iPhone hardware.

The Cocoa Touch layer provides the following frameworks for iPhone app development:

UIKit Framework (UIKit.framework)

❖ The UIKit framework is a vast and feature rich Objective-C based programming interface. It is, without question, the framework with which you will spend most of your time working. Entire books could, and probably will, be written about the UIKit framework alone.

Some of the key features of UIKit are as follows:

- User interface creation and management (text fields, buttons, labels, colors, fonts etc)
- Application lifecycle management
- Application event handling (e.g. touch screen user interaction)
- Multitasking
- Wireless Printing
- Data protection via encryption
- Cut, copy, and paste functionality
- Web and text content presentation and management
- Data handling
- Inter-application integration
- Push notification in conjunction with Push Notification Service
- Local notifications (a mechanism whereby an application running in the background can gain the user's attention)
- Accessibility
- Accelerometer, battery, proximity sensor, camera and photo library interaction

- Touch screen gesture recognition
- File sharing (the ability to make application files stored on the device available via iTunes)
- Blue tooth-based peer to peer connectivity between devices
- Connection to external displays

Map Kit Framework (MapKit.framework)

❖ The Map Kit framework provides a programming interface which enables you to build map-based capabilities into your own applications. This allows you to, amongst other things, display scrollable maps for any location, display the map corresponding to the current geographical location of the device and annotate the map in a variety of ways.

Push Notification Service

❖ The Push Notification Service allows applications to notify users of an event even when the application is not currently running on the device. Since the introduction of this service it has most commonly been used by news based applications.

❖ Typically, when there is breaking news, the service will generate a message on the device with the news headline and provide the user the option to load the corresponding news app to read more details. This alert is typically accompanied by an audio alert and vibration of the device. This feature should be used sparingly to avoid annoying the user with frequent interruptions.

Message UI Framework (MessageUI.framework)

❖ The Message UI framework provides everything you need to allow users to compose and send email messages from within your application. In fact, the framework even provides the user interface elements through which the user enters the email addressing information and message content.

❖ Alternatively, this information may be pre-defined within your application and then displayed for the user to edit and approve prior to sending

Game Kit Framework (GameKit.framework)

❖ The Game Kit framework provides peer-to-peer connectivity and voice communication between multiple devices and users allowing those running the same app to interact.

❖ When this feature was first introduced it was anticipated by Apple that it would primarily be used in multi-player games (hence the choice of name) but the possible applications for this feature clearly extend far beyond games development.

iAd Framework (iAd.framework)

❖ The purpose of the iAd Framework is to allow developers to include banner advertising within their applications. All advertisements are served by Apple‟s own ad service. Event Kit UI Framework (EventKit.framework) The Event Kit UI framework was introduced in iOS 4 and is provided to allow the calendar and reminder events to be accessed and edited from within an application.

❖ Accounts Framework (Accounts.framework) iOS 5 introduced the concept of system accounts. These essentially allow the account information for other services to be stored on the iOS device and accessed from within application code**.**

Social Framework (Social.framework)

❖ The Social Framework allows Twitter, Facebook and SinaWeibo integration to be added to applications. The framework operates in conjunction the Accounts Framework to gain access tothe user‟s social network account information.

7. <u>Explain in detail about the iOS Media Layer</u>.

The iOS Media Layer

❖ The role of the Media layer is to provide iOS with audio, video, animation and graphics capabilities. As with the other layers comprising the iOS stack, the Media layer comprises a number of frameworks which may be utilized when developing iPhone apps. In this section we will look at each one in turn.

Core Video Framework (CoreVideo.framework)

✓ The Core Video Framework provides buffering support for the Core Media framework. Whilst this may be utilized by application developers it is typically not necessary to use this framework.

Core Text Framework (CoreText.framework)

✓ The iOS Core Text framework is a C-based API designed to ease the handling of advanced text layout and font rendering requirements.

Image I/O Framework (ImageIO.framework)

✓ The Image I/O framework, the purpose of which is to facilitate the importing and exporting of image data and image metadata, was introduced in iOS 4. The framework supports a wide range of image formats including PNG, JPEG, TIFF and GIF.

Assets Library Framework (AssetsLibrary.framework)

✓ The Assets Library provides a mechanism for locating and retrieving video and photo files located on the iPhone device. In addition to accessing existing images and videos, this framework also allows new photos and videos to be saved to the standard device photo album.

Core Graphics Framework (CoreGraphics.framework)

✓ The iOS Core Graphics Framework (otherwise known as the Quartz 2D API) provides a lightweight two-dimensional rendering engine. Features of this framework include PDF document creation and presentation, vector-based drawing, transparent layers, path-based drawing, anti-aliased rendering, color manipulation and management, image rendering and gradients. Those familiar with the Quartz 2D API running on MacOS X will be pleased to learn that the implementation of this API is the same on iOS.

Core Image Framework (CoreImage.framework)

✓ A new framework introduced with iOS 5 providing a set of video and image filtering and manipulation capabilities for application developers.

Quartz Core Framework (QuartzCore.framework)

✓ The purpose of the Quartz Core framework is to provide animation capabilities on the iPhone. It provides the foundation for the majority of the

visual effects and animation used by the UIKit framework and provides an Objective-C based programming interface for creation of specialized animation within iPhone apps.

OpenGL ES framework (OpenGLES.framework)

✓ For many years the industry standard for high performance 2D and 3D graphics drawing has been OpenGL. Originally developed by the now defunct Silicon Graphics, Inc (SGI) during the 1990s in the form of GL, the open version of this technology (OpenGL) is now under the care of a non-profit consortium comprising a number of major companies including Apple, Inc., Intel, Motorola and ARM Holdings.

GLKit Framework (GLKit.framework) :

The GLKit framework is an Objective-C based API designed to ease the task of creating OpenGL ES based applications.

NewsstandKit Framework (NewsstandKit.framework)

The Newsstand application is a new feature of iOS 5 and is intended as a central location for users to gain access to newspapers and magazines. The NewsstandKit framework allows for the development of applications that utilize this new service.

iOS Audio Support

iOS is capable of supporting audio in AAC, Apple Lossless (ALAC), A-law, IMA/ADPCM, Linear PCM, μ-law, DVI/Intel IMA ADPCM, Microsoft GSM 6.10 and AES3-2003 formats through the support provided by the following frameworks.

AV Foundation framework (AVFoundation.framework)

✓ An Objective-C based framework designed to allow the playback, recording and management of audio content.

Core Audio Frameworks (CoreAudio.framework, AudioToolbox.framework and AudioUnit.framework)

✓ The frameworks that comprise Core Audio for iOS define supported audio types, playback and recording of audio files and streams and also provide access to the device‟s built-in audio processing units.

Open Audio Library (OpenAL)

✓ OpenAL is a cross platform technology used to provide high-quality, 3D audio effects (also referred to as positional audio). Positional audio may be used in a variety of applications though is typically used to provide sound effects in games.

Media Player Framework (MediaPlayer.framework)

✓ The iOS Media Player framework is able to play video in .mov, .mp4, .m4v, and .3gp formats at a variety of compression standards, resolutions and frame rates.

Core Midi Framework (CoreMIDI.framework)

✓ Introduced in iOS 4, the Core MIDI framework provides an API for applications to interact with MIDI compliant devices such as synthesizers and keyboards via the iPhone‟s dock connector.

8. Explain in detail about TheiOS Core Services Layer.

The iOS Core Services Layer

✓ The iOS Core Services layer provides much of the foundation on which the previously referenced layers are built and consists of the following frameworks.

Address Book Framework (AddressBook.framework)

✓ The Address Book framework provides programmatic access to the iPhone Address Book contact database allowing applications to retrieve and modify contact entries.

CFNetwork Framework (CFNetwork.framework)

✓ The CFNetwork framework provides a C-based interface to the TCP/IP networking protocol stack and low level access to BSD sockets. This enables application code to be written that works with HTTP, FTP and Domain Name servers and to establish secure and encrypted connections using Secure Sockets Layer (SSL) or Transport Layer Security (TLS).

Core Data Framework (CoreData.framework)

✓ This framework is provided to ease the creation of data modeling and storage in Model-ViewController (MVC) based applications. Use of the Core Data framework significantlyreduces the amount of code that needs to be written to

perform common tasks when working with structured data within an application.

Core Foundation Framework (CoreFoundation.framework)

✓ The Core Foundation framework is a C-based Framework which provides basic functionality such as data types, string manipulation, raw block data management, URL manipulation, threads and run loops, date and times, basic XML manipulation and port and socket communication.

✓ Additional XML capabilities beyond those included with this framework are provided via the libXML2 library. Though this is a C-based interface, most of the capabilities of the Core Foundation framework are also available with Objective-C wrappers via the Foundation Framework.

Core Media Framework (CoreMedia.framework)

✓ The Core Media framework is the lower level foundation upon which the AV Foundation layer is built. Whilst most audio and video tasks can, and indeed should, be performed using the higher level AV Foundation framework, access is also provided for situations where lower level control is required by the iOS application developer.

Core Telephony Framework (CoreTelephony.framework)

✓ The iOS Core Telephony framework is provided to allow applications to interrogate the device for information about the current cell phone service provider and to receive notification of telephony related events.

EventKit Framework (EventKit.framework)

✓ An API designed to provide applications with access to the calendar, reminders and alarms on the device.

Foundation Framework (Foundation.framework)

✓ The Foundation framework is the standard Objective-C framework that will be familiar to those who have programmed in Objective-C on other platforms (most likely Mac OS X). Essentially, this consists of Objective-C wrappers around much of the C-based Core Foundation Framework.

Core Location Framework (CoreLocation.framework)

✓ The Core Location framework allows you to obtain the current geographical location of the device (latitude, longitude and altitude) and compass readings from with your own applications.

✓ The method used by the device to provide coordinates will depend on the data available at the time the information is requested and the hardware support provided by the particular iPhone model on which the app is running (GPS and compass are only featured on recent models). This will either be based on GPS readings, Wi-Fi network data or cell tower triangulation (or some combination of the three).

Mobile Core Services Framework (MobileCoreServices.framework)

✓ The iOS Mobile Core Services framework provides the foundation for Apple‟s Uniform Type Identifiers (UTI) mechanism, a system for specifying and identifying data types.

✓ A vast range of predefined identifiers have been defined by Apple including such diverse data types as text, RTF, HTML, JavaScript, PowerPoint .ppt files, PhotoShop images and MP3 files.

Store Kit Framework (StoreKit.framework)

✓ The purpose of the Store Kit framework is to facilitate commerce transactions between your application and the Apple App Store. Prior to version 3.0 of iOS, it was only possible to charge a customer for an app at the point that they purchased it from the App Store. iOS 3.0 introduced the concept of the "in app purchase" whereby the user can be given the option to make additional payments from within the application.

SQLite library

✓ Allows for a lightweight, SQL based database to be created and manipulated from within your iPhone application.

System Configuration Framework (SystemConfiguration.framework)

✓ The System Configuration framework allows applications to access the network configuration settings of the device to establish information about the "reachability" of the device (for example whether Wi-Fi or cell connectivity is active and whether and how traffic can be routed to a server).

Quick Look Framework (QuickLook.framework)

✓ The Quick Look framework provides a useful mechanism for displaying previews of the contents of file types loaded onto the device (typically via an internet or network connection) for which the application does not already provide support. File format types supported by this framework include iWork, Microsoft Office document, Rich Text Format, Adobe PDF, Image files, public.text files and comma separated (CSV).

9. Explain about the iOS Core OS Layer.

The iOS Core OS Layer

➢ The Core OS Layer occupies the bottom position of the iOS stack and, as such, sits directly on top of the device hardware. The layer provides a variety of services including low level networking, access to external accessories and the usual fundamental operating system services such as memory management, file system handling and threads.

Accelerate Framework (Accelerate.framework)

❖ The Accelerate Framework provides a hardware optimized C-based API for performing complex and large number math, vector, digital signal processing (DSP) and image processing tasks and calculations.

External Accessory Framework (ExternalAccessory.framework)

❖ Provides the ability to interrogate and communicate with external accessories connected physically to the iPhone via the 30-pin dock connector or wirelessly via Bluetooth.

Security Framework (Security.framework)

❖ The iOS Security framework provides all the security interfaces you would expect to find on a device that can connect to external networks including certificates, public and private keys, trust policies, keychains, encryption, digests and Hash-based Message Authentication Code (HMAC).

System (LibSystem)

❖ As we have previously mentioned, iOS is built upon a UNIX-like foundation. The System component of the Core OS Layer provides much the same functionality as any other UNIX like operating system. This layer includes

the operating system kernel (based on the Mach kernel developed by Carnegie Mellon University) and device drivers.

❖ The kernel is the foundation on which the entire iOS platform is built and provides the low-level interface to the underlying hardware. Amongst other things, the kernel is responsible for memory allocation, process lifecycle management, input/output, inter-process communication, thread management, low level networking, file system access and thread management.

❖ As an app developer your access to the System interfaces is restricted for security and stability reasons. Those interfaces that are available to you are contained in a C-based library called LibSystem. As with all other layers of the iOS stack, these interfaces should be used only when you are absolutely certain there is no way to achieve the same objective using a framework located in a higher iOS layer.

10. Write the features of mobile OS. (April/May-2019)

Mobile devices are becoming less about the hardware and more about the operating system (OS) running atop of it. Here are four of the most important aspects of a mobile operating system:

1.Speed:

Menus and buttons are about as vital to a mobile experience as much as what the user can do with them. Impossible is it, to download apps and access the very things that a device is supposed to allow us to do if the settings and options to do are as complicated as figuring out quantum physics. It's the difference between 'Open App' and 'Begin Using Software Application'. It's also the difference between button shapes and if they're easily noticeable and understandable.

2. Power to the User:

When it comes to our gadgets, there are few things we enjoy more than a breadth of options. Given the option to change everything from colour schemes to simple concepts like being able to change the background image on the device or to decide how the device greets us on switching it on, usually, the more options the better.

The developer of the operating system want a completely different design to that of the user so while simplistic use may be fine for some, allowing them to add nuts, bolts, screws and brackets to the operating system bracket (besides apps) could be a great thing.

3. Apps

Smartphones and tablets made such a big splash when they were first introduced to the market in part because of how wondrous and fantastical were the devices' touchscreens, the other reason for the success, is likely down to apps.

- Eschewing having to load up the device's built-in web browser to fire up a website and access it that way, apps make that far easier, often providing even more features to their browser counterparts.

- In fact, apps are so advanced that you can even download a brand-new browser to access the Internet on. The only problem is that not every app is available on every OS.

- As it stands, Apple's iOS has far more apps listen on its store and some even have iOS exclusivity so if you want to get your hands on plenty of applications application, that's where you should look. So, while 'quality over quantity' is a good mantra to go by, the more apps available on an operating system, the better.

4. Multi-Tasking

The hardware of a device covers how well and how fast each app or process on the device runs. It's responsible for how many crashes an app will log in a use and what keeps them running as smoothly as possible and, with your hardware up to the challenge of running all of these apps as they should, why not take advantage and ask for more of it?

- Multi-tasking is mostly a new feature in terms of operation systems, with Apple's updated iOS allowing for multiple apps at one time and with Android's latest Jellybean update also letting users multi-task too.

- With devices more and more being used as extensions of offices and workspaces, it makes sense for them to allow us to run as many things as we'd want from our laptops and with devices that do that, it's a wonder why we'd find much use from computers at all.

11. Explain about iOS SDK(Software Development Kit):

The iOS SDK (Software Development Kit) (formerly iPhone SDK) is a software development kit developed by Apple Inc. The kit allows for the development of mobile apps on Apple's iOS operating system.

- While originally developing iPhone prior to its unveiling in 2007, Apple's then-CEO Steve Jobs did not intend to let third-party developers build native apps for iOS, instead directing them to make web applications for the Safari web browser.
- However, backlash from developers prompted the company to reconsider, with Jobs announcing in October 2007 that Apple would have a software development kit available for developers by February 2008. The SDK was released on March 6, 2008.
- The SDK is a free download for users of Mac personal computers. It is not available for Microsoft Windows PCs. The SDK contains sets giving developers access to various functions and services of iOS devices, such as hardware and software attributes.
- It also contains an iPhone simulator to mimic the look and feel of the device on the computer while developing. New versions of the SDK accompany new versions of iOS. In order to test applications, get technical support, and distribute apps through App Store, developers are required to subscribe to the Apple Developer Program.
- Combined with Xcode, the iOS SDK helps developers write iOS apps using officially supported programming languages, including Swift and Objective-C.

Other companies have also created tools that allow for the development of native iOS apps using their respective programming languages.

12. Explain about Mobile Operating Systems

1. Android OS (Google Inc.)

The Android mobile operating system is Google's open and free software stack that includes an operating system, middleware and also key applications for use on mobile devices, including smartphones.

Updates for the open source Android mobile operating system have been developed under "dessert-inspired" version names (Cupcake, Donut, Eclair, Gingerbread, Honeycomb, Ice Cream Sandwich) with each new version arriving in alphabetical order with new enhancements and improvements.

2. Bada (Samsung Electronics)

Bada is a proprietary Samsung mobile OS that was first launched in 2010. The Samsung Wave was the first smartphone to use this mobile OS. Bada provides mobile features such as multipoint-touch, 3D graphics and of course, application downloads and installation.

3. BlackBerry OS (Research In Motion)

The BlackBerry OS is a proprietary mobile operating system developed by Research In Motion for use on the company's popular BlackBerry handheld devices.

The BlackBerry platform is popular with corporate users as it offers synchronization with Microsoft Exchange, Lotus Domino, Novell GroupWise email and other business software, when used with the BlackBerry Enterprise Server.

4. iPhone OS / iOS (Apple)

Apple's iPhone OS was originally developed for use on its iPhone devices. Now, the mobile operating system is referred to as iOS and is supported on a number of Apple devices including the iPhone, iPad, iPad 2 and iPod Touch.

The iOS mobile operating system is available only on Apple's own manufactured devices as the company does not license the OS for third-party hardware. Apple iOS is derived from Apple's Mac OS X operating system.

5. MeeGo OS (Nokia and Intel)

A joint open source mobile operating system which is the result of merging two products based on open source technologies: Maemo (Nokia) and Moblin (Intel). MeeGo is a mobile OS designed to work on a number of devices including smartphones, netbooks, tablets, in-vehicle information systems and various devices using Intel Atom and ARMv7 architectures.

6. Palm OS (Garnet OS)

The Palm OS is a proprietary mobile operating system (PDA operating system) that was originally released in 1996 on the Pilot 1000 handheld.

Newer versions of the Palm OS have added support for expansion ports, new processors, external memory cards, improved security and support for ARM processors and smartphones.

Palm OS 5 was extended to provide support for a broad range of screen resolutions, wireless connections and enhanced multimedia capabilities and is called Garnet OS.

7. Symbian OS (Nokia)

Symbian is a mobile operating system (OS) targeted at mobile phones that offers a high-level of integration with communication and personal information management (PIM) functionality. Symbian OS combines middleware with wireless communications through an integrated mailbox and the integration of Java and PIM functionality (agenda and contacts).

Nokia has made the Symbian platform available under an alternative, open and direct model, to work with some OEMs and the small community of platform development collaborators. Nokia does not maintain Symbian as an open source development project.

8. webOS (Palm/HP)

WebOS is a mobile operating system that runs on the Linux kernel. WebOS was initially developed by Palm as the successor to its Palm OS mobile operating system. It is a proprietary Mobile OS which was eventually acquired by HP and now referred to as webOS (lower-case w) in HP literature.

HP uses webOS in a number of devices including several smartphones and HP TouchPads. HP has pushed its webOS into the enterprise mobile market by focusing on improving security features and management with the release of webOS 3.x. HP has also announced plans for a version of webOS to run within the Microsoft Windows operating system and to be installed on all HP desktop and notebook computers in 2012.

9. Windows Mobile (Windows Phone)

Windows Mobile is Microsoft's mobile operating system used in smartphones and mobile devices – with or without touchscreens. The Mobile OS is based on the Windows CE 5.2 kernel.

13. Differentiate the host and the guest Operating system. (Nov/Dec 2024)

Features	Guest Operating System	Host Operating System
Definition	A guest operating system is a piece of software that runs inside a virtual computer.	A host operating system is a piece of software that runs on a computer and connects with the hardware.
Resource Management:	the guest operating system is responsible for managing its own resources inside the virtual machine.	The host operating system is responsible for managing system resources such as memory, CPU, and disk space

Features	Guest Operating System	Host Operating System
Compatibility	the guest operating system must be compatible with the virtual machine software and hardware abstraction layer provided by the host operating system.	The host operating system must be compatible with the physical hardware or virtual machine,
Purpose	the guest operating system is a secondary operating system that runs inside the virtual machine.	The host operating system is the primary operating system that runs on the physical hardware or virtual machine,
Control	the guest operating system only has control over the virtual resources provided by the virtual machine.	The host operating system has complete control over the physical hardware or virtual machine,
Execution	It executes on a virtual machine	It executes directly on the hardware
Functionality	The guest operating system interacts with the virtual machine.	The host operating system interacts with the hardware.
Quantity	It is possible for the guest OS to be several or single.	It's possible that the host OS is all-in-one.

Features	Guest Operating System	Host Operating System
Operating system on computer	It is secondary to the originally installed operating system on a computer,	Host operating systems use container-based virtualization
Uses	It is used to run more than one application requiring different operating system on the same hardware.	It helps to partition the application in a server

14. Elucidate the architecture of the Linux operating system. (Nov/Dec 2024)

Linux is an open-source UNIX-based operating system. The main component of the Linux operating system is Linux kernel. It is developed to provide low-cost or free operating system service to personal system users, which includes an **X-window system, Emacs editor, IP/TCP GUI, etc**.

Linux distribution:

Linux distribution is developed using a set of software based on compatibility with the Linux core kernel, using which Linux-based operations in different systems, such as personal systems, embedded systems, etc. There are around 600 distributions available.

Some Linux distributions are: MX Linux, Manjaro, Linux Mint, elementary, Ubuntu, Debian, Solus, Fedora, openSUSE, Deepin

Components of Linux:

Like any operating system, Linux consists of software, computer programs, documentation, and hardware.

The main components of Linux operating system are: Application, Shell, Kernel, Hardware, Utilities as shown in figure 5.9.

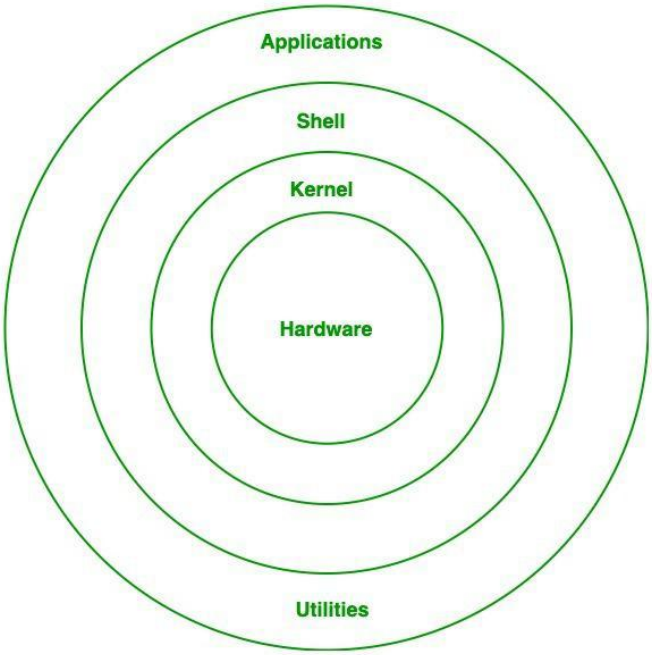


Figure 5.9 *Linux operating system architecture*

1. Kernel:

Kernel is the main core component if Linux, it controls the activity of other hardware components. It visualizes the common hardware resources and provide each process with necessary virtual resources. It makes the process to wait in the ready queue and execute in consequently to avoid any kind of conflict.

Different of types of kernel:

1.1. Monolithic Kernel:

Monolithic kernel is a type of operating system kernel, where all the concurrent processes are executed simultaneously in the kernel itself. All the processes share same memory recourses.

1.2. Micro kernel:

In micro kernel user services and kernel services are executed in separate address spaces. User services are kept in user address space and kernel services are kept in kernel address space.

1.3. Exokernel:

Exo-kernel is designed to manage hardware resources at application level. High level abstraction is used in this operating system to offer hardware resources access to kernel.

1.4. Hybrid kernel:

It is the combination of both monolithic kernel and microkernel. It has speed and design of monolithic kernel and modularity and stability of microkernel.

Main Subsystems of kernel:

- **Process scheduler**: Responsible for fairly distributing the the processing time among all the concurrently running process.
- **Memory management unit:** This kernel sub unit is responsible for proper distribution of memory resources among the concurrently running process.
- Virtual file system: This subsystem provides interface to access stored data across different file system and different physical media. As shown in figure 5.10.

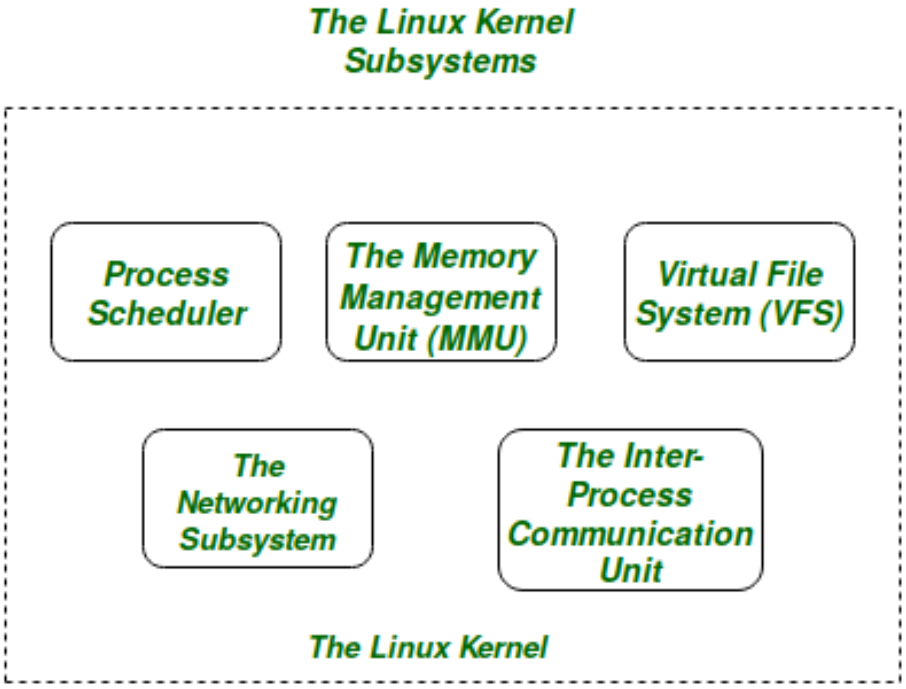


Figure 5.10 *Kernel subsystems*

2. System Library:

System libraries are some predefined functions by using which any application programs or system utilities can access kernel's features. These libraries are the foundation upon which any software can be built.

Some of the most common system libraries are:

1. **GNU C library**: This is the C library that provides the most fundamental system for the interface and execution of C programs. This provides may in-built functions for the execution.
2. **libpthread (POSIX Threads):** This library plays important role for multithreading in Linux, it allows users for creating and managing multiple threads.
3. **libdl (Dynamic Linker):** This library is responsible for the loading and linking file at the runtime.
4. **libm (Math Library):** This library provides user with all kind of mathematical function and their execution.

   Some other system libraries are: librt (Realtime Library), libcrypt (Cryptographic Library), libnss (Name Service Switch Library), libstdc++ (C++ Standard Library)

3. Shell:

Shell can be determined as the interface to the kernel, which hides the internal execution of functions of kernel from the user. Users can just enter the commend and using the kernel's function that specific task is performed accordingly.

Different types of shell:

3.1. Command Line shell:

Executes the command provided by user given in the form command. A special program called terminal in executed and the result is displayed in the terminal itself.

3.2. Graphical User Interface:

Executes the process provided by user in graphical way and output is displayed in the graphical window. As shown in figure 5.11.

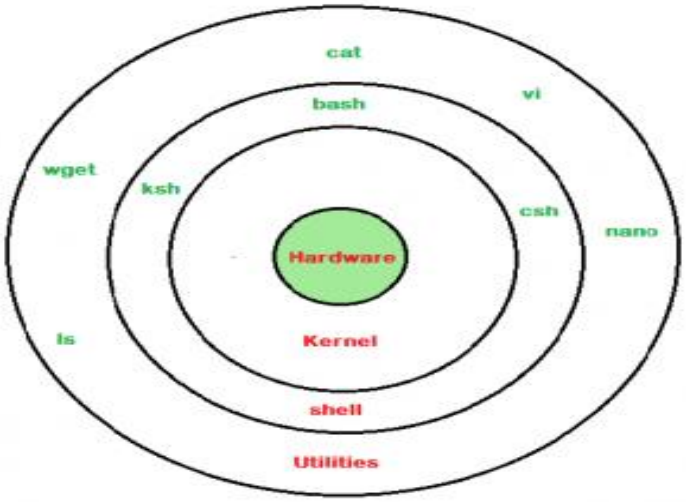


Figure 5.11 *Linux shell*

4. Hardware Layer:

Hardware layer of Linux is the lowest level of operating system track. It is plays a vital role in managing all the hardware components. It includes device drivers, kernel functions, memory management, CPU control, and I/O operations. This layer generalizes hard complexity, by providing an interface for software by assuring proper functionality of all the components.

5. System utility:

System utilities are the commend line tools that preforms various tasks provided by user to make system management and administration better. These utilities enables user to perform different tasks, such as file management, system monitoring, network configuration, user management etc.