F.Y.B.SC.(IT)

SEMESTER - I (CBCS)

OPERATING SYSTEM

SUBJECT CODE : USIT103

Prof. Suhas Pednekar
Vice Chancellor
University of Mumbai, Mumbai.

Prof. Ravindra D. Kulkarni
Pro Vice-Chancellor,
University of Mumbai.

Prof. Prakash Mahanwar
Director
IDOL, University of Mumbai.

Programme Co-ordinator : **Prof. Mandar Bhanushe**
Head, Faculty of Science & Technology,
IDOL, University of Mumbai - 400 098.

Course Co-ordinator **: Gouri S. Sawant**
Assistant Professor B.Sc.IT, IDOL,
University of Mumbai- 400098.

Course Writers **: Asst. Professor Seema Vishwakarma**
Vidyalankar School of Information Technology,
Wadala.

: Asst. Professor Zahirabbas Jainuddin Mulani
Svkm's Mithibai Autonomous College, Mumbai.

: Asst. Professor Rahul E. Borate
Sinhgad Institute of Management, Pune.

: Asst. Professor Madhavi Amondkar
Vidyalankar School of Information Technology,
Wadala.

: Asst. Professor Inumarthi V Srinivas
K. J. Somaiya Institute of Management, Mumbai.

June 2021, Print I

Published by **: Director**
Institute of Distance and Open Learning ,
University of Mumbai,
Vidyanagari, Mumbai - 400 098.

DTP Composed : Varda Offset and Typesetters
Andheri (W), Mumbai - 400 053.

Printed by :

CONTENT

*****

Syllabus

B. Sc (Information Technology)		Semester – I	
Course Name: Operating Systems		**Course Code: USIT103**	
Periods per week (1 Period is 50 minutes)		5	
Credits		2	
		Hours	**Marks**
Evaluation System	Theory	2½	75
	Examination	-	25

Unit	Details	Lectures
I	**Introduction:** What is an operating system? History of operating system, computer hardware, different operating systems, operating system concepts, system calls, operating system structure. **Processes and Threads:** Processes, threads, interprocess communication, scheduling, IPC problems.	**12**
II	**Memory Management:** No memory abstraction, memory abstraction: address spaces, virtual memory, page replacement algorithms, design issues for paging systems, implementation issues, segmentation. **File Systems:** Files, directories, file system implementation, file-system management and optimization, MS-DOS file system, UNIX V7 file system, CD ROM file system.	**12**
III	**Input-Output:** Principles of I/O hardware, Principles of I/O software, I/O software layers, disks, clocks, user interfaces: keyboard, mouse, monitor, thin clients, power management, **Deadlocks:** Resources, introduction to deadlocks, the ostrich algorithm, deadlock detection and recovery, deadlock avoidance, deadlock prevention, issues.	**12**
IV	**Virtualization and Cloud:** History, requirements for virtualization, type 1 and 2 hypervisors, techniques for efficient virtualization, hypervisor microkernels, memory virtualization, I/O virtualization, Virtual appliances, virtual machines on multicore CPUs, Clouds. **Multiple Processor Systems** Multiprocessors, multicomputers, distributed	**12**

	systems.	
V	**Case Study on LINUX and ANDROID:** History of Unix and Linux, Linux Overview, Processes in Linux, Memory management in Linux, I/O in Linux, Linux file system, security in Linux. Android **Case Study on Windows:** History of windows through Windows 10, programming windows, system structure, processes and threads in windows, memory management, caching in windows, I/O in windows, Windows NT file system, Windows power management, Security in windows.	**12**

Books and References:

Sr. No.	Title	Author/s	Publisher	Edition	Year
1.	Modern Operating Systems	Andrew S. Tanenbaum, Herbert Bos	Pearson	4th	2014
2.	Operating Systems – Internals and Design Principles	Willaim Stallings	Pearson	8th	2009
3.	Operating System Concepts	Abraham Silberschatz, Peter B. Galvineg Gagne	Wiley	8th	
4.	Operating Systems	Godbole and Kahate	McGraw Hill	3rd	

B. Sc (Information Technology)		Semester – I	
Course Name: Operating Systems Practical		**Practical Course Code: USIT1P3**	
Periods per week (1 Period is 50 minutes)		5	
Credits		2	
		Hours Marks	Theory Examination
Evaluation System	Theory Examination	2½	75
	Internal	-	25

List of Practical:

1.	Installation of virtual machine software.
2.	Installation of Linux operating system (RedHat / Ubuntu) on virtual machine.

3.	Installation of Windows operating system on virtial machine.
4.	**Linux commands: Working with Directories:**
a.	pwd, cd, absolute and relative paths, ls, mkdir, rmdir,
b.	file, touch, rm, cp. mv, rename, head, tail, cat, tac, more, less, strings, chmod
5.	**Linux commands: Working with files:**
a.	ps, top, kill, pkill, bg, fg,
b.	grep, locate, find, locate.
c.	date, cal, uptime, w, whoami, finger, uname, man, df, du, free, whereis, which.
d.	Compression: tar, gzip.
6.	**Windows (DOS) Commands – 1**
a.	Date, time, prompt, md, cd, rd, path.
b.	Chkdsk, copy, xcopy, format, fidsk, cls, defrag, del, move.
7.	**Windows (DOS) Commands – 2**
a.	Diskcomp, diskcopy, diskpart, doskey, echo
b.	Edit, fc, find, rename, set, type, ver
8.	**Working with Windows Desktop and utilities**
a.	Notepad
b.	Wordpad
c.	Paint
d.	Taskbar
e.	Adjusting display resolution
f.	Using the browsers
g.	Configuring simple networking
h.	Creating users and shares
9.	**Working with Linux Desktop and utilities**
a.	The vi editor.
b.	Graphics
c.	Terminal
d.	Adjusting display resolution
e.	Using the browsers
f.	Configuring simple networking
g.	Creating users and shares
10.	**Installing utility software on Linux and Windows**

UNIT I

1

INTRODUCTION

Unit Structure

1.0 OBJECTIVES

The objective of the chapter is as follow

- To get familiar with core component of operating systems
- To understand the different generation of operating system
- To understand the different functionality of system

1.1 INTRODUCTION

- Operating systems provides a clear, best and simple view of the computer to the users.

- Operating system performs the function of resource handling and distributing the resources to the different part of the system.
- It is the intermediary between users and the computer system and provide a level of abstraction due to which complicated details can be kept hidden from the user.

1.2 WHAT IS OPERATING SYSTEM

1.2.1 Definition:

- Operating System is a system software which acts as an intermediary between user and hardware.
- Operating System keeps the complicated details of the hardware hidden from the user and provides user with easy and simple interface.
- It performs functions which involves allocation of resources efficiently between user program, file system, Input Output device.

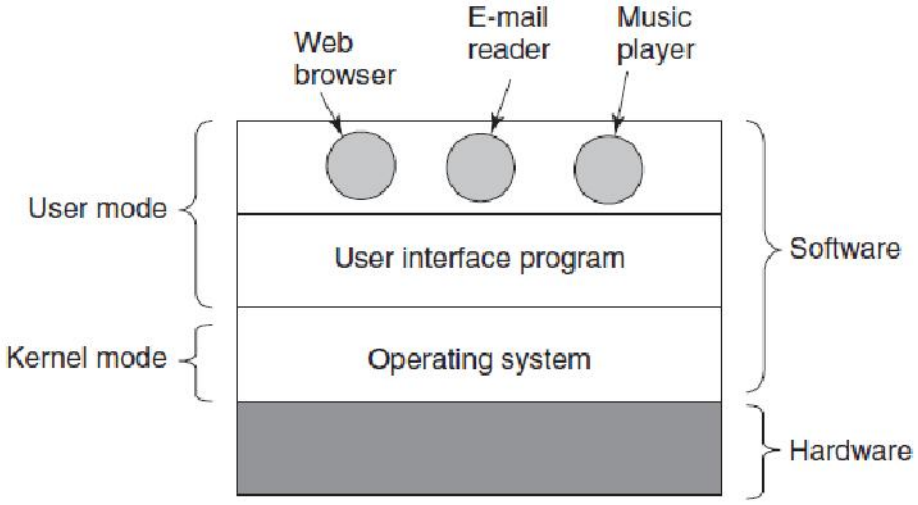


Figure 1. Abstract view of Operating system
Reference: Modern Operating system, Fourth edition, Andrew S. Tanenbaum, Herbert Bos

Explanation of Figure 1:

- The hardware components lies at the bottom of the diagram. It is considered as the most crucial part of computer system. To protect the hardware from direct access, it is kept at the lowest level of hierarchy. Hardware components includes circuits, input output device, monitor etc
- Operating system runs in the kernel mode of the system wherein the OS gets an access to all hardware and can execute all machine instructions. Other part of the system runs in user mode.

1.2.2 The Operating System as an Extended Machine:

- The structure of computers system at the machine-language level is complicated to program, especially for input/output. Programmers don't deal with hardware, so a level of abstraction is supposed to be maintained.
- Operating systems provides layer of abstraction for using disks: files.
- Abstraction allows a programs to create, write, and read files, without having to deal with the messy details of how the hardware actually works
- Abstraction is the key to managing all the complexity.
- Good abstractions turn a nearly impossible task into two manageable ones.
- The first is defining and implementing the abstractions.
- The second is using these abstractions to solve the problem at hand.
- operating system primarily provides abstractions to application programs in a top- down view
- E.g.: It is much easier to deal with photos, emails, songs, and Web pages than with the details of these files on SATA (or other) disks.

1.2.3 The Operating System as a Resource Manager:

- Modern computers consist of processors, memories, timers, disks, mice, network interfaces, printers, and a wide variety of other devices.
- In the bottom-up view, the operating system provides for an orderly and controlled allocation of the processors, memories, and I/O devices among the various programs.
- Operating system allows multiple programs to be in memory and run at the same time.
- Resource management includes multiplexing (sharing) resources in two different ways: in time and in space.
- In time multiplexed, different programs takes turns using CPU. First one of them gets to use the resource, then the another, and so on.
- E.g.: Sharing the printer. When multiple print jobs are queued up for printing on a single printer, a decision has to be made about which one is to be printed next
- In space multiplexing, Instead of the customers taking turns, each one gets part of the resource.
- E.g.: main memory is divided up among several running programs, so each one can be resident at the same time.

1.3 HISTORY OF OPERATING SYSTEM

English mathematician Charles Babbage (1792–1871) developed the first true digital computer which was purely mechanical, and the technology of his day could not produce

1.3.1 First Generation OS:

- First generation were also known as Vacuum Tube
- Single group of people were responsible for creating, building, programming, operating, and maintenance of each machine
- Programming was done by connecting the electrical circuit on plugboard with thousands of cables
- Programmer used to sign up for a block of time using the signup sheet on the wall then come down to the machine room, insert his or her plugboard into the computer, and spend the next few hours hoping that none of the 20,000 or so vacuum tubes would burn out during the run.

1.3.2 Second Generation OS:

- Second Generation computers were also known as Transistors and Batch Systems.
- Computer in this era was reliable and manufactured for the purpose of selling it to the customers like government agencies or universities.
- Separate groups were formed for working on designing, building and coding aspects of computer
- Computers were known as mainframes and were kept in separate rooms.
- Separate machines were build for calculation and for input/output.
- Programs were known as job. Jobs were entered in groups called as batch
- Second-generation computers were used for scientific and engineering calculations of physics and engineering.

Year	1955 -65
Programming language	Fortran, assembler
Operating system	IBM's operating system FM
Hardware	Transistors and Batch Systems, punch card, magnetic tape

1.3.3 Third Generation OS:

- Third Generation computers were known ICs and Multiprogramming
- Maintaining two computers were not easy so IBM introduced its first computer names System 360 made by using Integrated circuit

- The main purpose of this generation was all software, including the operating system, OS/360, worked on all models
- Important feature identified in this generation was multiprogramming where in when one job was waiting for I/O to complete, another job could be using the CPU. This way maximum utilization of CPU could be achieved
- Spooling that has an ability to read jobs from cards onto the disk
- Time sharing, which allocates the CPU in turns to number of users
- Third generation computers were used for Large scientific calculations and massive commercial data-processing runs

1.3.4 Fourth Generation OS:

- Fourth generation computers were also known as Personal computers
- Extremely small size computers could be created using microchips which made it possible for a single individual to have his own personal computer
- Companies like Intel and IBM started creating OS for their respective CPU
- User friendly GUI were built for general purpose usage
- Microsoft came up with different versions of Windows
- Network operating systems and distributed systems became popular in this era
- In network operating system, users log in to remote machines and copy files from one machine to another.
- A distributed operating system, is composed of multiple processors but appears to its users as a single uniprocessor unit

Year	1980 – Present
Programming language	High level programming language
Operating system	DOS, Windows, UNIX, FreeBSD
Hardware	LSI(large Scale Integration) circuit, chips, transistors
Computers	IBM 4341, DEC 10,STAR 100

1.3.5 Fifth Generation OS:

- Fifth generation was also known as Mobile computer, made by using Ultra Large Scale Integrated Chips
- New operating systems like symbians, Blackberry OS, iOS ,Android became popular in the market
- Devices become more portable and smaller in size
- Artificial intelligence is used on a large scale to construct a device

which uses natural language processing for analysis of input.

- Computers in this era were capable of self learning

Year	1990 – Present
Programming language	High level programming language
Operating system	iOS, Android, Symbians,RIM
Hardware	Ultra large scale integrated chip
Computers	Handheld devices, wearable devices, PDA, Smart phone

1.4.1 Processor:

- CPU is the most vital part of the computer. The instructions are fetched from the memory and executed by CPU using fetch-decode-execute
- CPUs contains registers inside to hold key variables and temporary data.
- Special registers called as program counter contains memory address of the next instruction to be fetched. Program Status Word contains the condition code bits
- The Intel Pentium 4 introduced multithreading or hyperthreading to the x86 processor, allowing the CPU to hold the state of two different threads and then switch back and forth in nanosecond.
- A GPU is a processor with thousands of tiny cores which are very good for many small computations done in parallel like rendering polygons in graphics applications

1.4.2 Memory:

- The basic expectations from the memory is its speed, storage and performance but a single memory is not capable of fulfilling the same
- The memory system is based on hierarchy of layers.
- Registers inside the CPU forms the top layer in the hierarchy which gives quick access to data.
- Cache memory is next in the hierarchy. Most heavily used data are kept inside the cache for high speed access using cache hit and cache miss.
- Two types of cache are present in the system depending upon the manufacturing company cache L1 and L2
- The cache that is always inside the CPU is L1 which enters decoded instructions inside the CPU
- L2 cache holds megabytes of memory words which were recently used The difference between the L1 and L2 caches lies in the timing.
- Main memory comes next in the hierarchy also known as RAM. Cache Miss request goes inside the main memory for

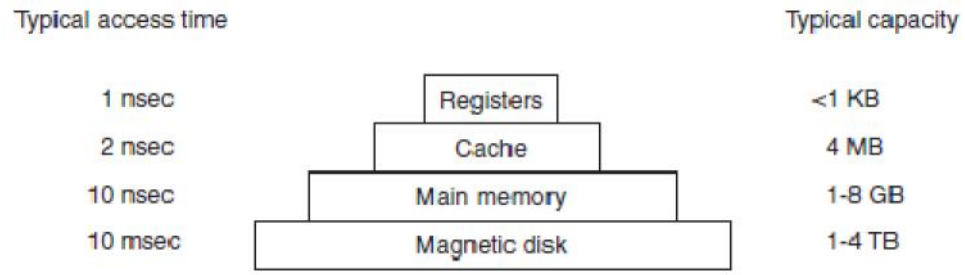


Figure 2. Memory hierarchy

Reference: ModernOperating system, Fourth edition, Andrew S. Tanenbaum, Herbert

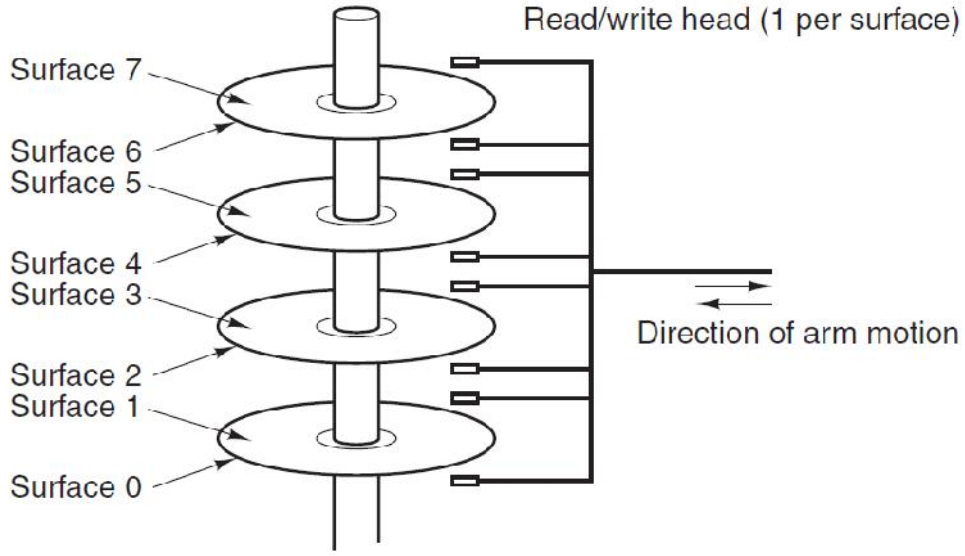


Figure 3. Disk Structure

Reference: Modern Operating system, Fourth edition, Andrew S. Tanenbaum, Herbert Bos

- Disk is a mechanical device capable of storage which is cheaper and larger than RAM.
- The only problem is that the time to randomly access data on it is slower.
- A disk consists metal plats that rotate at 5400, 7200, 10,800 RPM or more.
- An arm pivots over the plats from the corner
- Each of the heads can read an annular region called a track.
- Together, all the tracks for a given arm position form a cylinder.
- Each track is divided into some number of sectors, typically 512 bytes per sector.

1.4.3 Booting the computer:

- Process of loading the kernel is known as booting the system

- Parent board consist of a program called as BIOS(Basic Input Output System)
- BIOS starts with its responsibility of checking RAM, basic devices and PCI buses as soon as the system is booted. It scans and checks the response of the devices.
- Once the initial check is done, BIOS starts the boot device from the hard disk
- First section of the boot device is read into the memory and executed
- The secondary boot loader present inside the sector is read inside the memory
- The loader reads the operating system and starts it

1.5 LET US SUM UP

- Operating System is a system software which acts as an intermediary between user and hardware
- The Operating System acts as an Extended Machine by providing level of abstraction.
- Operating System is responsible for Resource allocation
- Five generations of computer operating systems have evolved.
- Different components of hardware interact with operating system which in turn interacts with the other applications

1.6 LIST OF REFERENCE

Modern Operating system, Fourth edition, Andrew S. Tanenbaum, Herbert Bos
https://www.geeksforgeeks.org/generations-of-computer/

1.7 BIBLIOGRAPHY

Modern Operating System by galvin

1.8 UNIT END QUESTIONS

1. Explain Third Generation operating system
2. Define Operating System. Explain the role of OS as an extended machine.
3. Write a short note on the fifth generation Operating System.
4. With a suitablediagram, explain the structure of the disk drive.
5. Explain the process of Booting in computer
6. Define Operating System. How it can be used as a resource manager.

*****

2

OPERATING SYSTEM CONCEPT

Unit Structure

2.0 OBJECTIVES

- The objective of the chapter is as follow

- To understand the operating system services provided to the users and processes,
- To understand the various operating system structures.
- To describe various types of operating system.

2.1 INTRODUCTION

An operating system provides the environment within which programs are executed. It is important to understand the goals of the system which will help us to select the algorithm and strategies for the designing of the system

2.2 DIFFERENT OPERATING SYSTEM

2.2.1 Mainframe Operating Systems:

- Mainframe operating systems are used in web servers of e commerce websites or servers dedicated for business-to-business transaction.
- The operating systems of Mainframe operating systems are oriented in such a way that it can handle many jobs simultaneously
- Mainframe Operating systems can deal with large amount of input output transaction.
- The main services of mainframe operating systems are
    - to handle batch processing of jobs.
    - to handle transaction processing of multiple request.
    - timesharing of servers allowing multiple remote users to have an access to the server.

2.2.2 Server Operating Systems:

- Server Operating Systems are the ones that runs on the machine which are dedicated servers.
- Solaris, Linux and Windows are some examples of Server Operating Systems
- Server Operating Systems allows sharing of multiple resources like hardware, files or print services
- Web pages are stored on a dedicated server to handle request and response.

2.2.3 Multiprocessor Operating System:

- Multiprocessor Operating Systems are also known as parallel computers or multicomputer depending upon how multiple processors are connected and shared.

- These computers have high speed communication mechanism with strong connectivity.
- Personal computer are also created and embedded with the multiprocessor technology.
- Multiprocessor operating system give high processing speed as multiple processors into single system.

2.2.4 Personal Operating Systems:

- Personal operating systems are installed in machines used by common and large number of users.
- They support multiprogramming, running multiple programs like word, excel, games, and Internet access simultaneously on one machine.
- Examples Linux, Windows, Mac.

2.2.5 Handheld Operating System:

- Handheld operating systems are found in all handheld devices like Smart phone and tablets. It is also known as Personal Digital Assistant.
- The most popular operating systems in today's market are android and iOS.
- These operating systems need high processing processor. It is also embedded with different types of sensor.

2.2.6 Embedded Operating Systems:

- Embedded operating systems are designed for those devices which are not considered as computers. These operating systems are preinstalled on the devices by the device manufacturer.
- All pre installed softwares are in ROM and no changes could be done to it by the users.
- The best example of embedded operating systems is washing machines, oven etc.

2.2.7 Real-Time Operating Systems:

- Real Time Operating systems have strict time constraints due to which it is used in applications that are very critical in terms of safety.
- Real time operating system are classified into hard real time and soft real time
- Hard real time systems have very stringent time constraints, certain actions should occur at that time only. Components are tightly coupled in hard real time
- Soft real time operating system is the one where missing of deadlines some time will not cause damage

2.2.8 Smart Card Operating System:

- Smart Card Operating Systems runs on smart cards. They contain processor chip embedded inside the CPU chip.
- They have very high processing power and memory constraints
- These operating systems can handle single function like making electronic payment and are license softwares.

2.3 OPERATING SYSTEM CONCEPTS

Operating Systems concepts deals with process, address space, file, input output devices

Process: A process is a program which is in execution mode. Each process has an address space. All data related to process is stored in a table called as process table. All details for running the program is contained in the process. A process can reside in anyone of the five states in its life time. There are background and foreground processes running in the systems carrying out different functions. These processes communicates with each other using interprocess communication

Address space: Computer need a mechanism to distinguish between process sitting inside the main memory. This is done by allocating he process to an address space. computers addresses are 32 or 64 bits, giving an address space of 232 or 264 bytes. Virtual address spaces are playing an important role in dealing with the problem of insufficient memory space.

Files are the data which the user want to retrieve back from the computer. Operating systems is supposed to maintain the data in hard disk and retrieve it from it when ever needed. Operating system arranges files in the form of hierarchy. The data goes inside the directory.

Shell : Shell is the command interpreter for UNIX. Shell become the intermediate between the user on the terminal and the operating systems. Every shell has a terminal for entering the data and to get the output. Instructions are given in the form of commands to the computer.

2.4 SYSTEM CALLS

It is a way by which user program request for services from the kernel. System calls provide an interface to the services made available by an operating system.

Step by step explanation of system call mechansim:

- A process running a user program in user mode want to execute read instruction a file, it has to execute a trap instruction to transfer control to the operating system.

- System call read has three parameters: the first one specifying the file, the second one pointing to the buffer, and the third one giving the number of bytes to read.
- count = read(fd, buffer, nbytes);
- the parameters are first pushed onto stack.
- library procedure are called in the step 4
- The library procedure, possibly written in assembly language, typically puts the system- call number in a place where the operating system expects it, such as a register (step 5)
- Then it executes a TRAP instruction to switch from user mode to kernel mode and start execution at a fixed address within the kernel (step 6).
- The kernel code that starts following the TRAP examines the system-call number and then dispatches to the correct system-call handler, usually via a table of pointers to system-call handlers indexed on system-call number (step 7).
- At that point the system-call handler runs (step 8).
- Once it has completed its work, control may be returned to the user-space library procedure at the instruction following the TRAP instruction (step 9).
- 12. This procedure then returns to the user program in the usual way procedure calls return (step 10).

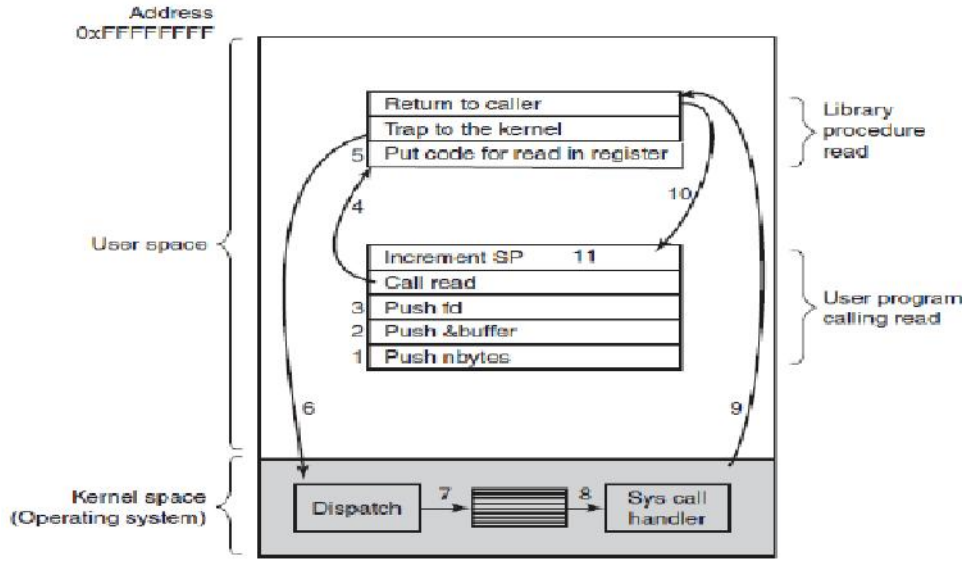


Figure 2.1 system call for read

Reference: Modern Operating system, Fourth edition, Andrew S. Tanenbaum, Herbert Bos

2.4.1 System calls for Process management:

- A system to create a new process or a duplicate process is fork

- The duplicate process will have all data in the file description and registers common.
- The original process is known as parent process and the duplicate is known as the child process
- The fork call returns a value, which is zero in the child and equal to the child's PID (Process IDentifier) in the parent.
- System calls like exit would request the services for terminating a process
- Loading of program or changing of the original image with duplicate needs execution of exec
- Pid would help to distinguish between child and parent process
- Eg of Process management system calls in Linux
- fork: for creating a duplicate process from parent process
- wait: process are supposed to wait for other processes to complete their work
- exec: loads the selected program into the memory
- exit: terminates the process

2.4.2 System calls for File management:

- A file is open using a system call open.
- The mode in which the file is supposed to be open is specified using the parameter. Parameters also consist of the names of the file to open or a new one to be created.
- The files are closed using the close systems.
- Associated with each file is a pointer that indicates the current position in the file.
- When reading (writing) sequentially, it normally points to the next byte to be read (written). The lseek call changes the value of the position pointer, so that subsequent calls to read or write can begin anywhere in the file.
- Lseek has three parameters: the first is the file descriptor for the file, the second is a file position, and the third tells whether the file position is relative to the beginning of the file, the current position, or the end of the file.
- Eg of systems calls for file management
- open: for opening the file for reading, writing
- close: to close the opened file
- read: for reading the data from the file into buffer
- write: for writing the data from the buffer into file

2.4.3 System calls for Directory management:

- mkdir is a system call that creates and empty directories, whereas rmdir removes an empty directories.
- link allows the same file to appear under two or more names, often in different directories for allowing several members of the same programming team to share a common file, with each of them having the file appear in his own directory, possibly under different names.
- By executing the mount system call, the USB file system can be attached to the root file system
- The mount call makes it possible to integrate removable media into a single integrated file hierarchy, without having to worry about which device a file is on

2.4.5 Windows Win32 API:

- Window's program are event driven. An event occurs that calls the procedure to handle it. Windows functioning is most driven by GUI based interactions like mouse movement. There are system calls which are exclusively present windows to deal with GUI and many of the systems calls which are present in UNIX are missing here.Following are some of it:
- CreateProcess: Creates a new process in Win32
- WaitForSingleObject: Waits for a process to exit
- ExitProcess: Terminates the execution of process
- CreateFile: Opens an existing file or creates a new one

2.5 OPERATING SYSTEM

2.5.1 Monolithic System:

- In the monolithic approach the entire operating system runs as a single program in kernel mode
- The operating system is written as a collection of procedures, linked together into a single large executable program.
- Each procedure in the system is free to call any other process
- Being able to call any procedure makes the system very efficient
- No information hiding —every procedure is visible to every other procedure
- E.g. MS DOS and LINUX
- This organization suggests a basic structure for the operating system:
- Main Function- invokes requested service procedure
- Service Procedures- carry out system calls
- Utility functions- Help service procedures to perform certain tasks

Disadvantage:

- Difficult and complicated structure
- A crash in any of these procedures will take down the entire operating system

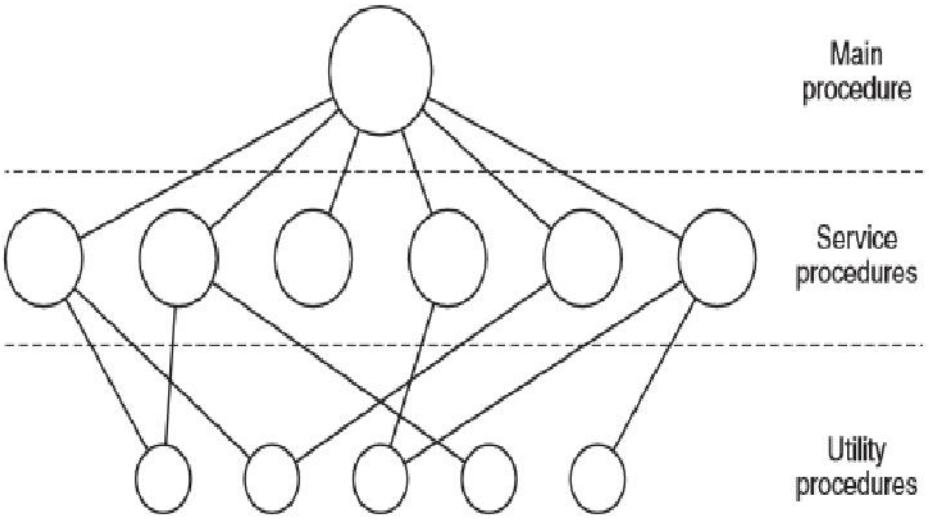


Figure 2.2 Monolithic Structure

Reference: Modern Operating system, Fourth edition, Andrew S. Tanenbaum, Herbert Bos

2.5.2 Layered System:

Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

Figure 2.2. Layered Struture

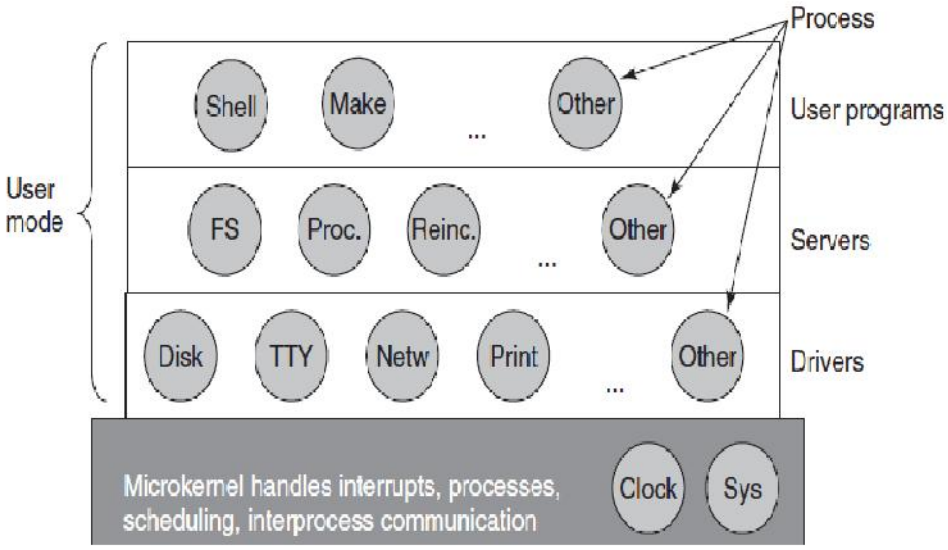
Reference: Modern Operating system, Fourth edition, Andrew S. Tanenbaum, Herbert Bos

- The operating system is organized as a hierarchy of layers, each one constructed upon the one below it. The first system constructed in this way was the THE system. The same concept of layered approach was also implemented by MULTICS with concentric rings. The procedures

in out rings are supposed to make a system call to access the process in the inner ring

- The diagram reflects the structure of The operating system with following details
- Layer 0 dealt with allocation of the processor, switching between processes when interrupts occurred or timers expired.
- Layer 1 did the memory management. It allocated space for processes in main memory.
- Layer2 handled communication between each process and the operator console·
- Layer 3 took care of managing the I/O devices and buffering the information streams·
- Layer 4 was where the user programs were found.
- Layer 5 : The system operator process was located.
- TRAP instruction whose parameters were carefully checked for validity before the call was allowed to proceed.

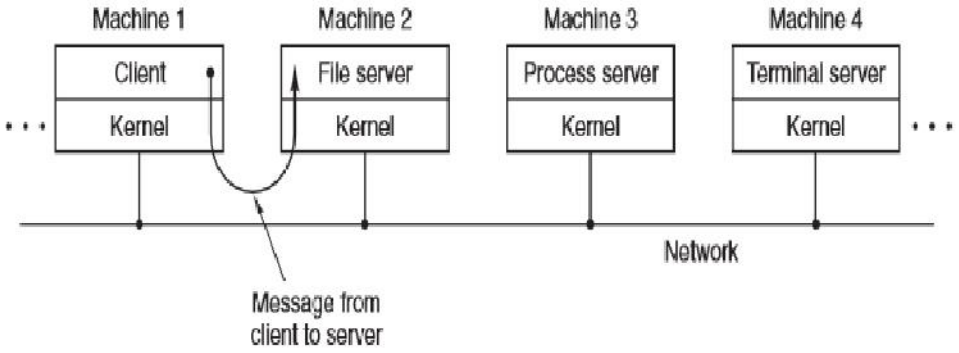
2.5.3 : Microkernels:



- Microkernel structure focusses on making the kernel smaller by reducing the non essential components from the kernel. These non essential components are placed in user space.
- The basic idea behind the microkernel design is to achieve high reliability by splitting the operating system up into small, well-defined modules.
- the microkernel—runs in kernel mode.
- The main function of microkernel is to provide a communication facility between the client program and various services that are also running in user space

- All new services are added to the user space and the kernel don't need to be modified.
- Microkernel provides high security and reliability as most of the services are running in user space , if a service fails the rest operating system remains untouched.
- Disadvantage
    - Performance decrease due to increased system function overhead

2.5.4 Client Server System:



- The servers, each of which provides some service, and the clients, which use these services. This model is known as the client-server model.
- Since clients communicate with servers by sending messages, the clients need not know whether the messages are handled locally on their own machines, or whether they are sent across a network to servers on a remote machine.
- As far as the client is concerned,: requests are sent and replies come back.
- Thus the client-server model is an abstraction that can be used for a single machine or for a network of machines

2.5.5 Exokernel:

- Exokernel runs in the bottom layer of kernel mode.
- Its job is to allocate resources to virtual machines and then check attempts to use them to make sure no machine is trying to use somebody else's resources.
- The advantage of the exokernel scheme is that it saves a layer of mapping whereas the virtual machine monitor must maintain tables to remap disk addresses
- The exokernel need only to keep track of which virtual machine
- has been assigned which resource

2.5 LET US SUM UP

1. Different types of operating systems are used in different types of machines depending upon the need of the user. Some of it are mainframe operating system, server operating system, embedded operating system, handheld operating system
2. System calls explains what the operating system does. Different types of system calls are used in operating system activities like file management, process creation, directory management.
3. The structure of the operating system has evolved with time. Most common ones includes monolithic, layered, microkernel etc

2.6 LIST OF REFERENCE

Modern Operating system, Fourth edition, Andrew S. Tanenbaum, Herbert Bos

2.7 BIBLIOGRAPHY

Operating System concepts, Eighth edition, Silberschatz, Galvin Gagne

2.8 UNIT END EXERCISE

1. Explain the micro kernel approach of Operating System design
2. Explain client-server model
3. List various Operating Systems. Explain any two.
4. With suitable diagram explain the structure of disk drive.
5. What do you mean by system call? Write system calls for directory management.
6. List and explain any five system calls used in file management

*****

3

PROCESSES AND THREADS

Unit Structure

3.0 OBJECTIVES

- The objective of the chapter is as follow.
- To understand process and thread and its importance in operating system.
- To understand various concepts related to process like scheduling, termination, creation.
- To understand interprocess communication in process.

3.1 INTRODUCTION

- The most important concept of any operating system is process which is an abstraction of a running program.
- They support the ability to perform concurrent operation even with single processorModern computing exists only because of process.
- Operating system can make the computer more productive by switching the CPU between processes

3.2PROCESS

- Definition : Process is a program in execution
- A running process are organized into sequential processes. Every process needs CPU for completing its execution. CPU switches back and forth between these running processes
- In any multiprogramming system, the CPU switches from process to process quickly, running each for tens or hundreds of milliseconds
- A process is an activity of some kind. It has a program, input, output, and a state.

o A single processor may be shared among several processes, with some scheduling algorithm being accustomed to determine when to stop work on one process and service a different one. In contrast, a program is something that may be stored on disk, not doing anythingProcessmemory is divided into four sections:

- The text section comprises the compiled program code, read in from non-volatile storage when the program is launched.
- The data section stores global and static variables, allocated and initialized prior to executing main.
- The heap is used for dynamic memory allocation, and is managed via calls to new, delete, malloc, free, etc.
- The stack is used for local variables

3.2.1 Process Creation:

Four principle events cause processes to be created:

1. System initialization:

- When an operating system is booted, numerous processes are created.
- Some of these are foreground processes: processes that interact with (human) users and perform work for them.
- Others run in the background also called as daemons and not associated with particular users, but instead have some specific function.

2. Execution of a process-creation system call by a running process.

- A running process will issue system calls to create one or more new processes to help it do its job

3. A user request to create a new process:

- A new process is created by having an existing process execute a process creation system call
- In UNIX, system call to create a new process: fork()
- In Windows, CreateProcess(), with 10 parameters handles both process creation and loading the correct program into the new process.

4. Initiation of a batch job:

- Users can submit batch jobs to the system.
- When the operating system creates a new process and runs the next job from the input queue in it

3.2.2 Process Termination:

Process can be terminated by a call to kill in UNIX or TerminateProcess in windows Process will be terminated due to following reason Normal exit:

- Most processes terminates when they have completed their work and executes a system call to exit
- This call is exit() in UNIX and ExitProcess in windows

Error exit:

- The third type of error occurs due to program bug like executing an illegal instruction, referencing
- 3on-existent memory or dividing by zero.

Fatal exit:

- A termination of a process occurs when it discovers a fatal error.
- For example, if a user types the command

- cc xyz.c
- to compile the program xyz.c and if no such file exists, the compiler simply announces this fact and exits.

Killed by another process:
A process executes a system call to kill some other process.
In UNIX this call is called as kill. The corresponding Win32 function is TerminateProcess.

3.2.3 Process States:

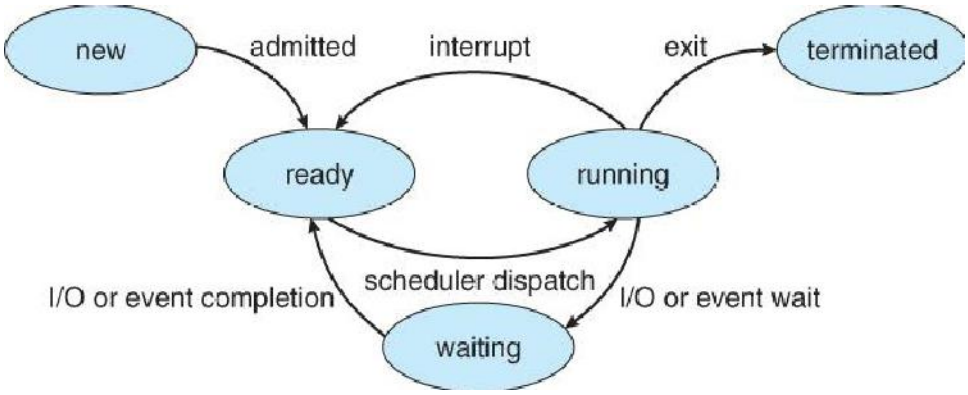


Figure 3.1

Reference: "Operating System Concepts" by Abraham Silberschatz, Greg Gagne, and Peter Baer Galvin

Process model makes it easier to understand what is going on inside the system. Some of the processes run programs that carry out commands typed in by a user other processes are part of the system processes.

When a disk interrupt occurs, the system makes a decision to stop running the current process and run the disk process, which was blocked waiting for that interrupt.

Any process in the system is present in any one of the given states

New – The process is in the stage of being created.

Ready – The process has all the resources available that it needs to run, but the CPU is not currently working on this process's instructions.

Running – The CPU is working on this process's instructions**.**

Waiting – The process cannot run at the moment, because it is waiting for some resource to become available or for some event to occur. For example the process may be waiting for keyboard input, disk access request, inter-process messages, a timer to go off, or a child process to finish.

Terminated – The process has completed.

3.3 THREAD

3.3.1 Thread Usage:

- A thread is a basic unit of CPU utilization, consisting of a program counter, a stack, and a set of registers.
- A process have a single thread of control – There is one program counter, and one sequence of instructions that can be carried out at any given time
- Decomposing an application into multiple sequential threads that run in quasi-parallel, the programming model becomes simpler
- Thread has an ability to share an address space and all of its data among themselves. This ability is essential for certain applications.
- Threads are lighter weight than processes, they are faster to create and destroy than processes.

3.3.2 Classical Thread Model:

- A process contains a number of resources such as address space, open files, accounting information, etc.
- In addition to these resources, a process has a thread of control, e.g., program counter, register contents, stack.
- The idea of threads is to permit multiple threads of control to execute within one process.
- This is often called multithreading and threads are also known as lightweight processes. Since threads in the same process share state and stack so switching between them is much less expensive than switching between separate processes.
- Individual threads within the same process are not completely independent but are cooperating and all are from the same process.
- The shared resources makes it easier between threads to use each other resources.
- A new thread in the same process is created by a library routine like thread_create; similarly thread_exit terminatea a thread.

3.3.3 Implementing thread in User Space:

- The entire thread package is kept in the user space and kernel has no knowledge about it.
- Kernel manages ordinary and single threaded processes
- The threads run on top of a run-time system.
- Run time system is a collection of procedures that manage threads.
- e.g. pthread create, pthread exit, pthread join, and pthread yield,
- Each process needs to have its own private thread table to keep track of the threads in that process.

- The thread table keeps a track of each thread's properties
- Thread tables are managed by runtime system

Advantages:

- Can be implemented on the OS that do not support thread and thread are implemented by library
- Requires no modification in the operating system.
- It gives better performance as there is no context switching involved from kernel.
- Each process is allowed to have its own customized scheduling algorithm.

Disadvantages

- Implementing blocking system calls would cause all thread to stop.
- If a thread starts running, no other thread would be able to run unless the thread voluntarily leaves the CPU.

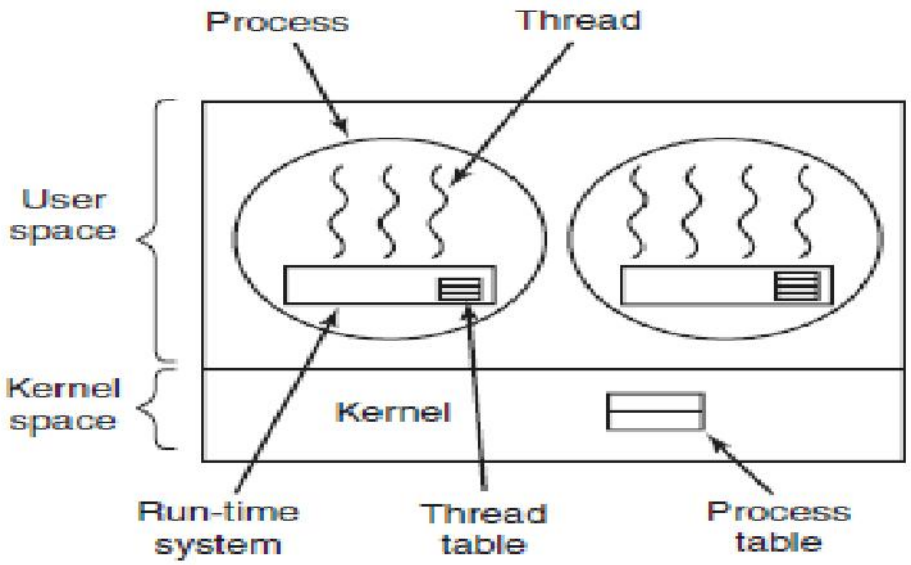


Figure 3.2

3.3.4 Implementing thread in User Kernel:

- Kernel manages the thread by keeping a track of all threads by maintaining a thread table in the system.
- When a thread wants to create a new thread or destroy an existing thread, it makes a kernel call, which then does the creation or destruction by updating the kernel thread table.
- The kernel's thread table holds each thread's registers, state, and other information and also maintains the traditional process table to keep track of processes

Advantages:

- Thread-create and friends are now systems and hence much slower.

- A thread that blocks causes no particular problem. The kernel can run another thread from this process or can run another process.
- Similarly a page fault in one thread does not automatically block the other threads in the process.

Disadvantages:

- Relatively greater cost of creating and destroying threads in the kernel
- When a signal comes in then which thread should handle it is a problem

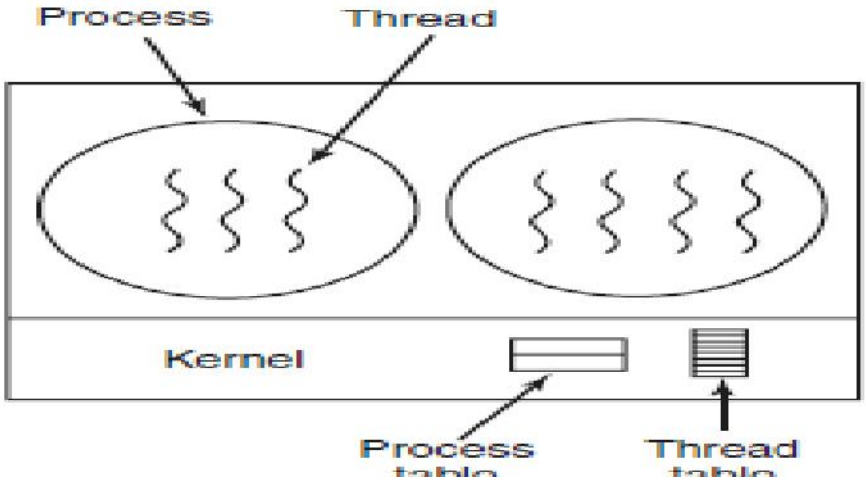


Figure 3.4

3.3.5 Hybrid implementation:

- Hybrid implementation combines the advantages of userlevel threads with kernel-level threads. One way is use kernel-level threads and then multiplex user-level threads onto some or all of them.
- This model provides maximum flexibility
- The kernel is aware of only the kernel-level threads and schedules those.
- These user-level threads are created, destroyed, and scheduled like the user-level threads in a process that runs on an operating system without multithreading capability

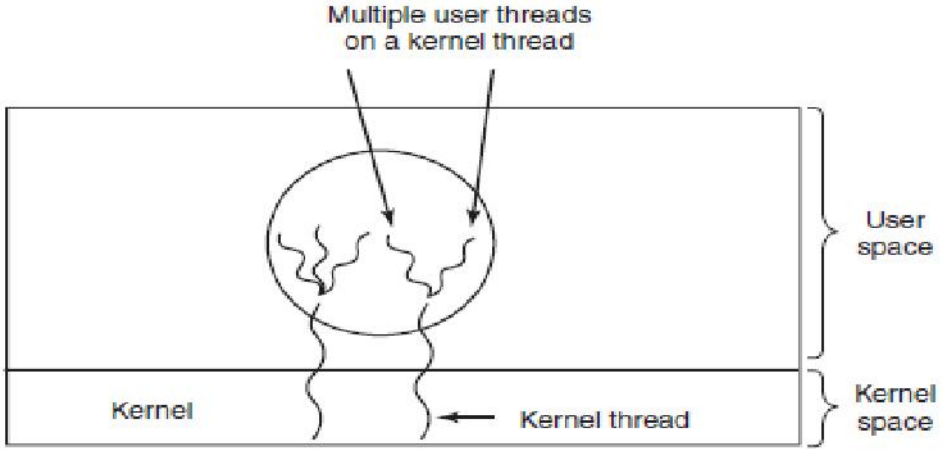


Figure 3.4

3.4 INTERPROCESS COMMUNICATION

- It is a mechanism that allows the exchange of data between processes
- Enables resource and data sharing between the processes without interference.
- To provide information about process status to other processes.
- Three problems which are faced,
- How one process can pass information to another?
- The second has to do with making sure two or more processes do not get in each other's way.
- The third concerns proper sequencing when dependencies are present.

3.4.1 Race Condition:

1. In operating system processes that are working together may share some common storage that each one can read and write.
2. The shared storage may be in main memory.
3. Several processes access and manipulates shared data simultaneously.
4. Final value of shared data depends upon which process finishes last. Fig. shows

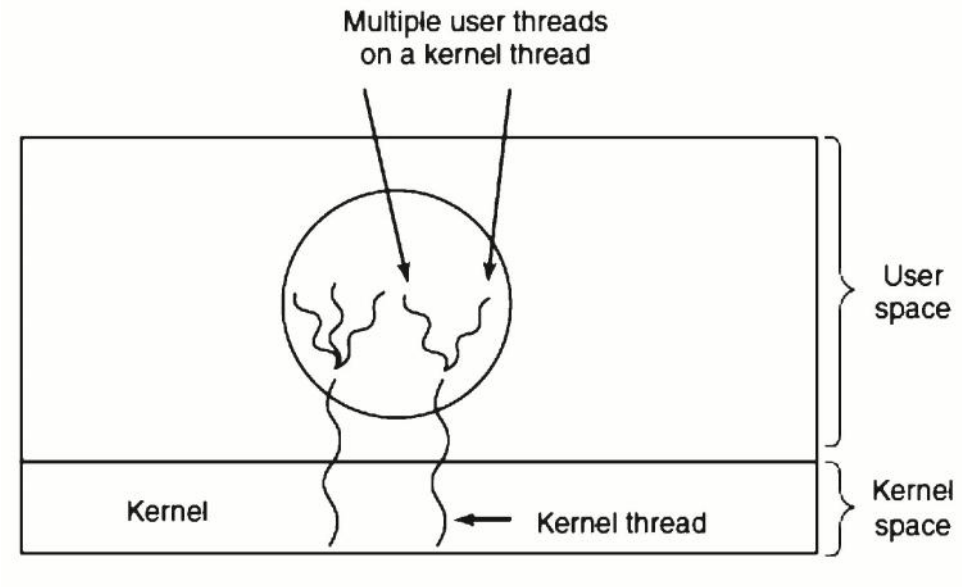


Figure 3.5

1. In the above example, file name is entered in the special spooler directory for printing.
2. he printer daemon prints the files and then removes their names from the directory.
3. Imagine that our spooler directory has a very large number of slots, numbered 0, 1, 2, …, each one capable of holding a file name.

4. two shared variables,
5. in – variable pointing to next free slot
6. out – variable pointing to next file to be printed.
7. Process A reads in and stores the value, 7, next free slot. Just then a clock interrupt occurs and the CPU decides that process switches to process B.
8. Process B also reads in and gets a 7 in local variable next free slot.
9. Process B now continues to run. It stores the name of its file in slot 7 and Then it goes off and does other things.
10. Eventually, process A runs again, starting from the place it left off.
11. It looks at next free slot, finds a 7 there, and writes its file name in slot 7, erasing the name that process B just put there.

3.4.2 Critical Region:

Definition: Part of the program where the shared memory is accessed is called the critical region or critical section

Race condition can be avoided by ensuring that no two processes are ever in their critical regions at the same time.

Following four conditions needed to have a good solution:

1. No two processes may be simultaneously inside their critical regions.
2. No assumptions may be made about speeds or the number of CPUs.
3. No process running outside its critical region may block any process.
4. No process should have to wait forever to enter its critical region.

3.4.3 Mutual Exclusion and busy waiting:

No other process will enter its critical region, when one process is in its critical region; Following are the ways for achieving mutual exclusion

3.4.3.1 Disabling interrupts:

- Each process disables all interrupts just after entering in its critical section and re-enable all interrupts just before leaving critical section.
- With interrupts turned off the CPU could not be switched to other process. Hence, no other process will enter its critical section and mutual exclusion will be achieved.
- But disabling interrupts is sometimes a useful technique within the kernel of an operating system, but it is not appropriate as a general mutual exclusion mechanism for user's process. The reason is that it is unwise to give user process the power to turn off interrupts.

3.4.3.2 Lock Variables:

A single, shared, lock variable is considered initially at 0. When a process wants to enter in its critical section, it first test the lock. If lock is 0, the process first sets it to 1 and then enters the critical section. If the lock is already 1, the process just waits until lock variable becomes 0. Thus, a 0 means that no process in its critical section, and 1 means to wait since some process is in its critical section.

But the technique has a drawback as explained Suppose that one process reads the lock and sees that it is 0. Before it can set the lock to 1, another process is scheduled, runs, and sets the lock to 1.

When the first process runs again, it will also set the lock to 1, and two processes will be in their critical regions at the same time.

3.4.2.3 Strict Alternation:

The integer variable 'turn' keeps track of whose turn is to enter the critical section. Initially, process A inspect turn, finds it to be 0, and enters in its critical section.

Process B also finds it to be 0 and sits in a loop continually testing 'turn' to see when it becomes 1. Continuously testing a variable waiting for some value to appear is called the Busy-Waiting.
Busy waiting wastes CPU time and should be avoided

3.4.4 Sleep and Wake:

sleep: A system call that causes the caller to block or remain suspended until another process wakes it up.

Wakeup: The process to be awakened is passed as a parameter to the wakeup system call.

3.4.4.1 Producer Consumer Problem (Bounded Buffer):

- The producer-consumer problem also known as bounded buffer problem which assumes that there is a fixed sized buffer available.
- To suspend the producers when the buffer is full, to suspend the consumers when the buffer is empty, and to make sure that only one process at a time manipulates a buffer so there are no race conditions.
- Two processes share a common, fixed-size (bounded) buffer. The producer puts information into the buffer and the consumer takes information out.
- Problem arises if the following scenario comes across:
- The producer wants to put a new data in the buffer, but buffer is already full.

Solution: Producer goes to sleep and to be awakened when the consumer has removed data. The consumer wants to remove data from the buffer but buffer is already empty.

Solution: Consumer goes to sleep until the producer puts some data in buffer and wakes consumer up.

Conclusion:

This approaches also leads to same race conditions we have seen in earlier approaches. Race condition can occur due to the fact that access to 'count' is unconstrained. The essence of the problem is that a wakeup call, sent to a process that is not sleeping, is lost.

3.4.5 Semaphore:

E. W. Dijkstra (1965) suggest semaphore, an integer variable to count the number of wakeups saved for future use. A semaphore could have the value 0, indicating that no wakeups were saved, or some positive value if one or more wakeups were pending.

Two operations are performed on semaphores called as down(sleep) and up(wakeup). Processes to synchronize their activities.

These operations are also known as: wait() denoted by P and signal() is denoted by V. wait(S)

```
{
while (S <= 0)
;
S—;
}

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

3.4.6 Mutex:

- Mutex is a simplified version of the semaphore used for managing mutual exclusion of shared resources.
- They are easy and efficient to implement and useful in thread packages that are implemented entirely in user space.
- A mutex is a shared variable that can be in one of two states: unlocked orLocked.
- The semaphore is initialized to the number of resources available.
- Each process that wishes to use a resource performs a wait() operation on the semaphore. When a process releases a resource, it performs a signal() operation.

- When the count for the semaphore goes to 0, all resources are being used.
- Any processes that wish to use a resource will block until the count becomes greater than 0.

3.5 SCHEDULING

- The part of the operating system that makes the choice is called the scheduler, and the algorithm it uses is called the scheduling algorithm
- Processes are of two types Compute bound or Input output bound Compute-bound processes have long CPU bursts and infrequent I/O waits I/O-bound processes have short CPU bursts and frequent I/O waits.
- The length of the CPU burst is an important factor
- It takes the same time to issue the hardware request to read a disk block no matter how much or how little time it takes to process the data after they arrive. Scheduling is of two types preemptive and non preemptive
- Scheduling algorithm are classified as Batch, Interactive and Real Time
- CPU scheduling takes place when one of the following condition is true Switching of process from the running state to waiting state
- Switching of process from the running state to ready state Switching of process from the waiting state to ready state When a process terminates
- scheduling under conditions 1 and 4 is called as non-preemptive scheduling. scheduling under conditions 2 and 3 is preemptive scheduling

3.5.1 First Come First Serve(Fcfs):

- It is a non preemtive algorithm where the ready queue is based on FIFO procedure.
- Processes are assigned to the CPU in the order they requested it.
- The strength of FCFS algorithm is that it is easy to understand and equally easy to program.
- It has a major disadvantage of high amount of waiting time
- It also suffers from convoy effect where in many small process have to wait for a longer process to release CPU.

Process	Burst Time	Arrival	Start	Wait	Finish	TA
1	24	0	0	0	24	24
2	3	0	24	24	27	27
3	3	0	27	27	30	30

Gantt chart:

average waiting time: (0+24+27)/3 = 17
average turnaround time: (24+27+30) = 27

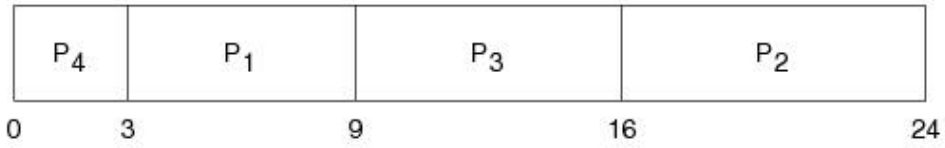
3.5.2 Shortest Job First(SJF):

- Each process is associated the length of its next CPU burst.
- According to the algorithm the scheduler selects the process with the shortest time SJF is of two types
- **non-preemptive:** A process once scheduled will continue running until the end of its CPU burst time preemptive also known as shortest remaining time next(SRTN): A process preempt if a new process arrives with a CPU burst of less length than the remaining time of the currently executing process. SJF is an optimal algorithm which gives minimum average waiting time for any set of processes but suffers from the drawback of assuming the run times are known in advance.

SJF (Non Preemptive)

Process	Burst Time	Arrival	Start	Wait	Finish	TA
1	6	0	3	3	9	9
2	8	0	16	16	24	24
3	7	0	9	9	16	16
4	3	0	0	0	3	3

Gantt chart:



average waiting time: (3+16+9+0)/4 = 7
average turnaround time: (9+24+16+3)/4 = 13

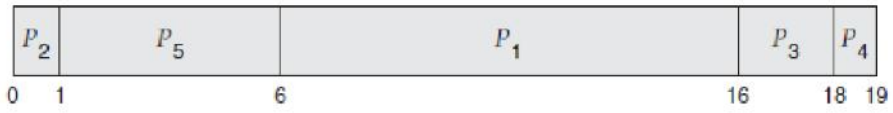
Process	Burst Time	Arrival	Start	Wait	Finish	TA
1	8	0	0	9	17	17
2	4	1	1	0	5	4
3	9	2	17	15	26	24
4	5	3	5	2	10	7

average waiting time: (9+0+15+2)/4 = 6.5
average turnaround time: (17+4+24+7)/4 = 13

3.5.3 Priority Scheduling:

- Priority scheduling associates a priority number with each process in its PCB block
- The runnable process with the highest priority is assigned to the CPU
- A clash of 2 processes with the same priority is handled using FCFS
- The need to take external factors into account leads to priority scheduling.
- To prevent high-priority processes from running indefinitely, the scheduler may decrease the priority of the currently running process at each clock tick
- Priorities can be assigned to processes statically or dynamically
- The algorithm faces with starvation low priority processes may never execute, they may have to wait indefinitely for the CPU therefore as a solution ageing is attached with each

Process	Burst Time	Priority Number	Arrival	Start	Wait	Finish	TA
1	10	3	0	6	6	16	16
2	1	1	0	0	0	1	1
3	2	4	0	16	16	18	18
4	1	5	0	18	18	19	19
5	5	2	0	1	1	6	6



average waiting time: (6+0+16+18+1)/5 = 8.2
average turnaround time: (16+1+18+19+6)/4 = 12

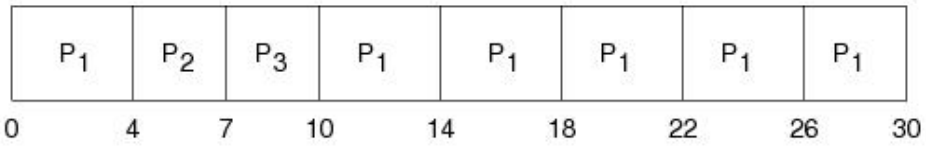
3.5.4 Round Robin Scheduling(RR):

- Round Robin scheduling algorithm has been designed specifically for time sharing system
- Time quantum or time slice is a small unit of time defined after which pre-emption of process would take place.
- The ready queue is based on FIFO order with each process getting access in circular manner
- The RR scheduling algorithm is thus preemptive. If there are n processes in the ready queue and the time quantum is q, then each process gets 1/n of the CPU time in chunks of at most q time units.
- Each process must wait no longer than (n 1) × q time units until its next time quantum.
- The selection of time slice (q) plays an important role.

if q is very large, Round Robin behaves like FCFS
if q is very small, it will result into too many context switch leading to the overhead time

Process	Burst Time	Arrival	Start	Wait	Finish	TA
1	24	0	0	6	30	30
2	3	0	4	4	7	7
3	3	0	7	7	10	10

Time quantum = 4
Gantt chart:



Average Waiting Time: (6+4+7)/3= 5.67

Average Turn around Time: (30+7+10) = 15.67

3.5.5 Multiple Queues:

- Division is made between foreground or interactive and background or batch processes and batch processes
- These two types of processes have different response-time requirements and so may have different scheduling needs.
- foreground processes may have priority over background processes.
- A multilevel queue scheduling algorithm partitions the ready queue into several separate queues
- The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type.
- Each queue has its own scheduling algorithm.
- The foreground queue might be scheduled by an RR algorithm, while the background queue is scheduled by an FCFS algorithm
- There must be scheduling among the queues, which is commonly implemented as fixed- priority preemptive scheduling

3.6 CLASSICAL IPC PROBLEM

3.6.1 Dinning Philosophers problem:

- Five silent philosophers sit at a round table with bowls of spaghetti. Forks are placed between each pair of adjacent philosophers.
- Each philosopher must alternately think and eat.

- However, a philosopher can only eat spaghetti when he has both left and right forks. Each fork can be held by only one philosopher and so a philosopher can use the fork only if it is not being used by another philosopher.
- After he finishes eating, he needs to put down both forks so they become available to others.
- A philosopher can take the fork on his right or the one on his left as they become available, but cannot start eating before getting both of them.
- The problem is how to design a discipline of behaviour (a concurrent algorithm) such that no philosopher will starve.
- Mutual exclusion is the basic idea of the problem; the dining philosophers create a generic and abstract scenario useful for explaining issues of this type.
- The failures these philosophers may experience are analogous to the difficulties that arise in real computer programming when multiple programs need exclusive access to shared resources

Problem:

Dinning philosopher suffers from the problem of deadlock when everyone want to eat simultaneously. If all five philosophers take their left forks simultaneously. None will be able to take their right forks, and there will be a deadlock.

The second problem of starvation arises when the philosophers could start the algorithm simultaneously, picking up their left forks, seeing that their right forks were not available, putting down their left forks, waiting, picking up their left forks again simultaneously, and so on, forever.

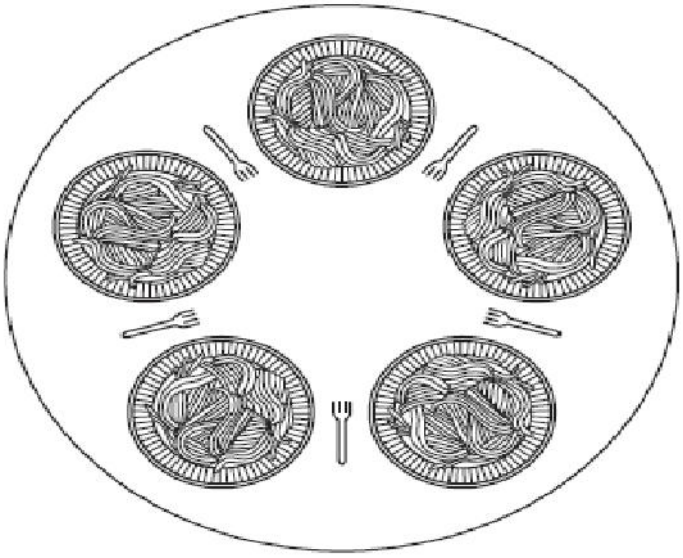


Figure 3.6 Dinning Philosopher Problem

3.6.2 Readers and Writers problem:

The dining philosophers problem is useful for modelling processes that are competing for exclusive access to a limited number of resources, such as I/O devices

- There is a data area that is shared among a number of processes.
- Any number of readers may simultaneously write to the data area. Only one writer at a time may write to the data area.
- If a writer is writing to the data area, no reader may read it.
- If there is at least one reader reading the data area, no writer may write to it.
- Readers only read and writers only write.
- The Reader and Writer problem, which models access to a database.
- For example, an airline reservation system, with many competing processes wishing to read and write it.
- It is acceptable to have multiple processes reading the database at the same time, but if one process is updating the database, no other process should access the database, not even readers.
- To avoid this situation, the program could be written slightly differently: when areader arrives and a writer is waiting, the reader is suspended behind the writer instead of being admitted immediately.

3.7 LET US SUM UP

- Processes can communicate with one another using interprocess communication
- Primitives
- A process can be running, runnable, or blocked and can change state when it or another process executes one of the interprocess communication primitives.
- Interprocess communication primitives can be used to solve such problems asthe producer-consumer, dining philosophers, and reader-writer

3.8 LIST OF REFERENCE

- Modern Operating system, Fourth edition, Andrew S. Tanenbaum, Herbert Bos Operating System concepts, Eighth edition, Silberschatz, Galvin Gagne
- http://academic.udayton.edu/SaverioPerugini/courses/cps346/lecture_notes/scheduling.ht ml

3.9 BIBLIOGRAPHY

Operating Systems – Internal Design and Principles , William Stallings

3.10 UNIT END QUSTIONSS

1. Write a short note on process termination
2. Write a short note on the process model.
3. What is race condition? How mutual exclusion handles race condition
4. With suitable example explain the shortest job first scheduling algorithm.
5. Explain round robin scheduling give proper example.

*****

UNIT II

4

MEMORY MANAGEMENT

Unit Structure

4.0 OBJECTIVES

- Description of various ways of organizing memory hardware.
- Techniques of allocating memory to processes.
- Paging works in contemporary computer systems.
- To describe the benefits of a virtual memory system.
- To explain the concepts of demand paging, page-replacement algorithms, and allocation of page frames.
- To discuss the principles of the working-set model.
- To examine the relationship between shared memory and memory-mapped files.
- To explore how kernel memory is managed

4.1 INTRODUCTION

We showed how the CPU can be shared by a set of processes. As a result of CPU scheduling, we can improve both the utilization of the CPU and the speed of the computer's response to its users. To realize this increase in performance, however, we must keep several processes in memory— that is, we must share memory. In this chapter, we discuss various ways to manage memory. The memory management algorithms vary from a primitive bare-machine approach to paging and segmentation strategies. Each approach has its own advantages and disadvantages.

Selection of a memory-management method for a specific system depends on many factors, especially on the hardware design of the system. As we shall see, many algorithms require hardware support, leading many

systems to have closely integrated hardware and operating- system memory management.

4.2 ADDRESS SPACE

An address space defines a range of discrete addresses, each of which may correspond to a network host, peripheral device, disk sector, a memory cell or other logical or physical entity.

For software programs to save and retrieve stored data, each unit of data must have an address where it can be individually located or else the program will be unable to find and manipulate the data. The number of address spaces available will depend on the underlying address structure and these will usually be limited by the computer architecture being used.

4.2.1 Logical Versus Physical Address Space:

An address generated by the CPU is commonly referred to as a logical address,whereas 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 a physical address.

The compile-time and load-time address-binding methods generate identical logical and physical addresses. However, the execution-time

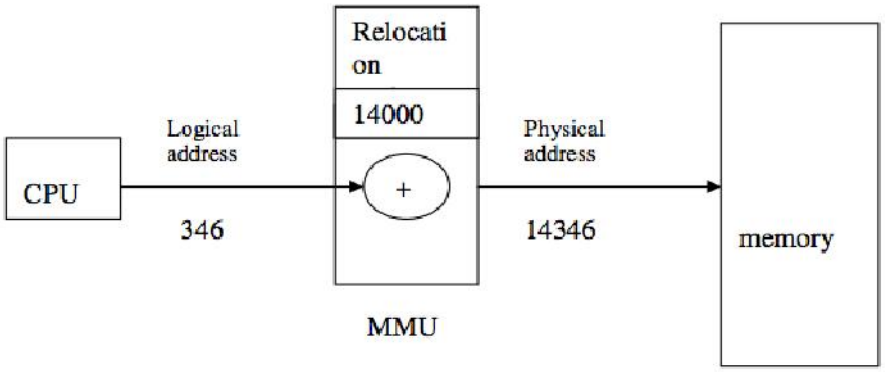


Fig: Convert logical addresses to physical addresses?

4.2.2 Address Mapping & Translation:

Another common feature of address spaces are mappings and translations, often forming numerous layers. This usually means that some higher-level address must be translated to lower- level ones in some way. For example, a file system on a logical disk operates linear sector numbers, which have to be translated to *absolute* LBA sector addresses, in simple cases, via addition of the partition's first sector address. Then, for a disk drive connected via Parallel ATA, each of them must be converted to *logical* cylinder-head-sector address due to the interface's historical shortcomings. It is converted back to LBA by the disk controller then, finally, to *physical* cylinder, head and sector numbers.

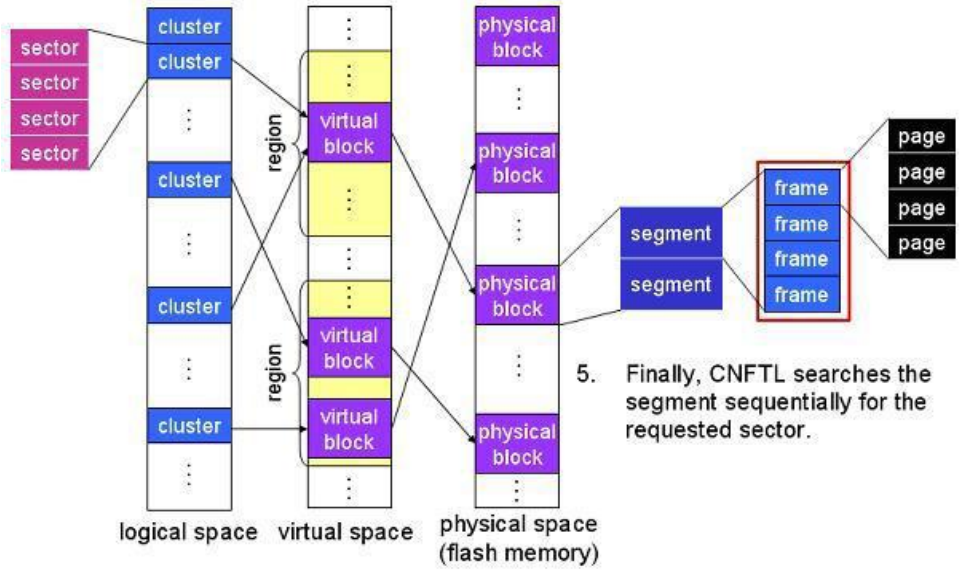


Fig: Illustration of translation from logical block addressing to physical geometry

4.2.3 virtual address space to physical address space:

The Domain Name System maps its names to (and from) network-specific addresses (usually IP addresses), which in turn may be mapped to link layer network addresses via Address Resolution Protocol. Also, network address translation may occur on the edge of different IP spaces, such as a local area network and the Internet.

An iconic example of virtual-to-physical address translation is virtual memory, where different pages of virtual address space map either to page file or to main memory physical address space. It is possible that several numerically different virtual addresses all refer to one physical address and hence to the same physical byte of RAM. It is also possible that a single virtual address maps to zero, one, or more than one physical address

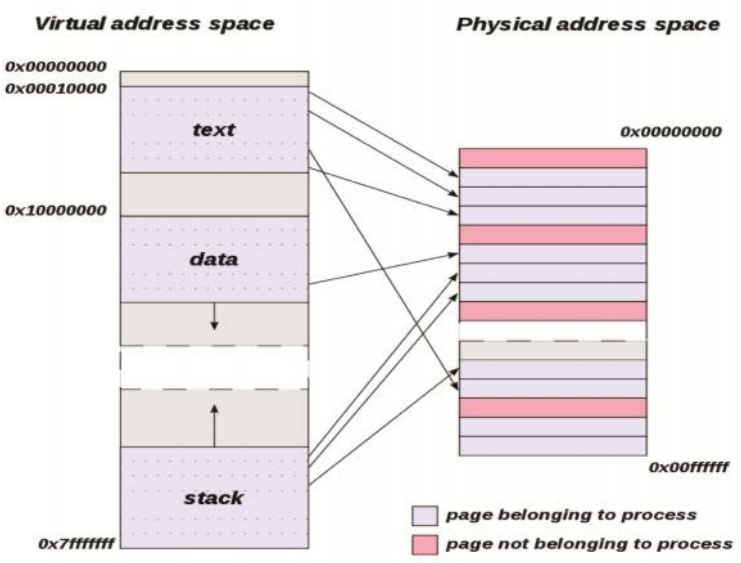


Fig: Illustration of translation from virtual address space to physical address space.

4.2.4 Types of Memory Address:

The operating system takes care of mapping the logical addresses to physical addresses at the time of memory allocation to the program. There are three types of addresses used in a program before and after memory is allocated −

S.N.	Memory Addresses & Description
1	**Symbolic addresses** The addresses used in a source code. The variable names, constants, and instruction labels are the basic elements of the symbolic address space.
2	**Relative addresses** At the time of compilation, a compiler converts symbolic addresses into relative addresses.
3	**Physical addresses** The loader generates these addresses at the time when a program is loaded into main memory

4.2.4.1 symbolic addressing :

An addressing scheme whereby reference to an address is made by some convenient symbol that (preferably) has some relationship to the meaning of the data expected to be located at that address. It serves as an aid to the programmer. The symbolic address is replaced by some form of computable/computed address during the operation of an assembler or compiler.

4.2.4.2 Relative addressing:

This is the technique of addressing instructions and data areas by designating their location in relation to the location counter or to some symbolic location. This type of addressing is always in bytes—never in bits, words, or instructions. Thus, the expression *+4 specifies an address that is 4 bytes greater than the current value of the location counter. In the sequence of instructions in the following example, the location of the CR machine instruction can be expressed in two ways, ALPHA+2, or BETA-4, because all the machine instructions in the example are for 2 byte instructions.

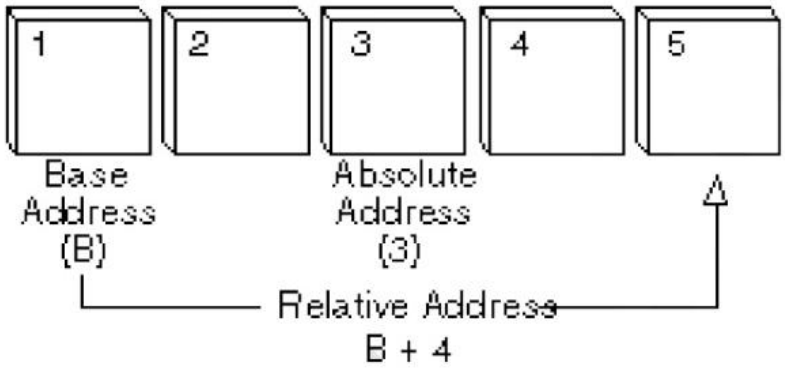
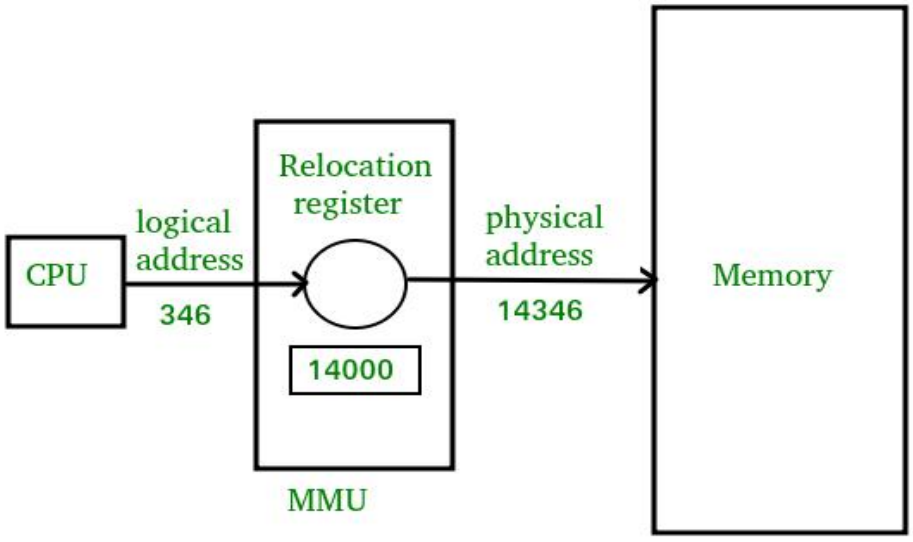


Fig: Relative Addressing

4.2.4.3 Physical Address:

This identifies a physical location of required data in a memory. The user never directly deals with the physical address but can access by its corresponding logical address. The user program generates the logical address and thinks that the program is running in this logical address but the program needs physical memory for its execution, therefore, the logical address must be mapped to the physical address by MMU before they are used. The term Physical Address Space is used for all physical addresses corresponding to the logical addresses in a Logical address space.



4.2.5 Difference between logical address and physical address:

Parameter	Logical Address	Physical Address
Basic	Generated by CPU	
Address	Logical Address Space is set of all logical addresses generated by cpu in	All physical addresses mapped to the corresponding logical addresses
Visibility	User can view the logical address of the programme	User can never view the address of the programme
Generation	Generated by CPU	Computed by MMU
Access	The user can use the logical address to access the physical address	User can indirectly access the physical address but not directly

4.3 VIRTUAL MEMORY

The memory-management algorithms outlined in Chapter 8 are necessary because of one basic requirement: The instructions being executed must be the entire logical address space in physical memory.

Dynamic loading can help to ease this restriction, but it generally requires special precautions and extra.

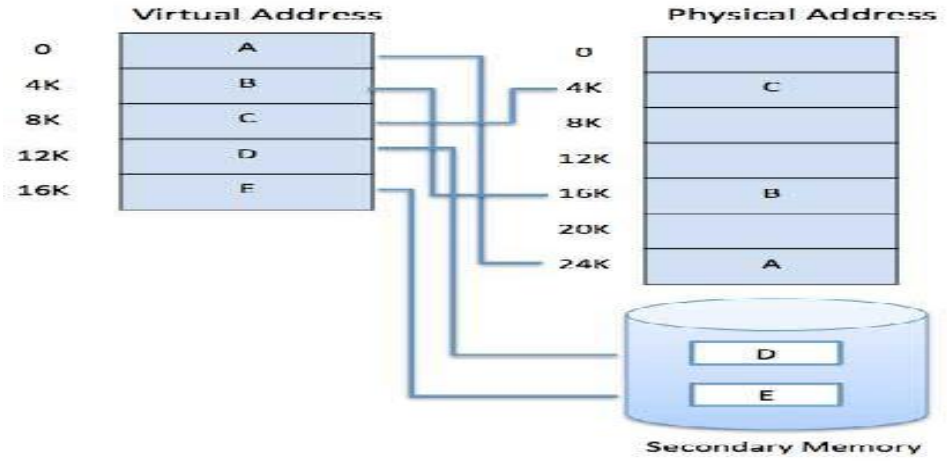
A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard disk that's set up to emulate the computer's RAM.

The main visible advantage of this scheme is that programs can be larger than physical memory. Virtual memory serves two purposes. First, it allows us to extend the use of physical memory by using a disk. Second, it allows us to have memory protection, because each virtual address is translated to a physical address.

Following are the situations, when the entire program is not required to be loaded fully in main memory.

- User written error handling routines are used only when an error occurred in the data or computation.
- Certain options and features of a program may be used rarely.
- Many tables are assigned a fixed amount of address space even though only a small amount of the table is actually used.
- The ability to execute a program that is only partially in memory would counter many benefits.
- Less number of I/O would be needed to load or swap each user program into memory. A program would no longer be constrained by the amount of physical memory that is available.
- Each user program could take less physical memory; more programs could be run at the same time, with a corresponding increase in CPU utilization and throughput.

Modern microprocessors intended for general-purpose use, a memory management unit, or MMU, is built into the hardware. The MMU's job is to translate virtual addresses into physical addresses. A basic example is given below

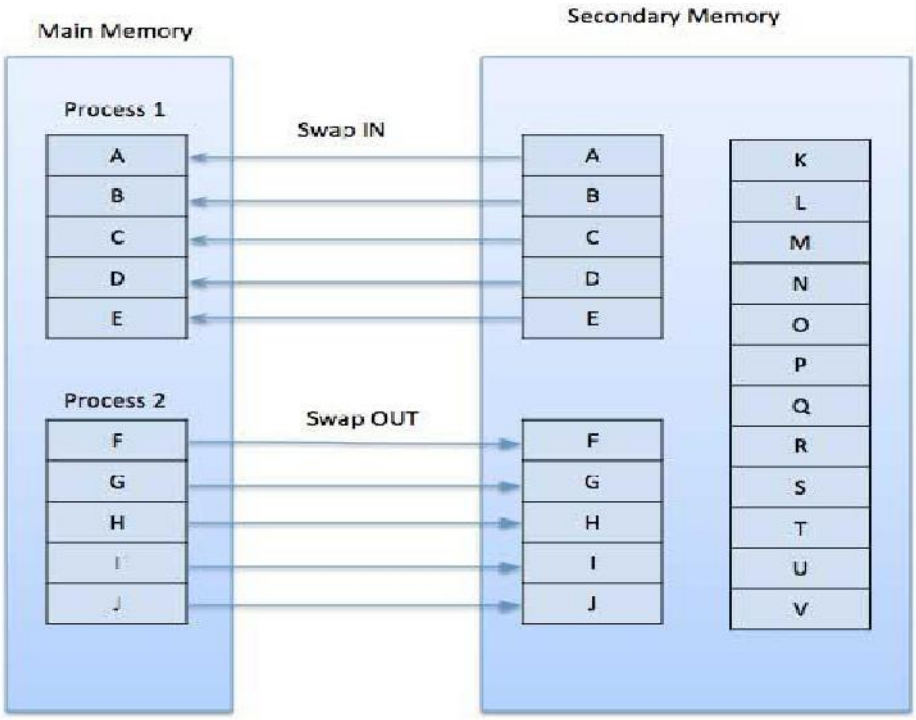


Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

Virtual memory involves the separation of logical memory as perceived by users from physical memory. This separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available.Virtual memory makes the task of programming much easier, because the programmer no longer needs to worry about the amount of physical memory available; she can concentrate instead on the problem to be programmed.

4.3.1 Demand Paging:

A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance. When a context switch occurs, the operating system does not copy any of the old program's pages out to the disk or any of the new program's pages into the main memory. Instead, it just begins executing the new program after loading the first page and fetches that program's pages as they are referenced.



While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a page fault and transfers control from the program to the operating system to demand the page back into the memory.

Advantages:

Following are the advantages of Demand Paging

- Large virtual memory.
- More efficient use of memory.
- There is no limit on the degree of multiprogramming.

Disadvantages:

Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.

4.3.4 Page Replacement Algorithm:

Page replacement algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated. Paging happens whenever a page fault occurs and a free page cannot be used for allocation purpose accounting to reason that pages are not available or the number of free pages is lower than required pages.

When the page that was selected for replacement and was paged out, is referenced again, it has to read in from disk, and this requires for I/O completion. This process determines the quality of the page replacement algorithm: the lesser the time waiting for page-ins, the better is the algorithm.

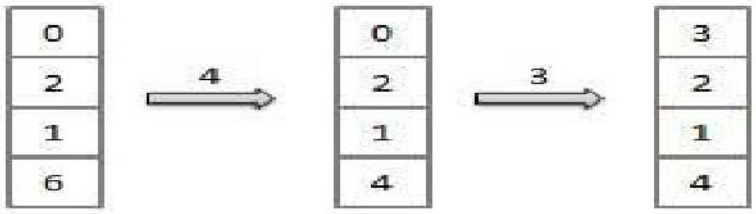
A page replacement algorithm looks at the limited information about accessing the pages provided by hardware, and tries to select which pages should be replaced to minimize the total number of page misses, while balancing it with the costs of primary storage and processor timeof the algorithm itself. There are many different page replacement algorithms. We evaluate an algorithm by running it on a particular string of memory reference and computing the number of page faults,

4.3.5 Optimal Page algorithm:

- An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.
- Replace the page that will not be used for the longest period of time. Use the time when a page is to be used

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x



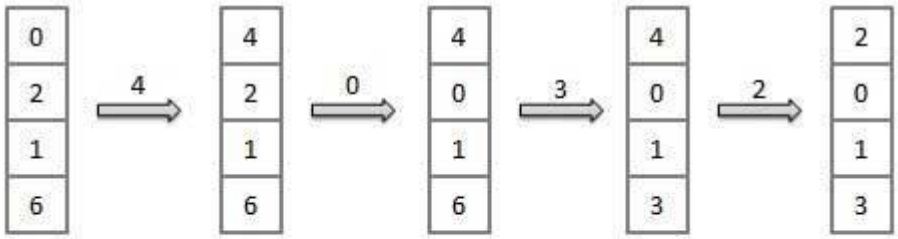
Fault Rate = 6 / 12 = 0.50

4.3.6 Least Recently Used (LRU) algorithm:

- Page which has not been used for the longest time in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages by looking back into time.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x



Fault Rate = 8 / 12 = 0.67

4.3.7 Page Buffering algorithm:

- To get a process start quickly, keep a pool of free frames.
- On page fault, select a page to be replaced.
- Write the new page in the frame of the free pool, mark the page table and restart the process.
- Now write the dirty page out of disk and place the frame holding replaced page in the free pool.

4.3.8 Least frequently Used (LFU) algorithm:

- The page with the smallest count is the one which will be selected for replacement.
- This algorithm suffers from the situation in which a page is used heavily during the initial phase of a process, but then is never used again.

4.3.9 Most frequently Used (MFU) algorithm:

- This algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used

4.7 SUMMARY

- Memory management is the process of controlling and coordinating computer memory, allocating portions called blocks to various running programs to optimize the complete performance of the system.
- It allows you to check how ample memory needs to be allocated to processes that decide which processor should get memory at what time.
- In Single Contiguous Allocation, all types of computer's memory excluding a small portion which is reserved for the OS is available for one application
- Partitioned Allocation method splits primary memory into various memory partitions, which is mostly contiguous areas of memory.

4.8 UNIT END QUESTIONS

1. What is Memory Management?
2. Why Use Memory Management?
3. Memory Management Techniques
4. What is Swapping?
5. What is Memory allocation?
6. Explain Disk replacement algorithms.
7. Disk replacement algorithms numerical

*****

5

PAGING AND SEGMENTATION

Unit Structure

5.0 OBJECTIVES OF A MEMORY MANAGEMENT (MM) SYSTEM

Relocation:

- *Relocatability* - the ability to move process around in memory without it affecting its execution
- OS manages memory, not programmer, and processes may be moved around in memory
- MM must convert program's logical addresses into physical addresses
- Process's first address is stored as virtual address 0
- *Static Relocation* - Program must be relocated before or during loading of process into memory. Programs must always be loaded into the same address space in memory, or relocator must be run again.
- *Dynamic Relocation* - Process can be freely moved around in memory. Virtual-to-physical address space mapping is done at run-time.

5.1MEMORY MANAGEMENT GOALS

Protection:

- *Write Protection* - to prevent data & instructions from being overwritten.
- *Read Protection* - To ensure privacy of data & instructions.
- OS needs to be protected from user processes, and user processes need to be protected from each other.
- Memory protection (to prevent memory overlaps) is usually supported by the hardware (limit registers), because most languages allow memory addresses to be computed at run-time.

Sharing:

- Sometimes distinct processes may need to execute the same process (e.g., many users executing the same editor), or even the same data (when one process prepares data for another process).
- When different processes signal or wait the same semaphore, they need to access the same memory address.
- OS has to allow sharing, while at the same time ensure protection.

Logical Organisation of Memory:

- Uni-dimensional address space

- If memory was *segmented* then it would be possible to code programs and subroutines separately, each with its own degree of protection.
- The MM would manage inter-segment references at run-time, and could allow a segment to be accessed by many different processes.

Physical Organisation of Memory:

- PM is expensive, so tends to be limited - but the amount of PM helps to determine the *degree of multiprogramming* (the number of runnable processes that can be simultaneously maintained)
- A two-level storage scheme (one RAM, the other slower secondary disk) can be used to virtually increase the overall amount of PM.
- Processes can be kept in secondary storage and only brought into PM when needed. MM and OS have to manage the operation of moving processes between the two levels.

Paging and segmentation:

Paging and segmentation are processes by which data is stored to, then retrieved from, a computer's storage disk.

Paging is a computer memory management function that presents storage locations to the computer's CPU as additional memory, called virtual memory. Each piece of data needs a storage address.

Segmentation is a virtual process that creates variable-sized address spaces in computer storage for related data, called segments. This process speed retrieval.

Managing computer memory is a basic operating system function -- both paging and segmentation are basic functions of the OS. No system can efficiently rely on limited RAM alone. So the computer’s memory

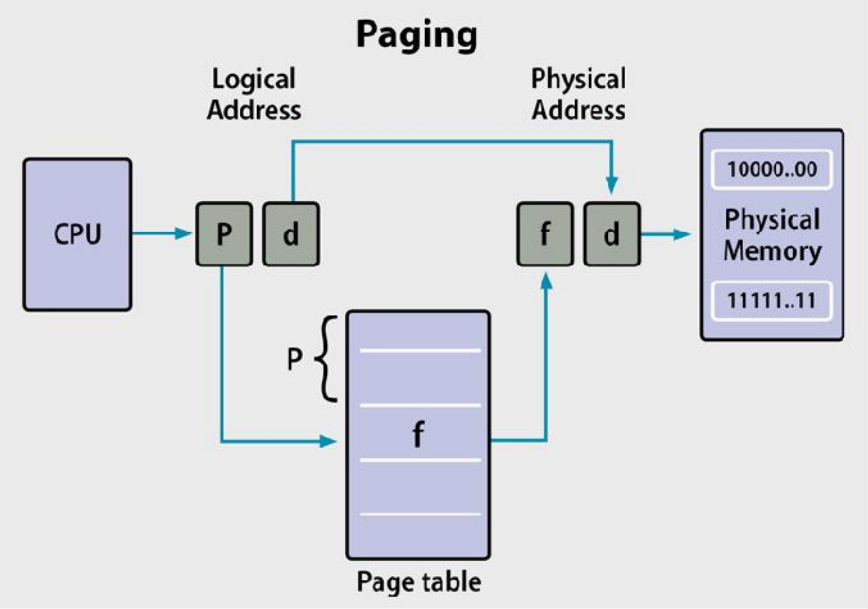
management unit (MMU) uses the storage disk, HDD or SSD, as virtual memory to supplement RAM.

What is Paging?:

As mentioned above, the memory management function called paging specifies storage locations to the CPU as additional memory, called virtual memory. The CPU cannot directly access storage disk, so the MMU emulates memory by mapping pages to frames that are in RAM.

Before we launch into a more detailed explanation of pages and frames, let's define some technical terms.

- **Page:** A fixed-length contiguous block of virtual memory residing on disk.
- **Frame:** A fixed-length contiguous block located in RAM; whose sizing is identical to pages.
- **Physical memory:** The computer's random access memory (RAM), typically contained in DIMM cards attached to the computer's motherboard.
- **Virtual memory:** Virtual memory is a portion of an HDD or SSD that is reserved to emulate RAM. The MMU serves up virtual memory from disk to the CPU to reduce the workload on physical memory.
- **Virtual address:** The CPU generates a virtual address for each active process. The MMU maps the virtual address to a physical location in RAM and passes the address to the bus. A virtual address space is the range of virtual addresses under CPU control.
- **Physical address:** The physical address is a location in RAM. The physical address space is the set of all physical addresses orresponding to the CPU's virtual addresses.A physical address space is the range of physical addresses under MMU control.



5.0 Fig: Paging concept

By assigning an address to a piece of data using a "page table" between the CPU andthe computer's physical memory, a computer's MMU enables the system to retrieve thatdata whenever needed**.**

The Paging Process:

A page table stores the definition of each page. When an active process requests data,the MMU retrieves corresponding pages intoframes located in physical memory forfaster processing. The process is called paging.

The MMU uses page tables to translate virtual addresses to physical ones. Each tableentry indicates where a page is located: in RAM or on disk as virtual memory. Tablesmay have a single or multi-level page table such as different tables for applications andsegments.

However, constant table lookups can slow down the MMU. A memory cache called theTranslation Lookaside Buffer (TLB) stores recenttranslations of virtual to physicaladdresses for rapid retrieval. Many systems have multiple TLBs, which may reside atdifferent locations, including between the CPU and RAM, or between multiple pagetable levels.

Different frame sizes are available for data sets with larger or smaller pages andmatching-sized frames. 4KB to 2MB are common sizes, and GB-sized frames areavailable in high-performance servers.

Paging with Example:

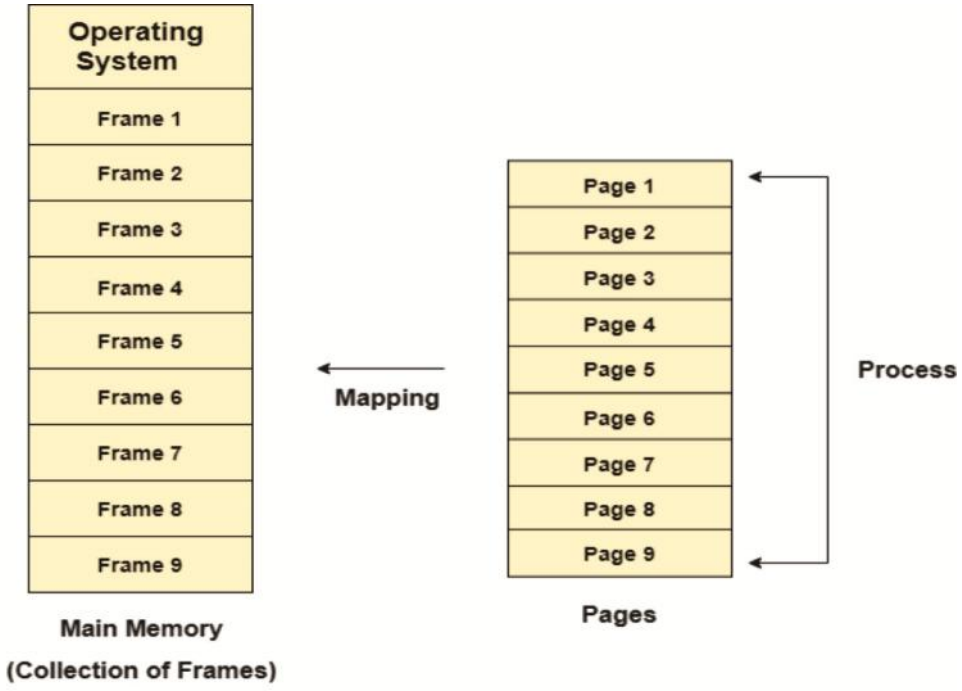
In Operating Systems, Paging is a storage mechanism used to retrieve processes from the secondary storage into the main memory in the form of pages.

The main idea behind the paging is to divide each process in the form of pages. The main memory will also be divided in the form of frames.

One page of the process is to be stored in one of the frames of the memory. The pagescan be stored at the different locations of the memory but the priority is always to find the contiguous frames or holes.

Pages of the process are brought into the main memory only when they are required otherwise they reside in the secondary storage.

Different operating system defines different frame sizes. The sizes of each frame must be equal. Considering the fact that the pages are mapped to the frames in Paging, page size needs to be as same as frame size.



5.1A fig: Paging process

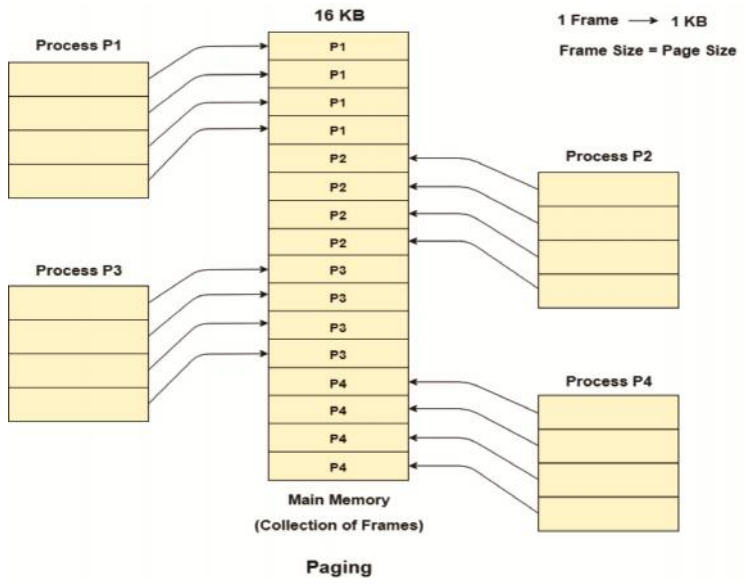
Example:

Let us consider the main memory size 16 Kb and Frame size is 1 KB therefore the main memory will be divided into the collection of 16 frames of 1 KB each.

There are 4 processes in the system that is P1, P2, P3 and P4 of 4 KB each. Each process is divided into pages of 1 KB each so that one page can be stored in one frame.

Initially, all the frames are empty therefore pages of the processes will get stored in the contiguous way.

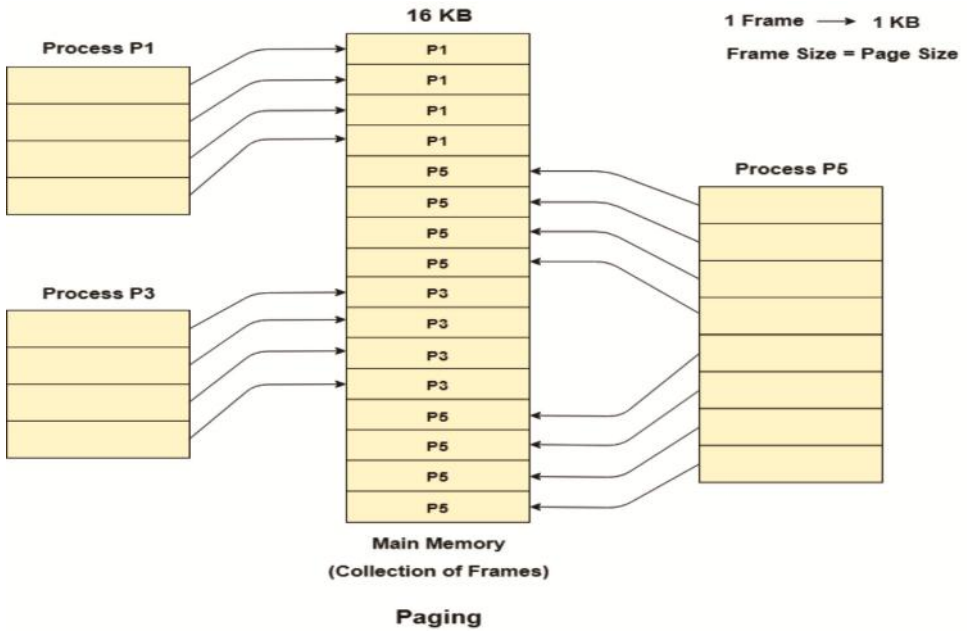
Frames, pages and the mapping between the two is shown in the image below.



5.1 B fig: Paging process

Let us consider that, P2 and P4 are moved to waiting state after some time. Now, 8 frames become empty and therefore other pages can be loaded in that empty place. The process P5 of size 8 KB (8 pages) is waiting inside the ready queue.

Given the fact that, we have 8 non contiguous frames available in the memory and paging provides the flexibility of storing the process at the different places. Therefore, we can load the pages of process P5 in the place of P2 and P4.



5.1 C fig: Paging process

Design Issues of Paging.

- The Working Set Model. In the purest form of paging, processes are started up with none of their pages in memory.
- Local versus Global Allocation Policies. In the preceding sections we have discussed several algorithms for choosing a page to replace when a fault occurs. ...
- Page Size. ...
- Virtual Memory Interface.

5.2 SEGMENTATION

In Operating Systems, Segmentation is a memory management technique in which the memory is divided into the variable size parts. Each part is known as a segment which can be allocated to a process.

What is Segmentation?

The process known as *segmentation* is a virtual process that creates address spaces of various sizes in a computer system, called segments.

Each segment is a different virtual address space that directly corresponds to process objects.

The details about each segment are stored in a table called a segment table. Segment table is stored in one (or many) of the segments. Segment table contains mainly two information about segment:

1. **Base:** It is the base address of the segment
2. **Limit:** It is the length of the segment.

5.2.1 Why Segmentation is required??

Till now, we were using Paging as our main memory management technique. Paging is more close to the Operating system rather than the User. It divides all the process into the form of pages regardless of the fact that a process can have some relative parts of functions which need to be loaded in the same page.

Operating system doesn't care about the User's view of the process. It may divide the same function into different pages and those pages may or may not be loaded at the same time into the memory. It decreases the efficiency of the system.

It is better to have segmentation which divides the process into the segments. Each segment contain same type of functions such as main function can be included in one segment and the library functions can be included in the other segment,

5.2.2 Translation of Logical address into physical address by segment table

CPU generates a logical address which contains two parts:

1. Segment Number
2. Offset

The Segment number is mapped to the segment table. The limit of the respective segment is compared with the offset. If the offset is less than the limit then the address is valid otherwise it throws an error as the address is invalid.

In the case of a valid address, the base address of the segment is added to the offset to get the physical address of the actual word in the main memory.

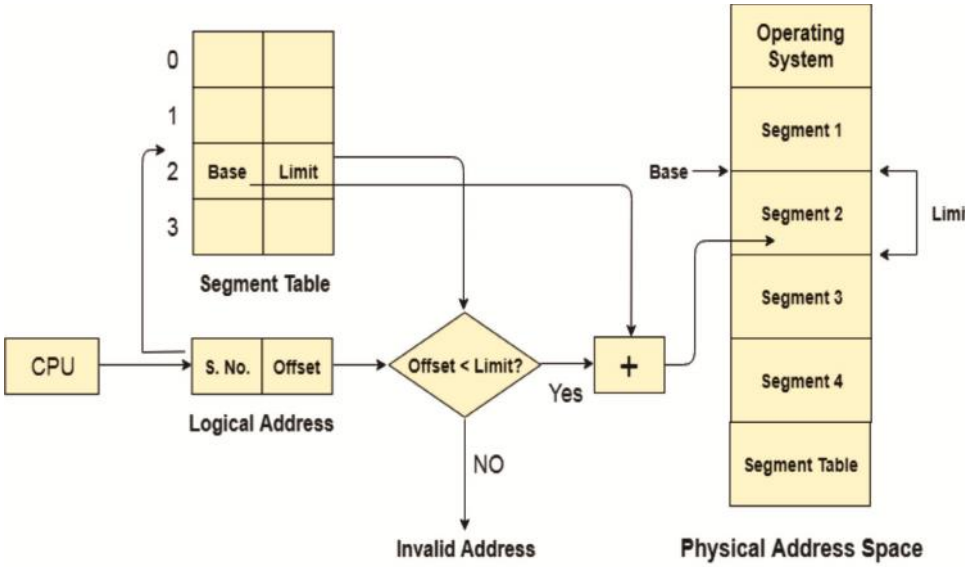


Fig : Segmentation

Advantages of Segmentation:

1. No internal fragmentation
2. Average Segment Size is larger than the actual page size.
3. Less overhead
4. It is easier to relocate segments than entire address space.
5. The segment table is of lesser size as compare to the pagetable in paging.

Disadvantages:

1. It can have external fragmentation.
2. It is difficult to allocate contiguous memory to variable sized partition.
3. Costly memory management algorithms.

Difference between Paging and Segmentation:

S.NO	PAGING	SEGMENTATION
1.	In paging, program is divided into fixed or mounted size pages.	In segmentation, program is divided into variable size sections
2.	For paging operating system is accountable	For segmentation compiler is accountable
3.	Page size is determined by hardware.	Here, the section size is given by the user
.4.	It is faster in the comparison of segmentation.	Segmentation is slow.
5.	Paging could result in internal fragmentation	Segmentation could result in external fragmentation.
6.	In paging, logical address is split into page number and page offset.	Here, logical address is split into section number and section offset

7.	Paging comprises a page table which encloses the base address of every page.	While segmentation also comprises the segment table which encloses segment number and segment offset

Page Replacement Algorithms in Operating Systems:

In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when a new page comes in.

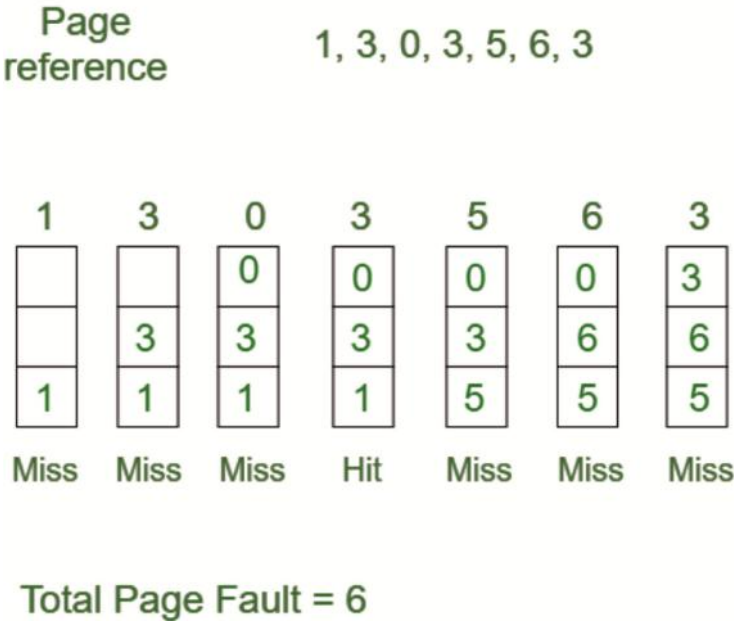
Page Fault: A page fault happens when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory. Since actual physical memory is much smaller than virtual memory, page faults happen. In case of page fault, the Operating System might have to replace one of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults.

Page Replacement Algorithms:

First In First Out (FIFO)

This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

Example-1 Consider page reference string 1, 3, 0, 3, 5, 6 with 3 page frames. Find number of page faults



Initially all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —>**3 Page Faults.** when 3 comes, it is already in memory so —>**0 Page Faults.** Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. —>**1 Page Fault.**

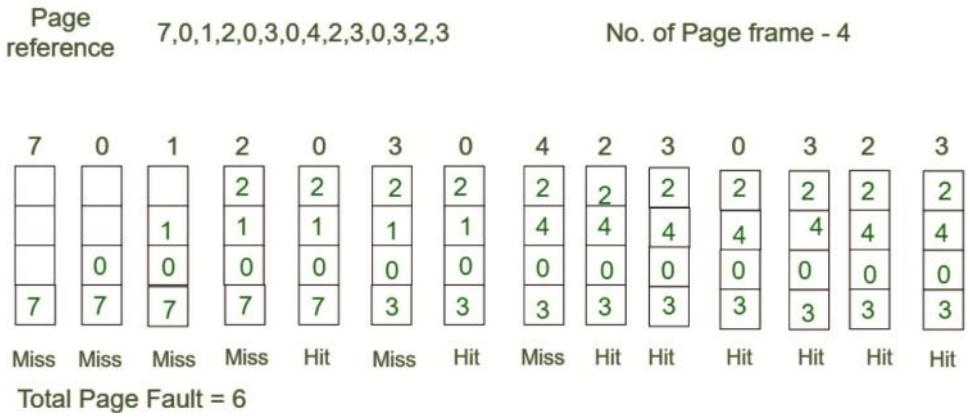
6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 >**1 Page Fault.**

Finally when 3 come it is not available so it replaces 0 **1 page fault**

Optimal Page replacement:

In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

Example-2: Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, with a 4 page frame. Find the number of page fault.



Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —>**4**

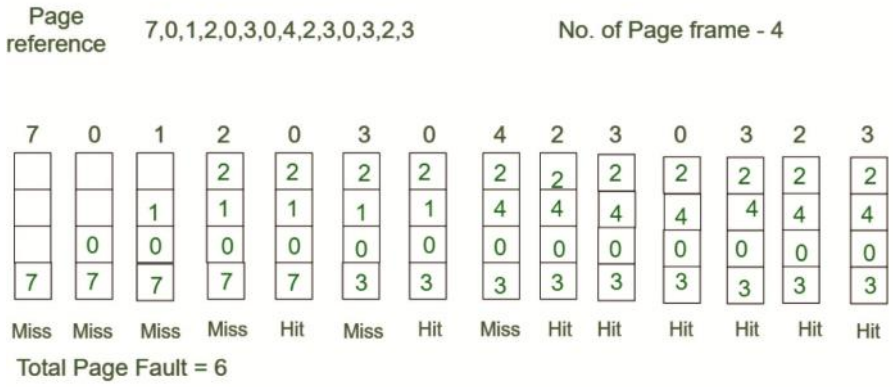
Page faults 0 is already there so —>**0 Page fault.**
when 3 came it will take the place of 7 because it is not used for the longest duration
of time in the future. —> **1 Page fault.**
0 is already there so —> **0 Page fault.**.
4 will takes place of 1 —> **1 Page Fault.**

Now for the further page reference string —>**0 Page fault** because they are already available in the memory.

Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

Least Recently Used:
In this algorithm page will be replaced which is least recently used.

Example-3 Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 with 4 page frames. Find number of page faults.



Here LRU has same number of page fault as optimal but it may differ according to question.

Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —>**4 Page faults**

0 is already their so —>**0 Page fault.**
when 3 came it will take the place of 7 because it is least recently used —>**1 Page fault**
0 is already in memory so —>**0 Page fault**.
4 will takes place of 1 —>**1 Page Fault**
Now for the further page reference string —>**0 Page fault** because they are already available in the memory.

5.5 DESIGN ISSUES OF PAGING

- The Working Set Model. In the purest form of paging, processes are started up with none of their pages in memory. .
- Local versus Global Allocation Policies. In the preceding sections we have discussed several algorithms for choosing a page to replace when a fault occurs. ...
- Page Size. ...
- Virtual Memory Interface.

5.6 SUMMARY

1. Paging is a storage mechanism that allows the OS to retrieve processes from the secondary storage into the main memory in the form of pages.
2. Fragmentation refers to the condition of a disk in which files are divided into pieces scattered around the disk.
3. Segmentation method works almost similarly to paging. The only

difference between the two is that segments are of variable-length, whereas, in the paging method, pages are always of fixed size.

4. Dynamic loading is a routine of a program which is not loaded until the program calls it.
5. Linking is a method that helps OS to collect and merge various modules of code and data into a single executable file.

5.7 UNIT END QUESTIONS

1. What is Paging?
2. What is Segmentation? and Paging vs. Segmentation
3. Advantages of Paging
4. Advantage of Segmentation and Disadvantages of Paging
5. Disadvantages of Segmentation
6. Page replacement algorithms numerical

*****

6

FILE SYSTEM

Unit Structure

6.0 OBJECTIVES

- Files
- Directories
- file system implementation
- file-system management and optimization
- MS-DOS file system
- UNIX V7 file system
- CDROM file system

6.1FILE

A file is a named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks. In general, a file is a sequence of bits, bytes, lines or records whose meaning is defined by the files creator and user.

6.2 FILE STRUCTURE

A File Structure should be according to a required format that the operating system can understand.

- A file has a certain defined structure according to its type.
- A text file is a sequence of characters organized into lines.
- A source file is a sequence of procedures and functions.

- An object file is a sequence of bytes organized into blocks that are understandable by the machine.
- When an operating system defines different file structures, it also contains the code to support these file structures. Unix, MS-DOS support a minimum number of file structures.

6.3 FILE TYPE

File type refers to the ability of the operating system to distinguish different types of file such as text files, source files and binary files etc. Many operating systems support many types of files. Operating system like MS-DOS and UNIX have the following types of files ''

1) Ordinary files:

- These are the files that contain user information.
- These may have text, databases or executable programs.
- The user can apply various operations on such files like add, modify, delete or even remove the entire file.

2) Directory files:

- These files contain a list of file names and other information related to these files.

3) Special files:

- These files are also known as device files.
- These files represent physical devices like disks, terminals, printers, networks, tape drive etc.

These files are of two types "

- **Character special files** "data is handled character by character as in case of terminals or printers.
- **Block special files** "data is handled in blocks as in the case of disks and tapes.

6.4 FILE ACCESS MECHANISMS

File access mechanism refers to the manner in which the records of a file may be accessed. There are several ways to access files "

- Sequential access
- Direct/Random access
- Indexed sequential access 1)Sequential access

A sequential access is that in which the records are accessed in some sequence, i.e., the information in the file is processed in order, one record after the other. This access method is the most primitive one. Example: Compilers usually access files in this fashion.

2) Direct/Random access:

- Random access file organization provides, accessing the records directly.
- Each record has its own address on the file with the help of which it can be directly accessed for reading or writing.
- The records need not be in any sequence within the file and they need not be in adjacent locations on the storage medium.

Indexed sequential access:

- This mechanism is built up on the basis of sequential access.
- An index is created for each file which contains pointers to various blocks.
- Index is searched sequentially and its pointer is used to access the file directly.

6.5 SPACE ALLOCATION

Files are allocated disk spaces by operating system. Operating systems deploy following three main ways to allocate disk space to files.

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

1) Contiguous Allocation

- Each file occupies a contiguous address space on disk.
- Assigned disk address is in linear order.
- Easy to implement.
- External fragmentation is a major issue with this type of allocation technique.

2) Linked Allocation:

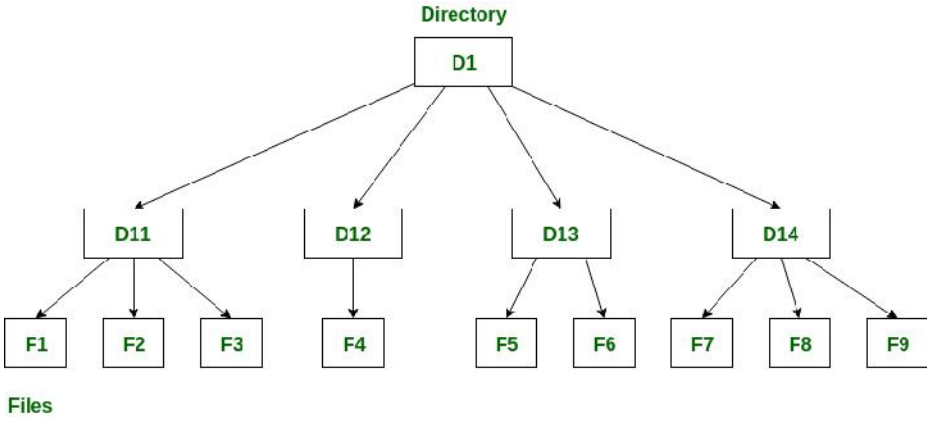
- Each file carries a list of links to disk blocks.
- Directory contains a link / pointer to the first block of a file.
- No external fragmentation
- Effectively used in a sequential access file.
- Inefficient in case of direct access file.

3) Indexed Allocation:

- Provides solutions to problems of contiguous and linked allocation.
- An index block is created having all pointers to files.
- Each file has its own index block which stores the addresses of disk space occupied by the file.
- Directory contains the addresses of index blocks of files.

Structure of directory in OS:

A **directory** is a container that is used to contain folders and files. It organizes files and folders into a hierarchical manner.

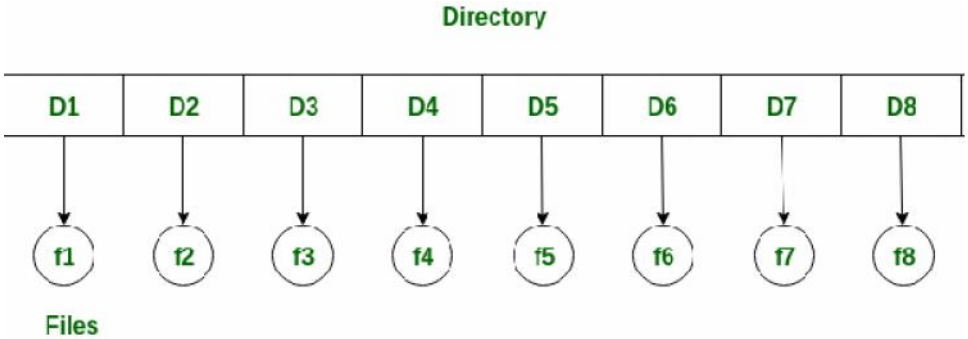


There are several logical structures of a directory, these are given below.

1. **Single-level directory:**

Single level directory is the simplest directory structure. In it all files are contained in the same directory which make it easy to support and understand.

A single level directory has a significant limitation, however, when the number of files increases or when the system has more than one user. Since all the files are in the same directory, they must have the unique name . If two users call their dataset test, then the unique name rule violated.



Advantages:

- Since it is a single directory, so its implementation is very easy.
- If the files are smaller in size, searching will become faster.
- The operations like file creation, searching, deletion, updating are very easy in such a directory structure.

Disadvantages:

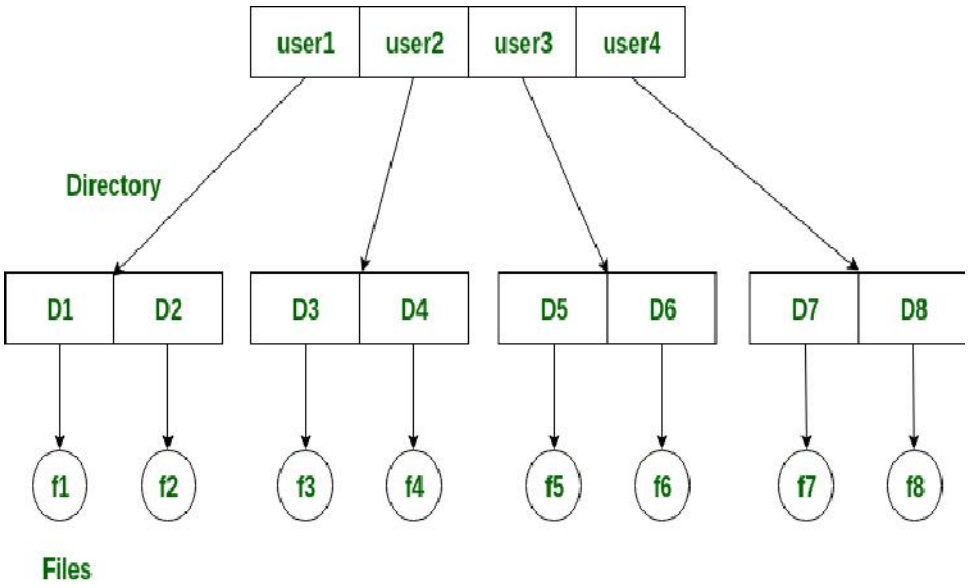
- There may be a chance of name collision because two files can not have the same name.

- Searching will become time taking if the directory is large.
- This can not group the same type of files together.

2. Two-level directory:

As we have seen, a single level directory often leads to confusion of files names among different users. The solution to this problem is to create a separate directory for each user.

In the two-level directory structure, each user has their own *user files directory (UFD)*. The UFDs have similar structures, but each lists only the files of a single user. The system's *master file directory (MFD)* is searched whenever a new user id=s logged in. The MFD is indexed by username or account number, and each entry points to the UFD for that user.



Advantages:

- We can give a full path like /User-name/directory-name/.
- Different users can have the same directory as well as file name.
- Searching for files becomes more easy due to path name and user-grouping.

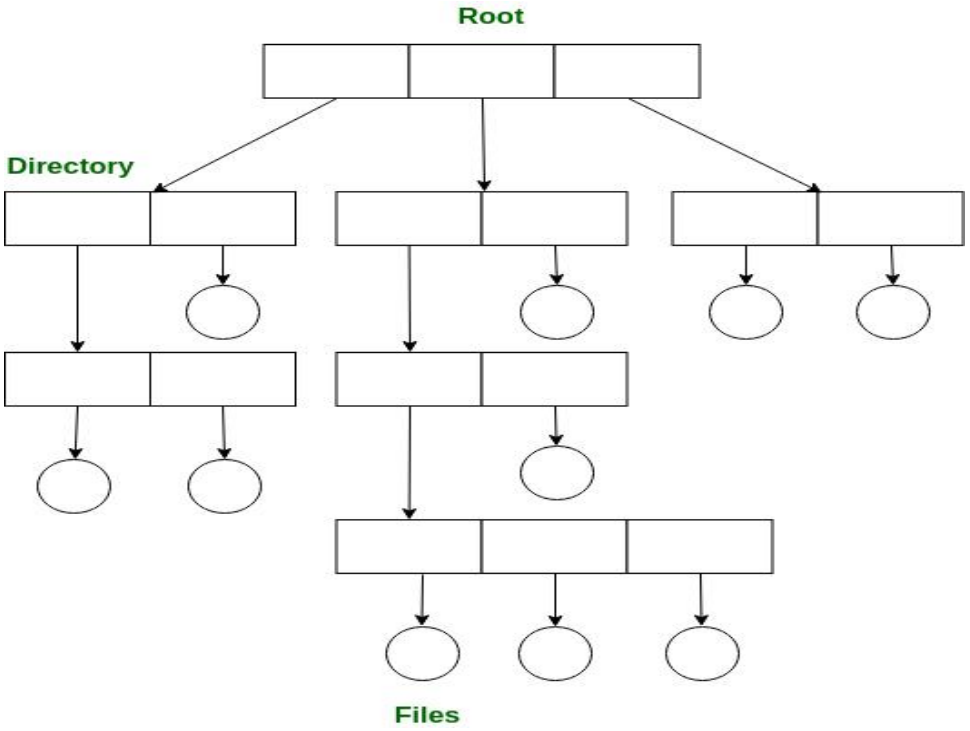
Disadvantages:

- A user is not allowed to share files with other users.
- Still it is not very scalable; two files of the same type cannot be grouped together in the same user.

3. Tree-structured directory:

Once we have seen a two-level directory as a tree of height 2, the natural generalization is to extend the directory structure to a tree of

arbitrary height. This generalization allows the user to create their own subdirectories and to organize their files accordingly.



A tree structure is the most common directory structure. The tree has a root directory, and every file in the system has a unique path.

Advantages:

- Very generalize, since full path names can be given.
- Very scalable, the probability of name collision is less.
- Searching becomes very easy; we can use both absolute paths as well as relative paths.

Disadvantages:

- Every file does not fit into the hierarchical model; files may be saved into multiple directories.
- We cannot share files.
- It is inefficient, because accessing a file may go under multiple directories.

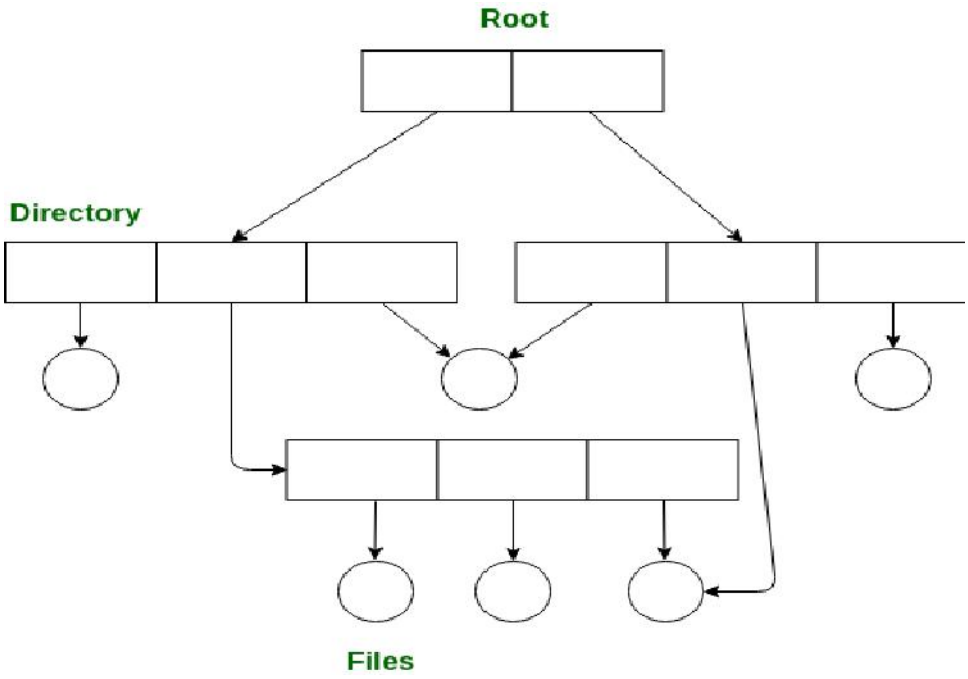
4. Acyclic graph directory:

An acyclic graph is a graph with no cycle and allows to share subdirectories and files. The same file or subdirectories may be in two different directories. It is a natural generalization of the tree-structured directory.

It is used in situations like when two programmers are working on a joint project and they need to access files. The associated files are stored

in a subdirectory, separating them from other projects and files of other programmers, since they are working on a joint project so they want the subdirectories to be into their own directories. The common subdirectories should be shared. So here we use Acyclic directories.

It is the point to note that shared file is not the same as copy file. If any programmer makes some changes in the subdirectory it will reflect in both subdirectories.



Advantages:

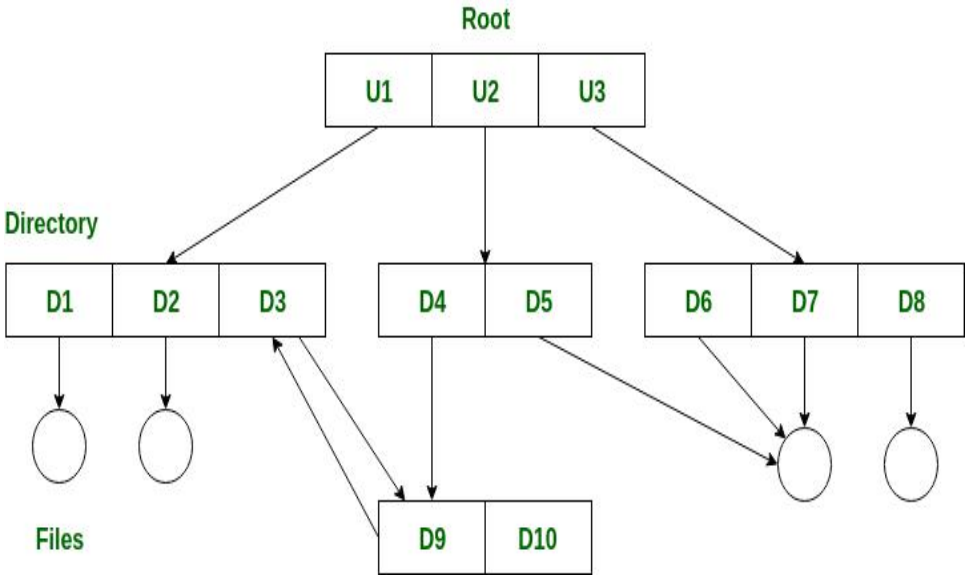
- We can share files.
- Searching is easy due to different-different paths.

Disadvantages:

- We share the files via linking, in case of deleting it may create the problem,
- If the link is softlink then after deleting the file we left with a dangling pointer.
- In the case of hardlink, to delete a file we have to delete all the references associated with it.

General graph directory structure:

In general graph directory structure, cycles are allowed within a directory structure where multiple directories can be derived from more than one parent directory. The main problem with this kind of directory structure is to calculate total size or space that has been taken by the files and directories.



Advantages:

- It allows cycles.
- It is more flexible than other directories structure.

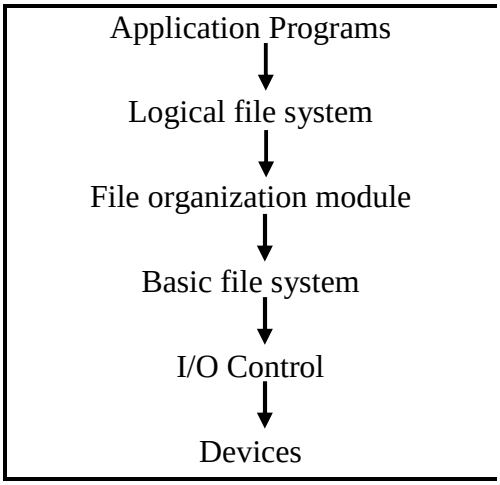
Disadvantages:

- It is more costly than others.
- It needs garbage collection.

File System Implementation:

A file is a collection of related information. The file system resides on secondary storage and provides efficient and convenient access to the disk by allowing data to be stored, located, and retrieved.

File system organized in many layers:



- **I/O Control level:**

Device drivers act as an interface between devices and Os, they help to transfer data between disk and main memory. It takes block

number as input and as output it gives low level hardware specific instruction.
/li>

- **Basic file system:**

It Issues general commands to device drivers to read and write physical blocks on disk. It manages the memory buffers and caches. A block in buffer can hold the contents of the disk block and cache stores frequently used file system metadata.

- **File organization Module:**

It has information about files, location of files and their logical and physical blocks. Physical blocks do not match with logical blocks numbered from 0 to N. It also has a free space which tracks unallocated blocks.

- **Logical file system:**

It manages metadata information about a file i.e includes all details about a file except the actual contents of the file. It also maintains via file control blocks. File control block (FCB) has information about a file – owner, size, permissions, location of file contents.

Advantages:

1. Duplication of code is minimized.
2. Each file system can have its own logical file system.

Disadvantages:

- If we access many files at same time then it results in low performance. We can **implement** file system by using two types data structures :

1.On-disk Structures:

Generally they contain information about total number of disk blocks, free disk blocks, location of them and etc. Below given are different on-disk structures:

1) Boot Control Block:

It is usually the first block of volume and it contains information needed to boot an operating system. In UNIX it is called boot block and in NTFS it is called as partition boot sector.

2) Volume Control Block:

It has information about a particular partition ex: - free block count, block size and block pointers etc. In UNIX it is called super block and in NTFS it is stored in the master file table.

3) Directory Structure:

They store file names and associated inode numbers. In UNIX, includes file names and associated file names and in NTFS, it is stored in the master file table.

3) Per-File FCB:

It contains details about files and it has a unique identifier number to allow association with directory entry. In NTFS it is stored in master file table

File Control Block (FCB)
File permissions
File dates (create, access, write)
File owner, group, ACL
File size
File data blocks or pointers to file data blocks

2) In-Memory Structure:

They are maintained in main-memory and these are helpful for file system management for caching. Several in-memory structures given below:

Mount Table :

It contains information about each mounted volume.

1) **Directory-Structure cache :**
   This cache holds the directory information of recently accessed directories.
2) **System wide open-file table :**
   It contains the copy of FCB of each open file.
3) **Per-process open-file table :**
   It contains information opened by that particular process and it maps with appropriate system wide open-file.

3) Directory Implementation:

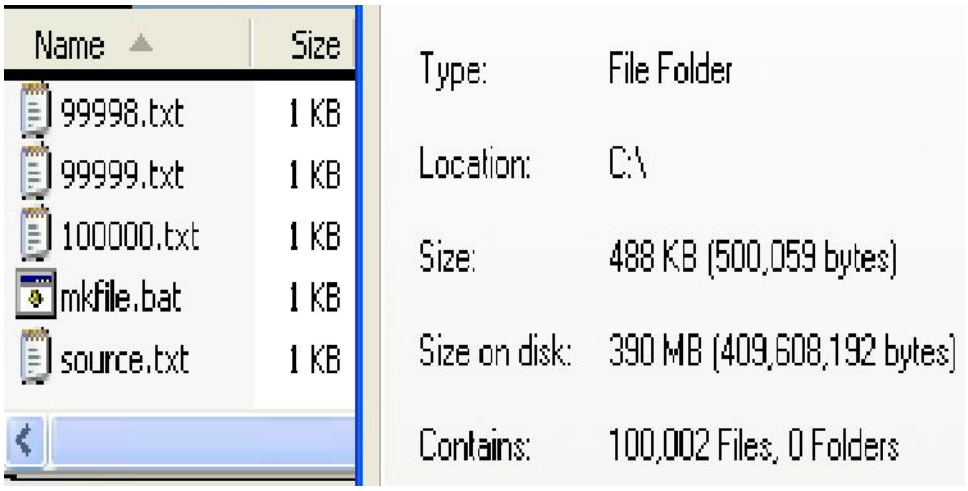
1) Linear List:

It maintains a linear list of filenames with pointers to the data blocks. It is time- consuming also. To create a new file, we must first search the directory to be sure that no existing file has the same name then we add a file at end of the directory. To delete a file, we search the directory for the named file and release the space. To reuse the directory entry either we can mark the entry as unused or we can attach it to a list of free directories.

2) Hash Table:

The hash table takes a value computed from the file name and returns a pointer to the file. It decreases the directory search time. The insertion and deletion process of files is easy. The major difficulty is hash tables are its generally fixed size and hash tables are dependent on hash function on that size.

File System Management and Optimization



1) Disk-Space Management

1) Disk-Space Management:

Since all the files are normally stored on disk one of the main concerns of file system is management of disk space.

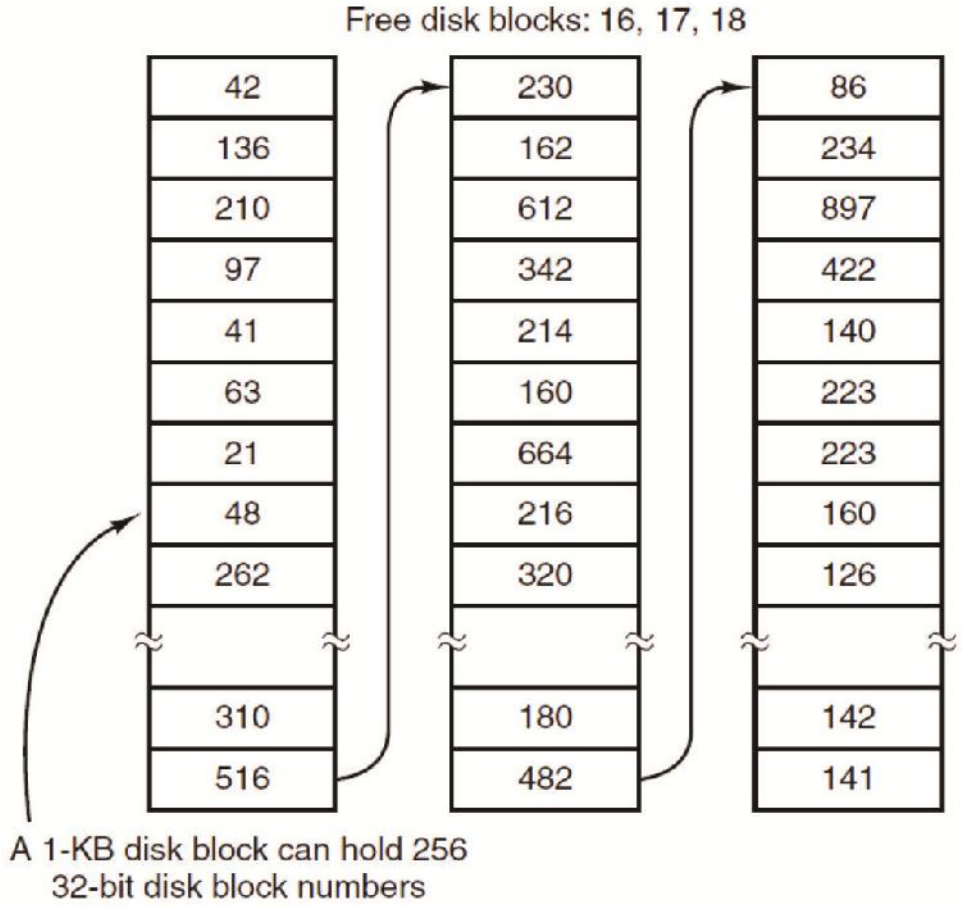
2) Block Size:

The main question that arises while storing files in a fixed-size block is the size of the block. If the block is too large, space gets wasted and if the block is too small, time gets wasted. So, to choose a correct block size some information about the file-size distribution is required. Performance

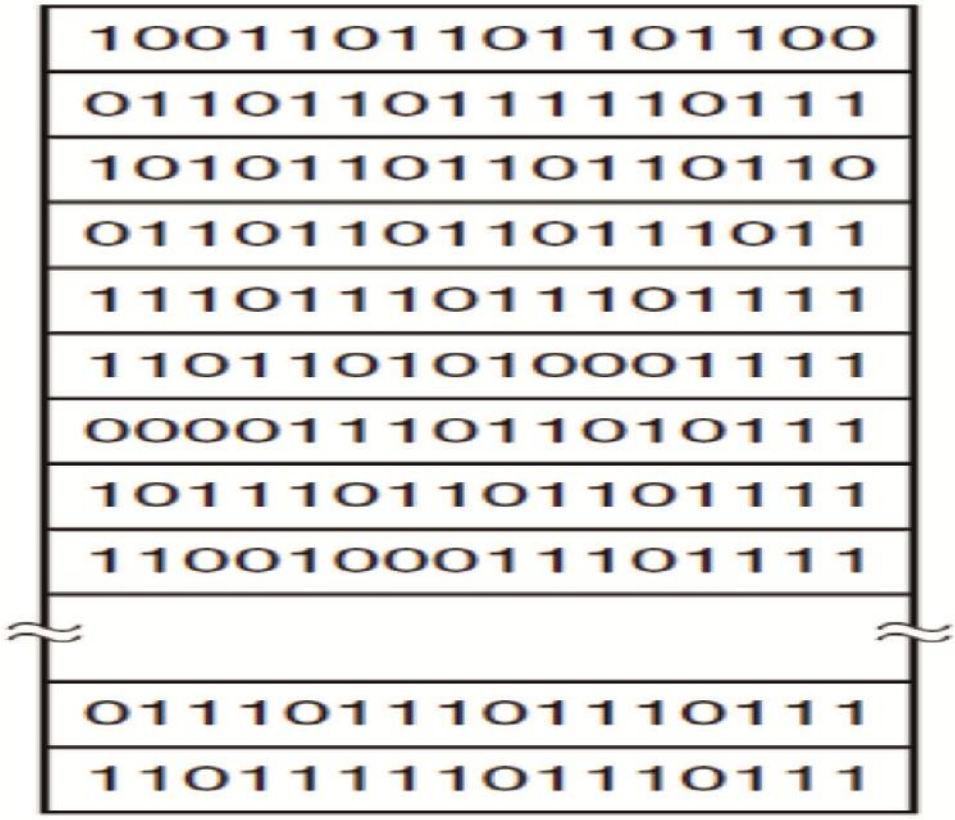
3) Keeping track of free blocks:

After a block size has been finalized the next issue that needs to be catered is how to keep track of the free blocks. In order to keep track there are two methods that are widely used:

Using a linked list: Using a linked list of disk blocks with eachblock holding as many free disk block numbers as will fit.



Bitmap: A disk with n blocks has a bitmap with n bits. Freeblocks are represented using 1's and allocated blocks as 0'sas seen below in the figure.



1) Disk quotas:

Multiuser operating systems often provide a mechanism for enforcing disk quotas. A system administrator assigns each user a maximum allotment of files and blocks and the operating system makes sure that the users do not exceed their quotas. Quotas are kept track of on a per-user basis in a quota table.

5) File-system Backups:

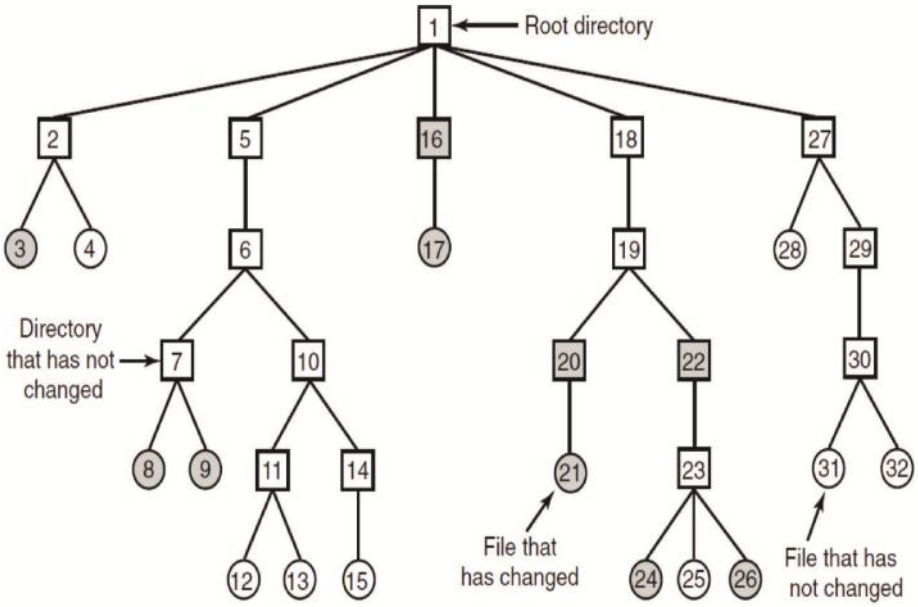
If a computer's file system is irrevocably lost, whether due to hardware or software restoring all the information will be difficult, time consuming and in many cases impossible. So it is advised to always have file-system backups.

- Backing up files is time consuming and as well occupies a large amount of space, so doing it efficiently and conveniently is important. Below are few points to be considered before creating backups for files.
- Is it required to backup the entire file system or only a part of it.
- Backing up files that haven't been changed from previous backup leads to **incremental dumps**. So it's better to take a backup of only those files which have changed from the time
- of previous backup. But recovery gets complicated in such cases.

- Since there is an immense amount of data, it is generally desired to compress the data before taking a backup for the same.
- It is difficult to perform a backup on an active file-system since the backup may be inconsistent.
- Making backups introduces many security issues

There are two ways for dumping a disk to the backup disk:

- **Physical dump:** In this way the dump starts at block 0 of the disk, writes all the disk blocks onto the output disk in order and stops after copying the last one.
- **Advantages:** Simplicity and great speed.
- **Disadvantages:** inability to skip selected directories, make incremental dumps, and restore individual files upon request
- **Logical dump:** In this way the dump starts at one or more specified directories and recursively dump all files and directories found that have been changed since some given base date. This is the most commonly used way



The above figure depicts a popular algorithm used in many UNIX systems wherein squares depict directories and circles depict files. This algorithm dumps all the files and directories that have been modified and also the ones on the path to a modified file or directory. The dump algorithm maintains a bitmap indexed by i-node number with several bits per i-node. Bits will be set and cleared in this map as the algorithm proceeds. Although logical dumping is straightforward, there are few issues associated with it.

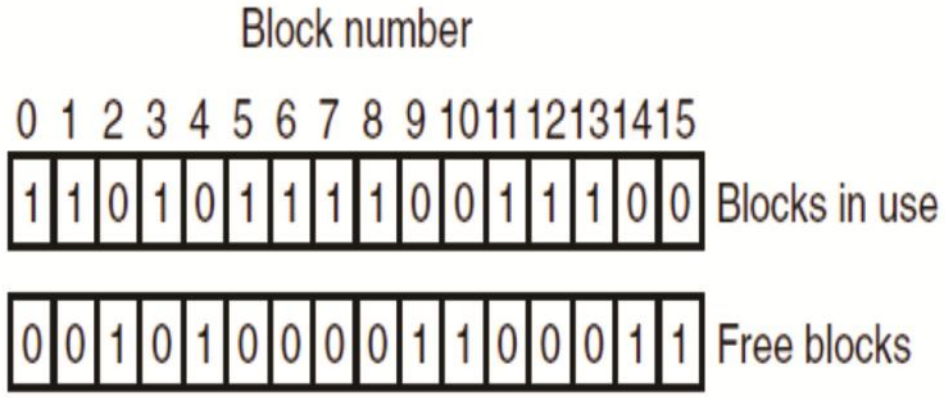
- Since the free block list is not a file, it is not dumped and hence it must be reconstructed from scratch after all the dumps have been restored

- If a file is linked to two or more directories, it is important that the file is restored only one time and that all the directories that are supposed to point to it do so
- UNIX files may contain holes
- Special files, named pipes and all other files that are not real should never be dumped.

6) File-system Consistency:

To deal with inconsistent file systems, most computers have a utility program that checks file-system consistency. For example, UNIX has fsck and Windows has sfc. This utility can be run whenever the system is booted. The utility programs perform two kinds of consistency checks.

- **Blocks:** To check block consistency the program builds two tables, each one containing a counter for each block, initially set to 0. If the file system is consistent, each block will have a 1 either in the first table or in the second table as you can see in the figure below.



In case if both the tables have 0 in it that may be because the block is missing and hence will be reported as a missing block. The two other situations are if a block is seen more than once in a free list and the same data block is present in two or more files.

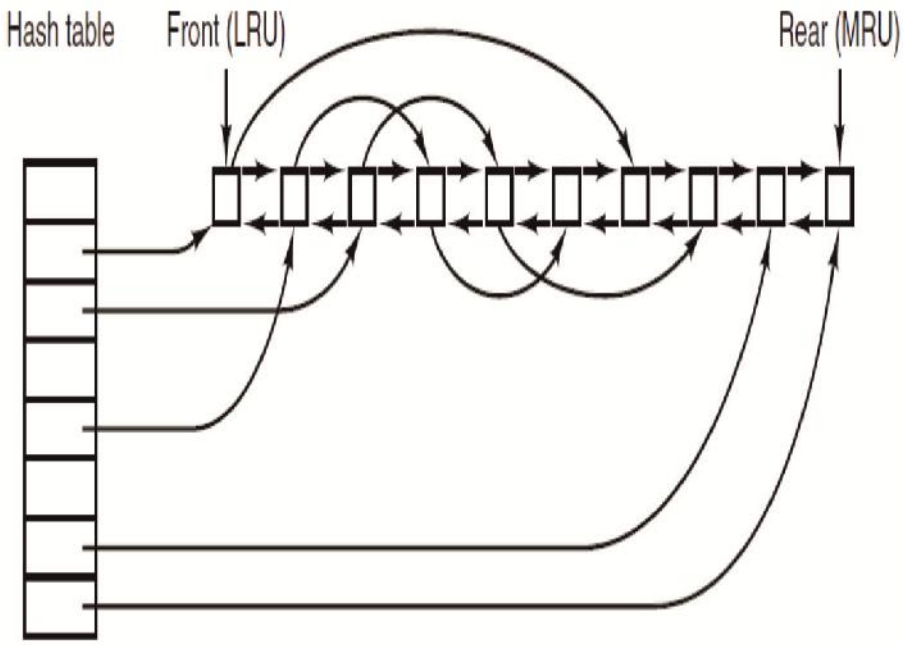
- In addition to checking to see that each block is properly accounted for, the file-system checker also checks the directory system. It too uses a table of counters but per file- size rather than per block. These counts start at 1 when a file is created and are incremented each time a (hard) link is made to the file. In a consistent file system, both counts will agree

7) File-system Performance:

Since the access to disk is much slower than access to memory, many file systems have been designed with various optimizations to improve performance as described below.

8) Caching:

The most common technique used to reduce disk access time is the block cache or buffer cache. Cache can be defined as a collection of items of the same type stored in a hidden or inaccessible place. The most common algorithm for cache works in such a way that if a disk access is initiated, the cache is checked first to see if the disk block is present. If yes then the read request can be satisfied without a disk access else the disk block is copied to cache first and then the read request is processed.



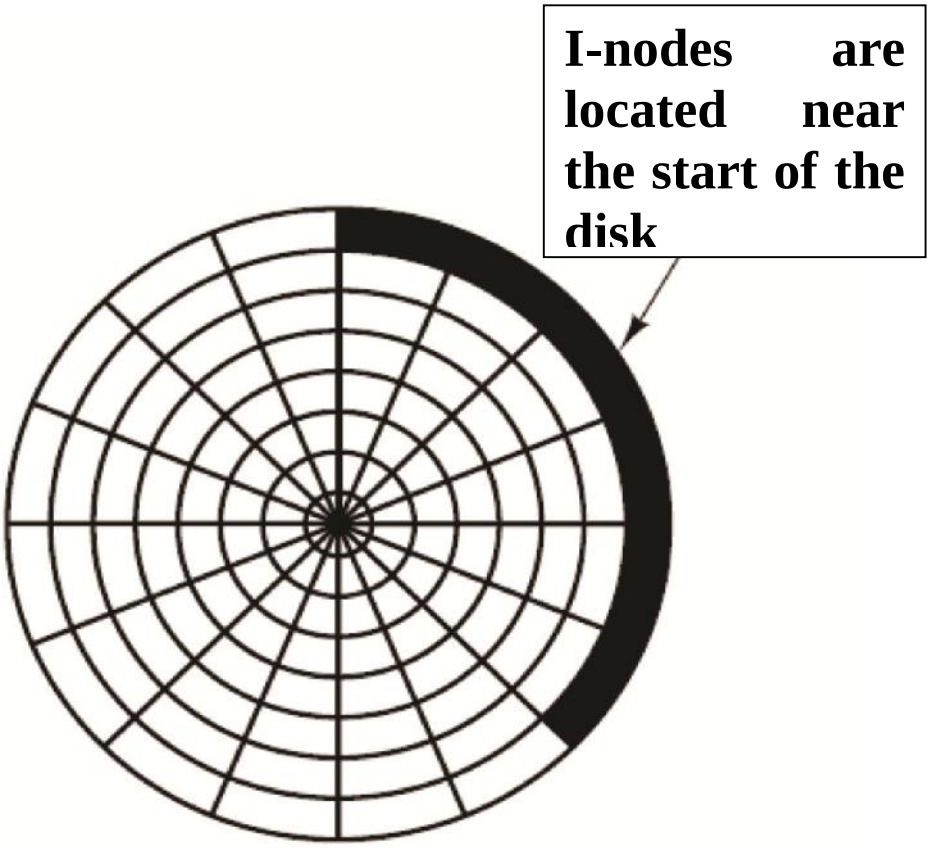
The above figure depicts how to quickly determine if a block is present in a cache or not. For doing so a hash table can be implemented and look up the result in a hash table.

9) Block Read Ahead:

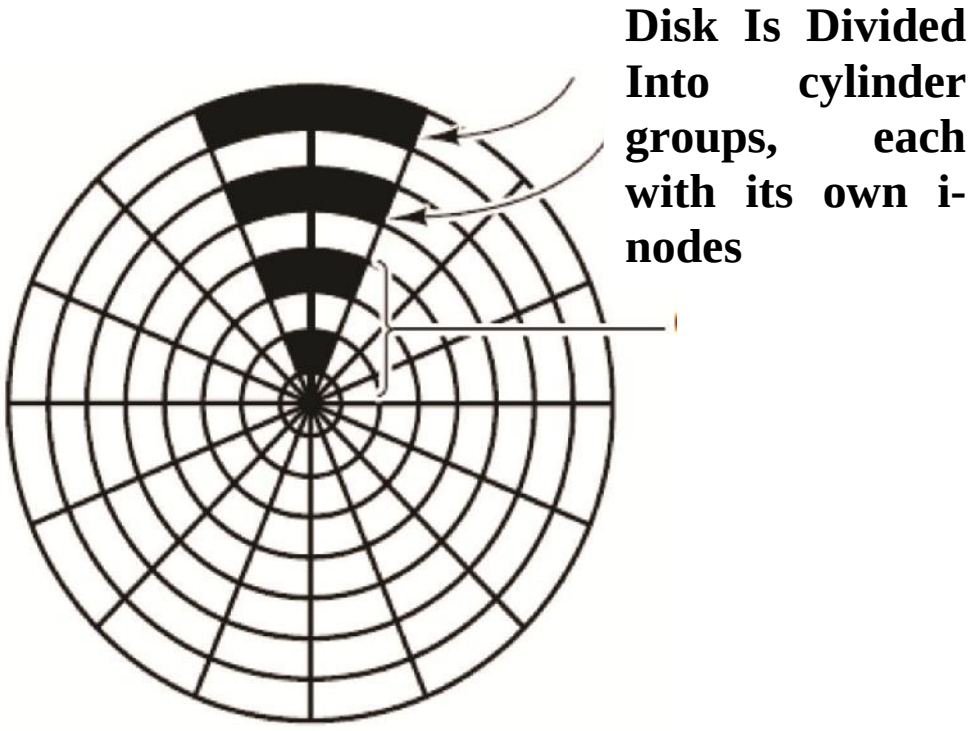
Another technique to improve file-system performance is to try to get blocks into the cache before they are needed to increase the hit rate. This works only when files are read sequentially. When a file system is asked for block ‘k’ in the file it does that and then also checks beforehand if ‘k+1’ is available if not it schedules a read for the block k+1 thinking that it might be of use later.

10) Reducing disk arm motion:

Another way to increase file-system performance is by reducing the disk- arm motion by putting blocks that are likely to be accessed in sequence close to each other, preferably in the same cylinder.



In the above figure all the i-nodes are near the start of the disk, so the average distance between an inode and its blocks will be half the number of cylinders, requiring long seeks. But to increase the performance the **placement of i-nodes can be modified as below next setting**



8) Defragmenting Disks:

Due to continuous creation and removal of files the disks get badly fragmented with files and holes all over the place. As a consequence, when a new file is created, the blocks used for it may be spread all over the disk, giving poor performance. The performance can be restored by moving files around to make them contiguous and to put all (or at least most) of the free space in one or more large contiguous regions on the disk.

MS-DOS Filesystem:

The MS-DOS filesystem is very straightforward. It is a 16-bit system based on a File Allocation Table, or FAT16 (FAT for short). The purpose of the file allocation table is to keep track of where to find files on the disk.

In MS-DOS, every DOS based partition has a letter: (A: or B: or C:). Typically, the drive letters A: and B: are reserved for floppy drives. You will most frequently find that the C: drive is the 'bootable partition'. Each drive has a root directory ('\') so the root directory on a given drive looks like this: C:\

Changing drives is as simple as typing the name of the drive letter:
A:> C: <*enter*>
C:>

MS-DOS then stores files to the system in any arrangement you choose. You can create directories, and store files within those directories. A typical file/path might look like this:
C:\ms-dos\dir\filename.txt

Limitations:

FAT16 file systems are compatible with all Microsoft operating systems, but it has severe limitations. First, all files on the system are limited to eight characters and a three letter extension. The MS-DOS file system also has a limit of approximately 2.1 Gigabytes owing to the fact that the MS-DOS operating system doesn't recognize 'Int 13' based commands, and therefore cannot issue commands to access the remainder of larger disks.

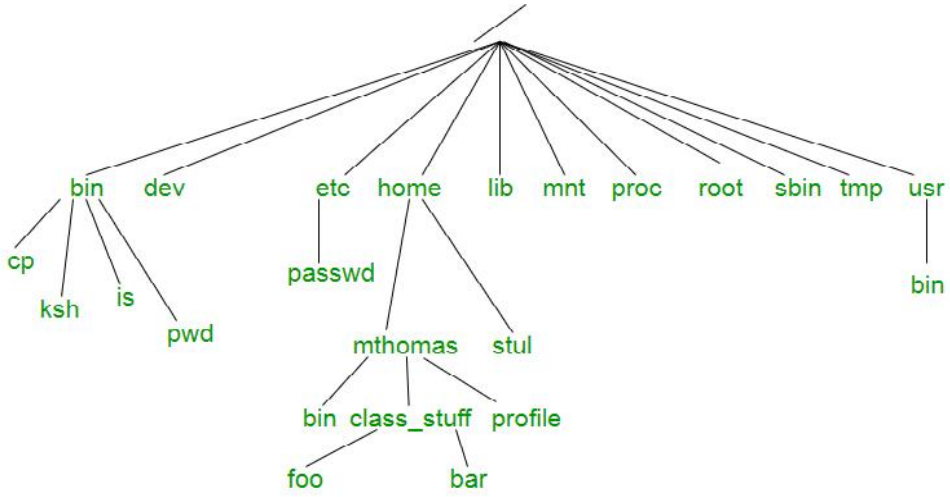
Keep in mind that MS-DOS is a legacy system kept around for doing command line based work in Windows.

Unix File System:

Unix file system is a logical method of **organizing and storing** large amounts of information in a way that makes it easy to manage. A file is a smallest unit in which the information is stored. The Unix file system has several important features. All data in Unix is organized into files. All files

are organized into directories. These directories are organized into a tree-like structure called the file system.

Files in the Unix System are organized into a multi-level hierarchy structure known as a directory tree. At the very top of the file system is a directory called “root” which is represented by a “/”. All other files are “descendants” of root.

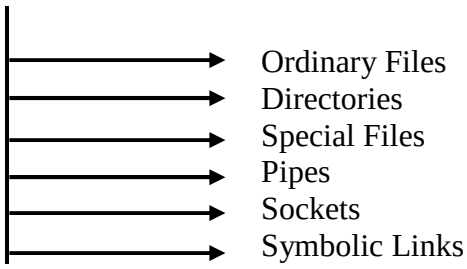


Directories or Files and their description:

- **/ :** The slash / character alone denotes the root of the filesystem tree.
- **/bin :** Stands for “binaries” and contains certain fundamental utilities, such as ls or cp, which are generally needed by all users.
- **/boot :** Contains all the files that are required for a successful booting process.
- **/dev :** Stands for “devices”. Contains file representations of peripheral devices and pseudo-devices.
- **/etc :** Contains system-wide configuration files and system databases. Originally also contained
- “dangerous maintenance utilities” such as init,but these have typically been moved to /sbin or elsewhere.
- **/home :** Contains the home directories for the users.
- **/lib :** Contains system libraries, and some critical files such as kernel modules or device drivers.
- **/media :** Default mount point for removable devices, such as USB sticks, media players, etc.
- **/mnt :** Stands for “mount”. Contains filesystem mount points. These are used, for example, if the
- system uses multiple hard disks or hard disk partitions. It is also often used for remote (network)
- filesystems, CD-ROM/DVD drives, and so on.

- **/proc :** procfs virtual filesystem showing information about processes as files.
- **/root :** The home directory for the superuser "root" – that is, the system administrator. This account's home directory is usually on the initial filesystem, and hence not in /home (which may be a mount point for another filesystem) in case specific maintenance needs to be performed, during
- which other filesystems are not available. Such a case could occur, for example, if a hard disk drive suffers physical failures and cannot be properly mounted.
- **/tmp :** A place for temporary files. Many systems clear this directory upon startup; it might have tmpfs mounted atop it, in which case its contents do not survive a reboot, or it might be explicitly cleared by a startup script at boot time.
- **/usr :** Originally the directory holding user home directories, its use has changed. It now holds executables, libraries, and shared resources that are not system critical, like the X Window System,
- KDE, Perl, etc. However, on some Unix systems, some user accounts may still have a home directory that is a direct subdirectory of /usr, such as the default as in Minix. (on modern systems, these user accounts are often related to server or system use, and not directly used by a person).
- **/usr/bin :** This directory stores all binary programs distributed with the operating system not residing in /bin, /sbin or (rarely) /etc.
- **/usr/include :** Stores the development headers used throughout the system. Header files are mostly used by the **#include** directive in C/C++ programming language.
- **/usr/lib :** Stores the required libraries and data files for programs stored within /usr or elsewhere.
- **/var :** A short for "variable." A place for files that may change often – especially in size, for example e-mail sent to users on the system, or process-ID lock files.
- **/var/log :** Contains system log files.
- **/var/mail :** The place where all the incoming mails are stored. Users (other than root) can access their own mail only. Often, this directory is a symbolic link to /var/spool/mail.
- **/var/spool :** Spool directory. Contains print jobs, mail spools and other queued tasks.
- **/var/tmp :** A place for temporary files which should be preserved between system reboots.

Types of Unix files – The UNIX files system contains several different types of files :



1. Ordinary files: An ordinary file is a file on the system that contains data, text, or program instructions.

- Used to store your information, such as some text you have written or an image you have drawn. This is the type of file that you usually work with.
- Always located within/under a directory file.
- Do not contain other files.
- In long-format output of ls -l, this type of file is specified by the "-" symbol.

2. Directories:

Directories store both special and ordinary files. For users familiar with Windows or Mac OS, UNIX directories are equivalent to folders. A directory file contains an entry for every file and subdirectory that it houses. If you have 10 files in a directory, there will be 10 entries in the directory. Each entry has two components.

(1) The Filename

(2) A unique identification number for the file or directory (called the inode number)

- Branching points in the hierarchical tree.
- Used to organize groups of files.
- May contain ordinary files, special files or other directories.
- Never contain "real" information which you would work with (such as text). Basically, just used for organizing files.
- All files are descendants of the root directory, (named /) located at the top of the tree.

In long-format output of ls –l , this type of file is specified by the "d" symbol.

3. Special Files:

Used to represent a real physical device such as a printer, tape drive or terminal, used for Input/Output (I/O) operations. **Device or**

special files are used for device Input/Output(I/O) on UNIX and Linux systems. They appear in a file system just like an ordinary file or a directory.

On UNIX systems there are two flavors of special files for each device, character special files and block special files:

- When a character special file is used for device Input/Output(I/O), data is transferred one character at a time. This type of access is called raw device access.
- When a block special file is used for device Input/Output(I/O), data is transferred in large fixed-size blocks. This type of access is called block device access.

For terminal devices, it's one character at a time. For disk devices though, raw access means reading or writing in whole chunks of data – blocks, which are native to your disk.

- In long-format output of ls -l, character special files are marked by the "c" symbol.
- In long-format output of ls -l, block special files are marked by the "b" symbol.

4. Pipes:

UNIX allows you to link commands together using a pipe. The pipe acts a temporaryfile which only exists to hold data from one command until it is read by another.A Unix pipe provides a one-way flow of data.The output or result of the first command sequence is used as the input to the second command sequence. To make a pipe, put a vertical bar (|) on the command line between two commands.For example: **who | wc -l**

7.0 What is CDFS (Compact Disc File System)?:

Introduction:

CDFS stands for Compact Disc File System. Before the era of CDFS, there was no medium for people to store their memories or files that they want to store for the long term purpose. The storing of data and information was a major problem because in that time the world needs a system that can store multiple files incompressed format. But the revolution of technology changed the culture of the world and new advanced things started coming to the market. CDFS came into the picture on 21 August 1999. At that time CDFS is considered the most advanced technology in the technology Industry. There were many features offered by CDFS that came into limelight immediately:

1. It is a file system for read-only and write-once CD-ROMs.
2. It exports all tracks and boot images on a CD as normal files.

3. CDFS provides a wide range of services which include creation, replacing, renaming, or deletion of files on write-oncemedia.
4. It uses a VCACHE driver to control the CD-ROM disc cache allowing for a smoother playback.
5. It includes several disc properties like volume attributes, file attributes, and file placement.

History:

CDFS was developed by Simson Garfinkel and J. Spencer Love at the MIT Media Lab between 1985 and 1986. CDFS was developed from the write-once CD-ROM simulator. They are designed to store any data and information on read-only and write-once media.A great setback for CDFS was that it never gets sold. The File System source code was published on the internet.

Disk Images can be saved using the CDFS standard, which may be used to burn ISO 9660 discs. **ISO 9660** also referred to as CDFS by some hardware and software providers, is a file system published by ISO (International Organization for Standardization) for optical disc media.

Applications:

A file system is a systematic organized way in which files have to get organized in a hard disk. The file system is initiated when a user opens a hard disk to access files. Here aresome applications of the Compact Disk

File System:

1. CDFS creates a way in which the system first sets up the root directory and then automatically creates all the subsequent folders for it.
2. The system also provides a wide range of services for all users. You can create new files or folders which are added to the main root file or we can say the “file tree” of the system.
3. There was also a problem of transferring data or files from CDs to a laptop or computer. But CDFS shows us a great solution to solve this problem. It is useful for burning discs that can be exchanged between different devices.
4. CDFS is not specific to a single Operating System, it means that a disc burned on Macintosh using CDFS can be read on a Windows or Linux based computer.
5. It can operate over numerous Operating Systems. It means if a user started shifting files from Macintosh using Compact Disk File System, he can also operate the files in Windows Operating System.
6. Disc Pictures are also saved using proper system standards. All files have a typical.ISO name extension.

Types:

There are different versions of Compact Disk File System:

1. **Clustered operated system. (can be Global or Grid)**
2. **Flash operated**

3. **Object file system**
4. **Semantic**
5. **Steganographic process**
6. **Versioning**
7. **Synthetic operated system**

6.6 SUMMARY

- A file is a collection of correlated information which is recorded on secondary or non-volatile storage like magnetic disks, optical disks, and tapes.
- It provides I/O support for a variety of storage device types.
- Files are stored on disk or other storage and do not disappear when a user logs off.
- A File Structure needs to be predefined format in such a way that an operating system understands it.
- File type refers to the ability of the operating system to differentiate different types of files like text files, binary, and source files.
- Create find space on disk and make an entry in the directory.
- Indexed Sequential Access method is based on simple sequential access
- In Sequential Access method records are accessed in a certain predefined sequence
- The random access method is also called direct random access
- Three types of space allocation methods are: Linked Allocation,Indexed Allocation,Contiguous Allocation
- Information about files is maintained by Directories

6.7 UNIT END QUESTIONS

1. Explain File System?
2. State the Objectives of File management System
3. Discuss the properties of a File System
4. Explain File structure
5. Discuss File Attributes
6. Explain File Type
7. List the Functions of File
8. State and explain Commonly used terms in File systems
9. Explain different File Access Methods
10. Explain Space Allocation
11. Discuss the File Directories
12. Explain File types- name, extension

*****

UNIT III

7

PRINCIPLES OF I/O HARDWARE AND SOFTWARE

Unit Structure

7.0 OBJECTIVES

- To understand principles of i/o hardware
- To learn principles of i/o software
- To learn different, i/o software layers

In addition to providing abstractions such as processes, address spaces, and files, an operating system also controls all the computer's I/O (Input/Output) devices. It must issue commands to the devices, catch interrupts, and handle errors. It should also provide an interface between the devices and the rest of the system that is simple and easy to use. To the extent possible, the interface should be the same for all devices (device independence). The I/O code represents a significant fraction of the total operating system. How the operating system manages I/O is the subject of this chapter.

This chapter is organized as follows. We will look first at some of the principles of I/O hardware and then at I/O software in general. I/O software can be structured in layers, with each having a well-defined task. We will look at these layers to see what they do and how they fit together. Next, we will look at several I/O devices in detail: disks, clocks, keyboards, and displays. For each device we will look at its hardware and software. Finally, we will consider power management.

7.1 INTRODUCTION

Different people look at I/O hardware in different ways. In this book we are concerned with programming I/O devices, not designing,

building, or maintaining them, so our interest is in how the hardware is programmed, not how it works inside.

7.1.1 I/O Devices:

I/O devices can be roughly divided into two categories: **block devices** and **character devices**. A block device is one that stores information in fixed-size blocks, each one with its own address. Common block sizes range from 512 to 65,536 bytes. All transfers are in units of one or more entire (consecutive) blocks. The essential property of a block device is that it is possible to read or write each block independently of all the other ones. Hard disks, Blu-ray discs, and USB sticks are common block devices.

The other type of I/O device is the character device. A character device delivers or accepts a stream of characters, without regard to any block structure. It is not addressable and does not have any seek operation. Printers, network interfaces, mice (for pointing), rats (for psychology lab experiments), and most other devices that are not disk-like can be seen as character devices. I/O devices cover a huge range in speeds, which puts considerable pressure on the software to perform well over many orders of magnitude in data rates. Figure 7.1 shows the data rates of some common devices.

Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Scanner at 300 dpi	1 MB/sec
Digital camcorder	3.5 MB/sec
4x Blu-ray disc	18 MB/sec
802.11n Wireless	37.5 MB/sec
USB 2.0	60 MB/sec
FireWire 800	100 MB/sec
Gigabit Ethernet	125 MB/sec
SATA 3 disk drive	600 MB/sec
USB 3.0	625 MB/sec
SCSI Ultra 5 bus	640 MB/sec
Single-lane PCIe 3.0 bus	985 MB/sec
Thunderbolt 2 bus	2.5 GB/sec
SONET OC-768 network	5 GB/sec

Figure 7.1 data rates of devices

7.1.2 Device Controllers:

I/O units often consist of a mechanical component and an electronic component. It is possible to separate the two portions to provide a more modular and general design. The electronic component is called

the **device controller** or **adapter**. On personal computers, it often takes the form of a chip on the parent board or a printed circuit

7.1.3 Memory-Mapped I/O:

Each controller has a few registers that are used for communicating with the CPU. By writing into these registers, the operating system can command the device to deliver data, accept data, switch itself on or off, or otherwise perform some action. By reading from these registers, the operating system can learn what the device's state is, whether it is prepared to accept a new command, and so on.

Each control register is assigned a unique memory address to which no memory is assigned. This system is called **memory-mapped I/O**. In most systems, the assigned addresses are at or near the top of the address space.

7.1.4 Direct Memory Access:

No matter whether a CPU does or does not have memory-mapped I/O, it needs to address the device controllers to exchange data with them. The CPU can request data from an I/O controller one byte at a time, but doing so wastes the CPU's time, so a different scheme, called **DMA** (**Direct Memory Access**) is often used. To simplify the explanation, we assume that the CPU accesses all devices and memory via a single system bus that connects the CPU, the memory, and the I/O devices. We already know that the real organization in modern systems is more complicated, but all the principles are the same. The operating system can use only DMA if the hardware has a DMA controller, which most systems do.

Sometimes this controller is integrated into disk controllers and other controllers, but such a design requires a separate DMA controller for each device. More commonly, a single DMA controller is available (e.g., on the parentboard) for regulating transfers to multiple devices, often concurrently

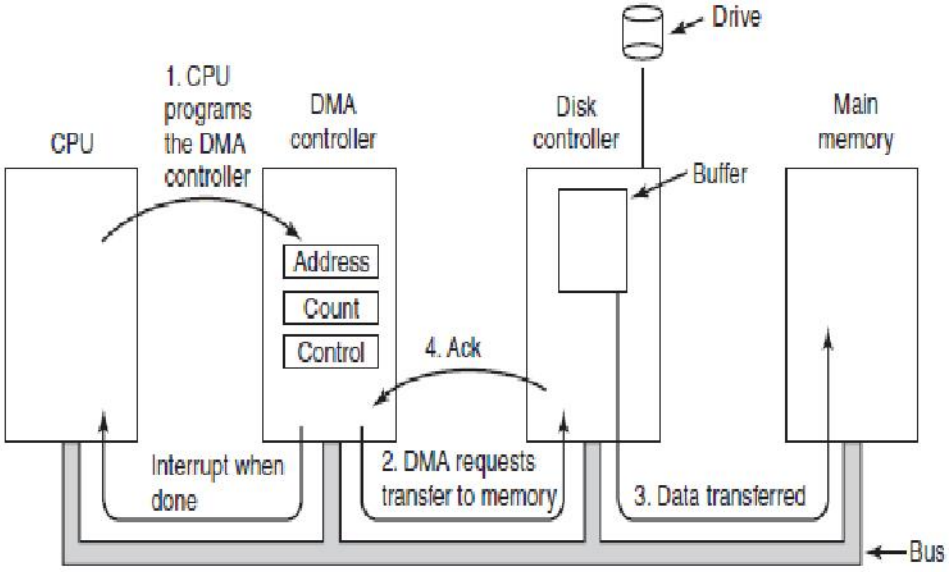


Fig 7.2 DMA controller

Some DMA controllers can also operate in either mode. In the former mode, the DMA controller requests the transfer of one word and gets it. If the CPU also wants the bus, it has to wait. The mechanism is called **cycle stealing** because the device controller sneaks in and steals an occasional bus cycle from the CPU once in a while, delaying it slightly.

In block mode, the DMA controller tells the device to acquire the bus, issue a series of transfers, then release the bus. This form of operation is called **burst mode**. It is more efficient than cycle stealing because acquiring the bus takes time and multiple words can be transferred for the price of one bus acquisition. The down side to burst mode is that it can block the CPU and other devices for a substantial period if a long burst is being transferred.

7.1.5 Interrupts Revisited:

In a typical personal computer system, the interrupt structure is as shown in Fig. 7.3 At the hardware level, interrupts work as follows. When an I/O device has finished the work given to it, it causes an interrupt (assuming that interrupts have been enabled by the operating system). It does this by asserting a signal on a bus line that it has been assigned. This signal is detected by the interrupt controller chip on the parent board, which then decides what to do.

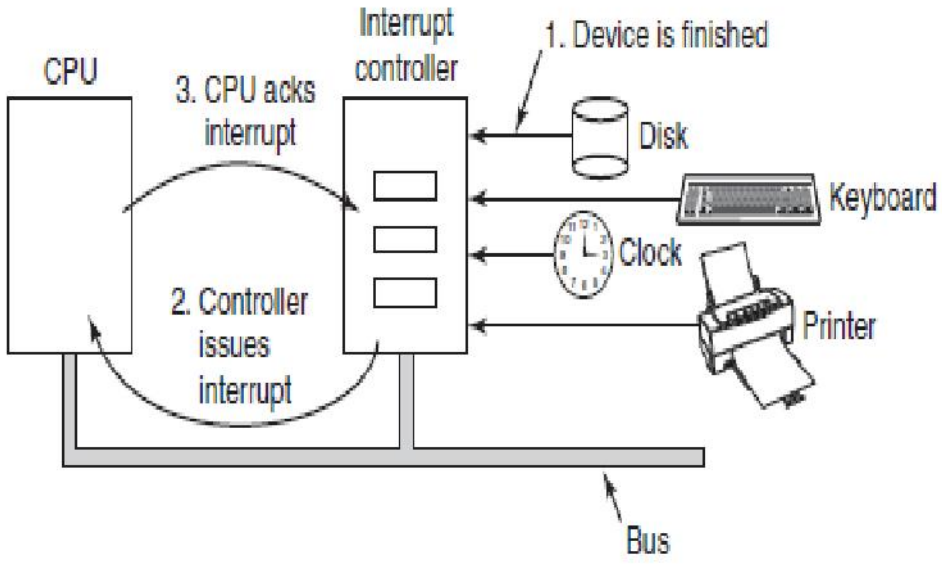


Fig 7.3 Interrupts

If no other interrupts are pending, the interrupt controller handles the interrupt immediately. However, if another interrupt is in progress, or another device has made a simultaneous request on a higher-priority interrupt request line on the bus, the device is just ignored for the moment. In this case it continues to assert an interrupt signal on the bus until it is serviced by the CPU. To handle the interrupt, the controller puts a number on the address lines specifying which device wants attention and asserts a signal to interrupt the CPU. The interrupt signal causes the CPU to stop

what it is doing and start doing something else. The number on the address lines is used as an index into a table called the **interrupt vector** to fetch a new program counter. This program counter points to the start of the corresponding interrupt-service procedure.

Precise Interrupts:

An interrupt that leaves the machine in a well-defined state is called a **precise interrupt.**

Such an interrupt has four properties:

1. The PC (Program Counter) is saved in a known place.
2. All instructions before the one pointed to by the PC have completed.
3. No instruction beyond the one pointed to by the PC has finished.
4. The execution state of the instruction pointed to by the PC is known.

7.2 PRINCIPLES OF I/O SOFTWARE

First we will look at its goals and then at the different ways I/O can be done from the point of view of the operating system.

7.2.1 Goals of the I/O Software:

A key concept in the design of I/O software is known as **device independence**.

What it means is that we should be able to write programs that can access any I/O device without having to specify the device in advance. For example, a program that reads a file as input should be able to read a file on a hard disk, a DVD, or on a USB stick without having to be modified for each different device.

It is up to the operating system to take care of the problems caused by the fact that these devices really are different and require very different command sequences to read or write.

Another important issue for I/O software is **error handling**. In general, errors should be handled as close to the hardware as possible. If the controller discovers a read error, it should try to correct the error itself if it can.

Still another important issue is that of **synchronous** (blocking) vs. **asynchronous** (interrupt-driven) transfers. Most physical I/O is asynchronous—the CPU starts the transfer and goes off to do something else until the interrupt arrives. User programs are much easier to write if the I/O operations are blocking—after a read system call the program is automatically suspended until the data are available in the buffer. It is up

to the operating system to make operations that are actually interrupt-driven look blocking to the user programs.

The final concept that we will mention here is sharable vs. dedicated devices. Some I/O devices, such as disks, can be used by many users at the same time. No problems are caused by multiple users having open files on the same disk at the same time. Other devices, such as printers, have to be dedicated to a single user until that user is finished. Then another user can have the printer.

7.2.2 Programmed I/O:

The simplest form of I/O is to have the CPU do all the work. This method is called **programmed I/O**.

It is simplest to illustrate how programmed I/O works by means of an example. Consider a user process that wants to print the eight-character string ‘‘ABCDEFGH’’on the printer via a serial interface. Displays on small embedded systems sometimes work this way. The software first assembles the string in a buffer in user space, as shown in Fig. 7.4

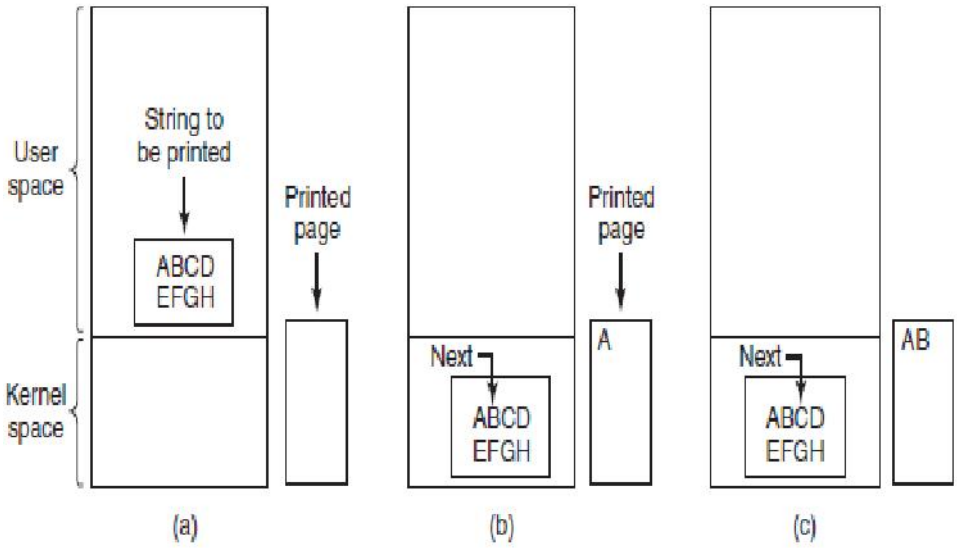


Fig 7.4 programmed I/O

The user process then acquires the printer for writing by making a system call to open it. If the printer is currently in use by another process, this call will fail and return an error code or will block until the printer is available, depending on the operating system and the parameters of the call. Once it has the printer, the user process makes a system call telling the operating system to print the string on the printer.

The operating system then (usually) copies the buffer with the string to an array, say, *p*, in kernel space, where it is more easily accessed (because the kernel may have to change the memory map to get at user space). It then checks to see if the printer is currently available. If not, it

waits until it is. As soon as the printer is available, the operating system copies the first character to the printer's data register, in this example using memory-mapped I/O. This action activates the printer. The character may not appear yet because some printers buffer a line or a page before printing anything. In Fig. 7.4 (b), however, we see that the first character has been printed and that the system has marked the ''B'' as the next character to be printed. As soon as it has copied the first character to the printer, the operating system checks to see if the printer is ready to accept another one. Generally, the printer has a second register, which gives its status. The act of writing to the data register causes the status to become not ready. When the printer controller has processed the current character, it indicates its availability by setting some bit in its status register or putting some value in it.

At this point the operating system waits for the printer to become ready again. When that happens, it prints the next character, as shown in Fig. 7.4 (c). This loop continues until the entire string has been printed. Then control returns to the user process.

7.2.3 Interrupt-Driven I/O:

Now let us consider the case of printing on a printer that does not buffer characters but prints each one as it arrives. If the printer can print, say 100 characters/ sec, each character takes 10 msec to print. This means that after every character is written to the printer's data register, the CPU will sit in an idle loop for 10 msec waiting to be allowed to output the next character. This is more than enough time to do a context switch and run some other process for the 10 msec that would otherwise be wasted.

The way to allow the CPU to do something else while waiting for the printer to become ready is to use interrupts. When the system call to print the string is made, the buffer is copied to kernel space, as we showed earlier, and the first character is copied to the printer as soon as it is willing to accept a character. At that point the CPU calls the scheduler and some other process is run. The process that asked for the string to be printed is blocked until the entire string has printed.

7.2.4 I/O Using DMA:

An obvious disadvantage of interrupt-driven I/O is that an interrupt occurs on every character. Interrupts take time, so this scheme wastes a certain amount of CPU time. A solution is to use DMA. Here the idea is to let the DMA controller feed the characters to the printer one at time, without the CPU being bothered. In essence, DMA is programmed I/O, only with the DMA controller doing all the work, instead of the main CPU. This strategy requires special hardware (the DMA controller) but frees up the CPU during the I/O to do other work.

The big win with DMA is reducing the number of interrupts from one per character to one per buffer printed. If there are many characters and interrupts are slow, this can be a major improvement. On the other hand, the DMA controller is usually much slower than the main CPU. If the DMA controller is not capable of driving the device at full speed, or the CPU usually has nothing to do anyway while waiting for the DMA interrupt, then interrupt-driven I/O or even programmed I/O may be better.

7.3 I/O SOFTWARE LAYERS

I/O software is typically organized in four layers, as shown in Fig. 7.5 Each layer has a well-defined function to perform and a well-defined interface to the adjacent layers

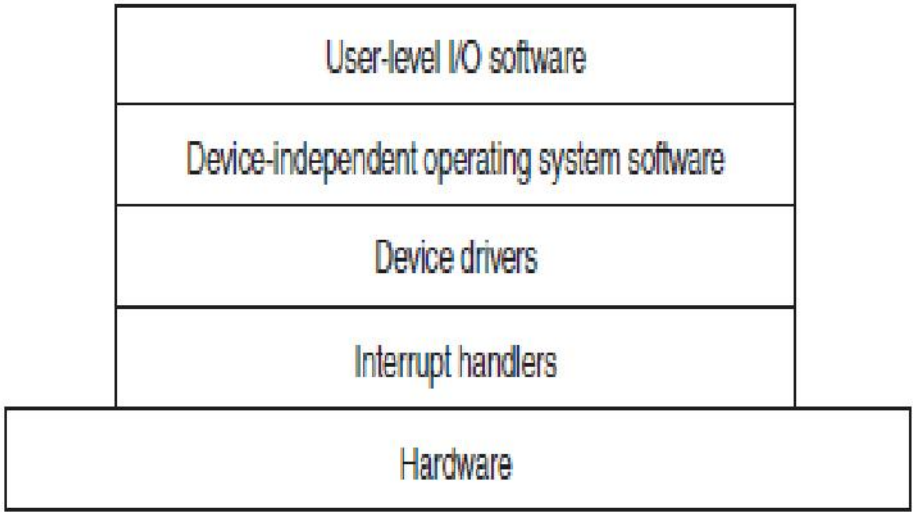


Figure 7.5 I/O SOFTWARE LAYERS

7.3.1 Interrupt Handlers:

When the interrupt happens, the interrupt procedure does whatever it has to in order to handle the interrupt. Then it can unblock the driver that was waiting for it. In some cases it will just complete up on a semaphore. In others it will do a signal on a condition variable in a monitor. In still others, it will send a message to the blocked driver. In all cases the net effect of the interrupt will be that a driver that was previously blocked will now be able to run. This model works best if drivers are structured as kernel processes, with their own states, stacks, and program counters. Of course, reality is not quite so simple. Processing an interrupt is not just a matter of taking the interrupt, doing an up on some semaphore, and then executing an IRET instruction to return from the interrupt to the previous process. There is a great deal more work involved for the operating system. We will now give an outline of this work as a series of steps that must be performed in software after the hardware interrupt has completed. It should be noted that the details are highly system dependent, so some of the steps listed below may not be needed on a particular machine, and

steps not listed may be required. Also, the steps that do occur may be in a different order on some machines.

1. Save any registers (including the PSW) that have not already been saved by the interrupt hardware.
2. Set up a context for the interrupt-service procedure. Doing this may involve setting up the TLB, MMU and a page table.
3. Set up a stack for the interrupt service-procedure.
4. Acknowledge the interrupt controller. If there is no centralized interrupt controller, re enable interrupts.
5. Copy the registers from where they were saved (possibly some stack) to the process table.
6. Run the interrupt-service procedure. It will extract information from the interrupting device controller's registers.
7. Choose which process to run next. If the interrupt has caused some high-priority process that was blocked to become ready, it may be chosen to run now.
8. Set up the MMU context for the process to run next. Some TLB setup may also be needed.
9. Load the new process' registers, including its PSW.
10. Start running the new process.

7.3.2 Device Drivers:

Earlier in this chapter we looked at what device controllers do. We saw that each controller has some device registers used to give it commands or some device registers used to read out its status or both. The number of device registers and the nature of the commands vary radically from device to device. For example, a mouse driver has to accept information from the mouse telling it how far it has moved and which buttons are currently depressed. In contrast, a disk driver may have to know all about sectors, tracks, cylinders, heads, arm motion, motor drives, head settling times, and all the other mechanics of making the disk work properly. Obviously, these drivers will be very different.

Consequently, each I/O device attached to a computer needs some device-specific code for controlling it. This code, called the **device driver**, is generally written by the device's manufacturer and delivered along with the device. Since each operating system needs its own drivers, device manufacturers commonly supply drivers for several popular operating systems.

Each device driver normally handles one device type, or at most, one class of closely related devices. For example, a SCSI disk driver can usually handle multiple SCSI disks of different sizes and different speeds, and perhaps a SCSI Blu-ray disk as well. On the other hand, a mouse and joystick are so different that different drivers are usually required.

However, there is no technical restriction on having one device driver control multiple unrelated devices. It is just not a good idea in most cases.

Sometimes though, wildly different devices are based on the same underlying technology. The best-known example is probably USB, a serial bus technology that is not called ‘‘universal’’ for nothing. USB devices include disks, memory sticks, cameras, mice, keyboards, mini-fans, wireless network cards, robots, credit card readers, rechargeable shavers, paper shredders, barcode scanners, disco balls, and portable thermometers. They all use USB and yet they all do very different things. The trick is that USB drivers are typically stacked, like a TCP/IP stack in networks.

In order to access the device’s hardware, actually, meaning the controller’s registers, the device driver normally has to be part of the operating system kernel, at least with current architectures. Actually, it is possible to construct drivers that run in user space, with system calls for reading and writing the device registers. This design isolates the kernel from the drivers and the drivers from each other, eliminating a major source of system crashes—buggy drivers that interfere with the kernel in one way or another. For building highly reliable systems, this is definitely the way to go.

7.3.3 Device-Independent I/O Software:

Although some of the I/O software is device specific, other parts of it are device independent. The exact boundary between the drivers and the device-independent software is system (and device) dependent, because some functions that could be done in a device-independent way may actually be done in the drivers, for efficiency or other reasons. The functions shown in Fig. 7.6 are typically done in the device independent software.

Uniform interfacing for device drivers
Buffering
Error reporting
Allocating and releasing dedicated devices
Providing a device-independent block size

Fig 7.6 Device-Independent I/O Software Function

The basic function of the device-independent software is to perform the I/O functions that are common to all devices and to provide a uniform interface to the user-level software. We will now look at the above issues in more detail.

Uniform Interfacing for Device Drivers:

A major issue in an operating system is how to make all I/O devices and drivers look more or less the same. If disks, printers, keyboards, and

so on, are all interfaced in different ways, every time a new device comes along, the operating system must be modified for the new device. Having to hack on the operating system for each new device is not a good idea

Buffering:

Buffering is also an issue, both for block and character devices, for a variety of reasons. To see one of them, consider a process that wants to read data from an (ADSL—Asymmetric Digital Subscriber Line) modem, something many people use at home to connect to the Internet. One possible strategy for dealing with the incoming characters is to have the user process do a read system call and block waiting for one character. Each arriving character causes an interrupt. The interrupt-service procedure hands the character to the user process and unblocks it. After putting the character somewhere, the process reads another character and blocks again.

Error Reporting:

Errors are far more common in the context of I/O than in other contexts. When they occur, the operating system must handle them as best it can. Many errors are device specific and must be handled by the appropriate driver, but the framework for error handling is device independent.

One class of I/O errors is programming errors. These occur when a process asks for something impossible, such as writing to an input device (keyboard, scanner, mouse, etc.) or reading from an output device (printer, plotter, etc.). Other errors are providing an invalid buffer address or other parameter, and specifying an invalid device (e.g., disk 3 when the system has only two disks), and so on. The action to take on these errors is straightforward: just report back an error code to the caller. Another class of errors is the class of actual I/O errors, for example, trying to write a disk block that has been damaged or trying to read from a camcorder that has been switched off. In these circumstances, it is up to the driver to determine what to do. If the driver does not know what to do, it may pass the problem back up to device independent software.

What this software does depends on the environment and the nature of the error. If it is a simple read error and there is an interactive user available, it may display a dialog box asking the user what to do. The options may include retrying a certain number of times, ignoring the error, or killing the calling process. If there is no user available, probably the only real option is to have the system call fail with an error code.

Allocating and Releasing Dedicated Devices:

Some devices, such as printers, can be used only by a single process at any given moment. It is up to the operating system to examine requests for device usage and accept or reject them, depending on whether

the requested device is available or not. A simple way to handle these requests is to require processes to perform opens on the special files for devices directly. If the device is unavailable, the open fails. Closing such a dedicated device then releases it.

An alternative approach is to have special mechanisms for requesting and releasing dedicated devices. An attempt to acquire a device that is not available blocks the caller instead of failing. Blocked processes are put on a queue. Sooner or later, the requested device becomes available and the first process on the queue is allowed to acquire it and continue execution.

Device-Independent Block Size:

Different disks may have different sector sizes. It is up to the device-independent software to hide this fact and provide a uniform block size to higher layers, for example, by treating several sectors as a single logical block. In this way, the higher layers deal only with abstract devices that all use the same logical block size, independent of the physical sector size. Similarly, some character devices deliver their data one byte at a time (e.g., mice), while others deliver theirs in larger units (e.g., Ethernet interfaces). These differences may also be hidden

7.3.4 User-Space I/O Software:

Although most of the I/O software is within the operating system, a small portion of it consists of libraries linked together with user programs, and even whole programs running outside the kernel. System calls, including the I/O system calls, are normally made by library procedures.
When a C program contains the call
count = write(fd, buffer, nbytes);

The library procedure *write* might be linked with the program and contained in the binary program present in memory at run time. In other systems, libraries can be loaded during program execution. Either way, the collection of all these library procedures is clearly part of the I/O system.

While these procedures do little more than put their parameters in the appropriate place for the system call, other I/O procedures actually do real work.

In particular, formatting of input and output is done by library procedures.

One example from C is *printf*, which takes a format string and possibly some variables as input, builds an ASCII string, and then calls write to output the string. As an example of *printf*, consider the statement.

printf("The square of %3d is %6d\n", i, i*i);

It formats a string consisting of the 14-character string ‘‘The square of ’’ followed by the value *i* as a 3-character string, then the 4-character string ‘‘ is ’’, then *i*2 as 6 characters, and finally a line feed.

An example of a similar procedure for input is *scanf*, which reads input and stores it into variables described in a format string using the same syntax as *printf*.

The standard I/O library contains a number of procedures that involve I/O and all run as part of user programs.

Not all user-level I/O software consists of library procedures. Another important category is the spooling system. **Spooling** is a way of dealing with dedicated I/O devices in a multiprogramming system.

Consider a typical spooled device: a printer. Although it would be technically easy to let any user process open the character special file for the printer, suppose a process opened it and then did nothing for hours. No other process could print anything.

Instead what is done is to create a special process, called a **daemon**, and a special directory, called a **spooling directory**. To print a file, a process first generates the entire file to be printed and puts it in the spooling directory. It is up to the daemon, which is the only process having permission to use the printer’s special file, to print the files in the directory. By protecting the special file against direct use by users, the problem of having someone keeping it open unnecessarily long is eliminated. Spooling is used not only for printers. It is also used in other I/O situations.

7.4 SUMMARY

Different people look at I/O hardware in different ways. In this book we are concerned with programming I/O devices, not designing, building, or maintaining them, so our interest is in how the hardware is programmed, not how it works inside. Different disks may have different sector sizes. It is up to the device-independent software to hide this fact and provide a uniform block size to higher layers, for example, by treating several sectors as a single logical block.

7.5 UNIT END QUESTIONS

1) What is the use of Boot Block?
2) What is Sector Sparing?
3) Explain Direct Memory Access?
4) Explain Programmed I/O in detail.
5) Explain Device-Independent I/O Software.

*****

8

I/O DEVICES

Unit structure

8.0 OBJECTIVES

- To understand the Disk concept
- To learn CLOCKS Concept
- To learn different, USER INTERFACES
- To understand THIN CLIENTS and Power management

In addition to providing abstractions such as processes, address spaces, and files, an operating system also controls all the computer's I/O (Input/Output) devices. It must issue commands to the devices, catch interrupts, and handle errors. It should also provide an interface between the devices and the rest of the system that is simple and easy to use. To the extent possible, the interface should be the same for all devices (device independence). The I/O code represents a significant fraction of the total operating system. How the operating system manages I/O is the subject of this chapter.

This chapter is organized as follows. We will look first at some of the principles of I/O hardware and then at I/O software in general. I/O software can be structured in layers, with each having a well-defined task. We will look at these layers to see what they do and how they fit together. Next, we will look at several I/O devices in detail: disks, clocks, keyboards, and displays. For each device we will look at its hardware and software. Finally, we will consider power management.

8.1 INTRODUCTION

We will begin with disks, which are conceptually simple, yet very important. After that we will examine clocks, keyboards, and displays.

8.1.1 Disk Hardware:

Disks come in a variety of types. The most common ones are the magnetic hard disks. They are characterized by the fact that reads and writes are equally fast, which makes them suitable as secondary memory (paging, file systems, etc.). Arrays of these disks are sometimes used to provide highly reliable storage. For distribution of programs, data, and movies, optical disks (DVDs and Blu-ray) are also important. Finally, solid-state disks are increasingly popular as they are fast and do not contain moving parts. In the following sections we will discuss magnetic disks as an example of the hardware and then describe the software for disk devices in general.

Magnetic Disks:

Magnetic disks are organized into cylinders, each one containing as many tracks as there are heads stacked vertically. The tracks are divided into sectors, with the number of sectors around the circumference typically being 8 to 32 on floppy disks, and up to several hundred on hard disks. The number of heads varies from 1 to about 16.

Older disks have little electronics and just deliver a simple serial bit stream. On these disks, the controller does most of the work. On other disks, in particular, **IDE** (**Integrated Drive Electronics**) and **SATA** (**Serial ATA**) disks, the disk drive itself contains a microcontroller that does considerable work and allows the real controller to issue a set of higher-level commands. The controller often does track caching, bad-block remapping, and much more.

A device feature that has important implications for the disk driver is the possibility of a controller doing seeks on two or more drives at the same time. These are known as **overlapped seeks**. While the controller and software are waiting for a seek to complete on one drive, the controller can initiate a seek on another drive. Many controllers can also read or write on one drive while seeking on one or more other drives, but a floppy disk controller cannot read or write on two drives at the same time. (Reading or writing requires the controller to move bits on a microsecond time scale, so one transfer uses up most of its computing power.) The situation is different for hard disks with integrated controllers, and in a system with more than one of these hard drives they can operate simultaneously, at least to the extent of transferring between the disk and the controller's buffer memory. Only one transfer between the controller and the main memory is possible at once, however. The ability to perform two or more operations at the same time can reduce the average access time considerably.

Figure 8.1 compares parameters of the standard storage medium for the original IBM PC with parameters of a disk made three decades later to show how much disks changed in that time. It is interesting to note

that not all parameters have improved as much. Average seek time is almost 9 times better than it was, transfer rate is 16,000 times better, while capacity is up by a factor of 800,000. This pattern has to do with relatively gradual improvements in the moving parts, but much higher bit densities on the recording surfaces.

Parameter	IBM 360-KB floppy disk	WD 3000 HLFS hard disk
Number of cylinders	40	36,481
Tracks per cylinder	2	255
Sectors per track	9	63 (avg)
Sectors per disk	720	586,072,368
Bytes per sector	512	512
Disk capacity	360 KB	300 GB
Seek time (adjacent cylinders)	6 msec	0.7 msec
Seek time (average case)	77 msec	4.2 msec
Rotation time	200 msec	6 msec
Time to transfer 1 sector	22 msec	1.4 μsec

Fig 8.1 compares parameters

One thing to be aware of in looking at the specifications of modern hard disks is that the geometry specified, and used by the driver software, is almost always different from the physical format. On old disks, the number of sectors per track was the same for all cylinders. Modern disks are divided into zones with more sectors on the outer zones than the inner ones.

RAID:

CPU performance has been increasing exponentially over the past decade, roughly doubling every 18 months. Not so with disk performance. In the 1970s, average seek times on minicomputer disks were 50 to 100 msec. Now seek times are still a few msec. In most technical industries (say, automobiles or aviation), a factor of 5 to 10 performance improvement in two decades would be major news (imagine 300-MPG cars), but in the computer industry it is an embarrassment.

Thus the gap between CPU performance and (hard) disk performance has become much larger over time. Can anything be done to help?

Yes! As we have seen, parallel processing is increasingly being used to speed up CPU performance. It has occurred to various people over the years that parallel I/O might be a good idea, too. In their 1988 paper, Patterson et al. suggested six specific disk organizations that could be used to improve disk performance, reliability, or both (Patterson et al., 1988).

These ideas were quickly adopted by industry and have led to a new class of I/O device called a **RAID**. Patterson et al.

defined RAID as **Redundant Array of Inexpensive Disks**, but industry redefined the I to be ‘‘Independent’’ rather than ‘‘Inexpensive’’ (maybe so they could charge more?). Since a villain was also needed (as in RISC vs. CISC, also due to Patterson), the bad guy here was the **SLED** (**Single Large Expensive Disk**).

The fundamental idea behind a RAID is to install a box full of disks next to the computer, typically a large server, replace the disk controller card with a RAID controller, copy the data over to the RAID, and then continue normal operation. In other words, a RAID should look like a SLED to the operating system but have better performance and better reliability. In the past, RAIDs consisted almost exclusively of a RAID SCSI controller plus a box of SCSI disks, because the performance was good and modern SCSI supports up to 15 disks on a single controller. Now a days, many manufacturers also offer (less expensive) RAIDs based on SATA. In this way, no software changes are required to use the RAID, a big selling point for many system administrators.

In addition to appearing like a single disk to the software, all RAIDs have the property that the data are distributed over the drives, to allow parallel operation. Several different schemes for doing this were defined by Patterson et al. Nowadays, most manufacturers refer to the seven standard configurations as RAID level 0 through RAID level 6. In addition, there are a few other minor levels that we will not discuss. The term ‘‘level’’ is something of a misnomer since no hierarchy is involved; there are simply seven different organizations possible.

8.1.2 Disk Formatting:

A hard disk consists of a stack of aluminum, alloy, or glass platters typically 3.5 inch in diameter (or 2.5 inch on notebook computers). On each platter is deposited a thin magnetizable metal oxide. After manufacturing, there is no information whatsoever on the disk. Before the disk can be used, each platter must receive a **low-level format** done by software. The format consists of a series of concentric tracks, each containing some number of sectors, with short gaps between the sectors. The format of a sector is shown in Fig. 8.2

Fig 8.2

The preamble starts with a certain bit pattern that allows the hardware to recognize the start of the sector. It also contains the cylinder and sector numbers and some other information. The size of the data

portion is determined by the low level formatting program. Most disks use 512-byte sectors. The ECC field contains redundant information that can be used to recover from read errors. The size and content of this field varies from manufacturer to manufacturer, depending on how much disk space the designer is willing to give up for higher reliability and how complex an ECC code the controller can handle. A 16-byte ECC field is not unusual.

Furthermore, all hard disks have some number of spare sectors allocated to be used to replace sectors with a manufacturing defect.

The position of sector 0 on each track is offset from the previous track when the low-level format is laid down. This offset, called **cylinder skew**, is done to improve performance. The idea is to allow the disk to read multiple tracks in one continuous operation without losing data.

8.1.3 Disk Arm Scheduling Algorithms:

In this section we will look at some issues related to disk drivers in general.

First, consider how long it takes to read or write a disk block. The time required is determined by three factors:

1. Seek time (the time to move the arm to the proper cylinder).
2. Rotational delay (how long for the proper sector to appear under the reading head).
3. Actual data transfer time.

If the disk driver accepts requests one at a time and carries them out in that order, that is, **FCFS** (**First-Come, First-Served**), little can be done to optimize seek time. However, another strategy is possible when the disk is heavily loaded. It is likely that while the arm is seeking on behalf of one request, other disk requests may be generated by other processes. Many disk drivers maintain a table, indexed by cylinder number, with all the pending requests for each cylinder chained together in a linked list headed by the table entries.

Given this kind of data structure, we can improve upon the first-come, first servedscheduling algorithm. To see how, consider an imaginary disk with 40 cylinders. A request comes in to read a block on cylinder 11. While the seek to cylinder 11 is in progress, new requests come in for cylinders 1, 36, 16, 34, 9, and 12, in that order.

They are entered into the table of pending requests, with a separate linked list for each cylinder. The requests are shown in Fig. 8.3 . When the current request (for cylinder 11) is finished, the disk driver has a choice of which request to handle next. Using FCFS, it would go next to cylinder 1,

then to 36, and so on. This algorithm would require arm motions of 10, 35, 20, 18, 25, and 3, respectively, for a total of 111 cylinders.

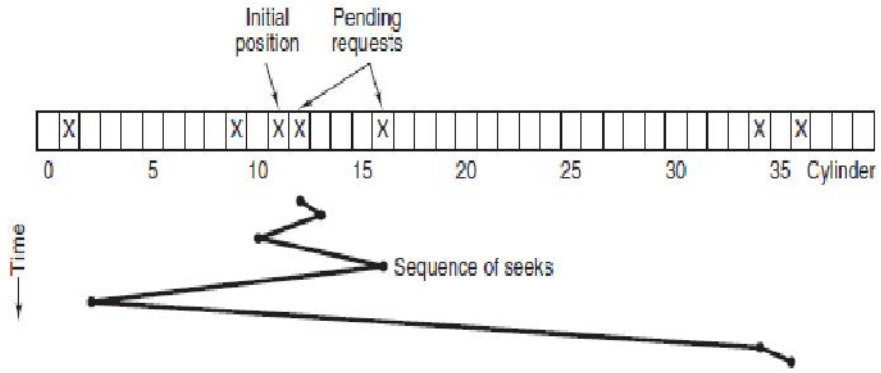


Fig 8.3

8.1.4 Error Handling:

Disk manufacturers are constantly pushing the limits of the technology by increasing linear bit densities. A track midway out on a 5.25-inch disk has a circumference of about 300 mm. If the track holds 300 sectors of 512 bytes, the linear recording density may be about 5000 bits/mm taking into account the fact that some space is lost to preambles, ECCs, and intersector gaps. Recording 5000 bits/mm requires an extremely uniform substrate and a very fine oxide coating. Unfortunately, it is not possible to manufacture a disk to such specifications without defects. As soon as manufacturing technology has improved to the point where it is possible to operate flawlessly at such densities, disk designers will go to higher densities to increase the capacity. Doing so will probably reintroduce defects.

Manufacturing defects introduce bad sectors, that is, sectors that do not correctly read back the value just written to them. If the defect is very small, say, only a few bits, it is possible to use the bad sector and just let the ECC correct the errors every time. If the defect is bigger, the error cannot be masked

There are two general approaches to bad blocks: deal with them in the controller or deal with them in the operating system. In the former approach, before the disk is shipped from the factory, it is tested and a list of bad sectors is written onto the disk. For each bad sector, one of the spares is substituted for it.

There are two ways to do this substitution. In Fig. 8.4 (a) we see a single disk track with 30 data sectors and two spares. Sector 7 is defective. What the controller can do is remap one of the spares as sector 7 as shown in Fig. 8.4(b). The other way is to shift all the sectors up one, as shown in Fig. 8.4 (c). In both cases the controller has to know which sector is which. It can keep track of this information through internal tables (one per track) or by rewriting the preambles to give the remapped sector numbers. If the preambles are rewritten, the method of Fig. 8.4(c) is more

work (because 23 preambles must be rewritten) but ultimately gives better performance because an entire track can still be read in one rotation.

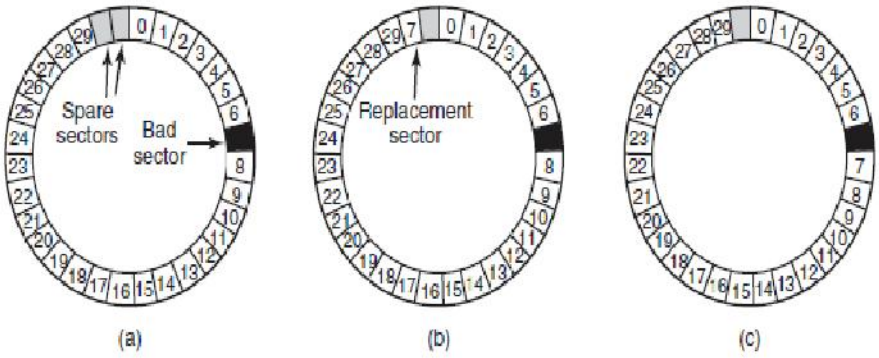


Fig 8.4

Errors can also develop during normal operation after the drive has been installed.

The first line of defense upon getting an error that the ECC cannot handle is to just try the read again. Some read errors are transient, that is, are caused by specks of dust under the head and will go away on a second attempt. If the controller notices that it is getting repeated errors on a certain sector, it can switch to a spare before the sector has died completely. In this way, no data are lost and the operating system and user do not even notice the problem. Usually, the method of Fig. 8.4(b) has to be used since the other sectors might now contain data. Using the method of Fig. 8.4(c) would require not only rewriting the preambles, but copying all the data as well.

8.1.5 Stable Storage:

As we have seen, disks sometimes make errors. Good sectors can suddenly become bad sectors. Whole drives can die unexpectedly. RAIDs protect against a few sectors going bad or even a drive falling out. However, they do not protect against write errors laying down bad data in the first place. They also do not protect against crashes during writes corrupting the original data without replacing them by newer data.

For some applications, it is essential that data never be lost or corrupted, even in the face of disk and CPU errors. Ideally, a disk should simply work all the time with no errors. Unfortunately, that is not achievable. What is achievable is a disk subsystem that has the following property: when a write is issued to it, the disk either correctly writes the data or it does nothing, leaving the existing data intact.

Such a system is called **stable storage** and is implemented in software (Lampson and Sturgis, 1979). The goal is to keep the disk consistent at all costs. Below we will describe a slight variant of the original idea.

Before describing the algorithm, it is important to have a clear model of the possible errors. The model assumes that when a disk writes a block (one or more sectors), either the write is correct or it is incorrect and this error can be detected on a subsequent read by examining the values of the ECC fields. In principle, guaranteed error detection is never possible because with a, say, 16-byte ECC field guarding a 512-byte sector, there are 24096 data values and only 2144 ECC values. Thus if a block is garbled during writing but the ECC is not, there are billions upon billions of incorrect combinations that yield the same ECC. If any of them occur, the error will not be detected. On the whole, the probability of random data having the proper 16- byte ECC is about 2−144, which is small enough that we will call it zero, even though it is really not.

The model also assumes that a correctly written sector can spontaneously go bad and become unreadable. However, the assumption is that such events are so rare that having the same sector go bad on a second (independent) drive during a reasonable time interval (e.g., 1 day) is small enough to ignore.

The model also assumes the CPU can fail, in which case it just stops. Any disk write in progress at the moment of failure also stops, leading to incorrect data in one sector and an incorrect ECC that can later be detected. Under all these conditions, stable storage can be made 100% reliable in the sense of writes either working correctly or leaving the old data in place. Of course, it does not protect against physical disasters, such as an earthquake happening and the computer falling 100 meters into a fissure and landing in a pool of boiling magma. It is tough to recover from this condition in software.

Stable storage uses a pair of identical disks with the corresponding blocks working together to form one error-free block. In the absence of errors, the corresponding blocks on both drives are the same. Either one can be read to get the same result. To achieve this goal, the following three operations are defined:

1. **Stable writes**. A stable write consists of first writing the block on drive 1, then reading it back to verify that it was written correctly. If it was not, the write and reread are done again up to *n* times until they work. After *n* consecutive failures, the block is remapped onto a spare and the operation repeated until it succeeds, no matter how many spares have to be tried. After the write to drive 1 has succeeded, the corresponding block on drive 2 is written and reread, repeatedly if need be, until it, too, finally succeeds. In the absence of CPU crashes, when a stable write completes, the block has correctly been written onto both drives and verified on both of them.

2. Stable reads: A stable read first reads the block from drive 1. If this yields an incorrect ECC, the read is tried again, up to *n* times. If all of these give bad ECCs, the corresponding block is read from drive 2. Given the fact that a successful stable writeleaves two good copies of the block

behind, and our assumption that the probability of the same block spontaneously going bad on both drives in a reasonable time interval is negligible, a stable read always succeeds.

3. Crash recovery: After a crash, a recovery program scans both disks comparing corresponding blocks. If a pairof blocks are both good and the same, nothing is done. If one of them has an ECC error, the bad block is overwritten with the corresponding good block. If a pairof blocks are both good but different, the block from drive 1 is written onto drive 2. In the absence of CPU crashes, this scheme always works because stable writes always write two valid copies of every block and spontaneous errors are assumed never to occur on both corresponding blocks at the same time. What about in the presence of CPU crashes during stable writes? It depends on precisely when the crash occurs. There are five possibilities, as depicted in Fig. 8.5

In the absence of CPU crashes, this scheme always works because stable writes always write two valid copies of every block and spontaneous errors are assumed never to occur on both corresponding blocks at the same time. What about in the presence of CPU crashes during stable writes? It depends on precisely when the crash occurs. There are five possibilities, as depicted in Fig. 8.5

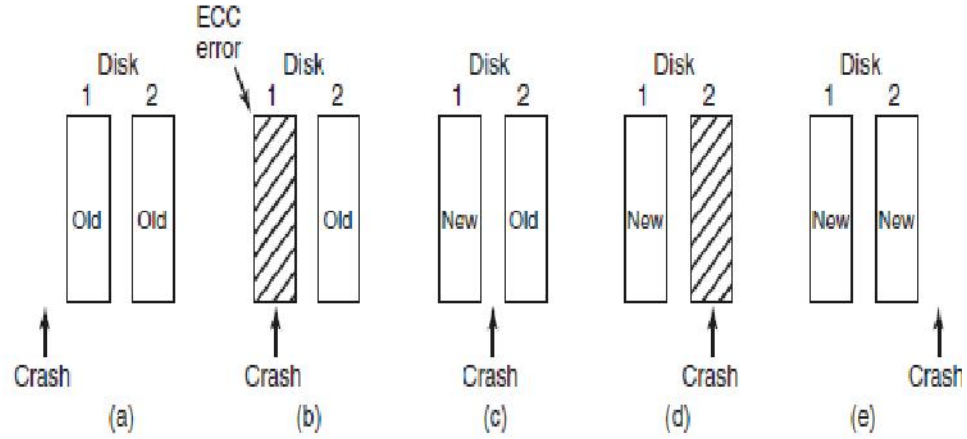


Fig 8.5

8.2 CLOCKS

Clocks (also called **timers**) are essential to the operation of any multiprogrammed system for a variety of reasons. They maintain the time of day and prevent one process from monopolizing the CPU, among other things. The clock software can take the form of a device driver, even though a clock is neither a block device, like a disk, nor a character device, like a mouse. Our examination of clocks will follow the same pattern as in the previous section: first a look at clock hardware and then a look at the clock software.

8.2.1 Clock Hardware:

Two types of 4 are commonly used in computers, and both are quite different from the clocks and watches used by people. The simpler clocks are tied to the 110- or 220-volt power line and cause an interrupt on every voltage cycle, at 50 or 60 Hz. These clocks used to dominate, but are rare nowadays.

When a piece of quartz crystal is properly cut and mounted under tension, it can be made to generate a periodic signal of very great accuracy, typically in the range of several hundred megahertz to a few gigahertz, depending on the crystal chosen. Using electronics, this base signal can be multiplied by a small integer to get frequencies up to several gigahertz or even more. At least one such circuit is usually found in any computer, providing a synchronizing signal to the computer's various circuits. This signal is fed into the counter to make it count down to zero. When the counter gets to zero, it causes a CPU interrupt.

Programmable clocks typically have several modes of operation. In **one-shot mode**, when the clock is started, it copies the value of the holding register into the counter and then decrements the counter at each pulse from the crystal. When the counter gets to zero, it causes an interrupt and stops until it is explicitly started again by the software. In **square-wave mode**, after getting to zero and causing the interrupt, the holding register is automatically copied into the counter, and the whole process is repeated again indefinitely. These periodic interrupts are called **clock ticks**

8.2.2 Clock Software:

All the clock hardware does is generate interrupts at known intervals. Everything else involving time must be done by the software, the clock driver. The exact duties of the clock driver vary among operating systems, but usually include most of the following:

1. Maintaining the time of day.
2. Preventing processes from running longer than they are allowed to.
3. Accounting for CPU usage.
4. Handling the alarm system call made by user processes.
5. Providing watchdog timers for parts of the system itself.
6. Doing profiling, monitoring, and statistics gathering.
   The first clock function, maintaining the time of day (also called the **real time**) is not difficult.

8.3 USER INTERFACES: KEYBOARD, MOUSE, MONITOR

Every general-purpose computer has a keyboard and monitor (and sometimes a mouse) to allow people to interact with it. Although the keyboard and monitor are technically separate devices, they work closely

together. On mainframes, there are frequently many remote users, each with a device containing a keyboard and an attached display as a unit. These devices have historically been called **terminals**.

People frequently still use that term, even when discussing personal computer keyboards and monitors (mostly for lack of a better term).

8.3.1 Input Software:

User input comes primarily from the keyboard and mouse (or sometimes touch screens), so let us look at those. On a personal computer, the keyboard contains an embedded microprocessor which usually communicates through a specialized serial port with a controller chip on the parentboard (although increasingly keyboards are connected to a USB port). An interrupt is generated whenever a key is struck and a second one is generated whenever a key is released. At each of these keyboard interrupts, the keyboard driver extracts the information about what happens from the I/O port associated with the keyboard. Everything else happens in software and is pretty much independent of the hardware.

Most of the rest of this section can be best understood when thinking of typing commands to a shell window (command-line interface). This is how programmers commonly work. We will discuss graphical interfaces below. Some devices, in particular touch screens, are used for input *and* output. We have made an (arbitrary) choice to discuss them in the section on output devices. We will discuss graphical interfaces later in this chapter.

Keyboard Software:

The number in the I/O register is the key number, called the **scan code**, not the ASCII code. Normal keyboards have fewer than 128 keys, so only 7 bits are needed to represent the key number. The eighth bit is set to 0 on a key press and to 1 on a key release. It is up to the driver to keep track of the status of each key (up or down). So all the hardware does is give press and release interrupts. Software does the rest.

When the *A* key is struck, for example, the scan code (30) is put in an I/O register. It is up to the driver to determine whether it is lowercase, uppercase, CTRLA, ALT-A, CTRL-ALT-A, or some other combination. Since the driver can tell which keys have been struck but not yet released (e.g., SHIFT), it has enough information to do the job.

For example, the key sequence DEPRESS SHIFT, DEPRESS A, RELEASE A, RELEASE SHIFT indicates an uppercase A. However, the key sequence DEPRESS SHIFT, DEPRESS A, RELEASE SHIFT, RELEASE A also indicates an uppercase A. Although this keyboard interface puts the full burden on the software, it is extremely flexible. For

example, user programs may be interested in whether a digit just typed came from the top row of keys or the numeric keypad on the side. In principle, the driver can provide this information.

Two possible philosophies can be adopted for the driver. In the first one, the driver's job is just to accept input and pass it upward unmodified. A program reading from the keyboard gets a raw sequence of ASCII codes. (Giving user programs the scan codes is too primitive, as well as being highly keyboard dependent.) This philosophy is well suited to the needs of sophisticated screen editors such as *emacs*, which allow the user to bind an arbitrary action to any character or sequence of characters. It does, however, mean that if the user types *dste* instead of *date* and then corrects the error by typing three backspaces and *ate*, followed by a carriage return, the user program will be given all 11 ASCII codes typed, as follows:

d s t e← ← ←a t e CR

Not all programs want this much detail. Often they just want the corrected input, not the exact sequence of how it was produced. This observation leads to the second philosophy: the driver handles all the intraline editing and just delivers corrected lines to the user programs. The first philosophy is character oriented; the second one is line oriented. Originally they were referred to as **raw mode** and **cooked mode**, respectively.

Mouse Software:

Most PCs have a mouse, or sometimes a trackball, which is just a mouse lying on its back. One common type of mouse has a rubber ball inside that protrudes through a hole in the bottom and rotates as the mouse is moved over a rough surface. As the ball rotates, it rubs against rubber rollers placed on orthogonal shafts. Motion in the east-west direction causes the shaft parallel to the *y*-axis to rotate; motion in the north-south direction causes the shaft parallel to the *x*-axis to rotate. Another popular type is the optical mouse, which is equipped with one or more light-emitting diodes and photodetectors on the bottom. Early ones had to operate on a special mousepad with a rectangular grid etched onto it so the mouse could count lines crossed. Modern optical mice have an image-processing chip in them and make continuous low-resolution photos of the surface under them, looking for changes from image to image. Whenever a mouse has moved a certain minimum distance in either direction or a button is depressed or released, a message is sent to the computer. The minimum distance is about 0.1 mm (although it can be set in software). Some people call this unit a **mickey**. Mice (or occasionally, mouses) can have one, two, or three buttons, depending on the designers' estimate of the users' intellectual ability to keep track of more than one button. Some mice have wheels that can send additional data back to the computer. Wireless mice are the same as wired mice except that instead of sending their data back to the computer over a wire, they use low-power radios, for example, using the **Bluetooth** standard.

8.3.2 Output Software:

Now let us consider output software. First we will look at simple output to a text window, which is what programmers normally prefer to use. Then we will consider graphical user interfaces, which other users often prefer.

Text Windows:

Output is simpler than input when the output is sequentially in a single font, size, and color. For the most part, the program sends characters to the current window and they are displayed there. Usually, a block of characters, for example, a line, is written
in one system call.

Screen editors and many other sophisticated programs need to be able to update the screen in complex ways such as replacing one line in the middle of the screen. To accommodate this need, most output drivers support a series of commands to move the cursor, insert and delete characters or lines at the cursor, and so on. These commands are often called **escape sequences**. In the heyday of the dumb 25 × 80 ASCII terminal, there were hundreds of terminal types, each with its own escape sequences. As a consequence, it was difficult to write software that worked on more than one terminal type.

One solution, which was introduced in Berkeley UNIX, was a terminal database called **termcap**. This software package defined a number of basic actions, such as moving the cursor to (*row*, *column*). To move the cursor to a particular location, the software, say, an editor, used a generic escape sequence which was then converted to the actual escape sequence for the terminal being written to. In this way, the editor worked on any terminal that had an entry in the termcap database.

Much UNIX software still works this way, even on personal computers. Eventually, the industry saw the need for standardizing the escape sequence, so an ANSI standard was developed.

8.4 THIN CLIENTS

Over the years, the main computing paradigm has oscillated between centralized and decentralized computing. The first computers, such as the ENIAC, were, in fact, personal computers, albeit large ones, because only one person could use one at once. Then came time sharing systems, in which many remote users at simple terminals shared a big central computer. Next came the PC era, in which the users had their own personal computers again.

While the decentralized PC model has advantages, it also has some severe disadvantages that are only beginning to be taken seriously. Probably the biggest problem is that each PC has a large hard disk and complex software that must be maintained. For example, when a new

release of the operating system comes out, aa great deal of work has to be done to perform the upgrade on each machine separately.

At most corporations, the labor costs of doing this kind of software maintenance dwarf the actual hardware and software costs. For home users, the labor is technically free, but few people are capable of doing it correctly and fewer still enjoy doing it. With a centralized system, only one or a few machines have to be updated and those machines have a staff of experts to do the work. A related issue is that users should make regular backups of their gigabyte file systems, but few of them do. When disaster strikes, a great deal of moaning and wringing of hands tends to follow. With a centralized system, backups can be made every night by automated tape robots.

Another advantage is that resource sharing is easier with centralized systems. A system with 256 remote users, each with 256 MB of RAM, will have most of that RAM idle most of the time. With a centralized system with 64 GB of RAM, it never happens that some user temporarily needs a lot of RAM but cannot get it because it is on someone else's PC. The same argument holds for disk space and other resources. Finally, we are starting to see a shift from PC-centric computing to Web Centric computing. One area where this shift is very far along is email. People used to get their email delivered to their home machine and read it there. Nowadays, many people log into Gmail, Hotmail, or Yahoo and read their mail there. The next step is for people to log into other Websites to do word processing, build spreadsheets, and other things that used to require PC software. It is even possible that eventually the only software people run on their PC is a Web browser, and maybe not even that. It is probably a fair conclusion to say that most users want high-performance interactive computing but do not really want to administer a computer. This has led researchers to reexamine timesharing using dumb terminals (now politely called **thin clients**) that meet modern terminal expectations.

8.5 POWER MANAGEMENT

The first general-purpose electronic computer, the ENIAC, had 18,000 vacuum tubes and consumed 140,000 watts of power. As a result, it ran up a nontrivial electricity bill. After the invention of the transistor, power usage dropped dramatically and the computer industry lost interest in power requirements. However, nowadays power management is back in the spotlight for several reasons, and the operating system is playing a role here.

Let us start with desktop PCs. A desktop PC often has a 200-watt power supply (which is typically 85% efficient, that is, loses 15% of the incoming energy to heat). If 100 million of these machines are turned on at once worldwide, together they use 20,000 megawatts of electricity. This is the total output of 20 average-sized nuclear power plants. If power requirements could be cut in half, we could get rid of 10 nuclear power plants. From an environmental point of view, getting rid of 10 nuclear

power plants (or an equivalent number of fossil-fuel plants) is a big win and well worth pursuing.

The other place where power is a big issue is on battery-powered computers, including notebooks, handhelds, and Webpads, among others. The heart of the problem is that the batteries cannot hold enough charge to last very long, a few hours at most. Furthermore, despite massive research efforts by battery companies, computer companies, and consumer electronics companies, progress is glacial. To an industry used to a doubling of performance every 18 months (Moore's law), having no progress at all seems like a violation of the laws of physics, but that is the current situation. As a consequence, making computers use less energy so existing batteries last longer is high on everyone's agenda. The operating system plays a major role here, as we will see below.

At the lowest level, hardware vendors are trying to make their electronics more energy efficient. Techniques used include reducing transistor size, employing dynamic voltage scaling, using low-swing and adiabatic buses, and similar techniques.

These are outside the scope of this book, but interested readers can find a good survey in a paper by Venkatachalam and Franz (2005). There are two general approaches to reducing energy consumption. The first one is for the operating system to turn off parts of the computer (mostly I/O devices) when they are not in use because a device that is off uses little or no energy. The second one is for the application program to use less energy, possibly degrading the quality of the user experience, in order to stretch out battery time. We will look at each of these approaches in turn, but first we will say a little bit about hardware design with respect to power usage.

8.6 SUMMARY

While using memory mapped IO, the OS allocates a buffer in memory and informs I/O device to use that buffer to send data to the CPU. I/O device operates asynchronously with CPU, interrupts CPU when finished. Memory mapped IO is used for most high-speed I/O devices like disks, communication interfaces.

8.7 UNIT END QUESTIONS

1) What problems could occur if a system allowed a file system to be mounted simultaneously at more than one location?
2) What criteria should be used in deciding which strategy is best utilized for a particular file?
3) What is meant by RAID?
4) What are the various Disk-Scheduling Algorithms?
5) What is Low-Level Formatting?
6) Explain POWER MANAGEMENT in details
7) Explain Thin Clients in detail.

*****

9

DEADLOCKS

Unit Structure

9.0 OBJECTIVES

- To understand what is a Deadlock
- learn how to make Resource acquisition
- learn different methods Detecting Deadlocks and Recovering
- To understand The Ostrich Algorithm

A **deadlock** occurs when every member of a set of processes is waiting for an event that can only be caused by a member of the set.

Often the event waited for is the release of a resource. In the automotive world deadlocks are called *gridlocks*.

The processes are the cars. The resources are the spaces occupied by the cars

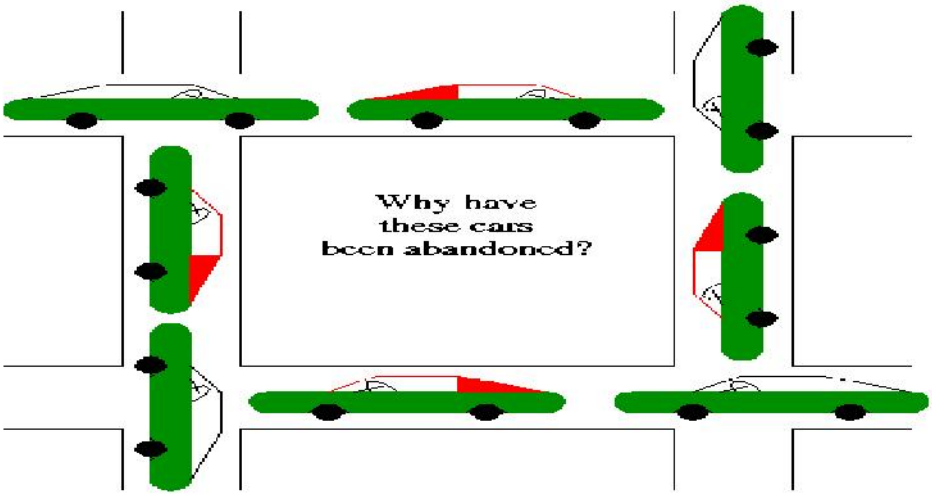


Figure 9.a Deadlock

For a computer science example consider two processes A and B that each want to print a file currently on tape.

1. A has obtained ownership of the printer and will release it after printing one file.
2. B has obtained ownership of the tape drive and will release it after reading one file.
3. A tries to get ownership of the tape drive, but is told to wait for B to release it.
4. B tries to get ownership of the printer, but is told to wait for A to release the printer.

9.1 RESOURCES

The **resource** is the object granted to a process.

9.1.1: Preemptable and Non-preemptable Resources

- Resources come in two types

1. Preemptable, meaning that the resource can be taken away from its current owner (and given back later). An example is memory.
2. Non-preemptable, meaning that the resource cannot be taken away. An example is a printer.

- The interesting issues arise with non-preemptable resources so those are the ones we study.

- Life history of a resource is a sequence of
  1. Request
  2. Allocate
  3. Use
  4. Release

- Processes make requests, use the resource, and release the resource. The allocation decisions are made by the system and we will study policies used to make these decisions.

Simple example of the trouble you can get into.

Two resources and two processes.

- Each process wants both resources.
- Use a semaphore for each. Call them S and T.
- If both processes execute P(S); P(T); --- V(T); V(S) all is well.
- But if one executes instead P(T); P(S); -- V(S); V(T) disaster! This was the printer/tape example just above

Recall from the semaphore/critical-section treatment, that it is easy to cause trouble if a process dies or stays forever inside its critical section. Similarly, we assume that no process maintains a resource forever. It may

obtain the resource an unbounded number of times (i.e. it can have a loop forever with a resource request inside), but each time it gets the resource, it must release it eventually.

9.2 INTRODUCTION TO DEADLOCKS

To repeat: A **deadlock** occurs when every member of a set of processes is waiting for an event that can only be caused by a member of the set.

Often the event waited for is the release of a resource.

9.2.1: (Necessary) Conditions for Deadlock

The following four conditions (Coffman; Havender) are *necessary* but *not* sufficient for deadlock. Repeat: They are **not** sufficient.

1. **Mutual exclusion:** A resource can be assigned to at most one process at a time (no sharing).
2. **Hold and wait:** A processing holding a resource is permitted to request another.
3. **No preemption:** A process must release its resources; they cannot be taken away.
4. **Circular wait:** There must be a chain of processes such that each member of the chain is waiting for a resource held by the next member of the chain.

The first three are characteristics of the system and resources. That is, for a given system with a fixed set of resources, the first three conditions are either true or false: They don't change with time. The truth or falsehood of the last condition does indeed change with time as the resources are equested/allocated/released.

9.2.2: Deadlock Modeling:

Following are several examples of a **Resource Allocation Graph**, also called a **Reusable Resource Graph**.

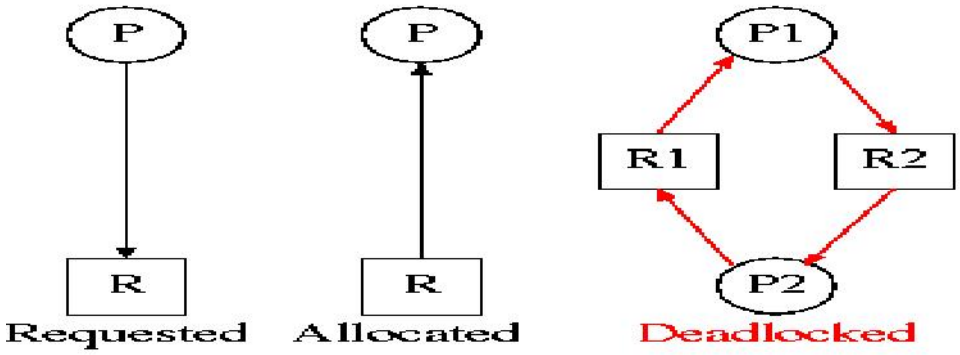


Figure 9.1 Resource Allocation Graph

- The processes are circles.
- The resources are squares.
- An arc (directed line) from a process P to a resource R signifies that process P has requested (but not yet been allocated) resource R.
- An arc from a resource R to a process P indicates that process P has been allocated resource R.

There are four strategies used for dealing with deadlocks

1. Ignore the problem
2. Detect deadlocks and recover from them
3. Avoid deadlocks by carefully deciding when to allocate resources.
4. Prevent deadlocks by violating one of the 4 necessary conditions.

9.3 IGNORING THE PROBLEM--THE OSTRICH ALGORITHM

The “put your head in the sand approach”.

- If the likelihood of a deadlock is sufficiently small and the cost of avoiding a deadlock is sufficiently high it might be better to ignore the problem.
- For example if each PC deadlocks once per 100 years, the one reboot may be less painful than the restrictions needed to prevent it.
- Clearly not a good philosophy for nuclear missile launchers.
- For embedded systems (e.g., missile launchers) the programs run are fixed in advance so many of the questions Tanenbaum raises (such as many processes wanting to fork at the same time) don't occur.

9.4 DETECTING DEADLOCKS AND RECOVERING

9.4.1Detecting Deadlocks with Single Unit Resources

- Consider the case in which there is **only one** instance of each resource.
- Thus a request can be satisfied by only one specific resource.
- In **this case** the 4 necessary conditions for deadlock are also sufficient.
- Remember we are making an assumption (single unit resources) that is often invalid. For example, many systems have several printers and a request is given for “a printer” not a specific printer. Similarly, one can have many tape drives.
- So the problem comes down to finding a directed cycle in the resource allocation graph. Why?
- Answer: Because the other three conditions are either satisfied by the system we are studying or are not in which case deadlock is not a

question. That is, conditions 1,2,3 are conditions on the system in general not on what is happening right now.

To find a directed cycle in a directed graph is not hard. The idea is simple.

1. For each node in the graph do a depth first traversal to see if the graph is a DAG(directed acyclic graph), building a list as you go down the DAG (and pruning it as you backtrack back up).
2. If you ever find the same node twice on your list, you have found a directed cycle, the graph is not a DAG, and deadlock exists among the processes in your current list.
3. If you never find the same node twice, the graph is a DAG and no deadlock occurs.
4. The searches are finite since there are a finite number of nodes.

9.4.2: Detecting Deadlocks with Multiple Unit Resources:

This is more difficult.

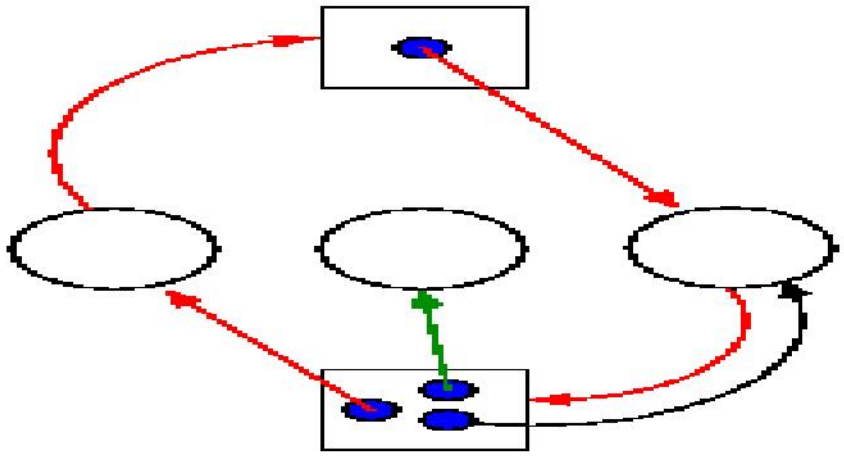


Figure 9.2 Resource allocation graph

- The figure on the above shows a resource allocation graph with multiple unit resources.
- Each unit is represented by a dot in the box.
- Request edges are drawn to the box since they represent a request for any dot in the box.
- Allocation edges are drawn from the dot to represent that this unit of the resource has been assigned (but all units of a resource are equivalent and the choice of which one to assign is arbitrary).
- Note that there is a directed cycle in red, but there is no deadlock. Indeed the middle process might finish, erasing the green arc and permitting the blue dot to satisfy the rightmost process.
- An algorithm for detecting deadlocks in this more general setting. The idea is as follows.

1. look for a process that might be able to terminate (i.e., all its request arcs can be satisfied).
2. If one is found pretend that it does terminate (erase all its arcs), and repeat step 1.
3. If any processes remain, they are deadlocked.

- The algorithm just given makes the most optimistic assumption about a running process: it will return all its resources and terminate normally. If we still find processes that remain blocked, they are deadlocked.

9.4.3 Recovery from deadlock:

Suppose that our deadlock detection algorithm has succeeded and detected a deadlock. What next? Some way is needed to recover and get the system going again.

In this section we will discuss various ways of recovering from deadlock.

Preemption:

In some cases it may be possible to temporarily take a resource away from its current owner and give it to another process

Perhaps you can temporarily preempt a resource from a process. Not likely.

For example, to take a laser printer away from its owner, the operator can collect all the sheets already printed and put them in a pile. Then the process can be suspended (marked as not runnable). At this point the printer can be assigned to another process. When that process finishes, the pile of printed sheets can be put back in the printer's output tray and the original process restarted.

Rollback:

If the system designers and machine operators know that deadlocks are likely, they can arrange to have processes checkpointed periodically. Checkpointing a process means that its state is written to a file so that it can be restarted later

Database (and other) systems take periodic checkpoints. If the system does take checkpoints, one can roll back to a checkpoint whenever a deadlock is detected.

Somehow must guarantee forward progress.

Kill processe:

The crudest but simplest way to break a deadlock is to kill one or more processes. One possibility is to kill a process in the cycle. With a little luck, the other processes will be able to continue. If this does not help, it can be repeated until the cycle is broken.

Can always be done but might be painful. For example some processes have had effects that can't be simply undone. Print, launch a missile, etc.

9.5 DEADLOCK AVOIDANCE

In the discussion of deadlock detection, we tacitly assumed that when a process asks for resources, it asks for them all at once (Figure 9.3 R Matrix). In most systems, however, resources are requested one at a time. The system must be able to decide whether granting a resource is safe or not and make the allocation only when it is safe. Thus, the question arises: Is there an algorithm that can always avoid deadlock bymaking the right choice all the time? The answer is a qualified yes—we can avoid deadlocks, but only if certain information is available in advance.

9.5.1 Resource Trajectories:

We plot progress of each process along an axis. In the example we show, there are two processes, hence two axes, i.e., planar. This procedure assumes that we know the entire request and release pattern of the processes *in advance* so it is not a practical solution. I present it as it is some motivation for the practical solution that follows, the Banker's Algorithm.

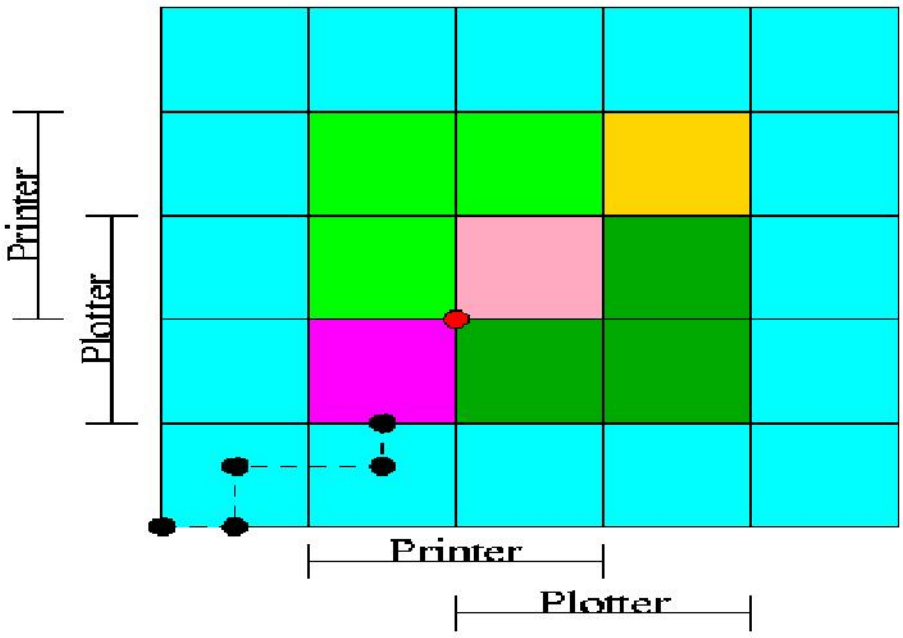


Figure 9.3 R Matrix

- We have two processes H (horizontal) and V.
- The origin represents them both starting.
- Their combined state is a point on the graph.
- The parts where the printer and plotter are needed by each process are indicated.
- The dark green is where both processes have the plotter and hence execution cannot reach this point.
- Light green represents both having the printer; also impossible.
- Pink is both having both a printer and plotter; impossible.
- Gold is possible (H has plotter, V has printer), but the system can't get there.
- The upper right corner is the goal; both processes have finished.
- The red dot is ... (cymbals) deadlock. We don't want to go there.
- The cyan is safe. From anywhere in the cyan we have horizontal and vertical moves to the finish point (the upper right corner) without hitting any impossible area.
- The magenta interior is very interesting. It is
- Possible: each processor has a different resource
- Not deadlocked: each processor can move within the magenta
- Deadly: deadlock is unavoidable. You will hit a magenta-green boundary
- and then will have no choice but to turn and go to the red dot.
- The cyan-magenta border is the danger zone.
- The dashed line represents a possible execution pattern.
- With a uniprocessor no diagonals are possible. We either move to the right meaning H is executing or move up indicating V is executing.
- The trajectory shown represents.
    1. H executing a little.
    2. V executing a little.
    3. H executes; requests the printer; gets it; executes some more.
    4. V executes; requests the plotter
- The crisis is at hand!
- If the resource manager gives V the plotter, the magenta has been entered and all is lost. “Abandon all hope ye who enter here” --Dante.
- The right thing to do is to deny the request, let H execute moving horizontally under the magenta and dark green. At the end of the dark green, no danger remains, both processes will complete successfully. Victory!

- This procedure is not practical for a general purpose OS since it requires knowing the programs in advance. That is, the resource manager, knows in advance what requests each process will make and in what order.

9.5.2: Safe States:

Avoiding deadlocks gives some extra knowledge.

- Not surprisingly, the resource manager knows how many units of each resource it had to begin with.
- Also it knows how many units of each resource it has given to each process.
- It would be great to see all the programs in advance and thus know all future requests, but that is asking for too much.
- Instead, when each process starts, it announces its maximum usage.
- That is each process, before making any resource requests, tells the resource manager the maximum number of units of each resource the process can possibly need.
- This is called the **claim** of the process.
  - If the claim is greater than the total number of units in the system the resource manager kills the process when receiving the claim (or returns an error code so that the process can make a new claim).
  - If during the run the process asks for more than its claim, the process is aborted (or an error code is returned and no resources are allocated).
  - If a process claims more than it needs, the result is that the resource manager will be more conservative than it needs to be and there will be more waiting.

Definition: A state is **safe** if there is an ordering of the processes such that: if the processes are run in this order, they will all terminate (assuming none exceeds its claim).

Recall the comparison made above between detecting deadlocks (with multi-unit) resources) and the banker's algorithm

- The deadlock detection algorithm given makes the most *optimistic* assumption about a running process: it will return all its resources and terminate normally. If we still find processes that remain blocked, they are deadlocked.
- The banker's algorithm makes the most *pessimistic* assumption about a running process: it immediately asks for all the resources it **can** (details later on “can”). If, even with such demanding processes, the resource manager can assure that all processesterminate, then we can assure that deadlock is avoided.

In the definition of a safe state no assumption is made about the running processes; that is, for a state to be safe termination must occur no matter what the processes do (providing the all terminate and to not exceed their claims). Making no assumption is the same as making the most pessimistic assumption.

Give an example of each of the four possibilities. A state that is

1. Safe and deadlocked--not possible.
2. Safe and not deadlocked--trivial (e.g., no arcs).
3. Not safe and deadlocked--easy (any deadlocked state).
4. Not safe and not deadlocked—interesting

Is the figure on the Below safe or not?

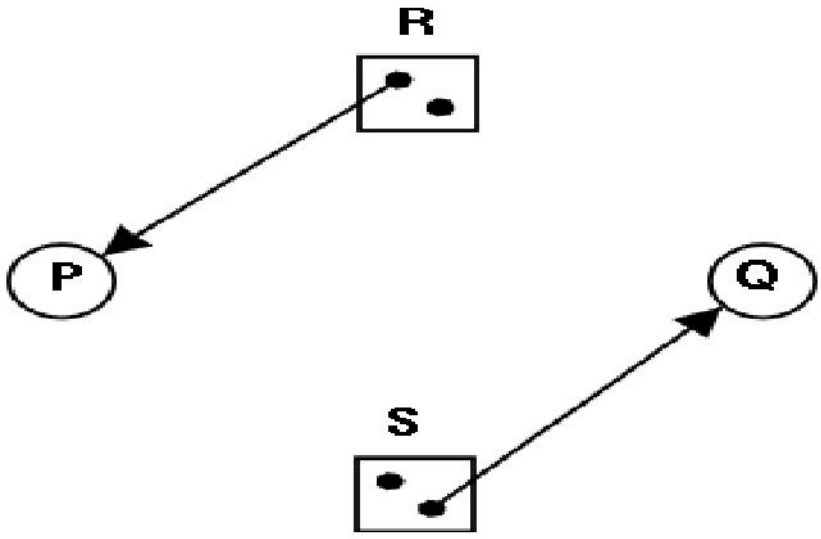


Figure 9.4 safe state

- You can **NOT** tell until I give you the initial claims of the process.
- Please do not make the unfortunately common exam mistake to give an example involving safe states without giving the claims.
- For the figure on the right, if the initial claims are: P: 1 unit of R and 2 units of S (written (1,2)) Q: 2 units of R and 1 units of S (written (2,1)) the state is **NOT** safe.
- But if the initial claims are instead: P: 2 units of R and 1 unit of S (written (2,1)) Q: 1 unit of R and 2 units of S (written (1,2)) the state **IS** safe.
- Explain why this is so.

A manager can determine if a state is safe.

- Since the manager knows all the claims, it can determine the maximum amount of additional resources each process can request.
- The manager knows how many units of each resource it has left.

The manager then follows the following procedure, which is part of **Banker's Algorithms** discovered by Dijkstra, to determine if the state is safe.

1. If there are no processes remaining, the state is **safe**.
2. Seek a process P whose max additional requests is less than what remains (for each resource type).
   - If no such process can be found, then the state is not safe.
   - The banker (manager) knows that if it refuses all requests except those from P, then it will be able to satisfy all of P's requests. Why? Ans: Look at how P was chosen.
3. The banker now pretends that P has terminated (since the banker knows that it can guarantee this will happen). Hence the banker pretends that all of P's currently held resources are returned. This makes the banker richer and hence perhaps a process that was not eligible to be chosen as P previously, can now be chosen.
4. Repeat these steps.

Example 1

process	initial claim	current alloc	max add'l
X	3	1	2
Y	11	5	6
Z	19	10	9
Total		16	
Available		6	

A safe state with 22 units of one resource

- One resource type R with 22 unit
- Three processes X, Y, and Z with initial claims 3, 11, and 19 respectively.
- Currently the processes have 1, 5, and 10 units respectively.
- Hence the manager currently has 6 units left.
- Also note that the max additional needs for the processes are 2, 6, 9 respectively.
- So the manager cannot assure (with its **current** remaining supply of 6 units) that Z can terminate. But that is **not** the question.
- This state is safe
  1. Use 2units to satisfy X; now the manager has 7 units.
  2. Use 6 units to satisfy Y; now the manager has 12 units.
  3. Use 9 units to satisfy Z; done

9.6 DEADLOCK PREVENTION

Attack one of the coffman/havender conditions.

9.6.1: Attacking Mutual Exclusion:

First let us attack the mutual exclusion condition. If no resource were ever assigned exclusively to a single process, we would never have deadlocks. For data, the simplest method is to make data read only, so that processes can use the data concurrently. However, it is equally clear that allowing two processes to write on the printer at the same time will lead to chaos. By spooling printer output, several processes can generate output at the same time. In this model, the only process that actually requests the physical printer is the printer daemon. Since the daemon never requests any other resources, we can eliminate deadlock for the printer. If the daemon is programmed to begin printing even before all the output is spooled, the printer might lie idle if an output process decides to wait several hours after the first burst of output. For this reason, daemons are normally programmed to print only after the complete output file is available. However, this decision itself could lead to deadlock. What would happen if two processes each filled up one half of the available spooling space with output and neither was finished producing its full output? In this case, we would have two processes that had each finished part, but not all, of their output, and could not continue. Neither process will ever finish, so we would have a deadlock on the disk.

9.6.2: Attacking Hold and Wait:

Require each process to request all resources at the beginning of the run. This is often called **One Shot**.

If we can prevent processes that hold resources from waiting for more resources, we can eliminate deadlocks. One way to achieve this goal is to require all processes to request all their resources before starting execution. If everything is available, the process will be allocated whatever it needs and can run to completion. If one or more resources are busy, nothing will be allocated and the process will just wait.

An immediate problem with this approach is that many processes do not know how many resources they will need until they have started running.

9.6.3: Attacking No Preempt:

If a process has been assigned the printer and is in the middle of printing its output, forcibly taking away the printer because a needed plotter is not available is tricky at best and impossible at worst. However, some resources can be virtualized to avoid this situation. Spooling printer output to the disk and allowing only the printer daemon access to the real

printer eliminates deadlocks involving the printer, although it creates a potential for deadlock over disk space. With large disks though, running out of disk space is unlikely.

9.6.4: Attacking Circular Wait:

The circular wait can be eliminated in several ways. One way is simply to have a rule saying that a process is entitled only to a single resource at any moment. If it needs a second one, it must release the first one. For a process that needs to copy a huge file from a tape to a printer, this restriction is unacceptable. Another way to avoid the circular wait is to provide a global numbering of all the resources, as shown in Fig. 9.b Now the rule is this: processes can request resources whenever they want to, but all requests must be made in numerical order. A process may request first a printer and then a tape drive, but it may not request first a plotter and then a printer.

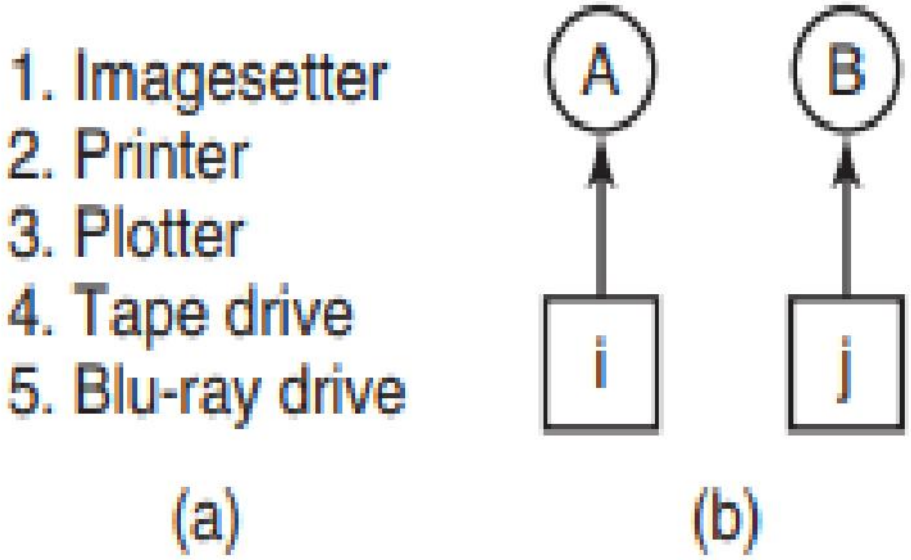


Figure 9. a) numerically order resource b) resource graph

9.7 ISSUES

9.7.1: Two-phase locking:

Although both avoidance and prevention are not terribly promising in the general case, for specific applications, many excellent special-purpose algorithms are known. As an example, in many database systems, an operation that occurs frequently is requesting locks on several records and then updating all the locked records. When multiple processes are running at the same time, there is a real danger of deadlock. The approach often used is called two-phase locking.

9.7.2: Starvation:

As usual FCFS is a good cure. Often this is done by priority aging and picking the highest priority process to get the resource. Also can

periodically stop accepting new processes until all old ones get their resources

A problem closely related to deadlock is starvation. In a dynamic system, requests for resources happen all the time. Some policy is needed to make a decision about who gets which resource when. This policy, although seemingly reasonable, may lead to some processes never getting service even though they are not deadlocked. As an example, consider allocation of the printer. Imagine that the system uses some algorithm to ensure that allocating the printer does not lead to deadlock. Now suppose that several processes all want it at once. Who should get it? One possible allocation algorithm is to give it to the process with the smallest file to print (assuming this information is available). This approach maximizes the number of happy customers and seems fair. Now consider what happens in a busy system when one process has a huge file to print. Every time the printer is free, the system will look around and choose the process with the shortest file. If there is a constant stream of processes with short files, the process with the huge file will never be allocated to the printer. It will simply starve to death (be postponed indefinitely, even though it is not blocked).

Problems on Deadlock:

Problem 01:

A system is having 3 user processes each requiring 2 units of resource R. The minimum number of units of R such that no deadlock will occur-

1. 3
2. 5
3. 4
4. 6

Solution:

In worst case,

The number of units that each process holds = One less than its maximum demand

So,

- Process P1 holds 1 unit of resource R
- Process P2 holds 1 unit of resource R
- Process P3 holds 1 unit of resource R

Thus,

- Maximum number of units of resource R that ensures deadlock = 1 + 2 + 3 = 6
- Minimum number of units of resource R that ensures no deadlock = 6 + 1 = 7

9.8 SUMMARY

A deadlock state occurs when two or more processes are waiting indefinitely for an event that can be caused only by one of the waiting processes.

There are three principal methods for dealing with deadlocks:

- Use some protocol to prevent or avoid deadlocks, ensuring that the system will never enter a deadlock state.
- Allow the system to enter a deadlock state, detect it, and then recover.
- Ignore the problem altogether and pretend that deadlocks never occur in the system. The third solution is the one used by most operating systems, including UNIX and Windows

A deadlock can occur only if four necessary conditions hold simultaneously in the system: mutual exclusion, hold and wait, no preemption, and circular wait. To prevent deadlocks, we can ensure that at least one of the necessary conditions never holds.

A method for avoiding deadlocks that is less stringent than the prevention algorithms requires that the operating system have a priori information on how each process will utilize system resources.

9.9 UNIT END QUSTIONS

1) Explain how the system can recover from the deadlock using
   (a) recovery through preemption.
   (b) recovery through rollback.
   (c) recovery through killing processes.
2) Explain deadlock detection and recovery.
3) How can deadlocks be prevented?
4) Explain deadlock prevention techniques in Details.
5) Explain Deadlock Ignorance.
6) Define the Deadlock with Suitable examples.
7) Explain Deadlock Avoidance in detail.
8) Explain other issues in deadlocks.
9) Explain Resource Acquisition in Deadlock.
10) A system has 3 user processes P1, P2 and P3 where P1 requires 21 units of resource R, P2 requires 31 units of resource R, P3 requires 41 units of resource R. The minimum number of units of R that ensures no deadlock is _____?

*****

UNIT IV

10

VIRTUALIZATION AND CLOUD

Unit Structure

10.0 OBJECTIVES

The objectives of this chapter is as follows:

i) The objective of this chapter is make students learn about the different Virtualization and Cloud technologies.
ii) To learn why there is a need of virtualization in a company or a data centre.
iii) What are the requirements of Virtualization

10.1 INTRODUCTION TO VIRTUALIZATION & CLOUD

In some situations, an organization needs a multi-computer, for example a company has an email server, a Web server, an FTP server, some e-commerce servers, and others. These all run on different computers in the same equipment rack, all connected by a high-speed network. The only objective to gain reliability, because a company can't trust on single operating system which is working 24X7. By putting each service on a separate computer, if one of the server crashes, at least the other ones are not affected. This is good for security also. Even if some malevolent intruder manages to compromise the Web server, he will not immediately have access to sensitive emails also this property sometimes referred to as sandboxing.

For instance, organizations often depend on more than one operating system for their daily operations: a Web server on Linux, a mail server on Windows, an e-commerce server for customers running on OS X, and a few other services running on various types of UNIX. The obvious solution to this is making use of virtual machine technology

10.1.1 About VMM:

1) The main idea is that a VMM (Virtual Machine Monitor) creates the illusion of multiple (virtual) machines on the same physical hardware.
2) VMM is also known as a hypervisor.
3) we distinguish between type 1 hypervisors which run on the bare metal, and type 2 hypervisors that may make use of all the wonderful services and abstractions offered by an underlying operating system.
4) Either way, virtualization allows a single computer to host multiple virtual machines, each potentially running a completely different operating system.
5) The advantage of this approach is that a failure in one virtual machine does not bring down any others
6) On a virtualized system, different servers can run on different virtual machines, thus maintaining the partial-failure model that a multicomputer has, but at a lower cost and with easier maintainability.
7) Moreover, we can now run multiple different operating systems on the same hardware, benefit from virtual machine isolation in the face of attacks.
8) With virtual machine technology, the only software running in the highest privilege mode is the hypervisor, which has two orders of magnitude fewer lines of code than a full operating system, and thus two orders of magnitude fewer bugs.

10.1.2 Having Virtualization has many advantages:

1. A failure in one virtual machine does not bring down any others.
2. Run multiple different operating systems on the same hardware
3. Having fewer physical machines saves money
4. Less hardware and electricity and takes up less rack space.
5. Helps in trying out new ideas
6. Each application can take its own environment with it.
7. Check-pointing and migrating virtual machines is much easier than migrating processes running on a normal operating system.
8. Easy to migrate from one operating system to another.
9. Helps to run legacy applications which are no longer supported or which do not work on current hardware.
10. Helps in software development.

10.2 CLOUD - INTRODUCTION

1. The key idea of a cloud is simple: Outsource.
2. Your computation or storage needs to a well-managed data center run by a company specializing in this and staffed by experts in the area.
3. Because the data center typically belongs to someone else, you will probably have to pay for the use of the resources, but at least you will not have to worry about the physical machines, power, cooling, and maintenance.
4. Because of the isolation offered by virtualization, cloud-providers can allow multiple clients, even competitors, to share a single physical machine.
5. Earlier the organizations were not comfortable sharing their information on cloud. By now, however, virtualized machines in the cloud are used by countless organization for countless applications, and while it may not be for all organizations and all data, there is no doubt that cloud computing has been a success.
6. After a lot of research from the year 1960, finally in the year 1990 researchers at Stanford University developed a new hypervisor and found VMware. VMware offers type 1 & type 2 hypervisors.

10.3 REQUIREMENTS OF VIRTUALIZATION

1. It is important that virtual machines act just like the real McCoy (real thing).
2. In particular, it must be possible to boot them like real machines and install arbitrary operating systems on them, just as can be done on the real hardware.
3. It is the task of hypervisor to provide this illusion and to do it efficiently. Every hypervisor measured on following three dimensions:
   a. **Safety:** The hypervisor should have full control of the virtualized resources.
   b. **Fidelity:** The behaviour of the program on a virtual machine should be identical to that of the same program running on bare hardware.
   c. **Efficiency:** Much of the code in a virtual machine should run without intervention of hypervisor.
4. The interpreter may be able to execute an INC (increment) as it is, but instructions that are not safe to execute directly must be simulated by the interpreter.
5. For instance, we cannot really allow the guest operating system to disable interrupts for the entire machine or modify the page-table mappings.

6. The idea is to make the operating system on top of the hypervisor think that it has disabled interrupts, or changed the machine's page mappings.
7. Every CPU with kernel mode and user mode has a set of instructions that behave differently when executed in kernelmode than when executed in user mode.
8. These include instructions that do I/O, change the MMU settings, and so on.
9. Popek and Goldberg called these sensitive instructions. There is also a set of instructions that cause a trap if executed in user mode.
10. Popek and Goldberg called these privileged instructions. Their paper stated for the first time that a machine is "virtualizable" only if the sensitive instructions are a subset of the privileged instructions.

10.4 TYPE 1 & TYPE 2 HYPERVISORS

1. It is important to mention that not all virtualization technology tries to trick the guest into believing that it has the entire system.
2. Sometimes, the aim is simply to allow a process to run that was originally written for a different operating system and/or architecture.
3. We therefore distinguish between full system virtualization and process-level virtualization.
4. In the year 1972, Goldberg distinguished between two approaches of virtualization.

a) **Type 1 Hypervisor:** Technically, it is like an operating system, since it is the only program running in the most privileged mode. Its job is to support multiple copies of the actual hardware, called virtual machines, similar to the processes a normal operating system runs.

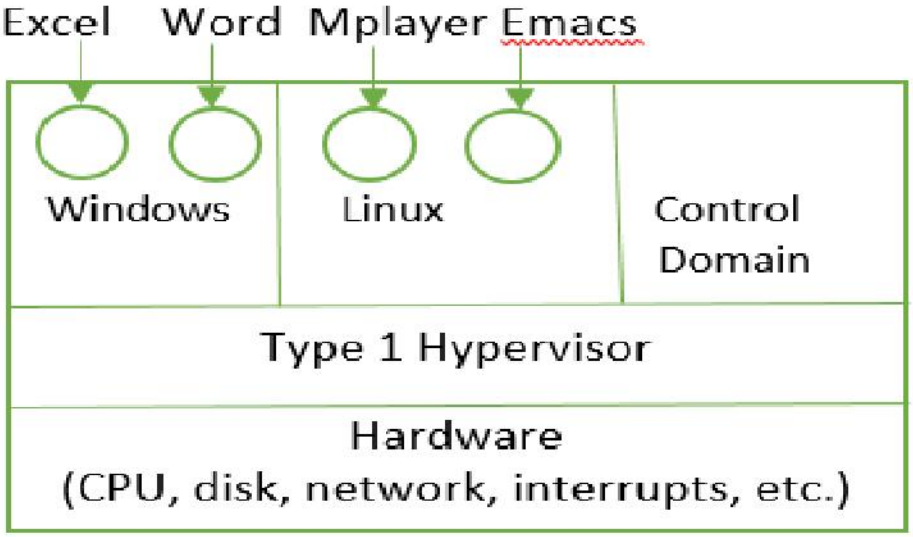


Fig shows Type 1 Hypervisor

b) Type 2 Hypervisor: is a different kind of animal. It is a program that relies on, say, Windows or Linux to allocate and schedule resources, very much like a regular process. Of course, the type 2 hypervisor still pretends to be a full computer with a CPU and various devices. Both types of hypervisor must execute the machine's instruction set in a safe manner. For instance, an operating system running on top of the hypervisor may change and even mess up its own page tables, but not those of others.

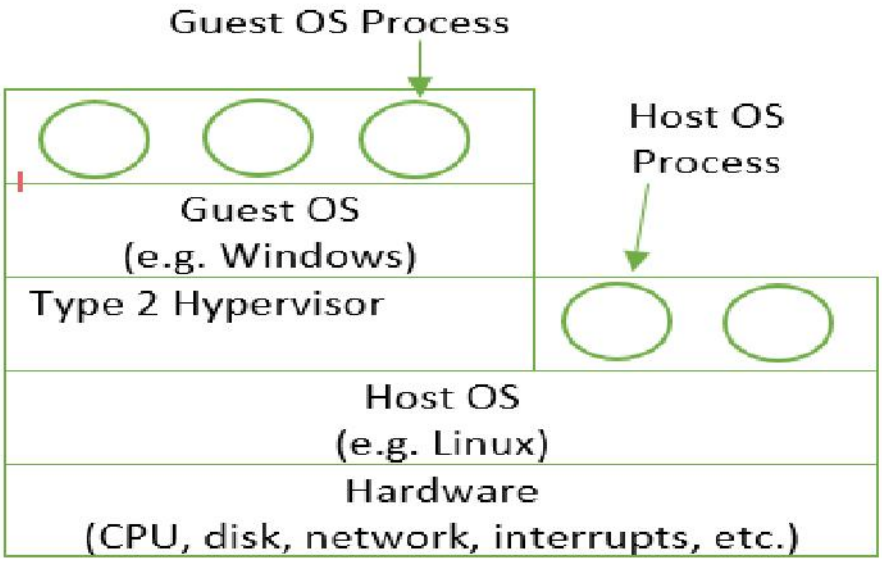


Fig shows Type 2 Hypervisor

5. The operating system running on top of the hypervisor in both cases is called the guest operating system.
6. For a type 2 hypervisor, the operating system running on the hardware is called the host operating system.
7. Type 2 hypervisors, sometimes referred to as hosted hypervisors, depend for much of their functionality on a host operating system such as Windows, Linux, or OS X.
8. When it starts for the first time, it acts like a newly booted computer and expects to find a DVD, USB drive, or CD-ROM containing an operating system in the drive. however, the drive could be a virtual device.

10.5 LET US SUM IT UP

1. VMM creates the illusion of multiple machines on the same physical hardware. 2. Virtualization gives a range of advantages from running different operating systems to developing software. 3. Outsourcing is the best option for storing data in the data centre. 4. Type 1 and type 2 are the two categories offered by VMM to achieve virtualization.

10.6 LIST OF REFERENCES

- Modern Operating system, Fourth edition, Andrew S. Tanenbaum, Herbert Bos.
- https://www.geeksforgeeks.org/generations-of-computer/

10.7 BIBLIOGRAPHY

- Modern Operating System by Galvin

10.8 UNIT END QUESTIONS

1. What is Cloud?
2. Explain Virtualization.
3. Explain types of Hypervisor with neat diagrams.

*****

11

MULTIPROCESSING SYSTEM

Unit Structure

11.0 OBJECTIVES

The objectives of this chapter is as follows:

1. The objective of this chapter is make students learn about the different Virtualization and Cloud technologies.
2. To learn what are the different techniques used for Virtualization.
3. Understand Memory Virtualization as well as I/O Virtualization.

11.1 PRE-REQUISITES

1. VMM creates the illusion of multiple machines on the same physical hardware.
2. Virtualization gives a range of advantages from running different operating systems to developing software.
3. Outsourcing is the best option for storing data in the data centre.
4. Type 1 and type 2 are the two categories offered by VMM to achieve virtualization.

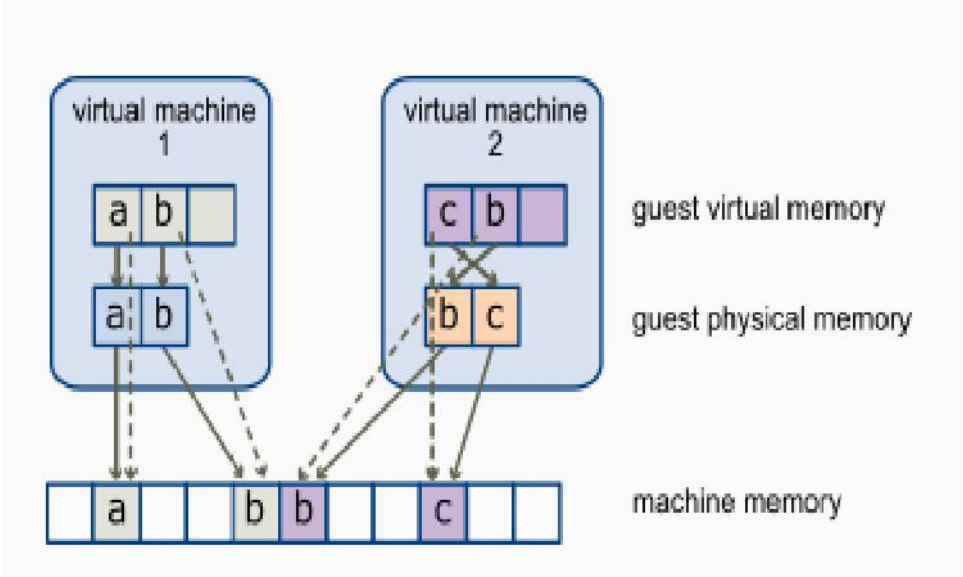
11.2 TECHNIQUES FOR EFFICIENT VIRTUALIZATION S

1. The Type 1 Hypervisor runs on bare metal.
2. The virtual machine runs as a user process in user mode, and as such is not allowed to execute sensitive instructions.

3. However, the virtual machine runs a guest operating system that thinks it is in kernel mode. We will call this virtual kernel mode.
4. The virtual machine also runs user processes, which think they are in user mode.
5. What happens when the guest operating system (which thinks it is in kernel mode) executes an instruction that is allowed only when the CPU really is in kernel mode? Normally, on CPUs without VT, the instruction fails and the operating system crashes.
6. On CPUs with VT, when the guest operating system executes a sensitive instruction, a trap to the hypervisor does occur.
7. Now, let's understand how to migrate from One Virtual machine to another.
   a) To move the virtual machine from the hardware to the new machine without taking it down at all.
   b) Modern virtualization solutions offer is something known as live migration. They move the virtual machine while it is still operational.
   c) For instance, they employ techniques like pre-copy memory migration.
   d) This means that they copy memory pages while the machine is still serving requests.
   e) Most memory pages are not written much, so copying them over is safe.
   f) Remember, the virtual machine is still running, so a page may be modified after it has already been copied.
   g) When memory pages are modified, we have to make sure that the latest version is copied to the destination, so we mark them as dirty.
   h) They will be recopied later. When most memory pages have been copied, we are left with a small number of dirty pages.
   i) We now pause very briefly to copy the remaining pages and resume the virtual machine at the new location. While there is still a pause, it is so brief that applications typically are not affected.
   j) When the downtime is not noticeable, it is known as a seamless live migration.

11.3 MEMORY VIRTUALIZATION

1. We have discussed the issue of how to virtualize the CPU so far. But a computer system has more than just a CPU.
2. It also has memory and I/O devices. They have to be virtualized, too.



a) The boxes represent pages, and the arrows show the different memory mappings.
b) The arrows from guest virtual memory to guest physical memory show the mapping maintained by the page tables in the guest operating system.
c) The arrows from guest physical memory to machine memory show the mapping maintained by the VMM.
d) The dashed arrows show the mapping from guest virtual memory to machine memory in the shadow page tables also maintained by the VMM.
e) The underlying processor running the virtual machine uses the shadow page table mappings

4. Modern operating systems nearly all support virtual memory, which is basically a mapping of pages in the virtual address space onto pages of physical memory.

5. This mapping is defined by (multilevel) page tables. Typically, the mapping is set in motion by having the operating system set a control register in the CPU that points to the top-level page table.

6. Virtualization greatly complicates memory management. In fact, it took hardware manufacturers two tries to get it right.

11.4 I/O VIRTUALIZATION

1. The guest operating system will typically start out probing the hardware to find out what kinds of I/O devices are attached. These probes will trap to the hypervisor. Hypervisor will do two things:

2. One approach is for it to report back that the disks, printers, and so on are the ones that the hardware actually has.
    i. The guest will then load device drivers for these devices and try to use them.
    ii. When the device drivers try to do actual I/O, they will read and write the device's hardware device registers.
    iii. These instructions are sensitive and will trap to the hypervisor, which could then copy the needed values to and from the hardware registers, as needed.
    iv. But here, too, we have a problem. Each guest OS could think it owns an entire disk partition, and there may be many more virtual machines (hundreds) than there are actual disk partitions.

3. The usual solution is for the hypervisor to create a file or region on the actual disk for each virtual machine's physical disk.
    i. Since the guest OS is trying to control a disk that the real hardware has (and which the hypervisor understands), it can convert the block number being accessed into an offset into the file or disk region being used for storage and do the I/O.
    ii. It is also possible for the disk that the guest is using to be different from the real one.

11.5 VIRTUAL APPLIANCES

1. Virtual machines offer a solution to a problem that has long plagued users, especially users of open source software: how to install new application programs?
2. The problem is that many applications are dependent on numerous other applications and libraries, which are themselves dependent on a host of other software packages, and so on.
3. Furthermore, there may be dependencies on particular versions of the compilers, scripting languages, and the operating system.
4. With virtual machines now available, a software developer can carefully construct a virtual machine, load it with the required operating system, compilers, libraries, and application code, and freeze the entire unit, ready to run.
5. This virtual machine image can then be put on a CD-ROM or a Website for customers to install or download.
6. This approach means that only the software developer has to understand all the dependencies. The customers get a complete package that actually works, completely independent of which operating system they are running and which other software, packages, and libraries they have installed.

7. These „„shrink wrapped"" virtual machines are often called "virtual appliances".
8. As an example, Amazon's EC2 cloud has many pre-packaged virtual appliances available for clients, which it offers as convenient software services („„Software as a Service"").

11.6 LET US SUM IT UP

1. There are two techniques used to migrate virtual machine:
    1. Migrate by pausing the virtual machine 2. Live Migration.

    2. Modern operating systems nearly all support virtual memory, which is basically a mapping of pages in the virtual address space onto pages of physical memory. 3. It is the job of the hypervisor to look after the virtualization of I/O. 4. Virtual machines offer a solution to a problem especially with users of open source software on how to install new application programs?

11.7 LIST OF REFERENCES

- Modern Operating system, Fourth edition, Andrew S. Tanenbaum, Herbert Bos.
- https://www.geeksforgeeks.org/generations-of-computer/
- Docs.vmware.com

11.8 BIBLIOGRAPHY

- Modern Operating System by Galvin

11.9 UNIT END QUESTIONS

1. How to migrate Virtual machine quickly?
2. What are virtual appliances?
3. Explain memory virtualization.
4. What is I/O Virtualization?

*****

12

MULTIPLE PROCESSING SYSTEMS

Unit Structure

12.0 OBJECTIVES

The objectives of this chapter is as follows:

i) The objective of this chapter is make students learn about the different Virtualization and Cloud technologies.

ii) Understand the characteristics, advantages of Cloud.

iii) Different types of Multiprocessor, multicomputer & Distributed systems.

12.1 PRE-REQUISITES

1. There are two techniques used to migrate virtual machine:
   a. Migrate by pausing the virtual machine
   b. Live Migration.
2. Modern operating systems nearly all support virtual memory, which is basically a mapping of pages in the virtual address space onto pages of physical memory.
3. It is the job of the hypervisor to look after the virtualization of I/O.

4. Virtual machines offer a solution to a problem especially with users of open source software on how to install new application programs?

12.2 VIRTUAL MACHINES ON MULTICORE CPUS

1. It has never been possible for an application designer to first choose how many CPUs he wants and then write the software accordingly.
2. The combination of virtual machines and multicore CPUs creates a whole new world in which the number of CPUs available can be set by the software.
3. This is clearly a new phase in computing. Moreover, virtual machines can share memory.
4. A typical example where this is useful is a single server hosting multiple instances of the same operating systems.
5. All that has to be done is map physical pages into the address spaces of multiple virtual machines.
6. Memory sharing is already available in deduplication solutions. Deduplication avoids storing the same data twice.
7. It is a common technique in storage systems, but is now appearing in virtualization as well.
8. In general, the technique revolves around scanning the memory of each of the virtual machines on a host and hashing the memory pages.
9. Should some pages produce an identical hash, the system has to first check to see if they really are the same, and if so, de-duplicate them, creating one page with the actual content and two references to that page. Since the hypervisor controls the nested (or shadow) page tables, this mapping is straightforward.
10. The combination of multicore, virtual machines, hypervisor, and microkernels is going to radically change the way people think about computer systems.
11. Current software cannot deal with the idea of the programmer determining how many CPUs are needed, whether they should be a multicomputer or a multiprocessor, and how minimal kernels of one kind or another fit into the picture.

12.3 LICENSING ISSUES

1. Some software is licensed on a per-CPU basis, especially software for companies. In other words, when they buy a program, they have the right to run it on just one CPU.
2. Does this contract give them the right to run the software on multiple virtual machines all running on the same physical machine? Many software vendors are somewhat unsure of what to do here.

3. The problem is much worse in companies that have a license allowing them to have ‘n’ machines running the software at the same time, especially when virtual machines come and go on demand.
4. In some cases, software vendors have put an explicit clause in the license forbidding the licensee from running the software on a virtual machine or on an unauthorized virtual machine.
5. For companies that run all their software exclusively on virtual machines, this could be a real problem. Whether any of these restrictions will hold up in court and how users respond to them remains to be seen.

12.4 CLOUDS

1. Virtualization technology played a crucial role in the dizzying rise of cloud computing. There are many clouds.
2. Some clouds are public and available to anyone willing to pay for the use of resources, others are private to an organization. 3. Likewise, different clouds offer different things. Some give their users access to physical hardware, but most virtualize their environments.
4. Some offer the bare machines, virtual or not, and nothing more, but others offer software that is ready to use and can be combined in interesting ways, or platforms that make it easy for their users to develop new services.
5. Cloud providers typically offer different categories of resources.

12.4.1 Characteristics of Cloud:

The National Institute of Standards and Technology has listed five essential characteristics:

a. On-demand self-service: Users should be able to provision resources automatically, without requiring human interaction.

b. Broad network access: All these resources should be available over the network via standard mechanisms so that heterogeneous devices can make use of them.

c. Resource pooling: The computing resource owned by the provider should be pooled to serve multiple users and with the ability to assign and reassign resources dynamically

d. Rapid elasticity: It should be possible to acquire and release resources elastically, perhaps even automatically, to scale immediately with the users’ demands.

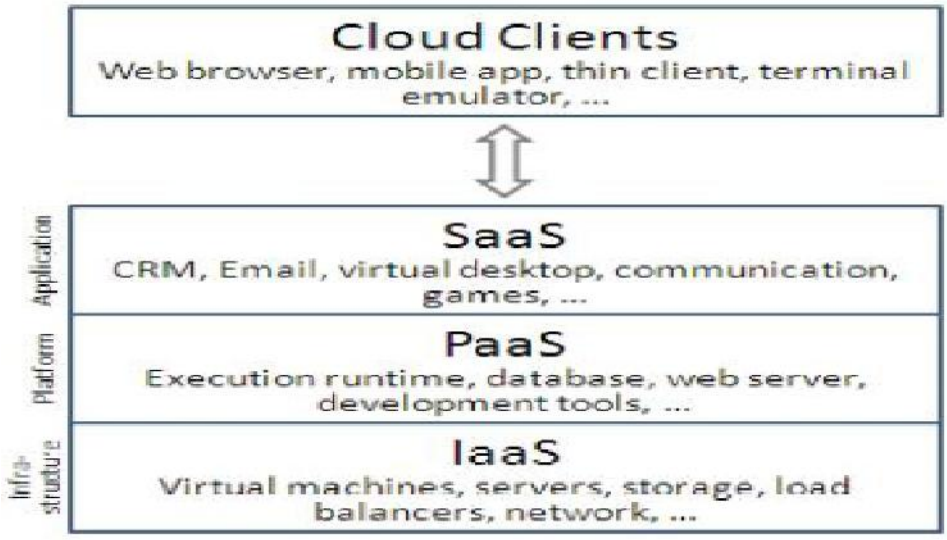
e. Measured service: The cloud provider meters the resources used in a way that matches the type of service agreed upon.

12.4.2 Services offered by Cloud:

12.4.2.1 Software as a Service (It offers specific software)

12.4.2.2 Platform as a Service (It creates environment which gives specific Operating system, database, web server etc.)

12.4.2.3 Infrastructure as a Service (Same cloud can run different Operating Systems) We can refer the diagram below for more understanding:



12.4.3 Advantages of Cloud

Following are the advantages of using Cloud:

1. **Unlimited storage:** Clouds provide unlimited storage.
2. **Flexibility:** If your needs increase, it's easy to scale up your cloud capacity. Likewise, if you need to scale down again, you can scale down the cloud capacity again.
3. **Disaster recovery:** Backup and recovery of data is possible.
4. Automatic software updates
5. **Capital-expenditure free:** Cloud computing cuts the high cost of hardware. You simply pay as you use subscription-based model.
6. **Work from anywhere:** With an internet connection, you can work from anywhere.
7. **Security:** Your data is stored in the cloud; you can access it no matter what happens to your machine

12.5 MULTIPLE PROCESSOR SYSTEMS

12.5.1.1 Each CPU has its own OS:

a. Memory is divided into equal sized partitions, where each partition belongs to one CPU.

b. Each CPU has its own private memory and its own private copy of the operating system.

c. Alternative to this scheme is to allow all the CPUs to share the operating system code and make private copies of only the data structures of OS.

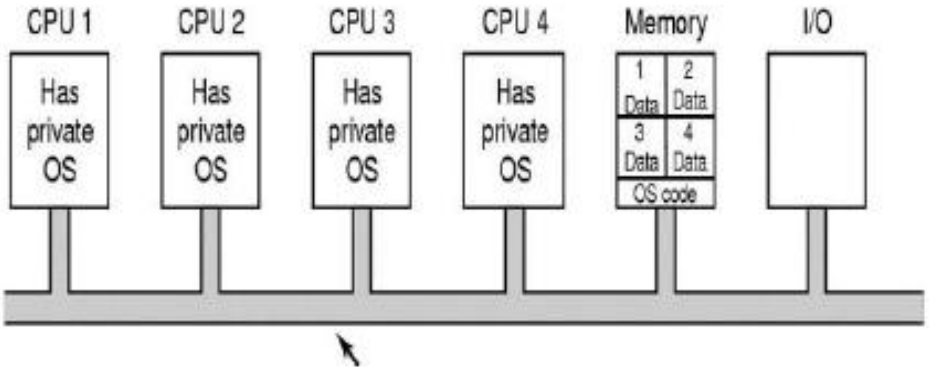
d. There are 4 aspects of this design,

   i. When a process makes a system call, the system call is caught and handled on its own CPU using the data structures in that operating system's tables.

   ii. Each operating system has its own tables; it also has its own set of processes that it schedules by itself.

   iii. Third, there is no sharing of physical pages. So some CPU is overburdened and some is idle, as there is no load sharing.

   iv. No additional memory, so programs cannot grow



12.5.1.2 Master-Slave Multiprocessor:

a. Only one copy of OS is present in memory.

b. Master CPU can only run the operating system from memory. So here, only CPU1 can run the OS and not any others.

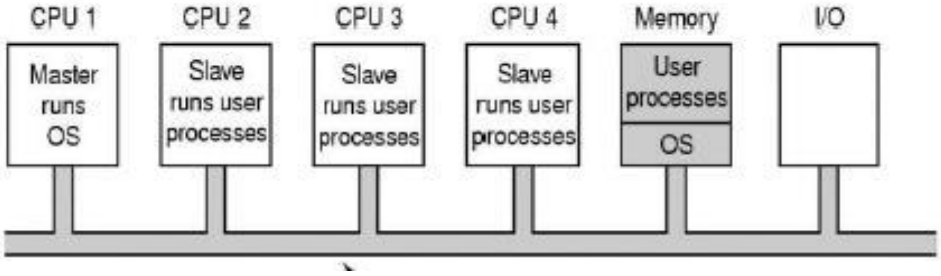
c. All system calls from other CPUs are redirected to CPU 1 for processing there.

d. CPU 1 is the master and all the others are slaves.

e. When a CPU goes idle, it asks the operating system on CPU 1 for a process to run and is assigned one.

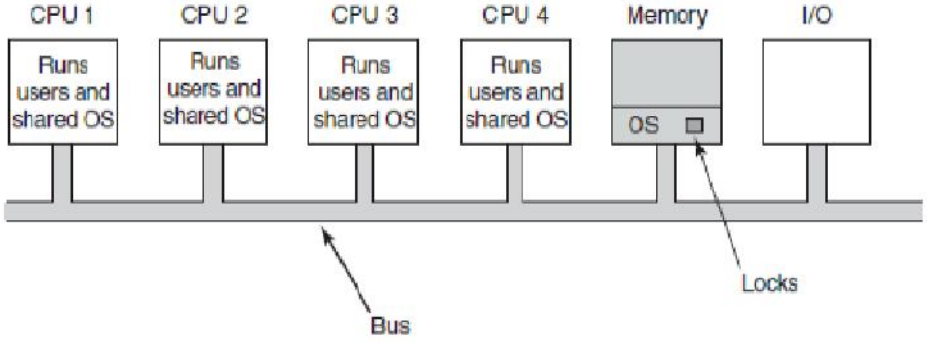
f. Thus it can never happen that one CPU is idle while another is overloaded. g. Similarly, pages can be allocated among all the processes dynamically and there is only one buffer cache, so inconsistencies never occur.

h. The problem with this model is that with many CPUs, the master will become a bottleneck.



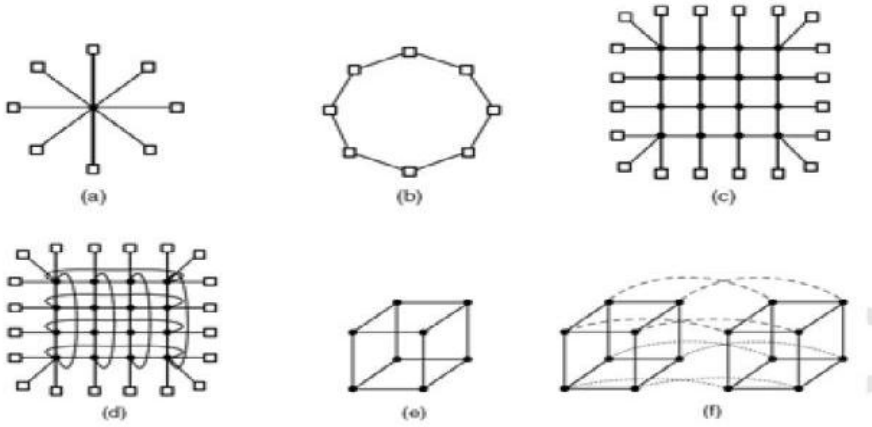
12.5.1.3 Symmetric Multiprocessor:

a. It eliminates the asymmetry in Master-Slave configuration.
b. There is one copy of the operating system in memory, but any CPU can run it.
c. It eliminates the master CPU bottleneck, since there is no master.
d. No need to redirect system calls to 1 CPU as each CPU can run the OS.
e. While running a process, the CPU on which the system call was made processes the system call.



12.5.2 Multi-computers:

a Following are the various inter-connection technologies used in Multi-computer:



(a) A single switch (b) A ring (c) A grid or mesh
(d) A double torus (e) A cube (f) A 4D hypercube

b. **Single Switch/ Star topology:** Every node contains a network interface card and all computers are connected to switches/hubs. Fast, expandable but single point failure systems. Failure in switch/hub can take down entire system.

c. **Ring Topology:** Each node has two wires coming out the network interface card, one into the node on the left and one going into the node on the right. There is no use of switches in this topology.

d. **Grid/mesh topology:** two dimensional design with multiple switches and can be expanded easily to large size. Its diameter is the longest path between any two nodes.

e. **Double Torus:** alternative to grid, which is a grid with the edges connected. With compare to grid its diameter is less and it is more fault tolerant. The diameter is less as opposite corners communicates in only two hops.

f. **Cube:** Fig e. shows 2 x 2 x 2 cube which is a regular three-dimensional topology. In general case it could be a n x n x n cube. No of nodes = 2n. So for 3D cube, 8 nodes can be attached.

g. **A 4-D Hypercube:** Fig (f) shows four –dimensional cube constructed from two three – dimensional cubes with the equivalent nodes connected. An n-dimensional cube formed this way is called a hypercube. Many parallel computers can be building using hypercube topology.

12.5.3 Distributed Systems:

a. A distributed system is defined as set of autonomous computers that appears to its users as a single coherent system.

b. Users of distributed system feel that, they are working with as a single system.

c. Distributed system is like multi-computers spread worldwide.

d. Each node in distributed system is having its own CPU, RAM, network board. OS, and disk for paging.

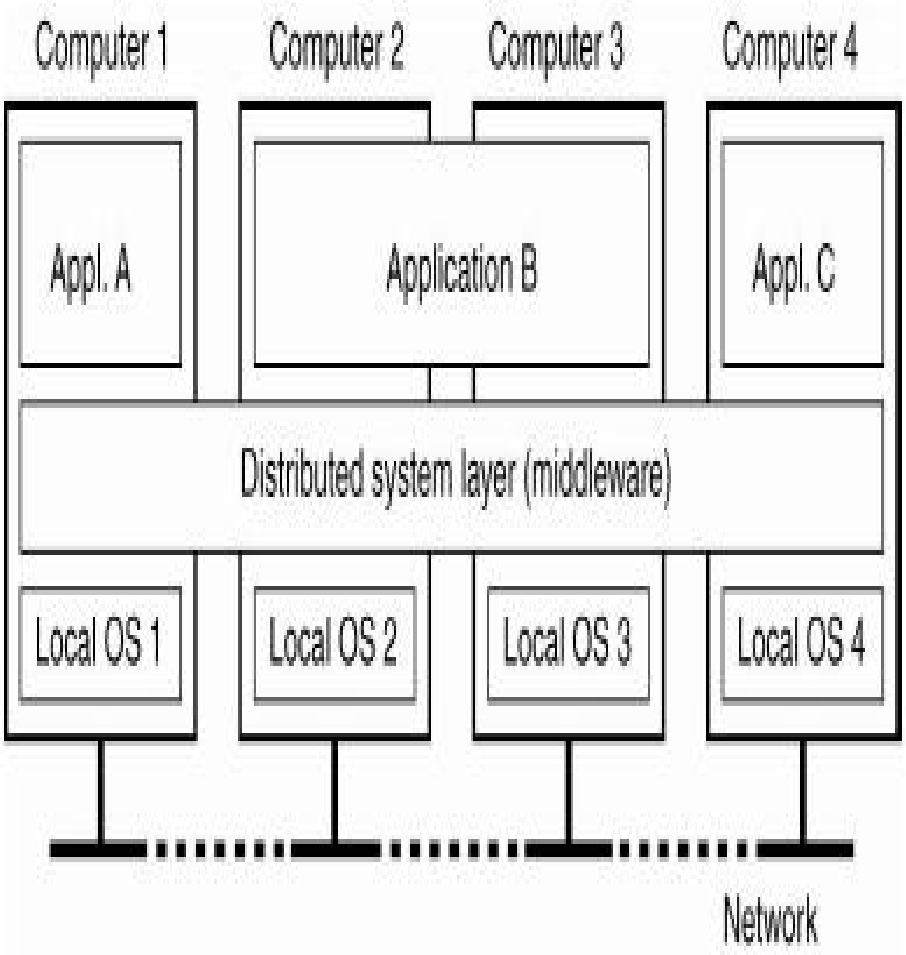
e. Following are the main characteristics of distributed systems:

   i. A distributed system comprises computers with different architecture and different OS. These dissimilarities and the ways all these machines communicate are hidden from users.

   ii. The manner in which distributed system is organized is also hidden from the users of the distributed system.

   iii. The interaction of users and applications with distributed system is in consistent and identical way.

   iv. It should be always available to the users and applications inspite of failures. Failure handling should be hidden from users and applications.



12.6 LET US SUM IT UP

ITEM	Multiprocessor	Multicomputer	Distributed System
Node Configuration	CPU	CPU, RAM, net interface	Complete computer
Node peripherals	All shared	Shared exc. Maybe disk	Full set per node
Location	Same rack	Same room	Possibly worldwide
Internode communication	Shared RAM	Dedicated interconnect	Traditional network
Operating systems	One, shared	Multiple, same	Possibly at different
File systems	One Shared	One Shared	Each Node Has Own
Administration	One organization	One organization	Many organizations

12.7 LIST OF REFERENCES

- Modern Operating system, Fourth edition, Andrew S. Tanenbaum, Herbert Bos.
- https://www.geeksforgeeks.org/generations-of-computer/

12.8 BIBLIOGRAPHY

- Modern Operating System by Galvin

12.9 UNIT END QUESTIONS

1. List and explain different types of multiprocessor operating system?
2. With neat diagram explain various interconnection technologies used in multicomputer.
3. Define and explain distributed systems with neat diagram.
4. List and explain different types of multiprocessor operating system.
5. Differentiate between Multiprocessor, Multicomputer and Distributed Systems.
6. What are the services and advantages of Cloud computing?
7. What is cloud? Write essential characteristic of cloud.

*****

13

LINUX CASE STUDY

Unit Structure

13.0 OBJECTIVES

- To understand principles of Linux
- To learn principles of Process Management, Memory Management
- To learn principles of I/O in Linux, File System and Security

13.1 HISTORY OF UNIX AND LINUX

13.1.1 History of UNIX:

I. The UNIX Operating System is derived from MULTICS (Multiplexed Operating System and Computer System).It was begun in mid 1960's.

II. In 1969, Kem Thompson wrote the first version of UNIX called UNICS. It stands for Uniplexed Operating and Computing System.

III. In 1973, Ken Thompson teamed up with Dennis Ritchie and rewrote the Unix kernel in C.

IV. Ken Thompson spent a year's sabbatical with the University of California at Berkeley. While there he and two graduate students, Bill Joy and Chuck Haley, wrote the first Berkely version of Unix, which was distributed to students.

V. This resulted in the source code being worked on and developed by many different people.

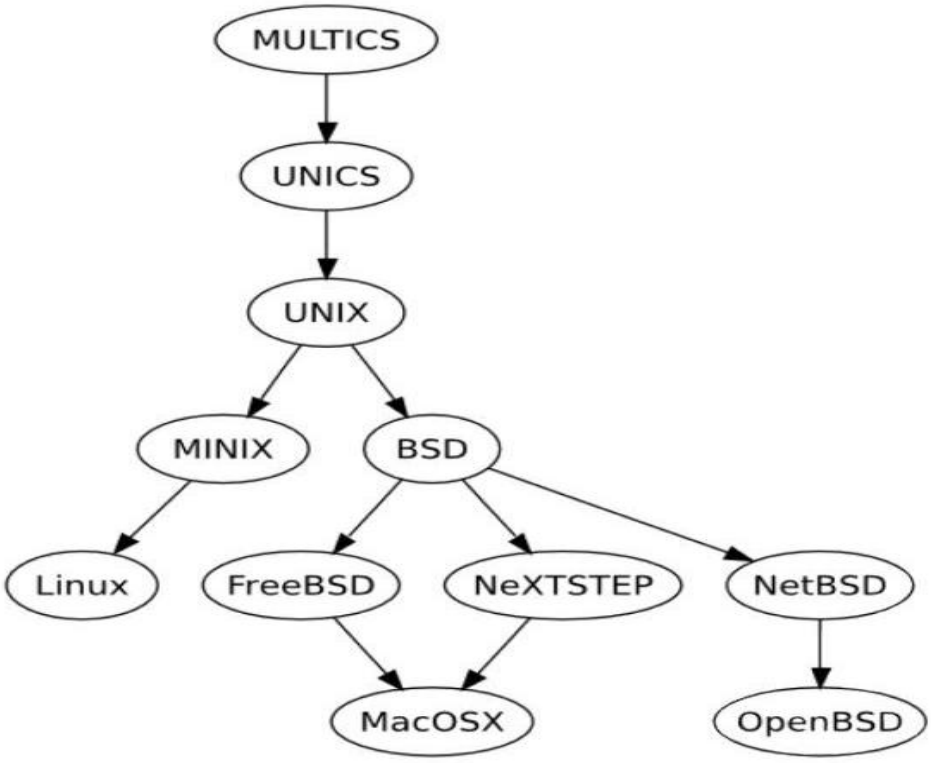
VI. The Berkeley version of Unix is known as BSD, Berkeley Software Distribution. From BSD came the vi editor, C shell, virtual memory, Sendmail, and support for TCP/IP.

VII. For several years SVR4 was the more conservative, commercial, and well supported.

VIII. Today SVR4 and BSD look very much alike. Probably the biggest cosmetic difference between them is the way the ps command functions.

IX. The Linux operating system was developed as a Unix look alike and has a user command interface that resembles SVR4.

Following figure shows history in better way



13.1.2 History of Linux:

Linux is an open source family of Unix-like Linux based kernel applications, a kernel operating system that was first released on September 17, 1991, by Linus Torvalds Linux usually included in the Linux distribution. Popular Linux distributions include Debian, Fedora, and Ubuntu. Commercial distribution includes Red Hat Enterprise Linux

and SUSE Linux Enterprise Server. Because Linux is distributed freely, anyone can create a distribution for any purpose.

Popular Linux distributions include Debian, Fedora, and Ubuntu. Commercial distribution includes Red Hat Enterprise Linux and SUSE Linux Enterprise Server. Because Linux is distributed freely, anyone can create a distribution for any purpose.

Linux was originally designed for computers based on the Intel x86 architecture, but has since been deployed to more platforms than any other operating system. Linux is a leading operating system on servers and other large-scale systems such as keyword computers.

The Unix operating system was developed in 1969 at AT & T Bell Laboratories in America by Ken Thompson and Dennis Ritchie. Unix's high-performance language acquisition has made it easy to be deployed across different computer platforms.

Creation:

In 1991, Torvalds became interested in the operating system. Torvalds introduced a switch from its original license, which prohibited commercial re-distribution to the GNU GPL. The developers worked to integrate the GNU components into the Linux kernel, creating a fully functional and free operating system.

Commercial and public reproduction:

Today the Linux systems are used in throughout computing that is from all theembedded systems on almost all the supercomputers, and on the server installations such as the very much popular the LAMP Application Stack. The use of Linux distribution on home and enterprise desktops is growing. Linux are also popular in the netbook market, many devices install customized Linux distributions, and Google hasreleased its own Chrome OS designed for netbooks.

13.2 OVERVIEW

13.2.1 An Overview of UNIX:

The UNIX operating system is designed to allow many programmers to simultaneously access the computer and share its resources.

The operating system controls all commands from all keyboards and all data generated, and allows each user to believe that he or she is the only person working on the computer.

The real-time sharing of resources makes UNIX one of the most powerful operating systems ever.

UNIX was developed by programmers for community of programmers, the functionality it provides is so powerful and flexible that it can be found in business, science, academia, and industry.

The uniqueness of UNIX and Features provided by UNIX are:

Multitasking capability:

Many computers can only do one thing at a time, and anyone with a PC or laptop can prove it. While opening the browser and opening the word processing program, try to log in to the company's network. When arranging multiple instructions, the processor may freeze for a few seconds.

Multiuser capability:

The same design that allows multitasking allows multiple users to use the computer. The computer can accept many user commands (depending on the design of the computer) to run programs, access files and print documents at the same time.

UNIX programs:

UNIX tools - Hundreds of programs come with UNIX, and these programs can be divided into two categories: Integrated utilities essential for computer operation, such as command interpreters and Tools that are not required for UNIX operation, but provide users with additional functions, such as typesetting functions and e-mail.

Library of application software:

In addition to the applications that come with UNIX, hundreds of UNIX applications can be purchased from third-party vendors. Although third-party vendors have written some tools for specific applications, there are hundreds of tools available for UNIX users.

Generally, tools are divided into categories for certain functions (such as word processing, business applications, or programming).

13.2.2 Overview of Linux|:

Linux is a UNIX - like computer OS which is assembled & made under the model of free and open source software development and distribution. The most defined component of Linux is the Linux kernel, an OS kernel was first released on 1991 by Linus Torvalds.

A Linux-based system is a modular Unix-like Operating System. It derives much of its basic design from principles established in Unix during the 1970 and 1980. Such a system uses a monolithic kernel, the Linux kernel, which handles process control, networking, & peripheral & file system access. Device drivers are integrated directly with the kernel or they added as modules loaded while the system is running.

- **A bootloader** - for example GRUB or LILO. This is a program which is executed by the computer when it is first turned on, & loads the Linux kernel into memory.
- **An init program** - This is a process launched by the Linux kernel, & is at the root of the process tree, in other words, all processes are launched through in it. It starts processes such as system services & login prompts (whether graphical or in terminal mode)
- Software libraries which contain code which can be used by running processes. On Linux OS using ELF-format executable files, the dynamic linker which manages use of dynamic libraries is "ld-linux.so".
- The most commonly used software library on Linux systems is the GNU C. Library. If the OS is set up for the user to compile software themselves, header files will also be included to describe the interface of installed libraries.
- User interface programs such as comm. & shells or windowing environments

Linux is a widely ported operating system kernel. Currently most of the distribution include a graphical user environment, with the 2 most popular environments which are GNOME (it basically utilizes additional shells such as the default GNOME Shell & Ubuntu Unity), & KDE Plasma Desktop.

13.3 PROCESS IN LINUX

A Linux-based system may be a modular Unix-like OS. It derives much of its basic design from principles established in Unix during the 1970s and 1980s. Such a system uses a monolithic kernel, the Linux kernel, which handles process control, networking, and peripheral and filing system access. Device drivers are either integrated directly with the kernel or added as modules loaded while the system is running.

Various parts of an OS UNIX and 'UNIX-like' operating systems (such as Linux) contains a kernel and a few system programs. There also are some application programs for doing work. The kernel is that the heart of the OS. In fact, it's often mistakenly considered to be the OS itself, but it's not. An OS provides more services than a clear kernel. It keeps track of files on the disk, starts programs and runs them concurrently, assigns memory and other resources to varied processes, receives packets from

and sends packets to the network, and so on. The kernel does little or noby itself, but it provides tools with which all services are often built. It also prevents anyone from accessing the hardware directly, forcing everyone to use the tools it provides. This manner the kernel provides some protection for users from one another. The tools provided by the kernel are used via system calls. The system programs use the tools provided by the kernel to implement the varied services required from an OS. System programs, and every one other programs, run `on top of the kernel', in what's called the user mode. The difference between system and application programs is one among intent: applications are intended for getting useful things done (or for enjoying, if it happens to be a game), whereas system programs are needed to urge the system working. A word processing system may be an application; mount is a systems program. The difference is usually somewhat blurry, however, and is vital only to compulsive categorizers. An OS also can contain compilers and therefore their corresponding libraries (GCC and the C library especially under Linux), although not all programming languages need be a part of the OS. Documentation, and sometimes even games, also can be a part of it. Traditionally, the OS has been defined by the contents of the installation tape or disks; with Linux it's not as clear since it's spread everywhere the FTP sites of the planet.

Important parts of the kernel:

The Linux kernel consists of several important parts: process management, memory management, hardware device drivers, filesystem drivers, network management, and various other bits and pieces. Memory management takes care of assigning memory areas and swap file areas to processes, parts of the kernel, and for the buffer cache. Process management creates processes, and implements multitasking by switching the active process on the processor. At rock bottom level, the kernel contains a hardware driver for every quite hardware it supports. There are often many otherwise similar pieces of hardware that differ in how they're controlled by software. The similarities make it possible to possess general classes of drivers that support similar operations; each member of the category has an equivalent interface to the remainder of the kernel but differs in what it must do to implement them. for instance, all disk drivers look alike to the remainder of the kernel, i.e., all of them have operations like `initialize the drive', `read sector N', and `write sector N'. Some software services provided by the kernel itself have similar properties, and may therefore be abstracted into classes. for instance, the varied network protocols are abstracted into one programming interface, the BSD socket library. Another example is that the virtual filesystem (VFS) layer that abstracts the filesystem operations faraway from their implementation. Each filesystem type provides an implementation of every filesystem operation. When some entity tries to use a filesystem, the request goes via the VFS, which routes the request to the right filesystem driver.

Major services during a UNIX :

Init the only most vital service during a UNIX is provided by init, init is started because the first process of each UNIX, because the last item the kernel does when it boots. When init starts, it continues the boot process by doing various start-up chores (checking and mounting filesystems, starting daemons, etc). When the system is pack up, it's init that's responsible of killing all other processes, unmounting all filesystems and stopping the processor, along side anything it's been configured to try to do

Syslog:

The kernel and lots of system programs produce error, warning, and other messages. It’s often important that these messages are often viewed later, even much later, in order that they should be written to a file. The program doing this is often syslog. It is often configured to sort the messages to different files consistent with writer or degree of importance. for instance, kernel messages are often directed to a separate file from the others, since kernel messages are often more important and wish to be read regularly to identify problems.

Both users and system administrators often got to run commands periodically. For instance, the supervisor might want to run a command to wash the directories with temporary files (/tmp and /var/tmp) from old files, to stay the disks from filling up, since not all programs pack up after themselves correctly.

The **cron** service is about up to try to this. Each user can have a crontab file, where she lists the commands she wishes to execute and therefore the times they ought to be executed. The cron daemon takes care of starting the commands when specified.

Graphical interface:

This arrangement makes the system more flexible, but has the disadvantage that it's simple to implement a special interface for every program, making the system harder to find out.

The graphical environment primarily used with Linux is named the X Window System (X for short). Some popular window managers are: fvwm, icewm, blackbox, and windowmaker. There also are two popular desktop managers, KDE and Gnome.

Networking:

Networking is that the act of connecting two or more computers in order that they will communicate with one another. the particular methods of connecting and communicating are slightly complicated, but the top result's very useful.

UNIX operating systems have many networking features. most elementary services (filesystems, printing, backups, etc) are often done over the network.

Network logins:

Network logins work a touch differently than normal logins. for every person logging in via the network there's a separate virtual network connection, and there are often any number of those counting on the available bandwidth. It's therefore impossible to run a separate getty for every possible virtual connection

13.4 MEMORY MANAGEMENT

It is the process of managing the computer memory. The goal is to keep track of which parts of memory are in use and which parts are not, to allocate memory to processes when they need it and de-allocate it when they are done.

UNIX operating system is works on two memory management schemes, These are as follows-

1. swapping
2. demand paging

Non-Contiguous Memory Allocation

Techniques are:

1.Paging:

It is a storage mechanism that allows OS to fetch processes from the non-volatile memory into the volatile memory in the form of pages. The partitions of nonvolatile are called as pages & volatile memory is divided into small fixed-size blocks of physical memory, which is called frames.

Example :

Consider a process is divided into 4 pages A0, A1, A2 and A3.

Depending upon the availability, these pages may be stored in the main memory frames as shown in the below diagram-

A2
A3
A0
A1

Main Memory

2. Segmentation:

A process is divided into division called Segments. These segments are not of same size. There are types of segmentation:

1. **Virtual memory segmentation:** Every process is divided into multiple number of segments, which do not reside at any one point in time.
2. **Simple segmentation:** Every process is divided into a number of segments, all the processes are loaded in run time.

Segment Table:

A table which stores the information about every segment of the process. It has two columns. Column 1 gives information about size or length of the segment. Column 2 gives information about the base address

Consider the below diagram for segment table.

Limit	**Base**
1500	1500
1000	4700
500	4500

Segment Table

According to the above table, the segments are stored in the main memory as:

Segment- 0
Segment- 3
Segment- 2

Main Memory

The advantages of segmentation are

- Segment table takes less space as compared to Page Table in paging.
- It solves the problem of internal fragmentation.

The disadvantages of segmentation are

- Unequal size is problem for swapping.
- Though it solves internal fragmentation, it do suffer from external fragmentation.

Paging vs Segmentation:

Paging divides program into fixed size Segmentation divides program into

Paging	Segmentation
Paging divides program into fixed size pages.	Segmentation divides program into variable size segments.
Operating System is responsible.	Compiler is responsible
Faster than segmentation.	It is slower than paging
It is closer to operating system.	Segmentation is closer to User.

Memory Management:

Demand Paging:

Deciding which pages need to be kept in the main memory and which need to be kept in the secondary memory, is going to be difficult because we can't say in advance that a process might require a particular page at a particular moment of time. So, to beat this problem, there comes a concept called Demand Paging. It advises keeping all pages of the frames in the secondary memory till they are required. We can also say that, it says that do not load other pages in the main memory till they are required. Whenever any page is referred for the 1st time in the main memory, then that page will appear in the secondary memory

Page fault:

If the mention page is not available in the main memory then there will be a gap and this theory is called Page miss or page fault. Then the CPU has to work on the missed page from the secondary memory. If the number of page faults is very high then the time to access a page of the system will become very high.

Thrashing:

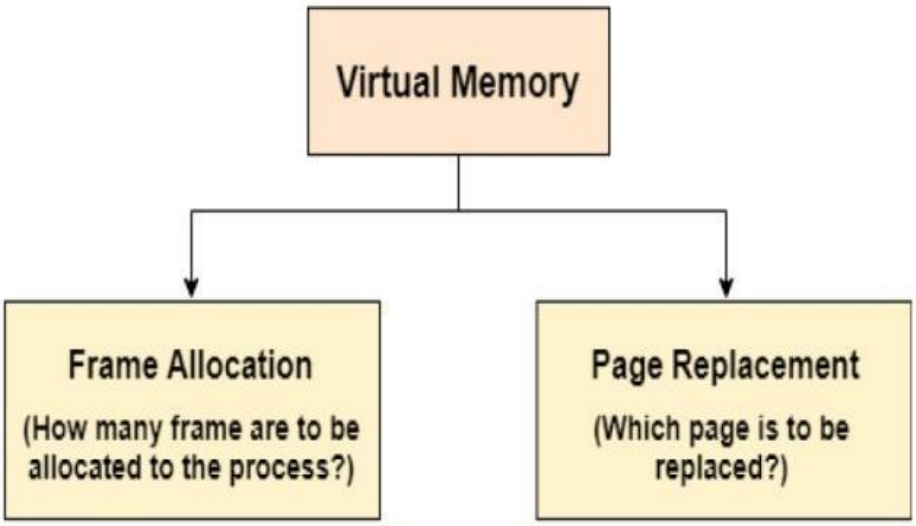
If number of page faults is equal to the number of mention pages or the number of page faults are so very high so that the CPU cannot remain vacant in just reading the pages from the secondary memory then the important access time will be the time taken by the CPU to read 1 word from the secondary memory and it will be very much high. The process is called thrashing. So, assume If the page fault rate is pp %, the time taken in getting a page from the secondary memory & again restarting is S(processing time) and the memory access time is m then the effective access time can be given as;

1. EAT = PF x S + (1 pp) x (m)

Page Replacement:

The page replacement algorithm tells us that which memory page is to be changed. This moment of replacement is sometimes called a swap

out or write to disk. Page replacement is to be done when the requested page is not been able to access or found in the main memory (page fault)



There are 2 main types of virtual memory, Frame allocation and Page Replacement. Frame allocation is all about how many frames can be allocated to a process while the page replacement tells us about determining the number of the pages which requires to be replaced in command to make space for the requested page

What If the algorithm is not optimal?

1. Due to the absence of frames, many of the pages will be occupying the main memory and however more page faults might occur. So, if the OS specifies more frames to the process then there can be interior fragmentation.

2. If the page replacement algorithm is not optimal the n there might also lead to the problem of thrashing. If the number of pages that are to be replaced by the requested pages will be referred to in the near future then there will be more number of swap-out & swap-in and so after the OS has to work on more replacements then usual which causes performance shortage. So, the task of an optimal page replacement algorithm is to check the page which can restrict the thrashing.

Types of Page Replacement Algorithms:

There are various types of page replacement algorithms. Each of the algorithms has a different way of working by which the pages can be replaced.

1. **Optimal Page Replacement algorithm** → The algorithms replaces the page which will not be mentioned for a long time in future. Anyways it cannot be practically implementable but it can be definitely used as a

benchmark. Other algorithms are balanced to this in terms of optimality.

2. **Least recent used (LRU) page replacement algorithm** → This algorithm replaces the page which has not been mentioned for a longer time. This algorithm is just an exchange to the optimal page replacement algorithm. In this, we check the past instead of staring at the future.

3. **FIFO → In** this type algorithm, a line is to be maintained. This page which is assigned the frame 1st will be replaced 1st. In other words, the page which stays at the rare end of the line will be changed on the every page fault.

13.5 INPUT

Linux operating system considers and works with the below devices, by the same way we open and close a file.

- Block devices(Hard-disks, Compact Disks, Floppy, Flash Memory)
- Serial devices (Mouse, keyboard)
- Network Devices

A user can do operations on these devices, as he does operations on a file.

I/O redirection allows you to alter input source of a command as well as its output and error messages are sent. And this is possible by the '<' and '>' redirection operators.

The main advantages with block devices are the fact that they can be read randomly. Also, serial devices are operated.

Another advantage of using block devices is that, if allows access to random location's on the device. Also data from the device is read with a fixed block size. Input & output to the block devices works on the "elevator algorithm" It says that it works on the same principle, as an elevator would.

Mechanical devices like hard-disks are very slow in nature, when it comes to data input and output compared to system memory (RAM) & Processor.

Sometimes applications have to wait for the input and output requests to complete, because different applications are in queue for its input output operations to complete.

The slowest part of any Linux system (or any other operating system), is the disk I/O systems. There is a large difference between, the speed and the duration taken to complete an input/output request of CPU, RAM and Hard-disk. Sometimes if one of the processes running on your system does a lot of read/write operations on the disk, there will be an intense lag or slow response from other processes since they are all waiting for their respective I/O operations to get completed.

Linux Io Commands:

Output Redirection:

The output from the command normally intended for normal output are often easily diverted to a file instead., this capability is understood as output redirection.

```
$ who > users
```

Notice that no output appears at the terminal. This is often because the output has been redirected from the default standard output device (the terminal) into the required file.

If a command has its output redirected to a file and therefore the file already contains some data, that data are going to be lost. Example

```
$ echo line 1 > users
$ cat users
line 1
$
```

The >> operator are often used to append the output in an existing file as follows

```
$ echo line 2 >> users
$ cat users
line 1
line 2
$
```

Input Redirection:

The commands that normally take their input from the standard input can have their input redirected from a file in this manner. For example, to count the number of lines in the file *users* generated above, you can execute the command as follows

```
$ wc -l users
2 users
$
```

Upon execution, you will receive the following output. You can count the number of lines in the file by redirecting the standard input of the wc command from the file users

```
$ wc -l < users
2
$
```

Note that there is a difference in the output produced by the two forms of the wc command. In the first case, the name of the file users is listed with the line count; in the second case, it is not.

In the first case, we knows that it is reading its input from the file users. In the second case, it only knows that it is reading its input from standard input so it does not display file name.

Redirection Commands:

Following is a complete list of commands which you can use for redirection

Sr. No	**Command & Description**
1	**Pgm > file** Output of pgm is redirected to file.
2	**Pgm < file** Program pgm reads its input from file.
3	**Pgm >> file** Output of pgm is appended to file.

4	**n > file** Output from stream with descriptor n redirected to file
5	**n >> file** Output from stream with descriptor n appended to file
6	**n >& m** Merges output from stream n with stream m
7	**n <& m** Merges input from stream n with stream m
8	**<< tag** Standard input comes from here through next tag at the start of line
9	**I** Takes output from one program, or process, and sends it to another.

Note that the file descriptor **0** is normally standard input (STDIN), **1** is standard output (STDOUT), and **2** is standard error output (STDERR).

13.6 LINUX FILE SYSTEM

Linux File System or any file system generally is a layer which is under the operating system that handles the positioning of your data on the storage. The starting and ending of file is not known by the system. Ever if you find any unsupported file system type.

Linux File System Directories:

/bin: Where core commands of Linux exists for example ls, mv.

/boot: Where boot loader and boot files located.

/dev: Physical drives like USBs DVDs are mounted in this.
/etc: Contains configurations for the installed packages.

/home: Here personal folders are allotted to the users to store his folders with his/her name like/home/like geeks.

/lib: Here the libraries are located of the installed package. You may find duplicates in different folders since libraries are shared among all packages unlike windows.

/media: Here is the external devices like DVDs and USB sticks that are mounted and you can access their files here.

/root: The home folder for the root user.

/sbin: Similar to /bin but difference is that the binaries here are for root user only.

/tmp: Contains the temporary files.

/usr: Where the utilities and files shared between users od linux.

/var: Contains system logs and other variable data.

Linux File System Types:

Following are the Linux File System types:

Ext: It is an older one which is not used due to its limitations.

Ext2: It is the first Linux file system which allows 2 terabytes of data allowed.

Ext3: It is arrived from Ext2 which is more upgraded and has backward compatibility.

Ext4: It is quite faster which allows larger files to operate with significant speed.

JFS: old file system made by IBM. Working of this is very well with small and big files but when used for longer time the files get orrupted.

XFS: It is old file system which works slowly with small files.

Btrfs: made by oracle. It is not stable as Ext in some distros, but you can say that it is replacement for it if you too. It has a good performance.

Working of file system in Linux:

The Linux file system unifies all physical hard drives and partitions into a single directory structure. It starts at the top-the root directory.

Storing file in Linux:

In Linux as in MS-DOS and Microsoft Windows, program is stored in files. It can be launched by simply typing its filename. However, this assumes that the file is stored in one of a series of directories known as path. A directory included in this series is said to be on a path.

Linux File Commands:

1. pwd – This command displays the present working directory where you are currently in.

2. ls – This command will list the content of present directory.

```
~$ ls
 HI   Hello   LinuxFiles   Numbers  'Welcome to CoCalc.term'   World   demo
 example.txt   exshell.sh   help
~$
```

3. ls -l – This command is used to show formatted listing of files and directory

```
~$ ls -l
total 8
drwxr-xr-x 2 user user   2 Apr  2 12:40  HI
drwxr-xr-x 2 user user   2 Apr  2 12:40  Hello
drwxr-xr-x 2 user user   2 Apr  2 12:41  LinuxFiles
-rw-r--r-- 1 user user 113 Apr  2 13:56  Shell.sh
-rw-r--r-- 1 user user   0 Apr  2 14:06 'Welcome to CoCalc.term'
drwxr-xr-x 2 user user   4 Apr  2 14:34  demo
drwxr-xr-x 2 user user   4 Apr  2 14:41  demo1
drwxr-xr-x 2 user user   2 Apr  2 14:09  example
-rw-r--r-- 1 user user  31 Apr  2 15:00  example.txt
-rw-r--r-- 1 user user  42 Apr  2 14:48  hello.txt
drwxr-xr-x 2 user user   2 Apr  2 12:41  help
-rw-r--r-- 1 user user 111 Apr  2 12:42  num.txt
~$
```

4. ls -la – This command will list all the content of present directory including the hidden files and directories.

```
~$ ls -la
total 27
drwxr-xr-x 12 user user    24 Apr  2 14:06  .
drwxr-xr-x  1 root root  4096 Mar 31 17:05  ..
-rw-r--r--  1 user user 28812 Apr  2 14:07 '.Welcome to CoCalc.term-0.term'
-rw-r--r--  1 user user   178 Apr  2 14:06 '.Welcome to CoCalc.term-1.term'
-rw-r--r--  1 user user    48 Apr  2 14:06 '.Welcome to CoCalc.term-2.term'
lrwxrwxrwx  1 user user    18 Apr  2 11:27  .bash_profile -> /home/user/.bashrc
-rw-r--r--  1 user user  2355 Apr  2 11:27  .bashrc
drwx------  3 user user     3 Apr  2 11:27  .config
-rw-r--r--  1 user user  8192 Apr  2 11:27  .jupyter-blobs-v0.db
drwxr-xr-x  2 user user     2 Apr  2 11:27  .sage
lrwxrwxrwx  1 user user    12 Apr  2 13:49  .smc -> /tmp/.cocalc
dr-xr-xr-x  2 user user     2 Apr  2 13:54  .snapshots
drwxr-xr-x  2 user user     3 Apr  2 13:49  .ssh
-rw-------  1 user user   974 Apr  2 13:56  .viminfo
drwxr-xr-x  2 user user     2 Apr  2 12:40  HI
drwxr-xr-x  2 user user     2 Apr  2 12:40  Hello
drwxr-xr-x  2 user user     2 Apr  2 12:41  LinuxFiles
-rw-r--r--  1 user user   111 Apr  2 12:42  Numbers
-rw-r--r--  1 user user     0 Apr  2 14:06 'Welcome to CoCalc.term'
drwxr-xr-x  2 user user     2 Apr  2 12:41  World
drwxr-xr-x  2 user user     4 Apr  2 12:33  demo
-rw-r--r--  1 user user    31 Apr  2 12:46  example.txt
-rw-r--r--  1 user user   113 Apr  2 13:56  exshell.sh
drwxr-xr-x  2 user user     2 Apr  2 12:41  help
~$
```

5. mkdir – This command will create a new directory

```
~$ mkdir example
~$ ls
 HI     LinuxFiles  'Welcome to CoCalc.term'   demo      example.txt   help
 Hello  Numbers      World                     example   exshell.sh
~$
```

6. rmdir – This command will delete specified directory, provided it is empty

```
~$ rmdir World
~$ ls
 HI     LinuxFiles  'Welcome to CoCalc.term'   example       exshell.sh
 Hello  Numbers      demo                      example.txt   help
~$
```

7. cd – This command is used to change directory

```
~$ cd demo
~/demo$
```

8. cd / - This command takes us to root directory

```
~/demo/hello/world$ cd /
/$
```

9. cd .. – This command takes us one level up the directory tree.

```
~/demo/hello/world$ cd ..
~/demo/hello$
```

10. rm filename – This command deletes specified file.

```
~/demo/hello$ rm hello.txt
~/demo/hello$ ls
world
```

11. rm -r directoryname – This command deletes the specified directory along with it's contents

```
~/demo$ rm -r hello
~/demo$ ls
example.txt   file.txt
~/demo$
```

12. cp file1 file2 – This command copies contents of file **file1** into file **file2**

```
~/demo$ cp example.txt file.txt
~/demo$
```

13. mv – This command is used to rename files and directories

```
~$ mv demo2 demo1
~$
```

14. cat>filename – This command is used to create a file and open it in write mode

```
~$ cat > hello.txt
Hello
This file is created as an example.
^Z
[1]+  Stopped                 cat > hello.txt
~$
```

15. cat filename – This command is used to display content of a file

```
~$ cat hello.txt
Hello
This file is created as an example.
~$
```

Linux Commands:

Linux is an open-source free OS. It supports all administrative tasks through the terminal. This also includes file manipulation, package installation and user management.

File Commands:

- ls = Listing the entire directory
- ls -at = Show formatted listing of hidden files
- ls -lt = Sort the formatted listing by time modified
- cd dir = To change the directory user is in

- cd = Shift to home directory
- pwd = To see which directory user is working
- mkdir dir = Creating a directory to work on
- cat >file = Places the standard input into the file
- more file = Shows output of the content of the file
- head file = Shows output of the first 10 files of file
- tail file = Shows output of the last 10 files of file
- tail -f file = Shows output content of the file as it grows, starting with the last 10 lines
- touch file = Used to create or upload a file
- rm file = For deleting a file
- rm -r dir = For deleting an entire directory
- rm -f file = This will force remove the file
- rm -rf dir = This will force remove a directory
- cp file1 file2 = It'll copy the contents of file1 to file2
- cp -r dir1 dir2 = it'll copy the contents of dir1 to dir2; Also create the directory if not present.
- mv file1 file2 = It'll rename or move file1 to file2, only if file2 is existing
- ln -s file link = It creates symbolic-link to a file

Process Management:

- ps = It displays the currently working processes
- top = It displays all running process
- kill pid = It'll kill the given process (as per specific PID)
- killall proc = It kills all the process named proc
- pgkill pattern = It will kill all processes matching the pattern given
- bg = It lists stopped or bg jobs, resume a stopped job in the background
- fg = It brings the most recent job to foreground

13.7 SECURITY IN LINUX

13.7.1 Security features:

Minimal set of security features were provided by kernel. Discretionary access control: Authentication is performed outside the kernel by user-level applications such as login. Authentication Allows system administrators to redefine access control policies. Customize the

way Linux authenticates users specify encryption algorithms that protect system resources.

13.7.2 Authentication:

Default authentication: User enters username and password via login. Passwords are hashed (using MD5 or DES).Encryption cannot be reversed and stored in /etc/passwd or /etc/shadow. Pluggable authentication modules (PAMs) can reconfigure the system at run time to include enhanced authentication techniques. Example: Disallow terms found in a dictionary and require users to choose new passwords regularly. It supports smart cards, Kerberos and voice authentication

13.7.3 Access Control Methods:

Access control attributes specify file permissions and file attributes

File permissions: Combination of read, write and/or execute permissions specified for three categories: user, group and other

File attributes: Additional security mechanism supported by some file systems allow users to specify constraints on file access beyond read write and execute. Examples: append-only, immutable

Linux security module (LSM) is a framework that allows a system administrator to customize the access control policy using loadable kernel modules. Kernel uses hooks inside the access control verification code to allow LSM to enforce its access control policy. Example: SELinux which is developed by NSA, It Replaces Linux's default discretionary access control policy with a mandatory access control (MAC) policy.

Privilege inheritance: Normally a process executes with same privileges as the user who launched it .Some applications require process to execute with other user privileges

Example: passwd – setuid and setgid allow process to run with the privileges of the file owner – Improper use of setuid and setgid can lead to security breaches – LSM

Capabilities allow administrator to assign privileges to applications as opposed to users to prevent this situation.

13.8 SUMMARY

- Linux is an open source family of Unix-like Linux based kernel applications, a kernel operating system that was first released on September 17, 1991, by Linus Torvalds Linux usually included in the Linux distribution.

- Many of the available software programs, utilities, games available on Linux are freeware or open source. Even such complex programs such as Gimp,OpenOffice, StarOffice are available for free or at a low cost.
- GUI makes the system more flexible, but has the disadvantage that it's simple to implement a special interface for every program, making the system harder to find out.
- The Linux kernel consists of several important parts: process management, memory management, hardware device drivers, filesystem drivers, network management, and various other bits and pieces.
- Memory management takes care of assigning memory areas and swap file areas to processes, parts of the kernel, and for the buffer cache. Process management creates processes, and implements multitasking by switching the active process on the processor.
- Linux File System or any file system generally is a layer which is under the operating system that handles the positioning of your data on the storage.
- The Linux file system unifies all physical hard drives and partitions into a single directory structure. It starts at the top-the root directory.
- Kernel provides a minimal set of security features

13.9 LIST OF REFERENCES

1. Modern Operating Systems, Andrew S. Tanenbaum, Herbert, Pearson 4 th, 2014
2. Operating Systems – Internals and Design Principles, Willaim Stallings, Pearson 8 th, 2009
3. Operating System -Concepts Abraham Silberschatz, Peter B. Galvineg Gagne Wiley ,8 th Edition.
4. Operating Systems Godbole and Kahate McGraw Hill 3 rd Edition.

13.10 BIBLIOGRAPHY

https://www.tutorialspoint.com

https://www.geeksforgeeks.org

https://www.javatpoint.com

https://guru99.com

www.slideshare.net

13.11 UNIT END QUESTIONS

1. Explain Architecture of Linux.
2. Explain memory management in Linux
3. Write short note on Process management in Linux
4. Write short note on security in Linux

*****

14

ANDROID CASE STUDY

Unit Structure

14.0 OBJECTIVES

- To understand principles of Android
- To learn principles of Process, Memory Management
- To learn principles of File System and Security

14.1 ANDROID HISTORY

Android OSis recognized by a consortium of developers: known as the Open Handset Alliance, with the main funder and commercial marketer being Google. It was being developed by Google for all the tablets, and smartphones. Android OS first industrialize by Android Incorporated, which is located in Silicon Valley before it was developed by Google in 2005. Versions of android are as follows:

Versions	Description
1) Android versions 1.0 to 1.1: The early days	Android made this version authorized in the year 2008, with Android 1.0. This version included a group Google apps; like Gmail, Maps, Calendar, and YouTube.
2) Android 1.5 Cupcake	The first official public code for Android didn't appear until; this version 1.5 Cupcake was released in April 2009. New few features and enhancements, compared to the first two versions i.e.; including ability to upload videos to YouTube; and a way for a phone's screen display to automatically rotate to the right positions.
3) Android 1.6 Donut	Google released this version of Android 1.6 Donut in September 2009. It included the support for carriers that used CDMA - based networks and phones to be sold by all carriers around the world. Others are; Quick Search Box, and quick toggling between the Camera, Camcorder, and Gallery to streamline the media-capture experience & even Power Control

	widget for the Wi-Fi, Bluetooth, GPS, etc.
4) Android 2.0 - 2.1 Éclair	Google launched the second version of Android and named it as éclair in October 2009.First version of Android; with a text-to-speech support feature & also included: multiple account support, navigation with Google Maps. The first smartphone with Éclair version was Motorola Droid, which was also the first one, that was sold by Verizon wireless company.
5) Android 2.2 Froyo	Released in May 2010, also called ―frozen yogurt‖. New features, including Wi-Fi mobile hotspot functions, push notifications via Android Cloud to Device Messaging (C2DM) service, flash support and also Wi-Fi mobile hotspot functions were introduced.
6) Android 2.3 Gingerbread	Launched in Sept. 2010, is currently, the oldest versions of the OS that Google. Android devices are currently running on this version. The first mobile phone was the Nexus S mobile to add both Gingerbread and NFC hardware, co-developed by Google and Samsung. It also introduced features like selfie, by adding in support for multiple cameras and video chat support within Google Talk.
7) Android 3.0 Honeycomb	This Version introduced in Feb, 2011, Motorola Xoom tablet along with, was released by Google only for tablets and other mobile devices with larger displays than normal smartphones. Honeycomb would offer specific features that could not be handled by the smaller displays found on smartphones at the time.
8) Android 4.0 Ice Cream Sandwich	Launched in October 2011. First to introduce the feature to unlock the phone using its camera. Other features are support for all the on-screen buttons, the ability to

	monitor the mobile and Wi-Fi data usage, and swipe gestures to dismiss notifications and browser tabs.
9) Android 4.1- 4.3 Jelly Bean	Google released versions 4.2 and 4.3, both under the Jelly Bean label, in Oct. 2012 and July 2013. Features include software updates, notifications that showed more content or action buttons, along with full support for the Android version of Google's Chrome web browser. Google now made Search, and "Project Butter to speed up and improve touch responsiveness
10) Android 4.4 KitKat	Officially launched in Sept. 2013, codename is —Key Lime Pie‖. It helped to expand the overall market &was optimized to run on the smartphones that had as little as 512 MB of RAM. This allowed many makers to get the latest version & installed it on a much cheaper handset.
11) Android 5.0 Lollipop	Released in the first month of 2014.This included the support for dual-SIM Card Feature, HD Voice calls, Device Protection to keep thieves locked out of your phone even after a factory reset.
12)Android 6.0	Marshmallow Initially called as Macadamia Nut Cookie, but later was released as Marshmallow in May 2015. Features are app drawer & the first version that had native support for unlocking the smartphone with biometrics, Type C support & Android pay was also there. Google's Nexus 6P and Nexus 5X were the first handsets.
13) Android 7.0 Nougat	Released in August 2016. Multitasking features that designed for smartphones with bigger screens. It included a split-screen feature and fast switching between applications. Other changes are made by Google such as switching to a new JIT compiler that could speed. Pixel, and Pixel XL, and LG

	V20 were released with this version.
14) Android 8.0 Oreo (August 21, 2017)	Second time Google used a trademarked name for it's Android version, the first was KitKat. Android 8.0 Oreo launched in August 2017. It included, visual changes such as native support for picture-in-picture mode feature, new autofill APIs;help in better managing the passwords and fill data, notification hannels. 15)Android 9.0 Pie (August 6, 2018)
15)Android 9.0 Pie (August 6, 2018)	Released in August 2018. New features & updates such as battery life. Home - button was added in this version. When swiped up it brings the apps that were used recently, a search bar, and suggestions of five apps at the bottom of the screen. New option added of swiping left to see the currently running applications.
16) Android 10 (September 3, 2019)	Finally, Google opted to drop the tradition of naming the Android version after sweets, desserts, and candies. It was launched in September 2019. Several new features were added such as support for the upcoming foldable smart phones with flexible displays. Android 10 also has a dark mode feature, along with the newly introduced navigation control using gestures, the feature for smart reply for all the messaging apps, and a sharing menu that is more effective.

14.2 ANDROID OVERVIEW

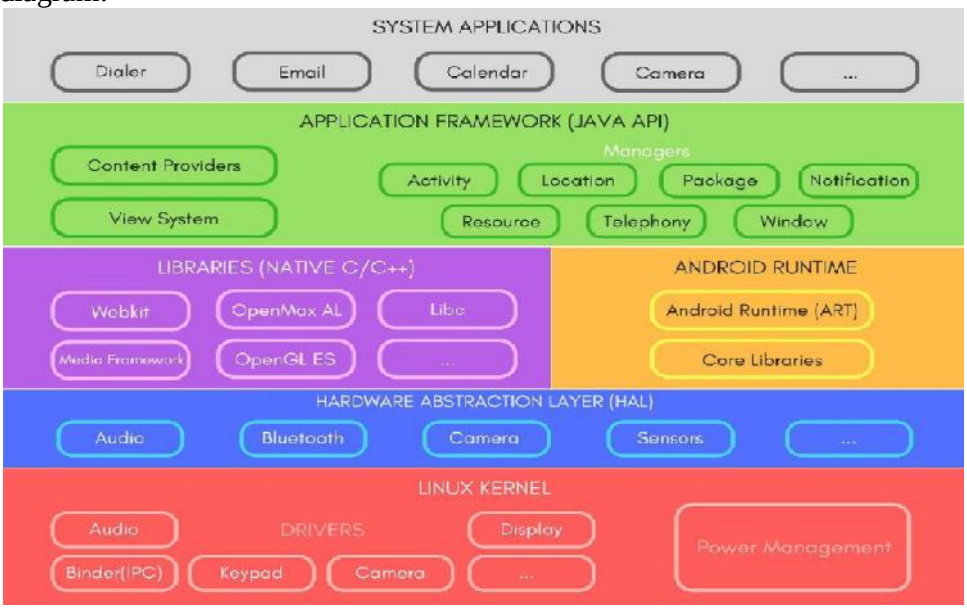
Android is an operating system based on the Linux kernel and other open-source software such as smartphones and table ts. Android approach to application development for mobile devices which means developers need only develop for Android and their applications should be able to run on different devices powered by Android. The source code for Android is available free and open-source software licenses.

14.2.1 Features:

Android is an operating system as well as supports great features. Few of them are listed below:

1. **Beautiful UI:** Android OS provides a beautiful and intuitive user interface.
2. **Connectivity:** Supports a large group of networks like GSM/EDGE, CDMA, UMTS, Bluetooth, WiFi, LTE, and WiMAX.
3. **Storage:** Uses SQLite, lightweight relational database storage for data storage. It is really helpful when there is limited mobile memory storage to be considered.
4. **Media support:** It Includes support for a large number of media formats, Audio as well as for Video, like H.263, H.264, MPEG 4 SP, AMR, AMR WB, AAC, MP3, JPEG, PNG, GIF & BMP.
5. **Messaging:** Both SMS and MMS are supported.
6. **Web Browser:** Based on Open Source WebKit, now known as Chrome.
7. **Multi-Touch:** Supports multi-touch screen. -
8. **Multi-Task:** Supports application multitasking.i.e, task to another and same time various applications can run simultaneously.
9. **Resizable widgets**: Widgets are resizable, so users can reuseit to show more content or to save space.
10. **Multi-Language**: Supports single direction and bi-irectionaltext.
11. **Hardware Support:** Accelerometer Sensor, Camera, DigitalN Compass, Proximity Sensor & GPS, and a lot more.

14.2.2 Android Architecture: Android is a stack of components of the software which is divided into five layers that are shown below in the diagram:



All these layers are responsible for different roles and features that have been discussed below.

Linux Kernel:

This layer provides a level of abstraction between hardware and it contains all the essential hardware drivers like camera, keypad, display. This layer is the foundation of the android platform.

Hardware Abstraction Layer:

It provides an abstraction between hardware and the rest ofthe software stack.

Libraries:

Libraries are present above the Linux kernel including open-source web browser engine WebKit it is the well-known library, SQLite database which is useful for storage and sharing of application data, libraries to play, and record audio and video, SSL libraries are responsible for Internet security, etc.

Android Runtime:

This layer provides a key component called Dalvik Virtual Machine which is a kind of Java Virtual Machine. JVM is specially designed and optimized for Android and designed to run apps in a constrained environment that has limited muscle power in terms of battery, processing, and memory. It contains a set of libraries that enables developers to write code for android apps using java programming.

Application Framework:

It provides a higher level of services to applications in the form of java classes. Developers are allowed to make the use of these services in their applications.

Android framework includes key services ofter are as follows:

1) **Activity Manager:** It controls all aspects of the application lifecycle and activity stack.
2) **Content Providers:** Allows applications to publish and share their data with other applications.
3) **Resource Manager:** Provides access to non-code embedded resources such as strings, color settings, and also user interface layouts.
4) **Notifications Manager:** Allows applications to display notifications to the user.

5) **View System**: Extensible set of views used to create application user interfaces.

Applications:

At the top, the layer you will find all android applications. This layer uses all the layers below it for the proper functioning of the mobile app, such applications are Contacts Books, Calendar, Browser, Games, and many more.

So Android holds layered or we can say a group of functionalities as software stack that makes Android work very fluently in any device.

14.3 ANDROID PROGRAMMING

If we want to develop Android apps, it is essential to pick a language. To differentiate between the various Android programming languages it may be a little complex. To choose which one to start with it requires an understanding of their strength and weakness.

The best way to develop an Android app is to download Android Studio. There is a piece of software called an Integrated Development Environment(IDE). It is offered as a package with the Android SDK, which is nothing but a set of tools used to facilitate Android development. It will give you everything you need in one place to get up and get going.

Features such as the visual designer make the process easier. Powerful features are being added to give developers access to things like cloud storage. While Java is the official.language for Android but there are so many other languages that can be used for Android App Development.

Below mentioned are these programming languages which are currently used for Android development:

1. Java:

- Java is the official.language for Android App Development and it is the most used language as well. Apps in the Play Store are most of built with Java and it is also the most supported language by Google. Java has a great online community for support in case of any problems.
- Java was developed by Sun Microsystems in 1995, and it is used for a wide range of programming.applications. Java code is run by a virtual machine. That runs on Android devices and interprets the code.
- However, Java is complicated. Language for a beginner to use as it contains complex topics like constructors, null pointer, exceptions, concurrency, checked exceptions, etc. Android Software Development Kit(SDK) increases the complexity at a greater extent.

- Development using java also requires a basic understanding of concepts like Gradle, Android Manifest, and the markup language XML.

2. Kotlin:

- Kotlin is a cross-platform programming language that is used as an alternative to Java for Android App Development. It has been introduced as a secondary - official‖ Java language in 2017.
- It can inter-operate with Java and it runs on the Java Virtual Machine.
- The only sizable difference is that Kotlin removes the superfluous features of Java such as null pointer exceptions and removes the necessity of ending every line with a semicolon.
- In short, It is much simpler for beginners to try as compared to Java and it can also be used as an entry point‖ for Android App Development.

3. C++:

- C++ is used for Android App Development using the Android Native Development Kit(NDK). An app cannot be created using C++ and the Native Development Kit is used to implement parts of the app in C++ native code. This helps in using C++ code libraries for the app as required.
- While C++ is useful for Android App Development in some cases, it is much more difficult to set up and is much less flexible. For applications like 3D games, this will use out some extra performance from an Android device, which means that you'll be able to use libraries written in C or C++.
- It may also lead to more bugs because of the increased complexity.So, it is better to use Java as compared to C++ as it does not provide enough gain to offset the efforts required.

4. C#:

- C# is a little bit similar to Java and so it is ideal for Android App Development. Like Java, C# also implements garbage collection sothere are fewer chances of memory leaks. And C# also has a cleaner and simpler syntax than Java which makes coding with it comparatively easier.
- Earlier, the biggest drawback of C# was that it could run only on Windows systems as it used the .NET Framework. However, this problem was handled by Xamarin.
- Android is a cross-platform implementation of the Common Language Infrastructure. Now, Xamarin. The android tool can be used to write native Android apps and share the code across multiple platforms.

5. Python:

- It is used for Android App Development even though Android doesn't support native Python development. This is done using various tools that convert the Python apps into Android Packages that can be run on Android devices.
- An example of this can be Kivy that is an open-source Python library used for developing mobile apps. It supports Android and also provides rapid app development. However, a downside to this is that there won't be native benefits for Kivy as it isn't natively supported.

6. Corona:

- It is a software development kit that is used for developing Android apps using Lua. It contains two operational modes, i.e. Corona Simulator and Corona Native. The Corona Simulator is used to build apps directly whereas the Corona Native is used to integrate the Lua code with an Android Studio project to build an app using native features.
- While Lua is a little limited as compared to Java, it is also much simpler and easy to learn. It is mostly used to create graphics applications and games but is by no means limited to that.
- We need to use a text editor like Notepad++ to enter your code and you can run said code on an emulator without even needing to compile first. When we are ready to create an APK and deploy, we will be able to do this using an online tool.

7. Unity:

- Unity is a "game engine," which means it provides things like physics calculations and 3D graphics rendering and an IDE like Android Studio.
- It is an open-source tool, which makes it incredibly easy to create our games, and the community is strong, which means we get a lot of support. With just a few lines of code, we have a basic platform game set up in less than an hour. It's multiplatform and is used by many game studios.
- It is a great way to learn object-oriented programming concept as the objects are an object.
- This is used to become a game developer.
- For a complete beginner, it is not the entry point to Android development – but for a small company wanting to create an app for iOS and Android, it makes sense and there's plenty of support and information out there to help you out.

8. PhoneGap:

- The last simple option you can choose to develop Android apps program.
- PhoneGap is powered by Apache Cordova and it allows you to create apps using the same code you would normally use to create a website: TML, CSS, and JavaScript. This is then shown through a "WebView" but packaged like an app. It acts like a mediator, which allows developers to access the basic features of the phone, such as the camera.
- This is not real Android development, though, and the only real programming will be JavaScrip

Conclusion:

- There are a lot of apps such as Chat messengers, Music players, Games. Calculators. etc. that can be created using the above languages.
- No language is correct for Android Development.
- So, it's upon you to make the correct choice of language based on your objectives and preferences for each project.

Databases that can be used with Android:

1. SQLite:

- SQLite is a relational database, a lite version of SQL designed for mobile.It is an in-process library that implements a self-contained, zero-configuration, transactional SQL database engine. Its an embedded SQL Database engine without any separate server process, unlike any other SQL database.
- SQLite supports all the relational database features.
- It is an open-source compact library that is by default present in two main Mobile OS i.e. Android and iOS.
- We can store SQLite both on disk as well as in memory. Each database file is a single disk file and it can be used for cross-platform. It requires very little memory to operate and is very fast.

2. Firebase:

- With Firebase, we can focus our time and attention on developing the best possible applications for our business. The operation and internal functions are very solid. They have taken care of the Firebase Interface. We can spend more time in developing high-quality apps that users want to use.

There are the following features which we can develop:

- **Cloud Messaging**: Firebase allows us to deliver and receive messages in a more reliable way across platforms.

- **Authentication**: Firebase has little friction with acclaimed authentication.
- **Hosting**: Firebase delivers web content faster.
- **Remote Configuration**: It allows us to customize our app on the go.
- **Dynamic Links**: Dynamic Links are smart URLs that dynamically change behavior for providing the best experience across different platforms.
- These links allow app users to take directly to the content of their interest after installing the app - no matter whether they are completely new or lifetime customers.
- **Crash Reporting**: It keeps our app stable.
- **Real-time Database**: It can store and sync app data in real-time.
- **Storage**: We can easily store the file in the database.

3. Realm DB:

The realm is a relational database management system which is like a conventional database that data can be queried, filtered, and persisted but also have objects which are life and fully reactive.

Realm database is developed by Realm and specially designed to run on mobile devices, Like SQLite, Realm is also serverless and cross-platform. It can be stored in the disk as well as in memory.

Realm has so many advantages over native SQLite, like:

- As we work with the real object there is no need to copy, modify, and save the object from the database.
- The realm is much faster than SQLite. Realm can query up to 57 record/sec, whereas SQLite can do only up to 20 record/sec.
- Data is secured with transparent encryption and decryption.
- Realm database has a reactive architecture, which means it can be directly connected to UI, if data changes it will automatically refresh and appear on the screen.
- One application can have multiple Realm database, both local and remote Can set different permissions for different users.

4. ORMLite:

- It is a lighter version of Object Relational Mapping which provides some simple functionality for persisting java object to SQL database. It is ORM wrapper over any mobile SQL related database.

- It is used to simplify complicated SQL operations by providing a flexible query builder. It also provides powerful abstract Database Access Object (DAO) classes.
- It is helpful in big size applications with complex queries because it handles "compiled" SQL statements for repetitive query tasks. It also supports for configuring of tables and fields without annotations and supports native calls to Android SQLite databases APIs.
- It doesn't fulfill all the requirements like it is bulky compared to SQLite or
- Realm, slower than SQLite and Realm but faster than most of the other ORMs present in the market.

14.4 PROCESS

14.4.1 Introduction:

A process is a - program in execution. It is generally used to accomplish a task, a process needs resources. For instance, CPU file, memory, etc. Resources are allocated to processes in two stages

- The stage when the process was created
- Dynamically allocate the process while they are running

A process is more than coding or program of code; it also includes current activity, the content of processor's registers, etc. A program can also be called a passive entity and a process can also be called an active entity. It also contains a feature which is known as a program counter which is responsible for specifying the next instruction to be executed. E.g. Word processor can be thought of as a process. So when we talk about a passive entity it would be like a file containing a set of instructions saved on a disk which is also known as an executable file, whereas a process is meant to be an active entity which is backed by a program counter which in turn specifies the next instruction to be executed along with a set of associated resources. In other words, the program converts into a process when it is loaded into memory.

14.4.2 Process in the application:

All components of the same application run in the same process and most applications do not change this. However, if Developer finds that Developer needs to control which process a certain component belongs to, the developer can do so in the manifest file. The manifest entry for each type of component element like

1. <activity>
2. <service>
3. <receiver>
4. <provider>
5. <application>-to set a default value that applies to all components.

It supports an android: process attribute that can specify a process in which that component should run. Developers can set this attribute so that each component runs in its processor so that some components share a process while others do not. Developers can also set android: process so that components of different applications run in the same process means it provided that the applications share the same Linux user ID and are signed with the same certificates.

Sometimes, Android might decide to shut down a process at some point, when memory is low and required by other processes that are more immediately serving the user. Application components running in the process that's killed are consequently destroyed and a process is started again for those components when there's again work for them to do. While determining which processes to kill, the Android device weighs their relative importance to the user. For instance, it simply shuts down a process hosting activities that can be not seen on the display screen, in comparison to a process hosting seen activities. The selection of whether or not to terminate a process, consequently, depends on the state of the components running in that procedure.

14.4.3 Structure of Process:

- **Stack:** contains temporary data such as parameters of the function and return addresses also local variables.
- **Heap:** is dynamically allocated memory during process run time.
- **Data:** includes global variables
- **Text:** includes the current activity which is represented by the value of the program counter and the contents of the processor's registers.

14.4.4 States of process:

The Major transition states of the process are as follows:

1. **New:** Process is created
2. **Running:** Instructions are executed
3. **Ready**: When the Process is ready to get executed and is waiting to get assigned to the processor
4. **Waiting:** Any event of the line need to be performed hence the process is waiting for that event to occur.
5. **Terminated:** Execution or process is accomplished The transition states that are represented above found on all systems but certain operating systems also more finely delineate states of the process. The names vary on different operating systems. There can be more than one process that is available which is in the ready or waiting state but at any instant of time, only one process can be in running state at one processor of the operating system.

14.4.5 Process Lifecycle:

The Android system tries to maintain a process for as long as its possible but eventually needs to remove the old processes to reclaim memory for more important processes. To find which processes to keep and which to kill, the system puts each process into an "importance hierarchy" based on the components running in the process and the state of components. Processes that have the lowest importance are eliminated first, then those with the next lowest importance, and so on, as necessary to recover system resources.

There are a total of five levels in the hierarchy. The following lists show the different types of processes in order of importance:

1. Foreground process:

A process that is required for what the user is doing. A process is considered to be in the foreground if any of the following conditions are true:

- It hosts an Activity that the user is interacting with (the on Resume() method).
- It hosts the Service that's bound to the activity that the user is interacting with.
- It hosts the Service that's running "in the foreground"—the service has called startForeground().
- It hosts the Service that's executing one of its lifecycle callbacks (onCreate()or onStart(), or onDestroy())
- It hosts another BroadcastReceiver that's executing its onReceive() method.

2. Generally, only a few foreground processes exist at any given time. They are killed only as a last resort—if memory is so low that they cannot all continue to run. Generally, at that point, the device has reached a memory paging state, so killing some foreground processes is required to keep the user interface responsive.

3. Visible process:

A process that does not have any foreground components, but still can affect what the user sees on the screen. A process is considered to be visible if all the following conditions are true: It hosts an Activity that is not in the foreground, but it is still visible to its user (its onPause() method called). This could occur, for eg if the foreground activity starts a dialog, which would allow the previous activity to be seen behind it. It hosts the Service that is bound to be visible (or foreground) activity. A visible process is considered extremely important and will not be killed unless doing so is required to keep all foreground processes running.

4. Service process:

A process that is running on the service that has been started with the startService() method and does not fall into either of the two higher categories. Although service processes are not directly tied to anything the user sees, they are generally doing what the user cares about (like playing music in the background or downloading data on the network), and so the system keeps them running unless there's not enough memory to remember them along with all foreground and visible processes in them.

5. Background process:

A process that is holding an activity which is not currently visible to the user (the activity's onStop() called). These processes have no direct impact on their user experience, and the system can kill them at any time to reclaim memory for a foreground, visible, or service process. Usually, many background processes are running, so they are kept in the least recently used list to ensure its process with the activity that was most recently seen by the user is last to be killed. If the activity implements the lifecycle methods correctly, and saves its current state, killing its process will have no visible effect on its user experience because when its user navigates back to its activity, the activity restores all of its visible states.

6. Empty process:

A process that does not hold any active application components is the only reason to keep the process alive is for its caching purposes, to improve startup time for its next time a component needs to run it. The system sometimes kills these processes to balance the system resources between its process caches and the underlying kernel caches

Android ranks process at the highest level it can, based on the importance of their components which are currently active in the process. For eg. if the process hosts a service and a visible activity, the process is ranked as a visible process, not a service process.

In addition to it, any process's ranking might increase because other processes are dependent on it so a process that is serving another process can be never ranked lower than another process it is serving. For example, if the content provider in process A is serving a client in process B, or if the service in process A is bound to a component in process B, process A is considered at least important as process B.

Because when a process is running the service is ranked higher than the process with its background activities, an activity that is long-running operation might do well to start the service for that operation, rather than simply creating the worker thread - particularly if the operation is likely to outlast the activity. For e.g. an activity that's uploading the picture to a web site will be starting a service to perform the upload so the

upload will continue in the background even if the user leaves the activity. Using the service guarantees that all the operations will be having at least "service process" priority, regardless of what happens to the activity. This is the same reason why the broadcast receivers should always employ services rather than simply put time-consuming operations in a thread.

14.4.6 Inter process Communication:

Android hosts a variety of applications and is designed in a way that removes any duplication or redundancy of functionalities in different applications or to allow functionality to be discovered, etc.

There are two major techniques related to the inter process communication and they are namely;

Intents: These enable the application to select an Activity based on the action you want to invoke and the data on which they operate. Path to an application is needed to use its functions and exchange data with it. With intent objects, data can be used to pass objects in both directions. It enables high-level communication.

Remote methods: By this we mean the remote procedure calls with these APIs can be accessed remotely. With this calls the methods to appear to be local which are executed in another process.

Android app avoids interprocess communication. It provides functions in terms of packages loaded by applications that require them. For applications to exchange data applications need to use file system or other traditional Unix/Linux mechanisms,

14.5 ANDROID MEMORY MANAGEMENT

14.5.1 Introduction:

In Android memory management instead of providing swap space, it uses paging and a map which means at your application touches cannot be paid until you release all your preferences now in Android the Dalvik virtual Machine heap size for the application process limited and the size of 2MB and the maximum allocation is limited to 36 MB examples of large applications are photo editor, camera, gallery, and home screen.

The background application processes in Android are stored in the LRU cache. According to the cat strategy, it will kill processes in the system when it runs slow and it will also consider the application which is the largest consumer.

If kind wants to make an app run and live longer in the background only to deallocate unnecessary memory in the four more into the

background system will generate an error message or terminate the application.

14.5.2 Garbage Collection:

The Dalvik virtual machine maintains track of memory allocation. Once it gets to know that memory is no longer used by any program if freeze into a heap without any participation from the programmer. it has basic two basic goals i.e. to find objects in a program that cannot be used in the future and second is to reclaim the resources used by the particular objects. Android memory heap is purely based on the life and size of an object been allocated.

The duration of garbage collection depends upon the generation of objects and collecting and how many active objects are there in each of the generations.

The memory heap of android may be a generalized one, meaning that there are different allocations that it tracks, supported the expected life and size of an object being allocated. For example, recently allocated objects belong within the Young Generation.

Each heap generation has its dedicated upper limit on the quantity of memory that objects there can occupy. Any time a generation starts to refill, the system executes a garbage pickup event to release memory.

Even though garbage pickup is often quite fast, it can still affect your app's performance. You don't generally control when a garbage pickup event occurs from within your code.

When the standards are satisfied, the system stops executing the method and begins garbage pickup. If garbage pickup occurs within the middle of an intensive processing loop like animation or during music playback, it can increase the time interval.

14.5.3 How to improve memory usage:

1. One should take care of the design pattern with fractions it can help to build a more flexible software architect. In the mobile world, abstraction may involve side effects for its extra code to be executed, which will cost more time and memory. Unless abstraction can provide application is a significant benefit.
2. Avoid using "enum". Do not use the enum because it doubles the memory allocation than ordinary static constant.
3. Instead of HashMap try to use the optimized sparse array, sparseboolean array, and long sparse array containers. Hashmap allocates and entry object during every mapping which is a memory inefficient action, also the low performance behavior,

"autoboxing/unboxing" is spread all over the usage. Instead, sparse array-like containers map keys into the plane array.

4. We should avoid creating unnecessary objects. Do not allocate memory especially for the short term temporary objects if you can avoid it and garbage collection will occur less when fewer objects are created.

14.5.4 How to avoid memory leaks:

1. After creating a database one should always close the cursor and if one wants to keep the cursor open for a long time in must be used carefully and close as another database task is finished.
2. To call unregisterReceiver() after calling registerReceiver ().
3. If you declare static member variable drawable inactivity then call view.setBackground(drawable) in onCreate(), a new activity instance will be created and the old activity instance can never be deallocated because drawable has a set the view as callback and you has reference to an activity
4. To avoid this kind of leakage do not keep long references to contacts activity and id3 using the context-application instead of context activity.
5. Threads in java are the root of garbage collection that is a DVM keeps data friends to all activity threads in the runtime system and threads are left running will never be eligible for garbage collection.

14.5.5Allocate and reclaim app memory:

- Dalvik Debug Monitor Service (DDMS) is a debugging tool included in the Android studio.
- IDE to the applications running on the device is connected by DDMS.
- Every application runs in its process in Android studio, each one of which hosts it is on a virtual machine (VM) and each process listens to a debugger on the various port.
- When it begins, DDMS connects to ADB (Android Debug Bridge which is a command-line utility included with Google's Android SDK.).
- This will notify the DDMS when the device is connected or disconnected the device when connected, a Virtual machine(VM)monitoring service is created between ADB and DDMS, which will tell the DDMS when a Virtual Machine on the device is started or terminated.
- The Dalvik heap is constrained to a single virtual memory range for every app process. This defines that the logical heap size grows as it needs a limit that the system defines for each app.

- The logical size of the heap is not like the amount of physical memory used by the heap.
- When we are inspecting our app's heap, a value called the Proportional Set Size (PSS) is computed by Android, that accounts for both dirty and clean pages which are shared with other processes—but only in an amount that's proportional to how many apps shared by that RAM.
- This (PSS) total is what the system considers to be the physical memory footprint. For more information regarding PSS, see the Investigating Your RAM Usage guide.
- The Dalvik heap enables a compact of the logical size of the heap, meaning Android does not defragment the heap to the close-up space.
- Android can only shrink by the logical heap size when there is unused space at the end of the heap. Therefore, the system can still reduce the physical memory used by the heap.

After the garbage collection process, Dalvik walks the heap and finds the unused pages, then returns these pages to the kernel using the advice.

14.6 FILE SYSTEM

The Android Operating System is a popular and universally used operating system for smartphones lately. While on the user's end it might appear simple and easy to use the Android File Systems Applications tend to be rather complicated and have several users scratching their head in amusement in daily. Let us now take a detailed look at the file systems and what they have to offer to the users as Android.

This informative piece is for people who are thinking to develop ROMs, Apps, and a lot of other things on the Android operating system. Without wasting a minute more let us begin with a detailed look at the Android file system. We would not just be naming the file systems in android we would also give you a brief explanation about a particular file system in detail understanding.

14.6.1 Flash Memory- Android OS File System:

1. **exFAT:** Created by Microsoft for flash memory, the exFAT file system is not a part of the standard Linux kernel. However, it still provides support for Android devices in some cases. It stands for Extended File Allocation Table Application.
2. **F2FS:** Users of Samsung smartphones are bound to have come across this type of file system Application if they have been using the smartphone for a while. F2FS stands for Flash-Friendly File System Application, which is an Open Source Linux file system. This was introduced by Samsung 4 years ago in the year 2012.

3. **JFFs2:** It stands for the Journal Flash File System version 2. This is the default flash file system for the Android Application Open Source Project kernels. This version of the Android File System has been around since the Android Ice Cream Sandwich Operating system was released.

14.6.2 Media-based Android File System:

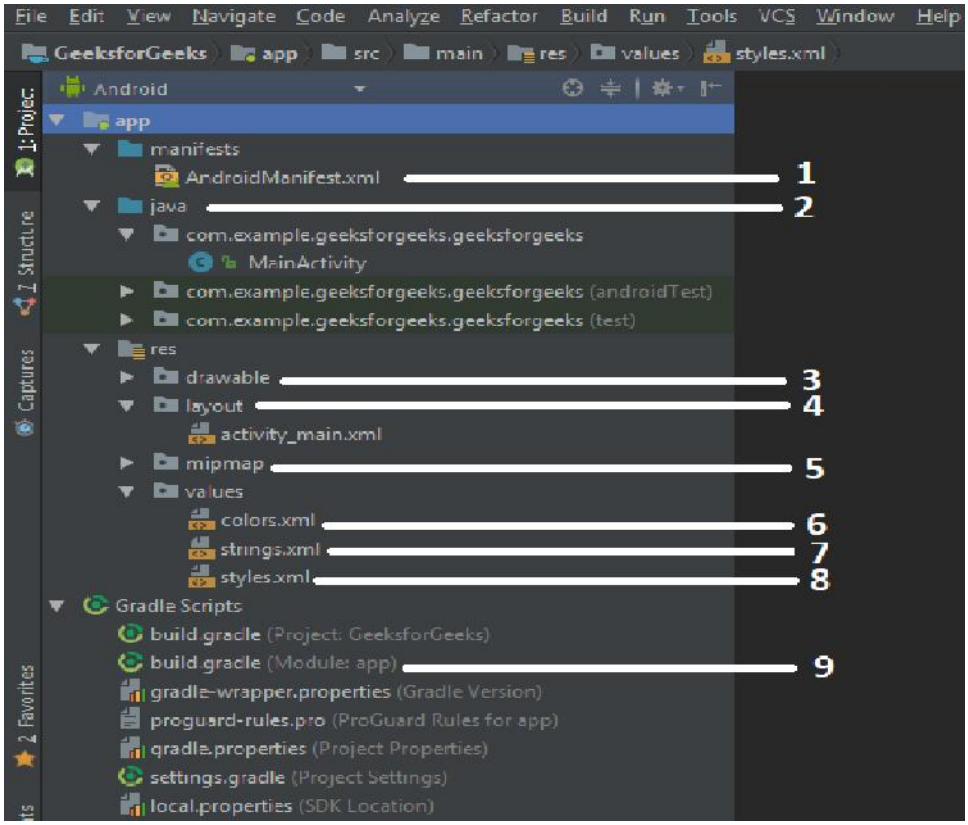
1. **EXT2/3/4:** Ext, which stands for the extended file systems, are the standards for the Linux file system. The latest out of these is the EXT4, which has now been replacing the YAFFS2 and the JFFS2 file systems on Android smartphones.
2. **MS-DOS:** Microsoft Disk Operating System is known to be one of the oldest names in the world of Operating Systems, and it helps FAT 12, FAT 16, and FAT 32 file systems to run seamlessly.
3. **vFAT:** An extension to the aforementioned FAT 12, FAT 16, and FAT 32 file systems, the vFAT is a kernel module seen alongside the msDOS module. External SD cards that help expand the storage space are formatted using VFAT.

14.6.3 Pseudo File Systems:

1. **CGroup:** Cgroup stands for Control Group. It is a pseudo-file system which allows access and meaning to various kernel parameters. Cgroups are very important for the Android File System as the Android OS makes use of these control groups for user accounting and CPU Control.
2. **Rootfs:** Rootfs acts as the mount point, and it is a minimal file system. It is located at the mount point "-".
3. **Process:** The process file system has files that showcase the live kernel data. Sometimes this file system Application development also reflects several kernel data structures. These numbers directories are reflective of process IDs for all the currently running tasks now.
4. **Systems:** Usually mounted on the /sys directory. The sysfs file system app helps the kernel identify the devices. Upon identifying a new device, the kernel builds an object.
5. **Tmpfs:** A temporary file system, tmpfs is usually mounted on /dev directory. Data on this is lost when the device is rebooted.

14.6.4 Android | Android Application File Structure :

It is very important to know about the basics of the Android StudioApplication file structure. In this, some important files/folders, and their significance is explained for the easy understanding of the android studio work environment. In the below image, several important files are marked here below in diagram picture:



14.6.5 Files in Android Studio and Explained below:

1. **AndroidManifest.xml:** Every project in Android includes a manifest file, which is manifest.xml stored in the root directory of its project hierarchy. The AndroidAppmanifest file is an important part of our app because it defines the structure and metadata of our application its components and its requirements. The file includes nodes for each of the Activities, Services providers Content ProvidersApplication and Broadcast Receiver App that make the application and using Intent Filters and Permissions, determines how they co-ordinate with each other and other Android applications.
2. **Java:** The Java folder contains the Java source code files in Application. These files are used as a controller for a controlled layout file. It gets the data from the layout file App and after processing that data output Android Application will be shown in the UI layout. It works on the backend of an Android application.
3. **Drawable:** A Drawable folder contains a resource type file (something that can be drawn). Drawables may take a variety of files like Bitmap Nine Patch, Vector (XML), Shape, Layers, States, Levels, and Scale.
4. **Layout:** A layout defines the visual structure for the user interface, such as the UI for an Android application. This Layout folder stores Layout files that are written in XML language we can add additional layout objects or widgets as child elements to gradually build a view hierarchy that defines your layout file.

5. **Mipmap:** Mipmap Android folder contains the Image Asset file that can be used in Android Studio Application. Generate the following icon types like Launcher icons, Action bar, and tab icons and Notification icons there.
6. **Colors.xml:** Colors.xml file contains color resources of the Android Application. Different color values are identified by a unique name that can be used in the Android application.
7. **Strings.xml:** The strings.xml file contains string resources of the Android application the different string value is identified by a unique name that can be used in the Android application program file also stores string array by using XML language in Application.
8. **Styles.xml:** Here styles.xml file contains resources of the theme style in the Android application. It is written in XML language for all activities in general in android.
9. **build.gradle:** This defines and implements the module-specific build configurations. We add dependencies need in the Android application here in the Gradle module.

File System provides an interface to a file system and is the factory for objects to access files and other objects in the file system. The default file system Application, obtained by invoking the method, provides defines methods to create file systems that provide access to other types of file systems.

A file system of Android is the factory for several Types:

- **GetPath method:** It converts a system-dependent *path string*, returning a Path object that may be used to locate and access a file.
- **GetPathMatcher method:** It is used to create a PathMatcher that performs match operations on paths.
- GetFileStores method: It returns an iterator over the underlying FileStore.
- **GetUserPrincipalLookupServicemethod:** It returns UserPrincipalLookupService to lookup users /groups by name of the services mentioned.
- **Watch service method:** It creates a WatchService that may be used to watch objects for changes and events.

File systems vary in some cases, the file system of android Application is a single hierarchy of files with one top-level root directory. In other cases, it may have several distinct file hierarchies, each with its top-level root directory Application. The getRootDirectories method may be used to iterate over the root directories in the system. A file system is typically composed of one or more underlying File Store that provides the storage for the files. File stores can also vary in the features they support,

and the file attributes and *meta-data* that they associate with files in this Applications.

A file system is open upon creation and can be closed by invoking a close method. Once closed any further attempt to access objects in the file system causes ClosedFileSystemException to be thrown. File systems created by the default FileSystemProvider cannot be closed in the Android application.

A FileSystem can provide read-only or read and write access to the file system. Whether or not a file system provides read-only access is established when the File System is created and can be tested by invoking it is a read-only method.

Attempts to write to file stores utilizing an object associated with a read-only file system throws ReadOnlyFileSystemException.

The android application provides many kinds of storage for applications to store their data. These storage places are shared preferences, internal and external storage SQLite storage browser, and storage via the network connection.

Internal storage is the storage of the private data on the device memory in the file system.

By default, these files are private and are accessed by only your application and get deleted, when the user deletes your android application.

14.7 SECURITY IN ANDROID

The security features provided in Android are:

14.7.1 Authentication:

Android uses the concept of user-authenticated cryptographic keys which requires cryptographic key storage facilities, service providers, and user authenticators.

On devices that possess a fingerprint sensor, the users can add more than one fingerprint to unlock the phone and accomplish different tasks. The Gatekeeper subsystem is used to perform the authentication of pattern/password in the TrustedExecution Environment (Trusty). Android 9 & higher versions also include Protected Confirmation which allows the user to formally confirm critical transactions.

14.7.2 Biometrics:

Android 9 and up consists of a BiometricPrompt API that allows the developers to integrate biometric authentication within their applications. Only strong biometrics can integrate with BiometricPrompt.

14.7.3 Encryption:

After the device is encrypted, all the data created by the user is automatically encrypted before committing it to the disk and also all the reads automatically decrypts the data before sending it back to their respective calling process. Encryption gives assurances that if an unauthorized user tried to access the data, they would not be able to read the content of the data.

14.7.4 Keystore:

The Keystore system allows you to store all cryptographic keys into a container to make it more difficult for the hacker to extract it from the device. Once keys are stored in the KeyStore, they can be used for various cryptographic procedures with the key remaining non-exportable. It offers features such as to restrict how and when the keys can be used, such as demanding authentication of users for key use and restricting usage of keys only in some cryptographic methods.

14.7.5 Trusty Trusted Execution Environment (TEE):

Trusty has access to the full power of a device's main processor and memory but remains completely isolated. Trusty's isolated position protects it from various malicious applications installed by the user and also from potential vulnerabilities that would be discovered in Android.

14.7.6 Verified Boot:

Verified boot cryptographically verifies all executable code and data which is part of the Android version that is being booted before it can be used. It ensures that all executable code comes from a trusted source, rather than from a hacker. It establishes a full chain of trust, starting from a hardware-protected root of trust to the bootloader, to the boot partition and also other verified partitions.

14.8 SUMMARY

- Android made this version authorized in the year 2008, with Android 1.0.
- Finally, Google opted to drop the tradition of naming the Android version after sweets, desserts, and candies. It was launched in September 2019.
- Android OS provides a beautiful and intuitive user interface.
- Android is a stack of components of the software which is divided into five layers.
- Android ranks process at the highest level it can, based on the importance of their components which are currently active in the process. For e.g. if the process hosts a service and a visible activity, the process is ranked as a visible process, not a service process.

- The Dalvik virtual machine maintains track of memory allocation. Once it gets to know that memory is no longer used by any program if freeze into a heap without any participation from the programmer.

14.9 LIST OF REFERENCES

1. Modern Operating Systems, Andrew S. Tanenbaum, Herbert , Pearson 4th Edition, 2014
2. Operating Systems – Internals and Design Principles, Willaim Stallings, Pearson 8th Edition, 2009
3. Operating System -Concepts Abraham Silberschatz, Peter B. Galvineg Gagne Wiley ,8 th Edition
4. Operating Systems Godbole and Kahate McGraw Hill 3 rd Edition

14.10 BIBLIOGRAPHY

https://www.tutorialspoint.com/

https://www.geeksforgeeks.org/

https://www.javatpoint.com/java-tutorial

https://guru99.com

14.11 UNIT END QUESTIONS

1. Explain Architecture of Android.
2. Explain how database connect in Android
3. Explain memory management in Android
3. Write short note on Process management in Android
4. Write short note on security in Linux

*****

15

WINDOWS CASE STUDY OBJECTIVES

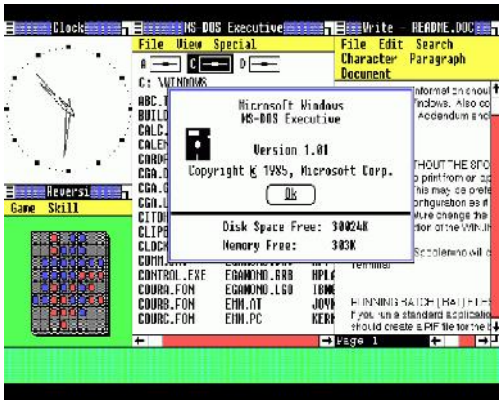
Unit Structure

15.0 OBJECTIVES

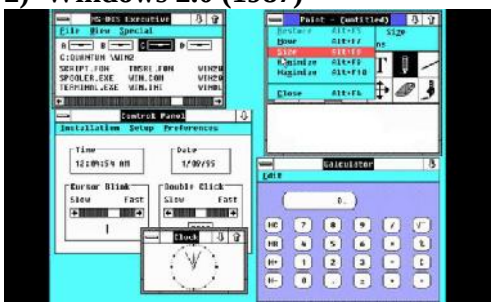
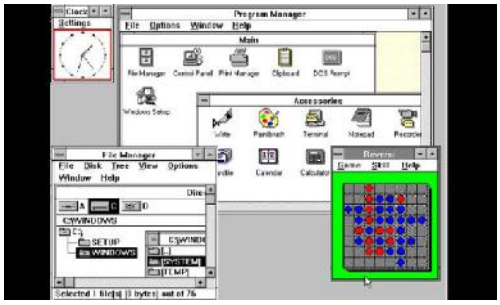
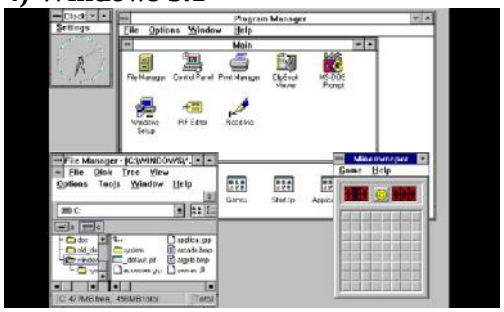
- To understand principles of Windows Operatig system
- To learn principles of Process, Memory Management
- To learn principles of IO Management, File System and Security

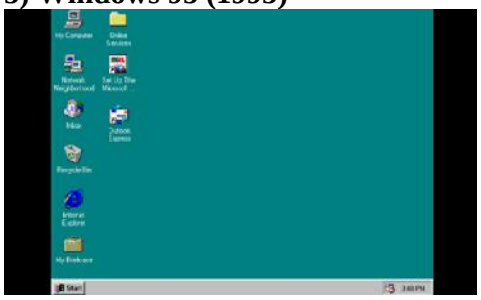
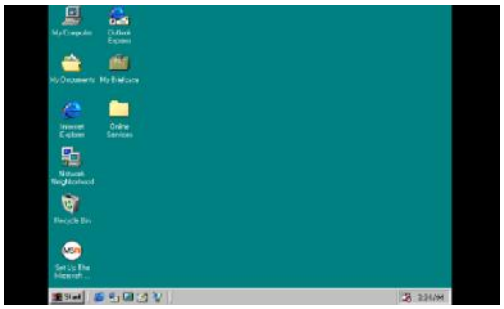
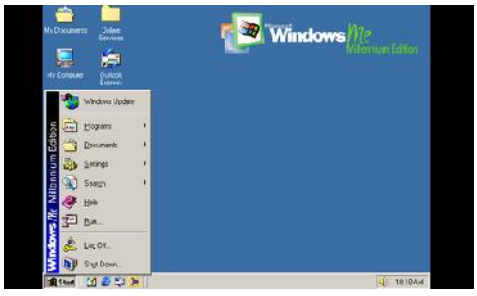
15.1 HISTORY OF WINDOWS

1) Windows 1.0 (1985)



The Windows 1 was released in November 1985 and was Microsoft's first true effort at a graphical user interface in 16-bit. It was prominent because it relied heavily on practice of a mouse before the mouse was a shared computer input device. To help users become familiar with this odd input system, Microsoft included a game, Reverse (visible in the screenshot) that relied on mouse control, not the keyboard,

	to get people used to moving the mouse everywhere and clicking onscreen elements.
2) Windows 2.0 (1987) 	• Two years after the release of Windows 1, Microsoft's Windows 2 substituted it in December 1987. • The control panel, where numerous system settings and formation options were collected together in one place, was existing in Windows 2 and survives to this day. • Microsoft Word and Excel also made their first arrivals running on Windows 2.
3) Windows 3.0 – 3.1 (1990–1994) 	• Windows 3 presented the skill to run MS-DOS programmers in windows, which took multitasking to heritage programmers, and sustained 256 colors bringing a more modern, colorful look to the interface. • More important - at least to the sum total of human time wasted - it presented the card-moving TimeSink (and mouse use trainer) Solitaire.
4) Windows 3.1 	• Minesweeper also made its original arrival. Windows 3.1 required 1MB of RAM to run and allowed maintained MS-DOS programs to be skillful with a mouse for the first time. Windows 3.1 was also the first Windows to be istributed on a CDROM, although once connected on a hard drive it only took up 10 to 15MB (a CD can naturally store up to 700MB).

5) Windows 95 (1995) 	• As the name implies, Windows 95 reached in August 1995 and with it brought the first ever Start button and Start menu. It also presented the idea of "plug and play" connect a peripheral and the operating system catches the suitable drivers for it and makes it work. • Windows 95 also presented a 32 bit environment, the task bar andengrossed on multitasking. MS - DOS still played an significant role for Windows 95, which compulsoryit to run some programmers and elements. Internet Explorer also made its debut on Windows 95, but was not installed by default needful the Windows 95 Plus! Pack•
6) Windows 98 (1998) 	• Released in June 1998, Windows 98 made on Windows 95 and transported with it IE 4, Outlook Express, Windows Address Book, Microsoft Chat and NetShow Player, which was substituted by Windows Media Player 6.2 in Windows 98 Second Edition in 1999. • USB support was much better in Windows 98 and led to its extensive adoption, including USB hubs and USB mice.
7) Windows 2000 (2000) 	• The initiative twin of ME, Windows 2000 was free in February 2000 and was formed on Microsoft's business-orientated system Windows NT and later became the foundation for Windows XP.

	• Microsoft's automatic informing played a significant role in Windows2000 and became the first Windows to support hibernation.
8) Windows ME (2000):	• Released in September 2000, it was the consumer-aimed operating system looped with Windows 2000 meant at the enterprise market. It presented some vital concepts to consumers, with more automated system recovery tools. • IE 5.5, Windows Media Player 7 and Windows Movie Maker all made their presence for the first time. Autocomplete also seemed in Windows Explorer, but the operating system was dishonorable for being buggy, failing to install properly and being generally poor.
9) Windows XP (2001) 	• It was built on Windows NT similar Windows 2000, but brought the consumer - friendly basics from Windows ME. The Start menu and task bar got a visual renovation, bringing the acquainted green Start button, blue task bar and vista wallpaper, along with several shadow and other visual effects.• • Its major problem was security though it had a firewall constructed in, it was turned off by default. Windows XP's vast approval turned out to be a boon for hackers and criminals, who browbeaten its flaws, especially in Internet Explorer, pitilessly - leading Bill Gates to pledge a Trustworthy Computing‖ initiative and the ensuing issuance of to Service Pack updates that tough XP against attack substantially.

10) Windows Vista (2007) 	• Windows XP remained the course for close to six years before being substituted by indows Vista in January 2007. Vista efficient the look and feel of Windows with more emphasis on transparent elements, search and security. Its growth, under the codename Longhorn‖, was troubled, with determined elements uncontrolled in order to get it into production. • It was buggy, loaded the user with hundreds of requirements for app permissions under User Account Control‖ the consequence of the Trustworthy Computing creativity which now meant that users had to approve or disapprove efforts by programs to make various changes. • It also ran gradually on older computers in spite of them being thought as Vista Ready‖ - a labelling that saw it sued since not all versions of Vista could run on PCs with that label.
11) Windows 7 (2009)	• It was sooner, more stable and easier to practice, becoming the operating system most users and business would advancement to from Windows XP, forgoing Vista completely. • Windows 7 saw Microsoft hit in Europe with antitrust inquiries over the pre-installing of IE, which led to a browser ballot screen being shown to original users

	allowing them to choose, which browser to connect on first boot.
12) Windows 8 (2012) 	● Released in October 2012, Windows 8 was Microsoft's most essential renovation of the Windows interface, scrapping the Start button and Start menu in favour of a more touch-friendly Start screen. The new smooth interface saw programmer icons and live tiles, which showed at-a-glance info normally related with widgets substitute the lists of programmers and icons. A desktop was still comprised, which look like Windows 7 An allowed point release to ● Windows 8 presented in ctober 2013, Windows 8.1 noticeable a shift towards yearly software updates from Microsoft and comprised the first step in Microsoft's U turn around its novel visual interface.
13) Windows 8.1 (2013) 	● Windows 8.1 re introduced the Start button, which took up the Start screen from the desktop view of Windows 8.1. Users could also select to boot straight into the desktop of Windows 8.1, which was more appropriate for those using a desktop computer with a mouse and console than the touch focused Start screen.
14) Windows 10 (2015)	● Windows 10 stays a computer operating system by Microsoft as fragment of its Windows family of operating systems. It was recognized as Threshold when it was being

	industrialized and proclaimed at a press event on 30 September 2014. Windows 10 is a Microsoft operating system for personal computers, tablets, encircled devices and internet of things devices. Microsoft free Windows 10 in July 2015 as a follow - up to Windows 8. The company has said it will update Windows 10 in eternity rather than release a new, complete operating system as a beneficiary.

15.2 PROGRAMMING WINDOWS

Microsoft Windows is a multi-tasking operating system that licenses many applications, pointy to here on out as processes. Every process in Windows is stated some quantity of time, recognized as a time slice, where the application is definite the right to control the system without being intermittent by the other processes. The runtime superiority and the quantity of time assigned to a process are acknowledged by the scheduler. The scheduler is measured as the manager of this multi-tasking operating system, making sure that each process is quantified the time and the importance it requires trusting on the current state of the system. Windows is what is acknowledged as an event-driven operating system. When that key is pushed Windows will record an event to the request that the key is down.

15.2.2 How Windows Program Work:

To make a basic application, you will originally require a compiler that performs on a Microsoft Windows operating system. Even however you can apply Win32 on many languages involving Pascal (namely Borland Delphi), we will use only one language. Actually the Win32 library is written in C, which is also the key
language of the Microsoft Windows operating systems.

Generating a Win32 Program:

All Win32 programs chiefly seem the same and act the same but, just like C++ programs, there are slight changes in terms of starting a program, trusting on the compiler you are utilizing. Here we will be challenging our programs on Borland C++ Builder, Microsoft Visual C++, and Microsoft Visual C++.NET.

For a important Win32 program, the contents of a Win32 program are alike. You will feel a difference only when you begin addition some

objects known as incomes. To create a Win32 program by means of Borland C++ Builder, you must make a console application by means of the Console Wizard.

Running Several Programs Simultaneously:

If you function with many programs concurrently you know how tedious is to run and launch them one by one! Actually, you need to find them between the other applications you have connected on your Window, click their icons and remain for them to open. This operation is fairly intolerable. That's why you require a specific shortcut (batch file) which is able to start all of them in one click! This trick will let you reach a great, time saving consequence. Opening numerous applications in a matter of a couple seconds! Stop glancing your computer folders, stop looking for the right icon. Handle all from the same place.

1. Click Start.
2. Click All Programs.
3. Click Accessories.
4. Click Notepad to open it.
5. Now write the following code:

Start "" *(confirm to leave a space before and after "") shadowed by the absolute path of the program you wish to open in quote.*

Example: Start " "
C:\Users\YourName\AppData\Local\Google\Chrome\Application\chrome.exe‖

6. Right after that press Enter and write additional line like the one overhead so as to open a new application.
7. Guarantee to write each Start ‘‘ ’’ command on a new line so that a line will encircle a Start ‘‘ ’’ command only, or else the batch file won't work and you won't be able to open many programs!
8. Now save the file with any name you wish and settle to save it as .bat extension (and not as .txt).

3. Code and Resources:
Resources are distinct as the data that you can comprise to the applications executable file resources can be:

- **standard**: icon, cursor, menu, dialog box, bitmap, improved metafile, font,
- accelerator table, message-table entry, string-table entry, or version.
- **custom**: any kind of data that doesn't fall into the previous category (for instance a mp3 file or a dictionary database).

Accessing Resources from Code:

The keys that know resources if they are clear during XAML are also used to improve specific resources if you request the resource in code. The meekest manner to recuperate a resource from code is to call also the FindResource or the TryFindResource method from framework-level objects in your application.

Creating Resources with Code:

If you want to produce a whole WPF application in code, you might also wish to make any resources in that application in code. To reach this, create a new Resource Dictionary example, and then add all the resources to the dictionary by means of succeeding calls to Resource Dictionary.Add. Then, use the Resource Dictionary thus produced to set the Resources property on an component that is current in a page scope, or the Application.Resources.

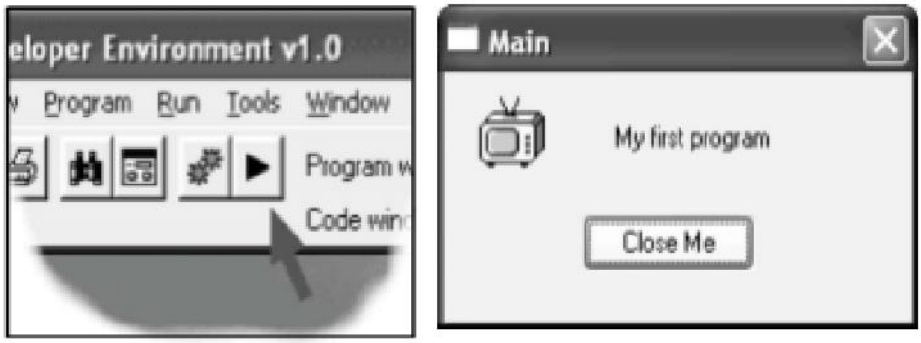
Different Data Types used in Resource File:

Microsoft Windows applications regularly depend on files that comprise nonexecutable data, such as Extensible Application Markup Language (XAML), images, video, and audio. Windows Presentation Foundation (WPF) offers superior provision for configuring, identifying, and consuming these types of data files, which are called application data files. This provision revolves around a specific set of application data file types, comprising

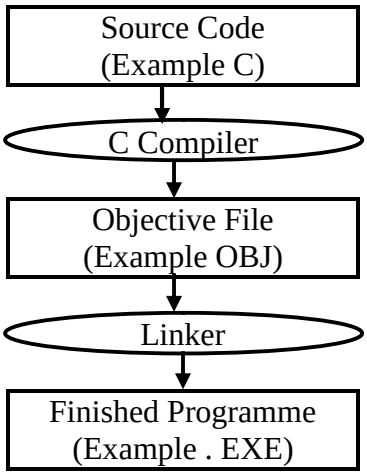
- **Resource Files:** Data files that are compiled into one or the other an executable or library WPF assembly.
- **Content Files:** Standalone data files that have an open association with an executable WPF assembly.
- **Site of Origin Files:** Standalone data files that have no suggestion with an executable WPF assembly.

6. Compiling Windows Program:

Click the Run button (displayed below with the arrow) and pause a few seconds. This compiles the program to an EXE file and runs it: When the program runs, you will detect the dialog on the screen. It appears like this:



Steps in creating a Dos Programme



Above figure shows a flow diagram for the formation of a Windows programs. In this figure the source code file gets converts to objective file from the compilers same as in the DOS. In windows programs the linker gets a few extra info from a small file called the "module definition file" with the file name extension ".DEF".This file tells the linker how to collect the program. The linker combines the module definition file info and the object file to make an incomplete .EXE file. The incomplete .EXE file absences the resource data. The main variance between Windows programs and DOS programs is in the compilation of the resource data file with the extension of ".RES". In DOS programs there is no resource data but in windows program the resource data is added to the incomplete.EXE file to create the complete executable program. The resource data is essentially stuck onto the end of the program's code and develops part of the programs file. In addition to adding the resource data the resource compiler writes the Windows version number into the program file.

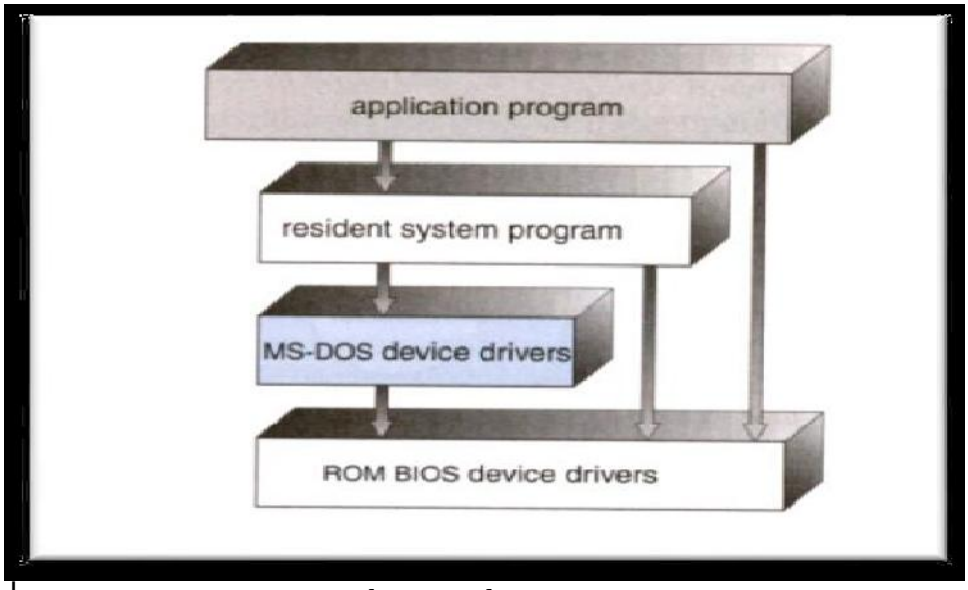
15.3 SYSTEM STRUCTURE

- User application program cooperates with system hardware through Operating System
- Operating system is such a composite structure; it should be shaped with utmost care so it can be used and adapted easily.

- An easy way to do this is to make the operating system in parts. Each of these parts should be well distinct with clear inputs, outputs and functions.
- There are two types of Structures in Windows OS: -

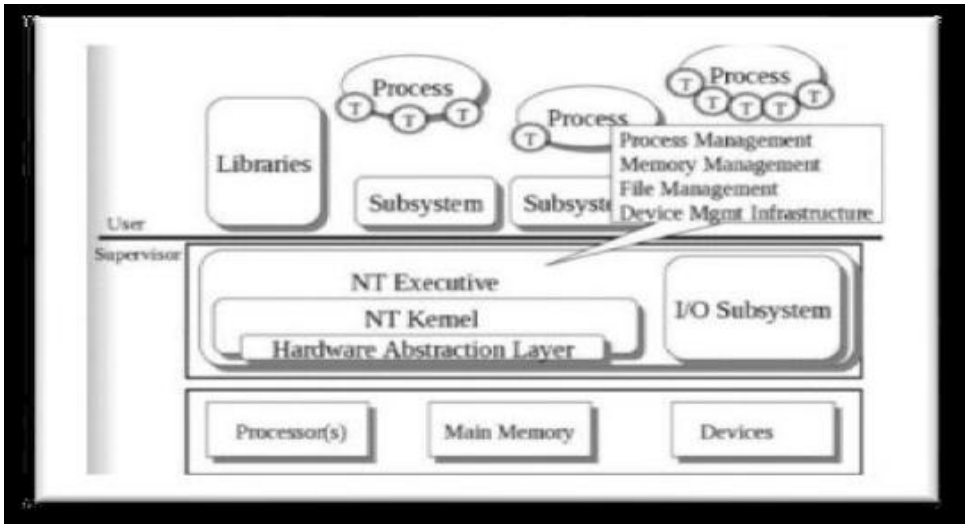
15.3.1 Simple Structure:

DIAGRAM :



- Many operating systems have modest structure.
- MS-DOS- written to deliver the most functionality in the smallest space
- Not distributed into modules
- Although MS-DOS has some structure, its interfaces and levels of functionality are not well detached

15.3.2 Layered Structure:



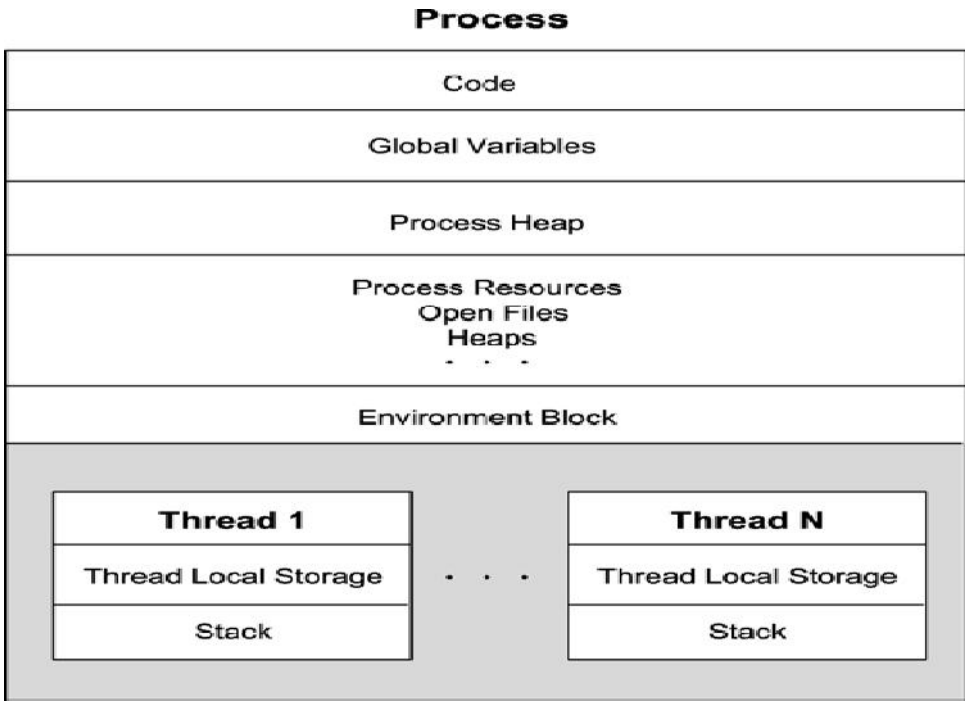
The operating system is divided into a numeral of layers (levels), each constructed on top of lower layers. The lowest layer (layer 0), is the hardware; the highest (layer N) is the user interface.

Advantage: Ease of construction and debugging. Difficulty: Defining the various layers OS inclines to be less well-organized than other applications.

15.4 PROCESS AND THREADS IN WINDOWS

An application contains of one or more processes. A *process*, in the humblest terms, is an executing program. One or more threads run in the perspective of the process. A *thread* is the basic unit to which the operating system assigns processor time.

A thread can achieve any part of the process code, with parts currently being executed by another thread. Each *process* brings the resources desirable to execute a program. A process consumes a virtual address space, executable code, open grips to system objects, a security setting, a unique process identifier, environment variables, an importance class, minimum and maximum working set sizes, and at smallest one thread of execution. Each process is continuing with a single thread, often called the *primary thread*, but can make additional threads from any of its threads. A *thread* is the object within a process that can be planned for execution. All threads of a process part its virtual address space and system resources. In adding, each thread supports exception handlers, a scheduling importance, thread local storage, a unique thread identifier, and a set of creations the system will use to save the thread setting until it is scheduled.



15.5 MEMORY MANAGEMENT IN WINDOW

Memory management is the process of directing and organizing computer memory, passing on portions called blocks to various running programs to optimize overall system performance. Memory management be located in hardware, in the OS (operating system), and in programs and applications.

Memory management is the functionality of an operating system that handles or manages primary memory and moves processes back and forth between main memory and disk throughout execution.

15.5.1 Importance of Memory Management:

Single contiguous allocation: Simplest allocation method used by MS-DOS. All memory is presented to a process.

Partitioned allocation: Memory is separated in different blocks or partitions. Each procedure is allocated according to the condition.

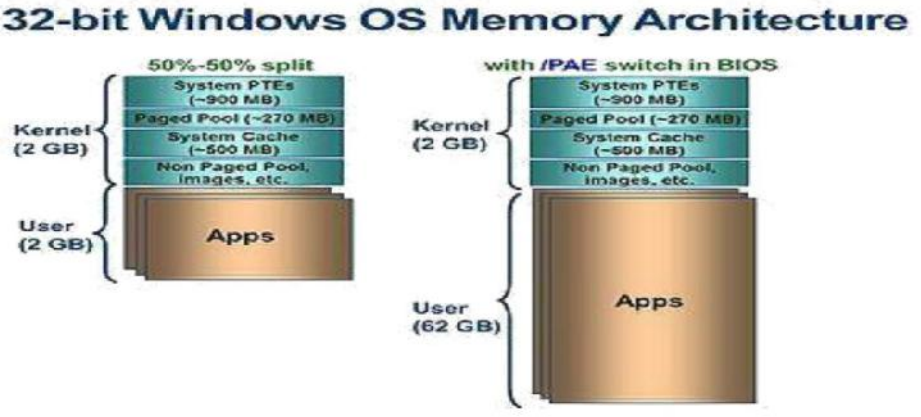
Paged memory management: Memory is divided into fixed sized units called page frames, used in a virtual memory environment.

Segmented memory management: Memory is separated in different segments (a segment is a logical grouping of the process' data or code). In this managing, assigned memory doesn't have to be contiguous. A process is separated into segments and individual segments have page

15.5.3 32-bit Windows Os Memory Management:

The 64-bit Windows Operating System addressable memory space is shared among active applications and the kernel as shown in Figure

The kernel address space contains a System Page Table Entry (PTE) area (kernel memory thread stacks), Paged Pool (page tables, kernel objects), System Cache (file cache, registry), and Non Paged Pool (images, etc.)



The default 64-bit Windows Operating System (OS) configuration offers up to 16 TB (2^54) of addressable memory space separated similarly among the kernel and the user applications.

With 16 TB of physical memory available, 8 TB virtual address (VA) space will be allocated to the kernel and 8 TB VA space to user application memory. The kernel virtual address space is shared across processes.

Each 64-bit process has its own space while each 32-bit application runs in a virtual 2 GB Windows-On-Windows (WOW).

15.5.5 Windows Uses Fifo:

- First in First Out Page Replacement Algorithm (F.I.F.O.).
- The eldest page is selected for replacement.
- It suffers from be lady's anomaly.
- Page fault rate may increase after we increase amount of frame.
- It has low performance.
- It has maximum number of page faults.

15.5.6 Catching in Window:

15.5.6.1 Caching:

Cache is a sort of memory that is used to increase the rapidity of information access. Normally, the data required for any process resides in the main memory. Though, it is moved to the cache memory temporarily if it is used commonly enough. The process of storing and retrieving information from a cache is known as caching.

15.5.6.2 Advantages of Cache Memory:

Some of the advantages of cache memory are as follows:

- Cache memory is faster than main memory as it is located on the processor chip itself. Its speed is similar to the processor registers and so often required information is stored in the cache memory.
- The memory access time is significantly fewer for cache memory as it is quite fast. This leads to faster execution of any process.
- The cache memory can store information temporarily as long as it is often required. After the use of any data has ended, it can be removed from the cache and replaced by new data from the main memory.

15.6 WINDOWS IO MANAGEMENT

A computer comprises of several devices that offer input and output (I/O) to and from the outside world. The Windows kernel-mode I/O

manager manages the communication among applications and the interfaces providing by device drivers.

15.6.1 File Buffering:

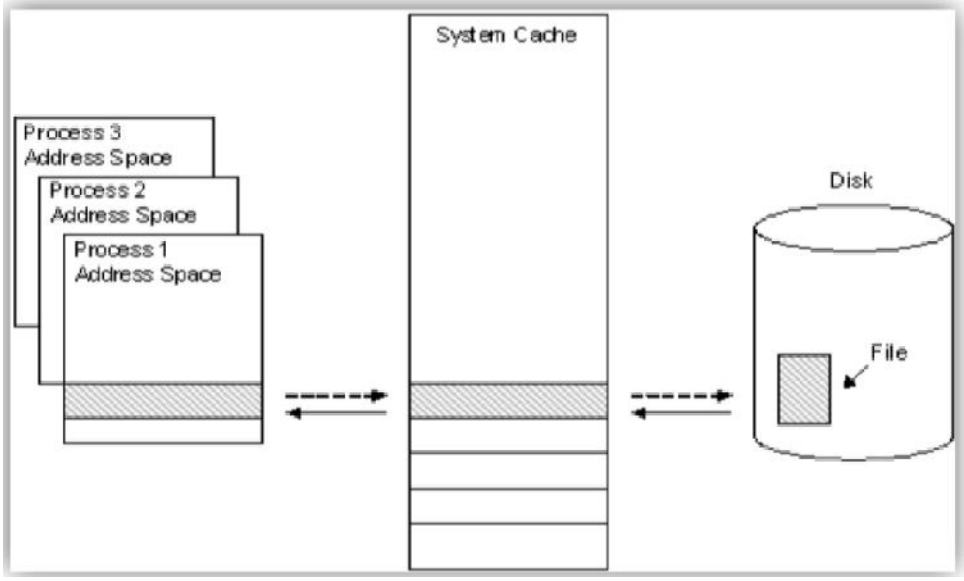
This covers the various considerations for application control of file buffering, also known as unbuffered file input/output (I/O). File buffering is usually handled by the system behind the scenes and is considered part of file caching within the Windows operating system. Although the terms *caching* and *buffering* are sometimes used interchangeably, this topic uses the term *buffering* specifically in the context of explaining how to interact with data that is not being cached (buffered) by the system, where it is then mostly out of the direct control of user-mode applications. When opening or creating a file through the CreateFile function, the FILE_FLAG_NO_BUFFERING flag can be specified to disable system caching of information being read from or written to the file. While this gives whole and direct control over data I/O buffering, in the instance of files and similar devices there are data alignment requirements that must be considered.

15.6.2 File Caching:

By default, Windows caches file information that is read from disks and written to disks. This indicates that read operations read file data from an area in system memory identified as the system file cache, instead of from the physical disk. Similarly, write operations write file data to the system file cache instead of to the disk, and this type of cache is stated to as a write-back cache. Caching is achieved per file object.

The time at which a block of file information is flushed is partly based on the quantity of time it has been kept in the cache and the amount of time later the data was last edited in a read operation. This confirms that file data that is often read will stay available in the system file cache for the maximum amount of time.

As shown by the solid arrows in the above figure, a 256 KB region of data is read into a 256 KB cache "slot" in system address space when it is first demanded by the cache manager through a file read operation. A user-mode process then copies the information in this slot to its individual address space



15.6.3 Synchronous and Asynchronous I/O:

There are two types of input/output (I/O) synchronization: synchronous I/O and asynchronous I/O. Asynchronous I/O is also denoted to as overlapped I/O.

In *synchronous file I/O*, a thread starts an I/O operation and directly enters a wait state till the I/O request has finished. A thread performs *asynchronous file I/O* sends an I/O request to the kernel by calling an proper function. If the request is acknowledged by the kernel, the calling thread remains processing another job till
the kernel signs to the thread that the I/O operation is finish. It then disturbs its current job and processes the information from the I/O operation as required.

15.7 WINDOWS NT FILE SYSTEM

NTFS (NT file system, sometimes New Technology File System) is the file system that the Windows NT operating system uses for storing and recovering files on a hard disk. NTFS is the Windows NT corresponding of the Windows 95 file allocation table (FAT) and the OS/2 High Performance File System (HPFS). However, NTFS offers a number of enhancements over FAT and HPFS in terms of performance, extendibility, and security.

Notable features of NTFS include:

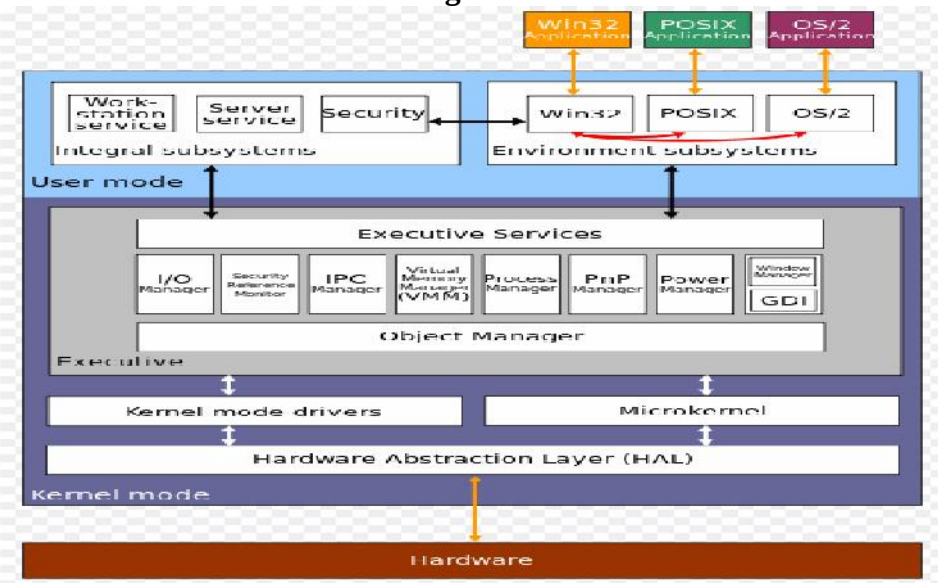
- Use of a b-tree directory structure to keep path of file clusters
- Info about a file's clusters and extra information is kept with each cluster, not just a governing table
- Support for very large files (up to 2 to the 64th power or around 16 billion bytesin extent)

- An access control list (ACL) that lets a server administrator control who can access detailed files
- Integrated file compression
- Support for names created on Unicode
- Support for long file names in addition to "8 by 3" names
- Data security on equally removable and fixed disks

15.7.1 Architecture of Windows NT:

- The design of Windows NT, a streak of operating systems formed and shifted by Microsoft, is a layered scheme that contains of two key constituents, user mode and kernel mode.
- To procedure input/output (I/O) requests, they use packet-driven I/O, which utilizes I/O request packets (IRPs) and asynchronous I/O.
- Kernel mode in Windows NT has full admission to the hardware and system resources of the computer. The Windows NT kernel is a hybrid kernel; the architecture includes a simple kernel, hardware abstraction layer (HAL), drivers, and a range of services (collectively named Executive), which all occur in kernel mode.
- User mode in Windows NT is made of subsystems accomplished of passing I/O requests to the suitable kernel mode device drivers by using the I/O manager.
- The kernel is also answerable for initializing device drivers at bootup.
- Kernel mode drivers occur in three levels: highest level drivers, intermediate drivers and low-level drivers.
- Windows Driver Model (WDM) exists in the intermediate layer and was mostly aimed to be binary and source compatible between Windows 98 and Windows 2000.
- The lowest level drivers are either legacy Windows NT device drivers that control a device straight or can be a plug and play (PnP) hardware bus.

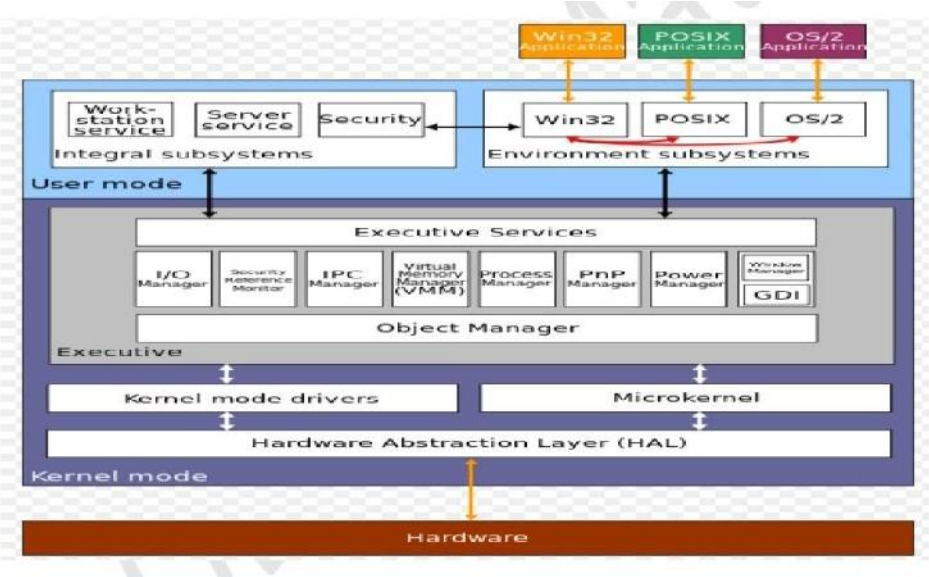
Diagram



15.7.2 Layout of NTFS volume:

The Windows NT file system (NTFS) offers a grouping of performance, dependability, and compatibility not found in the FAT file system.

The Windows NT file system (NTFS) offers a grouping of performance, dependability, and compatibility not found in the FAT file system.



15.7.2 Layout of NTFS volume:

The Windows NT file system (NTFS) offers a grouping of performance, dependability, and compatibility not found in the FAT file system.

Sections of layout of NTFS:

- Partition Boot Sector
- Master File Table (MFT)
- System Files
- File Area

15.8 WINDOWS POWER MANAGEMENT

The Windows operating system offers a complete and system-wide set of power management features. This enables systems to spread battery life and save energy, decrease heat and noise, and help ensure information reliability. The power management functions and messages retrieve the system power status, notify applications of power management events, and notify the system of each application's power necessities.

15.8.1 Edit Plan Setting in Windows:

15.8.2 Why Do we need Power Management?:

Windows power management makes computers rapidly available to users at the touch of a button or key. It also ensures that all elements of the system - applications, devices, and user interface—can take advantage of the vast improvements in power management technology and capabilities.

15.8.3 What are the benefits of Power Management?:

Eliminates start up and shutdown delays. The computer need not make a full system boot when exiting the sleep state or a complete system shutdown when the user initiates the sleep state. Allows automated tasks to run while the computer is in the sleep state. The Task Scheduler allows the user to schedule applications to run; scheduled events can run even after the system is in the sleep state. Enables perdevice power management. Enables users to generate power outlines, set alarms, and require battery options through the Power Options application in Control Panel. The operating system manages all power management activities, based on power policy settings. For more information, see the help file involved with the Power Options application. Progresses power efficiency. Power efficiency is mainly important on portable computers. Reducing system power consumption translates directly to lower energy costs and longer battery life.

15.8.4 System Power Status:

The system power status specifies whether the source of power for a computer is a system battery or AC power. For computers that use batteries, the system power status also specifies how much battery life remains and whether the battery is charging.

As of now, we are going to discuss only Six states of System Power:

- Working State (S0)
- Sleep State (Modern Standby)
- Sleep State (S1 – S3)
- Hibernate State (S4)
- Soft Off State (S5)
- Mechanical Off State (G3)

15.9 SECURITY IN WINDOWS

One of the basic beliefs of Windows Security is that each process runs on behalf of a user. So, each process running is connected with a security context.security context is a bit of cached data about a user, counting her SID, group SIDs,privileges. A security principal is an entity that can be positively recognized and confirmed via a technique known as authentication. Security principals in Windows are allocated on a process-by-process basis, via a little kernel object called a token. Each user, computer or group account is a security principal on th e system running Windows Server 2003, Windows 2000, and Windows XP. Security principal obtain permissions to access resources such as files and folders. There are 3 types of Security Principals

1) User principals

2) Machine principals

3) Service principals

Security Identifier: (SID)

Users reference their accounts by usernames but the Operating system, internally, references accounts by their security identifier. SID's are unique in their scope (domain or local) and are never reused. So, they are used to uniquely identify user and group account in Windows. By default, the operating system SID comprises of various parts S <revision><identifier authority><subauthorities><relative identifiers>

Access Token: A token is a kernel object that caches part of a user's security profile, containing the user SID, group SIDs, and privileges. A token contains of the following components. accountID, groupID, Rights, Owner, Primary group, Source, Type, Impersonation level, statistics, Restricted SID's, SessionID

Account Security: User accounts are core unit of Network security. In Win Server 2003 & Win2000, domain accounts are kept in Active Directory directories databases, where as in local accounts, they are kept in Security Accounts Manager database. The passwords for the accounts are stored and maintained by System Key. Though the accounts are protected by default, we can secure them even further. Go to Administrative tools in control panel (only when you are logged in as an admin) and click on Local Security and Settings".

Account Lock out policies: Account lockout period: Locks out the account after a specific period (1- 99,999 minutes). This feature is only exists is Win Ser 2003, Win 2000, but not in Windows XP.

Password Policies: Enforce password History: Enforces password history(0-24) Maximum password age: Set max password age(0-999) Minimum password age: Set min password age(0 to 999) Minimum password length: set min password length(0 to 14) Password must meet difficulty necessities: forces user to set complex alpha numeric passwords. Loading password using reversible encryption for users in the domain: We allow this if we want the password to be decrypted and related to pain text using procedures like Challenge Handshake authentication Protocol (CHAP) or Shiva password Authentication Protocol (SPAP)

Rights : Rights are actions or operations that an account can or cannot achieve. User Rights are of two types:

a) Privileges:

b**)** LOGON rights

Where are the passwords stored on the system?:

The system stores the passwords at machine's password strash, i.e., under HKLM/Security/Policy/Secretes. Type at 9:23am /interactive regedit.exe, substituting whatever time is appropriate: Make it one minute in the future.) Once regedit fires up, carefully look at the subkeys under HKLM/Security/Policy/Secrets. You're looking at the machine's password stash, more formally known as the LSA private data store. The operating system also, by default, caches (store locally), the last 10 passwords There are registry settings to turn this feature off or restrict the number of accounts cached.

a) Location: KEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\Current Version\Winlogon\

b) Type: REG_SZ

c) Key: CachedLogonsCount Default Value 10

d) Recommended value: 0-50 depending on your security needs.

15.10 SUMMARY

In this Chapter, we learn History of windows, Process and thread, System structure, Memory Management in Windows, Windows IO Management, Windows NT file System, Windows Power Management and Security in Windows.

15.11 LIST OF REFERENCES

1. Modern Operating Systems, Andrew S. Tanenbaum, Herbert , Pearson 4 th, 2014
2. Operating Systems – Internals and Design Principles, Willaim Stallings, Pearson 8e, 2009
3. Operating System -Concepts Abraham Silberschatz, Peter B. Galvineg Gagne Wiley ,8e
4. Operating Systems Godbole and Kahate McGraw Hill 3e

15.12 BIBLIOGRAPHY

https://www.tutorialspoint.com/

https://www.geeksforgeeks.org/

https://www.javatpoint.com/java-tutorial

https://guru99.com

https://docs.microsoft.com/

https://www.installsetupconfig.com/

15.13 UNIT END QUESTIONS

1. Explain Architecture of windows.
2. Write short note on memory management in windows
3. Explain Process management in windows
4. Write short note on security in Windows

*****