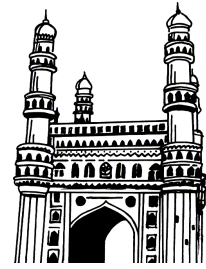


Rahul's ✓
Topper's Voice



M.C.A.

I Year II Sem

(Osmania University)

Latest 2023 Edition

OPERATING SYSTEMS

- ☞ Study Manual
- ☞ Important Questions
- ☞ Lab Practicals
- ☞ Solved Model Papers
- ☞ Previous Question Papers

- by -

WELL EXPERIENCED LECTURER

Price
199-00



Rahul PublicationsTM
Hyderabad. Cell : 9391018098, 9505799122.

All disputes are subjects to Hyderabad Jurisdiction only

M.C.A.

I Year II Sem

OPERATING SYSTEMS

Inspite of many efforts taken to present this book without errors, some errors might have crept in. Therefore we do not take any legal responsibility for such errors and omissions. However, if they are brought to our notice, they will be corrected in the next edition.

© No part of this publications should be reporduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording and/or otherwise without the prior written permission of the publisher

Price ` : 199-00

Sole Distributors :

Cell : 9391018098, 9505799122

VASU BOOK CENTRE

Shop No. 2, Beside Gokul Chat, Koti, Hyderabad.

Maternity Hospital Opp. Lane, Narayan Naik Complex, Koti, Hyderabad.

Near Andhra Bank, Subway, Sultan Bazar, Koti, Hyderabad -195.

OPERATING SYSTEMS

STUDY MANUAL

Important Questions	III - VIII
Unit - I	1 - 74
Unit - II	75 - 94
Unit - III	95 - 134
Unit - IV	135 - 164
Unit - V	165 - 177

SOLVED MODEL PAPERS

Model Paper - I	178 - 178
Model Paper - II	179 - 179
Model Paper - III	180 - 180
Lab Practicals	181 - 224

PREVIOUS QUESTION PAPERS

November - 2021	225 - 225
April - 2022	226 - 227
April / May - 2023	228 - 228

SYLLABUS

UNIT - I

Unix: Introduction, commands, file system, security and file permission, regular expression and grep, shell programming, awk.

Introduction to Operating Systems: OS structure and strategies, Process concepts, Multithreaded Programming, Process scheduling, Process synchronization, Deadlocks.

UNIT - II

Memory management strategies with example architectures: Swapping, Contiguous allocation, Paging, Segmentation, Segmentation with paging,

Virtual memory management: Demand paging, Page replacement, Thrashing.

UNIT - III

File System Interface: File concepts, Access methods and protection. File system implementation: File system structure, Allocation methods, Directory implementation of file systems, Mass storage structures, I/O systems

UNIT - IV

System Protection : Principles and Domain, Access Matrix and implementation, Access control and access rights, Capability based systems, Language based Protection,

System Security: Problem, Program threats, cryptography, user authentication, implementing security defenses, Firewalling, Computer security Classification

UNIT - V

Case Studies: The Linux System–Design principles, Kernel modules, Process management, Scheduling, Memory management, File systems, Input and Output, Inter process communication. Windows 7 –Design principles, System components, Terminal services and fast user switching File systems, Networking, Programmer interface.

Contents

UNIT - I

Topic	Page No.
1.1 Unix	1
1.1.1 Introduction, Commands	1
1.1.2 File System	3
1.1.3 Security and File Permission	3
1.1.4 Regular Expression and Grep	8
1.1.5 Shell Programming, awk	10
1.2 Introduction to Operating Systems	29
1.2.1 Os Structure And Strategies	29
1.2.2 Process Concepts	37
1.2.3 Multithreaded Programming	44
1.2.4 Process Scheduling	47
1.2.5 Process Synchronization	56
1.2.6 Deadlocks	64

UNIT - II

2.1 Memory Management Strategies with Example Architectures	75
2.1.1 Swapping	75
2.1.2 Contiguous Allocation	77
2.1.3 Paging Segmentation	80
2.1.4 Segmentation With Paging	84
2.2 Virtual Memory Management	87
2.2.1 Demand Paging	87
2.2.2 Page Replacement	87
2.2.3 Thrashing	93

UNIT - III

3.1 File System Interface	95
3.1.1 File Concepts	95
3.1.2 Access Methods And Protection	96
3.2 File System Implementation	100
3.2.1 File System Structure	100
3.2.2 Allocation Methods	102
3.2.3 Directory Implementation of File Systems	108

Topic	Page No.
3.2.4 Mass Storage Structures	114
3.2.5 I/O Systems	127
UNIT - IV	
4.1 System Protection	135
4.1.1 Principles and Domain	135
4.1.2 Domain of Protection	135
4.1.3 Access Matrix And Implementation	136
4.1.4 Access Control And Access Rights	139
4.1.5 Capability Based Systems	141
4.1.6 Language Based Protection	142
4.2 System Security	144
4.2.1 Problem	144
4.2.2 Program Threats	146
4.2.3 Cryptography	149
4.2.4 User Authentication	157
4.2.5 Implementing Security Defenses	159
4.2.6 Firewalling	162
4.2.7 Computer Security Classification	163
UNIT - V	
5.1 Case Studies	165
5.1.1 The Linux System	165
5.1.2 Design Principles	166
5.1.3 Kernel Modules	166
5.1.4 Process Management	167
5.1.5 Scheduling	168
5.1.6 Memory Management	168
5.1.7 File Systems	169
5.1.8 Input and Output	170
5.1.9 Inter Process Communication	170
5.2 Windows 7 –	171
5.2.1 Design Principles	171
5.2.2 System Components	173
5.2.3 Terminal Services And Fast User Switching File Systems	174
5.2.4 Networking	175
5.2.5 Programmer Interface	176

Important Questions

UNIT - I

1. Explain various basic unix commands with an examples.

Ans :

Refer Unit-I, Q.No. 2

2. Explain about File System in unix.

Ans :

Refer Unit-I, Q.No. 3

3. Explain about File Permission of unix.

Ans :

Refer Unit-I, Q.No. 4

4. Explain about Regular Expression in Grep of unix.

Ans :

Refer Unit-I, Q.No. 5

5. Explain briefly about various shell variables.

Ans :

Refer Unit-I, Q.No. 7

6. Explain briefly about awk.

Ans :

Refer Unit-I, Q.No. 9

7. Explain the basic syntax of awk.

Ans :

Refer Unit-I, Q.No. 11

8. What is Thread? Explain about various type of threads.

Ans :

Refer Unit-I, Q.No.23

9. Explain the concept of Process Scheduling.

Ans :

Refer Unit-I, Q.No. 26

10. Explain about various scheduling algorithms.

Ans :

Refer Unit-I, Q.No. 29

11. What is process synchronization? Explain critical section problem with its different solutions.

Ans :

Refer Unit-I, Q.No. 31

12. What is dining philosophers problem? Write about it.

Ans :

Refer Unit-I, Q.No. 36

13. Write about deadlock prevention techniques.

Ans :

Refer Unit-I, Q.No. 41

UNIT - II

1. What is Memory management? Explain briefly.

Ans :

Refer Unit-II, Q.No. 1

2. What is Swapping? Explain.

Ans :

Refer Unit-II, Q.No. 3

3. How memory is allocated to the process? Explain how to resolve dynamic storage allocation problem?

Ans :

Refer Unit-II, Q.No. 5

4. Explain the mechanism of paging segmentation.

Ans :

Refer Unit-II, Q.No. 7

5. Explain about Segmented Paging.

Ans :

Refer Unit-II, Q.No. 9

6. What is the use of page replacement algorithm? Explain about various page replacement algorithms.

Ans :

Refer Unit-II, Q.No. 11

7. Explain Belady's Anomaly with an example.

Ans :

Refer Unit-II, Q.No. 13

8. Define thrashing. Explain the techniques of thrashing.

Ans :

Refer Unit-II, Q.No. 14

UNIT - III

1. What is file? Explain about the structure of the file and its attributes.

Ans :

Refer Unit-III, Q.No. 1

2. What is file protection? Explain various types of file access control techniques

Ans :

Refer Unit-III, Q.No. 4

3. Explain the concept of file system structure.

Ans :

Refer Unit-III, Q.No. 5

4. Write about Directory structure of file system.

Ans :

Refer Unit-III, Q.No. 8

5. What are Mass (Secondary) Storage Devices? Explain.

Ans :

Refer Unit-III, Q.No. 11

6. Write about various communication I/O devices.

Ans :

Refer Unit-III, Q.No. 17

7. What is application I/O Interface? Describe various application I/O interfaces

Ans :

Refer Unit-III, Q.No. 19

UNIT - IV

1. What are the Goals and Principles of Protection?

Ans :

Refer Unit-IV, Q.No. 1

2. What is the Security Problem? Explain about the various types of Security Violations.

Ans :

Refer Unit-IV, Q.No. 9

3. What is Cryptography? Write about it.

Ans :

Refer Unit-IV, Q.No. 11

4. What is Encryption? Write about Symmetric and Asymmetric Encryption.

Ans :

Refer Unit-IV, Q.No. 12

5. Write about the implementation of cryptography in secure socket layers (SSL).

Ans :

Refer Unit-IV, Q.No. 14

6. Write about various user authentication processes.

Ans :

Refer Unit-IV, Q.No. 15

7. What is security policy? Write various security policies.

Ans :

Refer Unit-IV, Q.No. 16

8. Write about the use of firewalling in Operating system.

Ans :

Refer Unit-IV, Q.No. 17

9. Write about the Levels of Computer Security Classification.

Ans :

Refer Unit-IV, Q.No. 18

UNIT - V

1. What are the various components of Linux System ?

Ans :

Refer Unit-V, Q.No. 1

2. Discuss about the kernel modules of Linux Systems.

Ans :

Refer Unit-V, Q.No. 3

3. Write about the process management of Linux System.

Ans :

Refer Unit-V, Q.No. 4

4. Discuss how scheduling can happen in Linux System.

Ans :

Refer Unit-V, Q.No. 5

5. Explain about the inter process communication in LINUX.

Ans :

Refer Unit-V, Q.No. 9

6. Describe briefly about Windows Architecture.

Ans :

Refer Unit-V, Q.No. 11

7. Write about Windows File System.

Ans :

Refer Unit-V, Q.No. 14

8. Write about programmer interface of Windows 7.

Ans :

Refer Unit-V, Q.No. 16

UNIT I

Unix: Introduction, commands, file system, security and file permission, regular expression and grep, shell programming, awk

Introduction to Operating Systems: OS structure and strategies, Process concepts. Multithreaded Programming, Process scheduling. Process synchronization, Deadlocks.

1.1 Unix

1.1.1 Introduction, Commands

Q1. What is Unix ?

Ans :

The Unix operating system is a set of programs that act as a link between the computer and the user.

The computer programs that allocate the system resources and coordinate all the details of the computer's internals is called the operating system or the kernel.

Users communicate with the kernel through a program known as the shell. The shell is a command line interpreter; it translates commands entered by the user and converts them into a language that is understood by the kernel.

- Unix was originally developed in 1969 by a group of AT&T employees Ken Thompson, Dennis Ritchie, Douglas McIlroy, and Joe Ossanna at Bell Labs.
- There are various Unix variants available in the market. Solaris Unix, AIX, HP Unix and BSD are a few examples. Linux is also a flavor of Unix which is freely available.
- Several people can use a Unix computer at the same time; hence Unix is called a multiuser system.
- A user can also run multiple programs at the same time; hence Unix is a multitasking environment.

Q2. Explain various basic unix commands with an examples.

Ans :

(Imp.)

1. Displaying a Directory

ls—Lists the names of files in a particular Unix directory. If you type the ls command with no parameters or qualifiers, the command displays the files listed in your current working directory. When you give the ls command, you can add one or more modifiers to get additional information.

Example: ls

Result: Lists the names of files in your default directory, in alphabetical order.

Example: ls -l

Result: Gives a "long listing" of the files in your directory. In addition to the file name, the long listing shows protection information, file owner, number of characters in file, and the date and time of the last change to the file.

Example: ls -a

Result: Causes all your files to be listed, including those files that begin with a period (i.e., hidden files).

2. Displaying and Concatenating (Combining) Files

more—Enables examination of a continuous text one screenful at a time on a terminal. It normally pauses after each screenful, printing -- More -- at the bottom of the screen. Press RETURN to display one more line. Press the SPACE BAR to display another screenful. Press the letter Q to stop displaying the file.

Example: more newfile

Result: Displays the contents of "newfile" one screen ("page") at a time.

cat-- Displays the contents of a file on your terminal.

Example: cat newfile

Result: Displays the contents of the file "newfile" on your terminal.

Example: cat newfile oldfile

Result: Displays the contents of two files—"newfile" and "oldfile"—on your terminal as one continuous display.

While a file is being displayed, you can interrupt the output by pressing CTRL + C and return to the Unix system prompt. CTRL + S suspends the terminal display of the file and the processing of the command. To resume display, press CTRL + Q. The interrupted command displays lines beginning at the point at which processing was interrupted.

The cat command is also used to concatenate (combine) files and put them into another file. If you concatenate files to another one that already exists, the existing contents are permanently lost.

Example: cat fileone filetwo filethree > newfile

Result: Links together three files—fileone, filetwo, and filethree—into a new file called "newfile." The original files remain intact.

3. Copying Files

cp—Makes copies of your files. You can use it to make copies of files in your default directory, to copy files from one directory to another directory, or to copy files from other devices.

Example: cp fileone filetwo

Result: Copies the contents of fileone to a file named filetwo. Two separate files now exist.

Example: cp /usr/neighbor/testfile .

Result: Copies the file testfile from the directory /usr/neighbor to your Unix account. The period(.) at the end of the command line indicates that the file is to be copied to your current working directory and the name will remain the same.

Example: cp ~username/file1 yourfile

Result: Copies the file "file1" from user to your Unix account. The name of the file in your directory becomes yourfile. (Protections must be set for file to be readable by you in the other user's directory in order to be able to copy the file.)

4. Deleting Files

rm—Deletes specific files. You can enter more than one file specification on a command line by separating the file specifications with spaces.

Example: rm newfile

Result: Deletes the file named "newfile."

Example: rm newfile oldfile

Result: Deletes two files—"newfile" and "oldfile."

Example: rm new*

Result: Deletes all files that begin with the prefix new.

5. Renaming Files

mv—This command changes the identification (name) of one or more files.

Example: mv oldfile newfile

Result: Changes the name of the file "oldfile" to "newfile." Only one file will exist.

Example: mv oldfile bin/newfile

Result: Changes the name of the file "oldfile" to "newfile" and places it in the directory /bin. Only one file will exist.

6. Printing from Unix

The lpr command prints files on Unix. Use the -Pqueuname option to select a printer.

Example: lpr -Ppittprint sample.file

Result: This is the default output. Single-sided output, one page-worth of text per side, portrait format. Output is queued to the Pitt Print Stations.

The Unix operating system is case sensitive; type all commands in lower-case letters unless noted otherwise.

1.1.2 File System

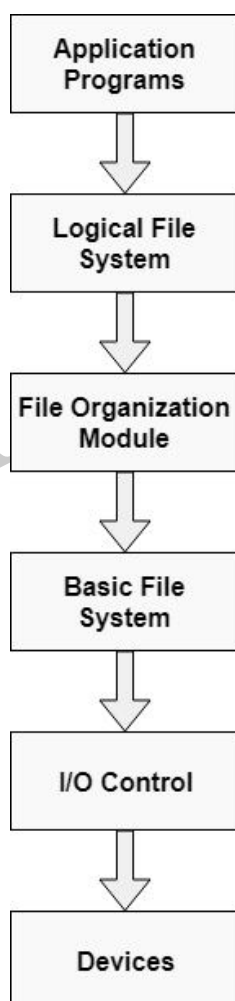
Q3. Explain about File System in unix.

Ans : (Imp.)

File System provide efficient access to the disk by allowing data to be stored, located and retrieved in a convenient way. A file System must be able to store the file, locate the file and retrieve the file.

Most of the Operating Systems use layering approach for every task including file systems. Every layer of the file system is responsible for some activities.

The image shown below, elaborates how the file system is divided in different layers, and also the functionality of each layer.



- When an application program asks for a file, the first request is directed to the logical file system. The logical file system contains the Meta data of the file and directory structure. If the application program doesn't have the required permissions of the file then this layer will throw an error. Logical file systems also verify the path to the file.
- Generally, files are divided into various logical blocks. Files are to be stored in the hard disk and to be retrieved from the hard disk. Hard disk is divided into various tracks and sectors. Therefore, in order to store and retrieve the files, the logical blocks need to be mapped to physical blocks. This mapping is done by File organization module. It is also responsible for free space management.
- Once File organization module decided which physical block the application program needs, it passes this information to basic file system. The basic file system is responsible for issuing the commands to I/O control in order to fetch those blocks.
- I/O controls contain the codes by using which it can access hard disk. These codes are known as device drivers. I/O controls are also responsible for handling interrupts.

1.1.3 Security and File Permission

Q4. Explain about File Permission of unix.

Ans : (Imp.)

1. The Permission Indicators

While using ls -l command, it displays various information related to file permission as follows -

```
$ls -l /home/amrood
```

```
-rwxr-xr-- 1 amrood users 1024 Nov 2 00:10 myfile
```

```
drwxr-xr-- 1 amrood users 1024 Nov 2 00:10 mydir
```

Here, the first column represents different access modes, i.e., the permission associated with a file or a directory.

The permissions are broken into groups of threes, and each position in the group denotes a specific permission, in this order: read (r), write (w), execute (x) -

- The first three characters (2-4) represent the permissions for the file's owner. For example, -rwxr-xr-- represents that the owner has read (r), write (w) and execute (x) permission.
- The second group of three characters (5-7) consists of the permissions for the group to which the file belongs. For example, -rwxr-xr-- represents that the group has read (r) and execute (x) permission, but no write permission.
- The last group of three characters (8-10) represents the permissions for everyone else. For example, -rwxr-xr-- represents that there is read (r) only permission.

2. File Access Modes

The permissions of a file are the first line of defense in the security of a Unix system. The basic building blocks of Unix permissions are the read, write, and execute permissions, which have been described below -

Read

Grants the capability to read, i.e., view the contents of the file.

Write

Grants the capability to modify, or remove the content of the file.

Execute

User with execute permissions can run a file as a program.

3. Directory Access Modes

Directory access modes are listed and organized in the same manner as any other file. There are a few differences that need to be mentioned -

Read

Access to a directory means that the user can read the contents. The user can look at the filenames inside the directory.

Write

Access means that the user can add or delete files from the directory.

Execute

Executing a directory doesn't really make sense, so think of this as a traverse permission.

A user must have execute access to the bin directory in order to execute the ls or the cd command.

4. Changing Permissions

To change the file or the directory permissions, you use the chmod (change mode) command. There are two ways to use chmod — the symbolic mode and the absolute mode.

5. Using chmod in Symbolic Mode

The easiest way for a beginner to modify file or directory permissions is to use the symbolic mode. With symbolic permissions you can add, delete, or specify the permission set you want by using the operators in the following table.

Sr.No.	Chmod operator & Description
1	+ Adds the designated permission(s) to a file or directory.
2	- Removes the designated permission(s) from a file or directory.
3	= Sets the designated permission(s).

Here's an example using testfile. Running `ls -l` on the testfile shows that the file's permissions are as follows -

```
$ls -l testfile
```

```
-rwxrwxr-- 1 amrood users 1024 Nov 2 00:10 testfile
```

Then each example chmod command from the preceding table is run on the testfile, followed by `ls -l`, so you can see the permission changes -

```
$chmod o+wx testfile
```

```
$ls -l testfile
```

```
-rwxrwxrwx 1 amrood users 1024 Nov 2 00:10 testfile
```

```
$chmod u-x testfile
```

```
$ls -l testfile
```

```
-rw-rwxrwx 1 amrood users 1024 Nov 2 00:10 testfile
```

```
$chmod g = rx testfile
```

```
$ls -l testfile
```

```
-rw-r-xrwx 1 amrood users 1024 Nov 2 00:10 testfile
```

Here's how you can combine these commands on a single line -

```
$chmod o+wx,u-x,g = rx testfile
```

```
$ls -l testfile
```

```
-rw-r-xrwx 1 amrood users 1024 Nov 2 00:10 testfile
```

6. Using chmod with Absolute Permissions

The second way to modify permissions with the chmod command is to use a number to specify each set of permissions for the file.

Each permission is assigned a value, as the following table shows, and the total of each set of permissions provides a number for that set.

Number	Octal Permission Representation	Ref
0	No permission	---
1	Execute permission	--x
2	Write permission	-w-
3	Execute and write permission: 1 (execute) + 2 (write) = 3	-wx
4	Read permission	r--
5	Read and execute permission: 4 (read) + 1 (execute) = 5	r-x
6	Read and write permission: 4 (read) + 2 (write) = 6	rw-
7	All permissions: 4 (read) + 2 (write) + 1 (execute) = 7	rwX

Here's an example using the testfile. Running `ls -l` on the testfile shows that the file's permissions are as follows -

```
$ls -l testfile
-rwxrwxr-- 1 amrood users 1024 Nov 2 00:10 testfile
```

Then each example `chmod` command from the preceding table is run on the testfile, followed by `ls -l`, so you can see the permission changes -

```
$ chmod 755 testfile
$ls -l testfile
-rwxr-xr-x 1 amrood users 1024 Nov 2 00:10 testfile
$ chmod 743 testfile
$ls -l testfile
-rwxr---wx 1 amrood users 1024 Nov 2 00:10 testfile
$ chmod 043 testfile
$ls -l testfile
---r---wx 1 amrood users 1024 Nov 2 00:10 testfile
```

7. Changing Owners and Groups

While creating an account on Unix, it assigns a owner ID and a group ID to each user. All the permissions mentioned above are also assigned based on the Owner and the Groups.

Two commands are available to change the owner and the group of files -

- **chown** - The `chown` command stands for "change owner" and is used to change the owner of a file.
- **chgrp** - The `chgrp` command stands for "change group" and is used to change the group of a file.

8. Changing Ownership

The chown command changes the ownership of a file. The basic syntax is as follows -

```
$ chown user filelist
```

The value of the user can be either the name of a user on the system or the user id (uid) of a user on the system.

The following example will help you understand the concept -

```
$ chown amrood testfile
$
```

Changes the owner of the given file to the user amrood.

Note: The super user, root, has the unrestricted capability to change the ownership of any file but normal users can change the ownership of only those files that they own.

9. Changing Group Ownership

The chgrp command changes the group ownership of a file. The basic syntax is as follows -

```
$ chgrp group filelist
```

The value of group can be the name of a group on the system or the group ID (GID) of a group on the system.

Following example helps you understand the concept -

```
$ chgrp special testfile
$
```

Changes the group of the given file to special group.

10. SUID and SGID File Permission

Often when a command is executed, it will have to be executed with special privileges in order to accomplish its task.

As an example, when you change your password with the passwd command, your new password is stored in the file /etc/shadow.

As a regular user, you do not have read or write access to this file for security reasons, but when you change your password, you need to have the write permission to this file. This means that the passwd program has to give you additional permissions so that you can write to the file /etc/shadow.

Additional permissions are given to programs via a mechanism known as the Set User ID (SUID) and Set Group ID (SGID) bits.

When you execute a program that has the SUID bit enabled, you inherit the permissions of that program's owner. Programs that do not have the SUID bit set are run with the permissions of the user who started the program.

This is the case with SGID as well. Normally, programs execute with your group permissions, but instead your group will be changed just for this program to the group owner of the program.

The SUID and SGID bits will appear as the letter "s" if the permission is available. The SUID "s" bit will be located in the permission bits where the owners' execute permission normally resides.

For example, the command -

```
$ ls -l /usr/bin/passwd
-r-sr-xr-x 1 root bin 19031 Feb 7 13:47 /usr/bin/passwd*
$
```

Shows that the SUID bit is set and that the command is owned by the root. A capital letter S in the execute position instead of a lowercase s indicates that the execute bit is not set.

If the sticky bit is enabled on the directory, files can only be removed if you are one of the following users -

- The owner of the sticky directory
- The owner of the file being removed
- The super user, root

To set the SUID and SGID bits for any directory try the following command -

```
$ chmod ug+s dirname
$ ls -l
drwxr-sr-x 2 root root 4096 Jun 19 06:45 dirname
$
```

1.1.4 Regular Expression and Grep

Q5. Explain about Regular Expression in Grep of unix.

Ans :

(Imp.)

Regular Expression provides an ability to match a "string of text" in a very flexible and concise manner. A "string of text" can be further defined as a single character, word, sentence or particular pattern of characters.

Like the shell's wild-cards which match similar filenames with a single expression, grep uses an expression of a different sort to match a group of similar patterns.

- []: Matches any one of a set characters
- [] with hyphen: Matches any one of a range characters
- ^: The pattern following it must occur at the beginning of each line
- ^ with []: The pattern must not contain any character in the set specified
- \$: The pattern preceding it must occur at the end of each line
- . (dot): Matches any one character
- \ (backslash): Ignores the special meaning of the character following it
- *: zero or more occurrences of the previous character
- (dot).*: Nothing or any numbers of characters.

Examples

(a) []: Matches any one of a set characters

1. \$grep "New[abc]" filename

It specifies the search pattern as :

Newa , Newb or Newc

2. \$grep "[aA]g[ar][ar]wal" filename

It specifies the search pattern as

Agarwal , Agaawal , Agrawal , Agrawal

agarwal , agaawal , agrawal , agrawal

- (b) Use [] with hyphen: Matches any one of a range characters
1. `$grep "New[a-e]" filename`
It specifies the search pattern as
Newa , Newb or Newc , Newd, Newe
 2. `$grep "New[0-9][a-z]" filename`
It specifies the search pattern as: New followed by a number and then an alphabet.
New0d, New4f etc
- (c) Use ^ : The pattern following it must occur at the beginning of each line
1. `$grep "^san" filename`
Search lines beginning with san. It specifies the search pattern as
sanjeev ,sanjay, sanrit , sanchit , sandeep etc.
 2. `$ls -l |grep "^d"`
Display list of directories only
 3. `$ls -l |grep "^-"`
Display list of regular files only
- (d) Use ^ with []: The pattern must not contain any character in the set specified
1. `$grep "New[^a-c]" filename`
It specifies the pattern containing the word "New" followed by any character other than an 'a', 'b', or 'c'
 2. `$grep "^[^a-zA-Z]" filename`
Search lines beginning with a non-alphabetic character
- (e) Use \$: The pattern preceding it must occur at the end of each line
- `$ grep "vedik$" file.txt`
- (f) Use . (dot): Matches any one character
- `$ grep "..vik" file.txt`
- `$ grep "7..9$" file.txt`
- (g) Use \ (backslash): Ignores the special meaning of the character following it
1. `$ grep "New\[abc\]" file.txt`
It specifies the search pattern as New.[abc]
 2. `$ grep "S\[K\].Kumar" file.txt`
It specifies the search pattern as
S.K.Kumar
- (h) Use *: zero or more occurrences of the previous character
- `$ grep "[aA]gg*[ar][ar]wal" file.txt`
- (i) Use (dot).*: Nothing or any numbers of characters.
- `$ grep "S.*Kumar" file.txt`

1.1.5 Shell Programming, awk

Q6. What is Shell?

Ans :

A Shell provides you with an interface to the Unix system. It gathers input from you and executes programs based on that input. When a program finishes executing, it displays that program's output.

Shell is an environment in which we can run our commands, programs, and shell scripts. There are different flavors of a shell, just as there are different flavors of operating systems. Each flavor of shell has its own set of recognized commands and functions.

Shell Prompt

The prompt, \$, which is called the command prompt, is issued by the shell. While the prompt is displayed, you can type a command.

Shell reads your input after you press Enter. It determines the command you want executed by looking at the first word of your input. A word is an unbroken set of characters. Spaces and tabs separate words.

Example

Following is a simple example of the date command, which displays the current date and time-

```
$date
```

```
Thu Jun 25 08:30:19 MST 2009
```

Q7. Explain briefly about various shell variables.

Ans :

(Imp.)

A variable is a character string to which we assign a value. The value assigned could be a number, text, filename, device, or any other type of data.

A variable is nothing more than a pointer to the actual data. The shell enables you to create, assign, and delete variables.

Variable Names

The name of a variable can contain only letters (a to z or A to Z), numbers (0 to 9) or the underscore character (_).

By convention, Unix shell variables will have their names in UPPERCASE.

Example :

The following examples are valid variable names -

```
_ALI
TOKEN_A
VAR_1
VAR_2
```

Following are the examples of invalid variable names -

```
2_VAR
-VARIABLE
VAR1-VAR2
VAR_A!
```

The reason you cannot use other characters such as !, *, or - is that these characters have a special meaning for the shell.

Defining Variables

Variables are defined as follows -

```
variable_name=variable_value
```

For example

```
NAME="Zara Ali"
```

The above example defines the variable NAME and assigns the value "Zara Ali" to it. Variables of this type are called scalar variables. A scalar variable can hold only one value at a time.

Shell enables you to store any value you want in a variable. For example -

```
VAR1="Zara Ali"
VAR2=100
```

Accessing Values

To access the value stored in a variable, prefix its name with the dollar sign (\$) -

For example, the following script will access the value of defined variable NAME and print it on STDOUT -

```
#!/bin/sh
NAME="Zara Ali"
echo $NAME
```

The above script will produce the following value -

Zara Ali

Read-only Variables

Shell provides a way to mark variables as read-only by using the read-only command. After a variable is marked read-only, its value cannot be changed.

For example, the following script generates an error while trying to change the value of NAME -

```
#!/bin/sh
NAME="Zara Ali"
readonly NAME
NAME="Qadiri"
```

The above script will generate the following result -

```
/bin/sh: NAME: This variable is read only.
```

Unsetting Variables

Unsetting or deleting a variable directs the shell to remove the variable from the list of variables that it tracks. Once you unset a variable, you cannot access the stored value in the variable.

Following is the syntax to unset a defined variable using the unset command -

```
unset variable_name
```

The above command unsets the value of a defined variable. Here is a simple example that demonstrates how the command works -

```
#!/bin/sh
NAME="Zara Ali"
unset NAME
echo $NAME
```

The above example does not print anything. You cannot use the unset command to unset variables that are marked readonly.

Variable Types

When a shell is running, three main types of variables are present -

- (i) **Local Variables** - A local variable is a variable that is present within the current instance of the shell. It is not available to programs that are started by the shell. They are set at the command prompt.
- (ii) **Environment Variables** - An environment variable is available to any child process of the shell. Some programs need environment variables in order to function correctly. Usually, a shell script defines only those environment variables that are needed by the programs that it runs.
- (iii) **Shell Variables** - A shell variable is a special variable that is set by the shell and is required by the shell in order to function correctly. Some of these variables are environment variables whereas others are local variables.

The following table shows a number of special variables that you can use in your shell scripts -

Sl. No.	Variable & Description
1	\$0 The filename of the current script.
2	\$n These variables correspond to the arguments with which a script was invoked. Here n is a positive decimal number corresponding to the position of an argument (the first argument is \$1, the second argument is \$2, and so on).
3	\$# The number of arguments supplied to a script.
4	\$* All the arguments are double quoted. If a script receives two arguments, \$* is equivalent to \$1 \$2.
5	\$@ All the arguments are individually double quoted. If a script receives two arguments, \$@ is equivalent to \$1 \$2.
6	\$? The exit status of the last command executed.
7	\$\$ The process number of the current shell. For shell scripts, this is the process ID under which they are executing.
8	\$_ The process number of the last background command.

Q8. Explain various operators of shell.

Ans :

Arithmetic Operators

The following arithmetic operators are supported by Bourne Shell.

Assume variable a holds 10 and variable b holds 20 then -

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator	`expr \$a + \$b` will give 30
- (Subtraction)	Subtracts right hand operand from left hand operand	`expr \$a - \$b` will give -10
* (Multiplication)	Multiplies values on either side of the operator	`expr \$a * \$b` will give 200
/ (Division)	Divides left hand operand by right hand operand	`expr \$b / \$a` will give 2

% (Modulus)	Divides left hand operand by right hand operand and returns remainder	`expr \$b % \$a` will give 0
= (Assignment)	Assigns right operand in left operand	a = \$b would assign value of b into a
== (Equality)	Compares two numbers, if both are same then returns true.	[\$a == \$b] would return false.
!= (Not Equality)	Compares two numbers, if both are different then returns true.	[\$a != \$b] would return true.

It is very important to understand that all the conditional expressions should be inside square braces with spaces around them, for example [\$a == \$b] is correct whereas, [\$a == \$b] is incorrect.

All the arithmetical calculations are done using long integers.

Relational Operators

Bourne Shell supports the following relational operators that are specific to numeric values. These operators do not work for string values unless their value is numeric.

For example, following operators will work to check a relation between 10 and 20 as well as in between "10" and "20" but not in between "ten" and "twenty".

Assume variable a holds 10 and variable b holds 20 then -

Operator	Description	Example
-eq	Checks if the value of two operands are equal or not; if yes, then the condition becomes true.	[\$a -eq \$b] is not true.
-ne	Checks if the value of two operands are equal or not; if values are not equal, then the condition becomes true.	[\$a -ne \$b] is true.
-gt	Checks if the value of left operand is greater than the value of right operand; if yes, then the condition becomes true.	[\$a -gt \$b] is not true.
-lt	Checks if the value of left operand is less than the value of right operand; if yes, then the condition becomes true.	[\$a -lt \$b] is true.
-ge	Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then the condition becomes true.	[\$a -ge \$b] is not true.
-le	Checks if the value of left operand is less than or equal to the value of right operand; if yes, then the condition becomes true.	[\$a -le \$b] is true.

It is very important to understand that all the conditional expressions should be placed inside square braces with spaces around them. For example, [\$a <= \$b] is correct whereas, [\$a <= \$b] is incorrect.

Boolean Operators

The following Boolean operators are supported by the Bourne Shell.

Assume variable a holds 10 and variable b holds 20 then -

Operator	Description	Example
!	This is logical negation. This inverts a true condition into false and vice versa.	[! false] is true.
-o	This is logical OR . If one of the operands is true, then the condition becomes true.	[\$a -lt 20 -o \$b -gt 100] is true.
-a	This is logical AND . If both the operands are true, then the condition becomes true otherwise false.	[\$a -lt 20 -a \$b -gt 100] is false.

String Operators

The following string operators are supported by Bourne Shell.

Assume variable a holds "abc" and variable b holds "efg" then -

Operator	Description	Example
=	Checks if the value of two operands are equal or not; if yes, then the condition becomes true.	[\$a = \$b] is not true.
!=	Checks if the value of two operands are equal or not; if values are not equal then the condition becomes true.	[\$a != \$b] is true.
-z	Checks if the given string operand size is zero; if it is zero length, then it returns true.	[-z \$a] is not true.
-n	Checks if the given string operand size is non-zero; if it is nonzero length, then it returns true.	[-n \$a] is not false.
str	Checks if str is not the empty string; if it is empty, then it returns false.	[\$a] is not false.

File Test Operators

We have a few operators that can be used to test various properties associated with a Unix file.

Assume a variable file holds an existing file name "test" the size of which is 100 bytes and has read, write and execute permission on -

Operator	Description	Example
-b file	Checks if file is a block special file; if yes, then the condition becomes true.	[-b \$file] is false.
-c file	Checks if file is a character special file; if yes, then the condition becomes true.	[-c \$file] is false.
-d file	Checks if file is a directory; if yes, then the condition becomes true.	[-d \$file] is not true.
-f file	Checks if file is an ordinary file as opposed to a directory or special file; if yes, then the condition becomes true.	[-f \$file] is true.
-g file	Checks if file has its set group ID (SGID) bit set; if yes, then the condition becomes true.	[-g \$file] is false.
-k file	Checks if file has its sticky bit set; if yes, then the condition becomes true.	[-k \$file] is false.
-p file	Checks if file is a named pipe; if yes, then the condition becomes true.	[-p \$file] is false.
-t file	Checks if file descriptor is open and associated with a terminal; if yes, then the condition becomes true.	[-t \$file] is false.
-u file	Checks if file has its Set User ID (SUID) bit set; if yes, then the condition becomes true.	[-u \$file] is false.
-r file	Checks if file is readable; if yes, then the condition becomes true.	[-r \$file] is true.
-w file	Checks if file is writable; if yes, then the condition becomes true.	[-w \$file] is true.
-x file	Checks if file is executable; if yes, then the condition becomes true.	[-x \$file] is true.
-s file	Checks if file has size greater than 0; if yes, then condition becomes true.	[-s \$file] is true.
-e file	Checks if file exists; is true even if file is a directory but exists.	[-e \$file] is true.

Q9. Explain briefly about awk.

Ans :

(Imp.)

AWK is an interpreted programming language. It is very powerful and specially designed for text processing. Its name is derived from the family names of its authors - Alfred Aho, Peter Weinberger, and Brian Kernighan.

The version of AWK that GNU/Linux distributes is written and maintained by the Free Software Foundation (FSF); it is often referred to as GNU AWK.

Types of AWK

Following are the variants of AWK -

AWK - Original AWK from AT & T Laboratory.

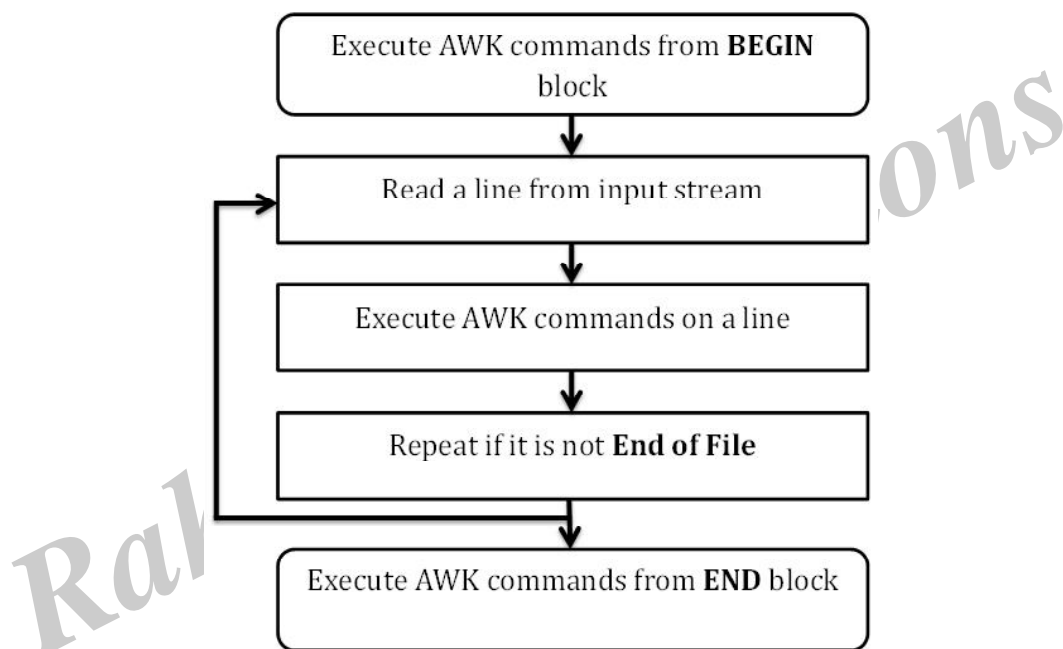
NAWK - Newer and improved version of AWK from AT & T Laboratory.

GAWK - It is GNU AWK. All GNU/Linux distributions ship GAWK. It is fully compatible with AWK and NAWK.

Q10. Explain the working mechanism of awk.

Ans :

To become an expert AWK programmer, you need to know its internals. AWK follows a simple workflow - Read, Execute, and Repeat. The following diagram depicts the workflow of AWK -



Read

AWK reads a line from the input stream (file, pipe, or stdin) and stores it in memory.

Execute

All AWK commands are applied sequentially on the input. By default AWK execute commands on every line. We can restrict this by providing patterns.

Repeat

This process repeats until the file reaches its end.

Program Structure

Let us now understand the program structure of AWK.

BEGIN block

The syntax of the BEGIN block is as follows -

Syntax

```
BEGIN {awk-commands}
```

The BEGIN block gets executed at program start-up. It executes only once. This is good place to initialize variables. BEGIN is an AWK keyword and hence it must be in upper-case. Please note that this block is optional.

Body Block

The syntax of the body block is as follows -

Syntax

```
/pattern/ {awk-commands}
```

The body block applies AWK commands on every input line. By default, AWK executes commands on every line. We can restrict this by providing patterns. Note that there are no keywords for the Body block.

END Block

The syntax of the END block is as follows -

Syntax

```
END {awk-commands}
```

The END block executes at the end of the program. END is an AWK keyword and hence it must be in upper-case. Please note that this block is optional.

Let us create a file marks.txt which contains the serial number, name of the student, subject name, and number of marks obtained.

- 1) Amit Physics 80
- 2) Rahul Maths 90
- 3) Shyam Biology 87
- 4) Kedar English 85
- 5) Hari History 89

Let us now display the file contents with header by using AWK script.

Example

```
[jerry]$ awk 'BEGIN{printf "Sr No\tName\tSub\tMarks\n"} {print}' marks.txt
```

When this code is executed, it produces the following result -

Output

```
Sr No Name Sub Marks
```

- 1) Amit Physics 80
- 2) Rahul Maths 90
- 3) Shyam Biology 87
- 4) Kedar English 85
- 5) Hari History 89

At the start, AWK prints the header from the BEGIN block. Then in the body block, it reads a line from a file and executes AWK's print command which just prints the contents on the standard output stream. This process repeats until file reaches the end.

Q11. Explain the basic syntax of awk.

Ans :

(Imp.)

AWK is simple to use. We can provide AWK commands either directly from the command line or in the form of a text file containing AWK commands.

AWK Command Line

We can specify an AWK command within single quotes at command line as shown -

awk [options] file ...

Example

Consider a text file marks.txt with the following content -

1) Amit	Physics	80
2) Rahul	Maths	90
3) Shyam	Biology	87
4) Kedar	English	85
5) Hari	History	89

Let us display the complete content of the file using AWK as follows -

Example

[jerry]\$ awk '{print}' marks.txt

On executing this code, you get the following result -

Output

1) Amit	Physics	80
2) Rahul	Maths	90
3) Shyam	Biology	87
4) Kedar	English	85
5) Hari	History	89

AWK Program File

We can provide AWK commands in a script file as shown -

awk [options] -f file

First, create a text file command.awk containing the AWK command as shown below -

{print}

Now we can instruct the AWK to read commands from the text file and perform the action. Here, we achieve the same result as shown in the above example.

Example

```
[jerry]$ awk -f command.awk marks.txt
```

On executing this code, you get the following result -

Output

```
1) Amit   Physics   80
2) Rahul  Maths     90
3) Shyam  Biology    87
4) Kedar  English    85
5) Hari   History    89
```

AWK Standard Options

AWK supports the following standard options which can be provided from the command line.

The -v option

This option assigns a value to a variable. It allows assignment before the program execution. The following example describes the usage of the -v option.

Example

```
[jerry]$ awk -v name=Jerry 'BEGIN{printf "Name = %s\n", name}'
```

On executing this code, you get the following result -

Output

```
Name = Jerry
```

The --dump-variables[=file] option

It prints a sorted list of global variables and their final values to file. The default file is awkvars.out.

Example

```
[jerry]$ awk --dump-variables "
```

```
[jerry]$ cat awkvars.out
```

On executing the above code, you get the following result -

Output

```
ARGC: 1
ARGIND: 0
ARGV: array, 1 elements
BINMODE: 0
CONVFMT: "%.6g"
ERRNO: ""
FIELDWIDTHS: ""
FILENAME: ""
FNR: 0
FPAT: "[^[:space:]]+"
FS: " "
```

```

IGNORECASE: 0
LINT: 0
NF: 0
NR: 0
OFMT: "%.6g"
OFS: " "
ORS: "\n"
RLENGTH: 0
RS: "\n"
RSTART: 0
RT: ""
SUBSEP: "\034"
TEXTDOMAIN: "messages"

```

The --help option

This option prints the help message on standard output.

Example

```
[jerry]$ awk --help
```

On executing this code, you get the following result -

Output

Usage: awk [POSIX or GNU style options] -f progfile [--] file ...

Usage: awk [POSIX or GNU style options] [--] 'program' file ...

POSIX options : GNU long options: (standard)

-f progfile	--file=progfile
-F fs	--field-separator=fs
-v var=val	--assign=var=val

Short options : GNU long options: (extensions)

-b	--characters-as-bytes
-c	--traditional
-C	--copyright
-d[file]	--dump-variables[=file]
-e 'program-text'	--source='program-text'
-E file	--exec=file
-g	--gen-pot
-h	--help
-L [fatal]	--lint[=fatal]
-n	--non-decimal-data
-N	--use-lc-numeric
-O	--optimize
-p[file]	--profile[=file]

-P	--posix
-r	--re-interval
-S	--sandbox
-t	--lint-old
-V	--version

The --lint[=fatal] option

This option enables checking of non-portable or dubious constructs. When an argument fatal is provided, it treats warning messages as errors. The following example demonstrates this -

Example

```
[jerry]$ awk --lint " /bin/l
```

On executing this code, you get the following result -

Output

```
awk: cmd. line:1: warning: empty program text on command line
awk: cmd. line:1: warning: source file does not end in newline
awk: warning: no program text at all!
```

The --posix option

This option turns on strict POSIX compatibility, in which all common and gawk-specific extensions are disabled.

The --profile[=file] option

This option generates a pretty-printed version of the program in file. Default file is awkprof.out. Below simple example illustrates this -

Example

```
[jerry]$ awk --profile 'BEGIN{printf"---|Header|--\n"} {print}
END{printf"---|Footer|--\n"}' marks.txt > /dev/null
[jerry]$ cat awkprof.out
```

On executing this code, you get the following result -

Output

```
# gawk profile, created Sun Oct 26 19:50:48 2014
# BEGIN block(s)
BEGIN {
  printf "---|Header|--\n"
}
# Rule(s) {
  print $0
}
# END block(s)
END {
```

```
printf "---| Footer | ---\n"
}
```

The --traditional option

This option disables all gawk-specific extensions.

The --version option

This option displays the version information of the AWK program.

Example

```
[jerry]$ awk --version
```

When this code is executed, it produces the following result -

Output

```
GNU Awk 4.0.1
```

```
Copyright (C) 1989, 1991-2012 Free Software Foundation.
```

Q12. What are the built in variables present in awk?

Ans :

AWK provides several built-in variables. They play an important role while writing AWK scripts. This chapter demonstrates the usage of built-in variables.

Standard AWK Variables

The standard AWK variables are discussed below.

ARGC

It implies the number of arguments provided at the command line.

Example

```
[jerry]$ awk 'BEGIN {print "Arguments =", ARGC}' One Two Three Four
```

On executing this code, you get the following result -

Output

```
Arguments = 5
```

But why AWK shows 5 when you passed only 4 arguments? Just check the following example to clear your doubt.

ARGV

It is an array that stores the command-line arguments. The array's valid index ranges from 0 to ARGC-1.

Example

```
[jerry]$ awk 'BEGIN {
    for (i = 0; i < ARGC - 1; ++i) {
        printf "ARGV[%d] = %s\n", i, ARGV[i]
    }
}' one two three four
```


On executing this code, you get the following result -

Output

```
ARGV[0] = awk
ARGV[1] = one
ARGV[2] = two
ARGV[3] = three
```

CONVFMT

It represents the conversion format for numbers. Its default value is %.6g.

Example

```
[jerry]$ awk 'BEGIN { print "Conversion Format =", CONVFMT }'
```

On executing this code, you get the following result -

Output

```
Conversion Format = %.6g
```

ENVIRON

It is an associative array of environment variables.

Example

```
[jerry]$ awk 'BEGIN { print ENVIRON["USER"] }'
```

On executing this code, you get the following result -

Output

```
jerry
```

To find names of other environment variables, use env command.

FILENAME

It represents the current file name.

Example

```
[jerry]$ awk 'END {print FILENAME}' marks.txt
```

On executing this code, you get the following result -

Output

```
marks.txt
```

Please note that FILENAME is undefined in the BEGIN block.

FS

It represents the (input) field separator and its default value is space. You can also change this by using -F command line option.

Example

```
[jerry]$ awk 'BEGIN {print "FS = " FS}' | cat -vte
```

On executing this code, you get the following result -

Output

```
FS = $
```

NF

It represents the number of fields in the current record. For instance, the following example prints only those lines that contain more than two fields.

Example

```
[jerry]$ echo -e "One Two\nOne Two Three\nOne Two Three Four" | awk 'NF > 2'
```

On executing this code, you get the following result -

Output

```
One Two Three
One Two Three Four
```

NR

It represents the number of the current record. For instance, the following example prints the record if the current record number is less than three.

Example

```
[jerry]$ echo -e "One Two\nOne Two Three\nOne Two Three Four" | awk 'NR < 3'
```

On executing this code, you get the following result -

Output

```
One Two
One Two Three
```

FNR

It is similar to NR, but relative to the current file. It is useful when AWK is operating on multiple files. Value of FNR resets with new file.

OFMT

It represents the output format number and its default value is %.6g.

Example

```
[jerry]$ awk 'BEGIN {print "OFMT = " OFMT}'
```

On executing this code, you get the following result -

Output

```
OFMT = %.6g
```

OFS

It represents the output field separator and its default value is space.

Example

```
[jerry]$ awk 'BEGIN {print "OFS = " OFS}' | cat -vte
```

On executing this code, you get the following result -

Output

```
OFS = $
```

ORS

It represents the output record separator and its default value is newline.

Example

```
[jerry]$ awk 'BEGIN {print "ORS = " ORS}' | cat -vte
```

On executing the above code, you get the following result -

Output

```
ORS = $  
$
```

RLENGTH

It represents the length of the string matched by match function. AWK's match function searches for a given string in the input-string.

Example

```
[jerry]$ awk 'BEGIN { if (match("One Two Three", "re")) { print RLENGTH } }'
```

On executing this code, you get the following result -

Output

```
2
```

RS

It represents (input) record separator and its default value is newline.

Example

```
[jerry]$ awk 'BEGIN {print "RS = " RS}' | cat -vte
```

On executing this code, you get the following result -

Output

```
RS = $  
$
```

RSTART

It represents the first position in the string matched by match function.

Example

```
[jerry]$ awk 'BEGIN { if (match("One Two Three", "Thre")) { print RSTART } }'
```

On executing this code, you get the following result -

Output

```
9
```

SUBSEP

It represents the separator character for array subscripts and its default value is \034.

Example

```
[jerry]$ awk 'BEGIN { print "SUBSEP = " SUBSEP }' | cat -vte
```

On executing this code, you get the following result -

Output

```
SUBSEP = ^\
```

\$0

It represents the entire input record.

Example

```
[jerry]$ awk '{print $0}' marks.txt
```

On executing this code, you get the following result -

Output

```
1) Amit    Physics    80
2) Rahul   Maths     90
3) Shyam   Biology    87
4) Kedar   English    85
5) Hari    History     89
```

\$n

It represents the nth field in the current record where the fields are separated by FS.

Example

```
[jerry]$ awk '{print $3 "\t" $4}' marks.txt
```

On executing this code, you get the following result -

Output

```
Physics    80
Maths      90
Biology    87
English    85
History    89
```

GNU AWK Specific Variables

GNU AWK specific variables are as follows -

ARGIND

It represents the index in ARGV of the current file being processed.

Example

```
[jerry]$ awk '{
    print "ARGIND  = ", ARGIND; print "Filename = ", ARGV[ARGIND]
}' junk1 junk2 junk3
```

On executing this code, you get the following result -

Output

```
ARGIND  = 1
Filename = junk1
ARGIND  = 2
Filename = junk2
ARGIND  = 3
Filename = junk3
```

BINMODE

It is used to specify binary mode for all file I/O on non-POSIX systems. Numeric values of 1, 2, or 3 specify that input files, output files, or all files, respectively, should use binary I/O. String values of r or w specify that input files or output files, respectively, should use binary I/O. String values of rw or wr specify that all files should use binary I/O.

ERRNO

A string indicates an error when a redirection fails for getline or if close call fails.

Example

```
[jerry]$ awk 'BEGIN { ret = getline < "junk.txt"; if (ret == -1) print "Error:", ERRNO }'
```

On executing this code, you get the following result -

Output

Error: No such file or directory

FIELDWIDTHS

A space separated list of field widths variable is set, GAWK parses the input into fields of fixed width, instead of using the value of the FS variable as the field separator.

IGNORECASE

When this variable is set, GAWK becomes case-insensitive. The following example demonstrates this-

Example

```
[jerry]$ awk 'BEGIN{IGNORECASE = 1} /amit/' marks.txt
```

On executing this code, you get the following result -

Output

1) Amit Physics 80

LINT

It provides dynamic control of the --lint option from the GAWK program. When this variable is set, GAWK prints lint warnings. When assigned the string value fatal, lint warnings become fatal errors, exactly like --lint=fatal.

Example

```
[jerry]$ awk 'BEGIN {LINT = 1; a}'
```

On executing this code, you get the following result -

Output

awk: cmd. line:1: warning: reference to uninitialized variable `a'

awk: cmd. line:1: warning: statement has no effect

PROCINFO

This is an associative array containing information about the process, such as real and effective UID numbers, process ID number, and so on.

Example

```
[jerry]$ awk 'BEGIN { print PROCINFO["pid"] }'
```

On executing this code, you get the following result -

Output

4316

TEXTDOMAIN

It represents the text domain of the AWK program. It is used to find the localized translations for the program's strings.

Example

```
[jerry]$ awk 'BEGIN { print TEXTDOMAIN }'
```

On executing this code, you get the following result -

Output

messages

Q13. Explain the various operators of awk.

Ans :

S.No.	Operators & Description
1	Arithmetic Operators AWK supports the following arithmetic operators.
2	Increment and Decrement Operators AWK supports the following increment and decrement operators.
3	Assignment Operators AWK supports the following assignment operators.
4	Relational Operators AWK supports the following relational operators.
5	Logical Operators AWK supports the following logical operators.
6	Ternary Operator We can easily implement a condition expression using ternary operator.
7	Unary Operators AWK supports the following unary operators.
8	Exponential Operators There are two formats of exponential operators.
9	String Concatenation Operator Space is a string concatenation operator that merges two strings.
10	Array Membership Operator It is represented by in. It is used while accessing array elements.
11	Regular Expression Operators This example explains the two forms of regular expressions operators.

1.2 INTRODUCTION TO OPERATING SYSTEMS

1.2.1 Os Structure And Strategies

Q14. What is Operating System? Explain about it.

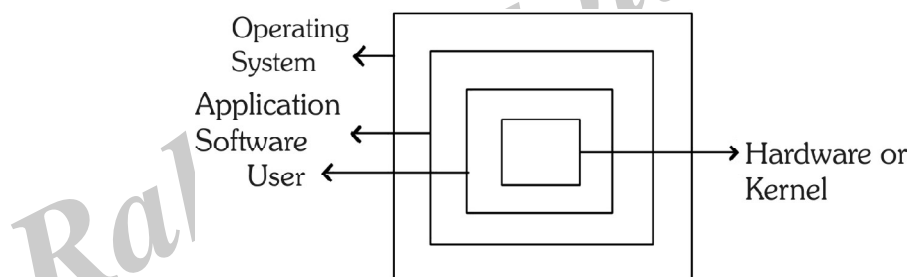
Ans :

Meaning

Operating System is software that works as an interface between a user and the computer hardware. The primary objective of an operating system is to make computer system convenient to use and to utilize computer hardware in an efficient manner. The operating system performs the basic tasks such as receiving input from the keyboard, processing instructions and sending output to the screen.

Operating system is software that is required in order to run application programs and utilities. It works as a bridge to perform better interaction between application programs and hardware of the computer. Various types of 'operating systems' are UNIX, MS-DOS, MS-Windows - 98/XP/Vista, Windows-NT/2000, OS/2 and Mac OS.

Operating system manages overall activities of a computer and the input/output devices attached to the computer. It is the first software you see when you turn on the computer, and the last software you see when the computer is turned off. It is the software that enables all the programs you use. At the simplest level, an operating system does two things:



- The first, it manages the hardware and software resources of the computer system. These resources include the processor, memory, disk space, etc.
- The second, it provides a stable, consistent way for applications to deal with the hardware without having to know all the details of the hardware.
- The first task is very important i.e. managing the hardware and software resources, as various processes compete to each other for getting the CPU time and memory space to complete the task. In this regard; the operating system acts as a manager to allocate the available resources to 'satisfy the requirements' of each process.
- The second task i.e. providing a consistent application interface is especially important. A consistent application program interface (API) allows a user (or S/W developer) to write an application program on any computer and to run this program on another computer, even if the hardware configuration is different like as amount of memory, type of CPU or storage disk. It shields the user of the machine from the low-level details of the machine's operation and provides frequently needed facilities.

- When you turn on the computer, the operating system program is loaded into the main memory. This program is called the kernel. Once initialized, the system program is prepared to run the user programs and permits them to use the hardware efficiently. Windows 98/XP is an excellent example that supports different types of hardware configurations from thousands of vendors and accommodates thousands of different I/O devices like printers, disk drives, scanners and cameras.
- Operating systems may be classified based on if multiple tasks can be performed simultaneously, and if the system can be used by multiple users. It can be termed as single-user or multi-user OS, and single-tasking or multi-tasking OS. A multi-user system must be multi-tasking. MS-DOS and Windows 3x are examples of single user operating system. Whereas UNIX is an example of multi-user and multitasking operating system.

For Better understanding you can see the Working of the Operating System.

So we can say that the Operating System have the Following Characteristics:

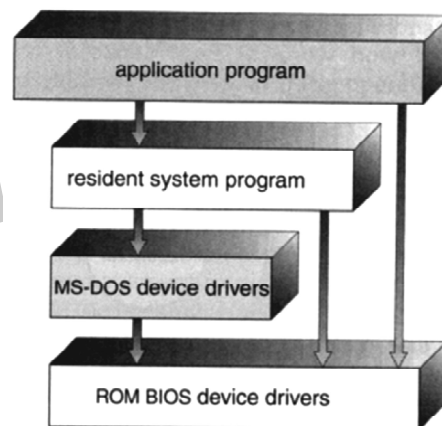
1. Operating System is a Collection of Programs those are Responsible for the Execution of other Programs.
2. Operating System is that which Responsible is for Controlling all the Input and Output Devices those are connected to the System.
3. Operating System is that which Responsible is for Running all the Application Software's.
4. Operating System is that which Provides Scheduling to the Various Processes Means Allocates the Memory to various Process those Wants to Execute.
5. Operating System is that which provides the Communication between the user and the System.

6. Operating System is Stored into the BIOS Means in the Basic Input and Output System means when a user Starts his System then this will Read all the instructions those are Necessary for Executing the System Means for Running the Operating System, Operating System Must be Loaded into the Computer For this, this will use the Floppy or Hard Disks Which Stores the Operating System.

Q15. Explain the structure of the Operating system.

Ans :

Operating System Structure

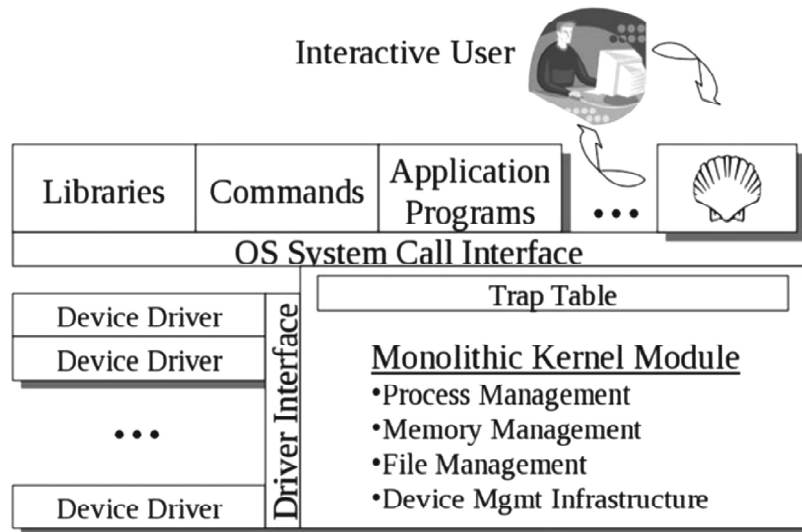


In MS-DOS, applications may bypass the operating system.

- Operating systems such as MS-DOS and the original UNIX did not have well-defined structures.
- There was no CPU Execution Mode (user and kernel), and so errors in applications could cause the whole system to crash.

Monolithic Approach

- Functionality of the OS is invoked with simple function calls within the kernel, which is one large program.
- Device drivers are loaded into the running kernel and become part of the kernel.

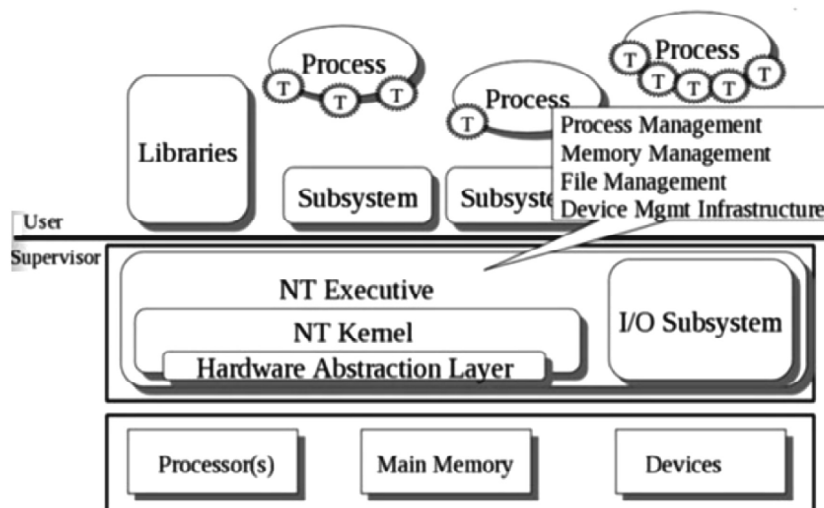


A monolithic kernel, such as Linux and other Unix systems

Layered Approach

This approach breaks up the operating system into different layers.

- This allows implementers to change the inner workings, and increases modularity.
- As long as the external interface of the routines don't change, developers have more freedom to change the inner workings of the routines.
- With the layered approach, the bottom layer is the hardware, while the highest layer is the user interface.
 - ▶ The main advantage is simplicity of construction and debugging.
 - ▶ The main difficulty is defining the various layers.
 - ▶ The main disadvantage is that the OS tends to be less efficient than other implementations.



The Microsoft Windows NT Operating System. The lowest level is a monolithic kernel, but many OS components are at a higher level, but still part of the OS.

Microkernels

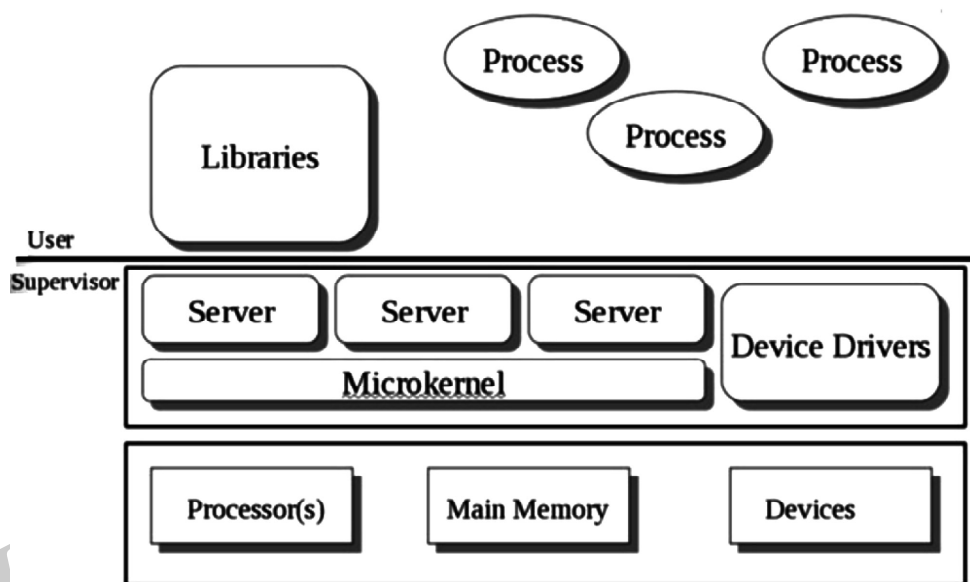
This structures the operating system by removing all nonessential portions of the kernel and implementing them as system and user level programs.

- Generally they provide minimal process and memory management, and a communications facility.
- Communication between components of the OS is provided by message passing.

The benefits of the microkernel are as follows:

- Extending the operating system becomes much easier.
- Any changes to the kernel tend to be fewer, since the kernel is smaller.
- The microkernel also provides more security and reliability.

Main disadvantage is poor performance due to increased system overhead from message passing.



A Microkernel architecture.

Q16. Write about the strategies performed by the OS.

Or

Write about various types of Operating Systems.

Ans :

Operating system strategies

Depending on the needs of its users, a computer's operating system may be designed to use different strategies to serve the users' needs best.

Types of Operating Systems

Following are some of the most widely used types of Operating system.

1. Simple Batch System
2. Multiprogramming Batch System

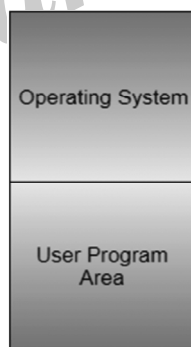
3. Multiprocessor System
4. Desktop System
5. Distributed Operating System
6. Client Server
7. Realtime Operating System
8. Handheld System

1. Simple Batch Systems

- In this type of system, there is no direct interaction between user and the computer.
- The user has to submit a job (written on cards or tape) to a computer operator.
- Then computer operator places a batch of several jobs on an input device.
- Jobs are batched together by type of languages and requirement.
- Then a special program, the monitor, manages the execution of each program in the batch.
- The monitor is always in the main memory and available for execution.

Following are some disadvantages of this type of system :

1. No interaction between user and computer.
2. No mechanism to prioritise the processes.



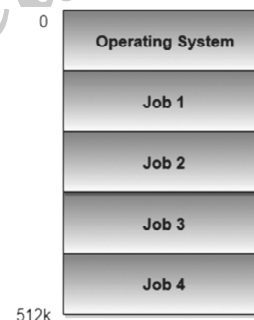
2. Multiprogramming Batch Systems

- In this the operating system picks up and begins to execute one of the jobs from memory.
- Once this job needs an I/O operation operating system switches to another job (CPU and OS always busy).

- Jobs in the memory are always less than the number of jobs on disk(Job Pool).
- If several jobs are ready to run at the same time, then the system chooses which one to run through the process of CPU Scheduling.
- In Non-multiprogrammed system, there are moments when CPU sits idle and does not do any work.
- In Multiprogramming system, CPU will never be idle and keeps on processing.

Time-Sharing Systems are very similar to Multiprogramming batch systems. In fact time sharing systems are an extension of multiprogramming systems.

In time sharing systems the prime focus is on minimizing the response time, while in multiprogramming the prime focus is to maximize the CPU usage.



3. Multiprocessor Systems

A multiprocessor system consists of several processors that share a common physical memory. Multiprocessor system provides higher computing power and speed. In multiprocessor system all processors operate under single operating system. Multiplicity of the processors and how they do act together are transparent to the others.

Following are some advantages of this type of system.

1. Enhanced performance
2. Execution of several tasks by different processors concurrently, increases the system's throughput without speeding up the execution of a single task.

3. If possible, system divides task into many subtasks and then these subtasks can be executed in parallel in different processors. Thereby speeding up the execution of single tasks.

4. Desktop Systems

Earlier, CPUs and PCs lacked the features needed to protect an operating system from user programs. PC operating systems therefore were neither **multiuser** nor **multitasking**. However, the goals of these operating systems have changed with time; instead of maximizing CPU and peripheral utilization, the systems opt for maximizing user convenience and responsiveness. These systems are called **Desktop Systems** and include PCs running Microsoft Windows and the Apple Macintosh. Operating systems for these computers have benefited in several ways from the development of operating systems for **mainframes**.

Microcomputers were immediately able to adopt some of the technology developed for larger operating systems. On the other hand, the hardware costs for microcomputers are sufficiently **low** that individuals have sole use of the computer, and CPU utilization is no longer a prime concern. Thus, some of the design decisions made in operating systems for mainframes may not be appropriate for smaller systems.

5. Distributed Operating Systems

The motivation behind developing distributed operating systems is the availability of powerful and inexpensive microprocessors and advances in communication technology.

These advancements in technology have made it possible to design and develop distributed systems comprising of many computers that are inter connected by communication networks. The main benefit of distributed systems is its low price/performance ratio.

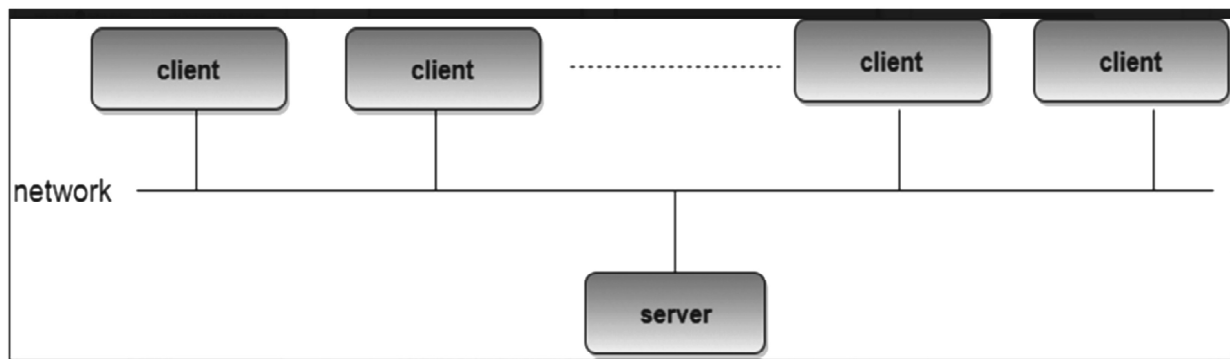
Following are some advantages of this type of system.

1. As there are multiple systems involved, user at one site can utilize the resources of systems at other sites for resource-intensive tasks.
2. Fast processing.
3. Less load on the Host Machine.

The two types of Distributed Operating Systems are: Client-Server Systems and Peer-to-Peer Systems.

6. Client-Server Systems

Centralized systems today act as **server systems** to satisfy requests generated by **client systems**. The general structure of a client-server system is depicted in the figure below:



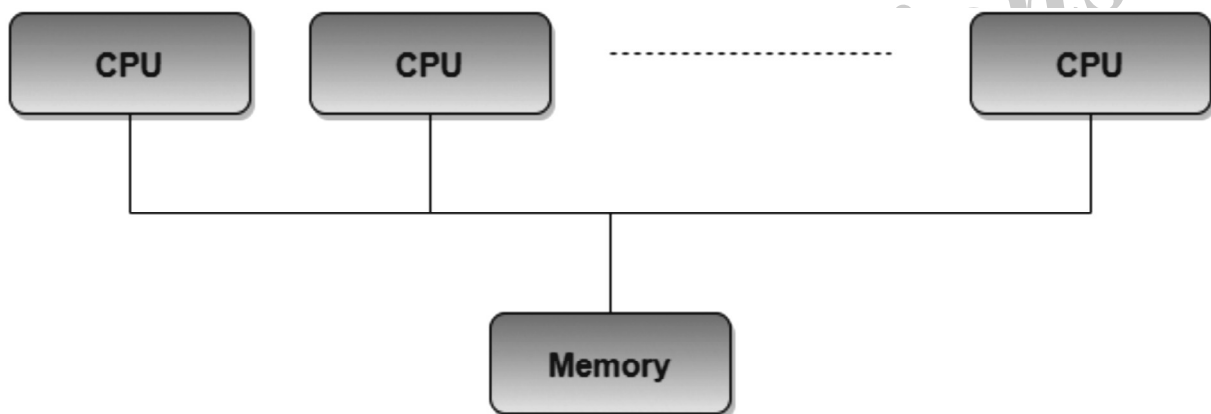
Server Systems can be broadly categorized as compute servers and file servers.

- **Compute-server systems** provide an interface to which clients can send requests to perform an action, in response to which they execute the action and send back results to the client.
- **File-server systems** provide a file-system interface where clients can create, update, read, and delete files.

Peer-to-peer Systems

The growth of computer networks - especially the Internet and World Wide Web (WWW) – has had a profound influence on the recent development of operating systems. When PCs were introduced in the 1970s, they were designed for **personal** use and were generally considered standalone computers. With the beginning of widespread public use of the Internet in the 1980s for electronic mail and ftp many PCs became connected to computer networks.

In contrast to the **tightly coupled** systems, the computer networks used in these applications consist of a collection of processors that do not share memory or a clock. Instead, each processor has its own local memory. The processors communicate with one another through various communication lines, such as high-speed buses or telephone lines. These systems are usually referred to as loosely coupled systems (or distributed systems). The general structure of a client-server system is depicted in the figure below:



Clustered Systems

- Like parallel systems, clustered systems gather together multiple CPUs to accomplish computational work.
- Clustered systems differ from parallel systems, however, in that they are composed of two or more individual systems coupled together.
- The definition of the term clustered is **not concrete**; the general accepted definition is that clustered computers share storage and are closely linked via LAN networking.
- Clustering is usually performed to provide **high availability**.
- A layer of cluster software runs on the cluster nodes. Each node can monitor one or more of the others. If the monitored machine fails, the monitoring machine can take ownership of its storage, and restart the application(s) that were running on the failed machine. The failed machine can remain down, but the users and clients of the application would only see a brief interruption of service.
- **Asymmetric Clustering** - In this, one machine is in hot standby mode while the other is running the applications. The hot standby host (machine) does nothing but monitor the active server. If that server fails, the hot standby host becomes the active server.

- **Symmetric Clustering** - In this, two or more hosts are running applications, and they are monitoring each other. This mode is obviously more efficient, as it uses all of the available hardware.
- **Parallel Clustering** - Parallel clusters allow multiple hosts to access the same data on the shared storage. Because most operating systems lack support for this simultaneous data access by multiple hosts, parallel clusters are usually accomplished by special versions of software and special releases of applications.

Clustered technology is rapidly changing. Clustered system use and features should expand greatly as **Storage Area Networks(SANs)**. SANs allow easy attachment of multiple hosts to multiple storage units. Current clusters are usually limited to two or four hosts due to the complexity of connecting the hosts to shared storage.

7. Real-time Operating System

It is defined as an operating system known to give maximum time for each of the critical operations that it performs, like OS calls and interrupt handling.

The Real-Time Operating system which guarantees the maximum time for critical operations and complete them on time are referred to as **Hard Real-Time Operating Systems**.

While the real-time operating systems that can only guarantee a maximum of the time, i.e. the critical task will get priority over other tasks, but no assurance of completing it in a defined time. These systems are referred to as **Soft Real-Time Operating Systems**.

8. Handheld Systems

Handheld systems include **Personal Digital Assistants(PDAs)**, such as Palm-Pilots or Cellular. Telephones with connectivity to a network such as the Internet. They are usually of limited size due to which most handheld devices have a small amount of memory, include slow processors, and feature small display screens.

- Many handheld devices have between **512 KB** and **8 MB** of memory. As a result, the operating system and applications must manage memory efficiently. This includes returning all allocated memory back to the memory manager once the memory is no longer being used.
- Currently, many handheld devices do **not use virtual memory** techniques, thus forcing program developers to work within the confines of limited physical memory.
- Processors for most handheld devices often run at a fraction of the speed of a processor in a PC. Faster processors require **more power**. To include a faster processor in a handheld device would require a **larger battery** that would have to be replaced more frequently.
- The last issue confronting program designers for handheld devices is the small display screens typically available. One approach for displaying the content in web pages is **web clipping**, where only a small subset of a web page is delivered and displayed on the handheld device.

Some handheld devices may use wireless technology such as **BlueTooth**, allowing remote access to e-mail and web browsing. **Cellular telephones** with connectivity to the Internet fall into this category. Their use continues to expand as network connections become more available and other options such as cameras and MP3 players, expand their utility.

Q17. Explain various functions of OS.

Ans :

There are Many Functions those are Performed by the Operating System But the Main Goal of Operating System is to Provide the Interface between the user and the hardware Means Provides the Interface for Working on the System by the user. The various Functions those are Performed by the Operating System are as Explained below:

1. Operating System as a Resource Manager

Operating System Also Known as the Resource Manager Means Operating System will Manages all the Resources those are Attached to the System means all the Resource like Memory and Processor and all the Input output Devices those are Attached to the System are Known as the Resources of the ComputerSystem and the Operating system will Manage all the Resources of the System. The Operating System will identify at which Time the CPU will perform which Operation and in which Time the Memory is used by which Programs. And which Input Device will respond to which Request of the user means When the Input and Output Devices are used by the which Programs. So this will manage all the Resources those are attached to the Computer System.

2. Storage Management

Operating System also Controls the all the Storage Operations means how the data or files will be Stored into the computers and how the Files will be Accessed by the users etc. All the Operations those are Responsible for Storing and Accessing the Files is determined by the Operating System. Operating System also Allows us Creation of Files, Creation of Directories and Reading and Writing the data of Files and Directories and also Copy the contents of the Files and the Directories from One Place to Another Place.

3. Process Management

The Operating System also Treats the Process Management means all the Processes those are given by the user or the Process those are System 's own Process are Handled by the Operating System. The Operating SystemwillCreate the Priorities foe the user and also Start or Stops the Execution of the Process and Also Makes the Child Process after dividing the Large Processes into the Small Processes

4. Memory Management

Operating System also Manages the Memory of the Computer System means Provide the Memory to the Process and Also Deallocate the Memory from the Process. And also defines that if a Process gets completed then this will deallocate the Memory from the Processes.

5. Extended Machine

Operating System also behaves like an Extended Machine means Operating system also Provides us Sharing of Files between Multiple Users, also Provides Some Graphical Environments and also Provides Various Languages for Communications and also Provides Many Complex Operations like using Many Hardware's and Software's.

Operating System also controls the Errors those have been Occurred into the Program and Also Provides Recovery of the System when the System gets Damaged Means When due to Some Hardware Failure.

1.2.2 Process Concepts

Q18. What is a Process? Explain about it.

Ans :

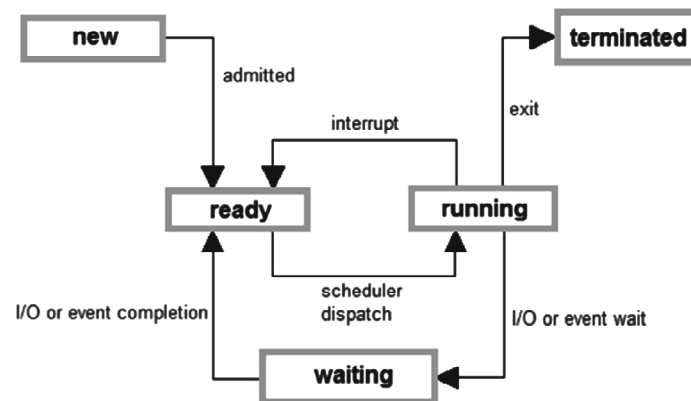
A program in the execution is called a Process. Process is not the same as program. A process is more than a program code. A process is an 'active' entity as opposed to program which is considered to be a 'passive' entity. Attributes held by process include hardware state, memory, CPU etc.

Process memory is divided into four sections for efficient working :

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

Q19. What is process state? And Explain Various Process States.*Ans :*

A process which is Executed by the Process have various States, the State of the Process is also called as the **Status of the process**, The Status includes whether the Process has Executed or Whether the process is Waiting for Some input and output from the user and whether the Process is Waiting for the CPU to Run the Program after the Completion of the Process.



The various States of the Process are as Followings :

1. New State

When a user request for a Service from the System , then the System will first initialize the process or the System will call it an initial Process . So Every new Operation which is Requested to the System is known as the New Born Process.

2. Running State

When the Process is Running under the CPU, or When the Program is Executed by the CPU , then this is called as the Running process and when a process is Running then this will also provides us Some Outputs on the Screen.

3. Waiting

When a Process is Waiting for Some Input and Output Operations then this is called as the Waiting State. And in this process is not under the Execution instead the Process is Stored out of Memory and when the user will provide the input then this will Again be on ready State.

4. Ready State

When the Process is Ready to Execute but he is waiting for the CPU to Execute then this is called as the Ready State. After the Completion of the Input and outputs the Process will be on Ready State means the Process will Wait for the Processor to Execute.

5. Terminated State

After the Completion of the Process , the Process will be Automatically terminated by the CPU . So this is also called as the Terminated State of the Process. After Executing the Whole Process the Processor will Also deallocate the Memory which is allocated to the Process. So this is called as the Terminated Process.

As we know that there are many processes those are running at a Time, this is not true. **A processor can execute only one Process at a Time.** There are the various States of the Processes those determined

which Process will be executed. The Processor will Execute all the processes by using the States of the Processes, the Processes those are on the Waiting State will not be executed and CPU will Also divides his time for Execution if there are Many Processes those are Ready to Execute.

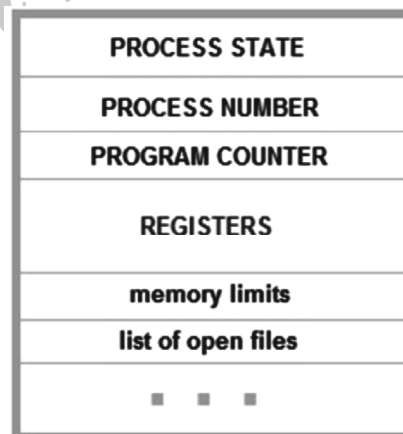
When a Process Change his State from one State to Another, then this is also called as the **Process State Transition**. In this a Running Process may goes on Wait and a ready Process may goes on the Wait State and the Wait State can be goes on the Running State.

Q20. Write a note on Process Control Block (PCB).

Ans :

Process Control block is used for storing the collection of information about the Processes and this is also called as the Data Structure which Stores the information about the process. The information of the Process is used by the CPU at the Run time. The various information which is Stored into the PCB as followings:

1. Name of the Process.
2. State of the Process. Means Ready, Active, Wait.
3. Resources allocated to the Process
4. Memory which is provided to the Process.
5. Scheduling information.
6. Input and Output Devices used by the Process.
7. Process ID or a Identification Number which is given by the CPU when a Process Request for a Service.



Q21. What are the various operations performed on the processor?

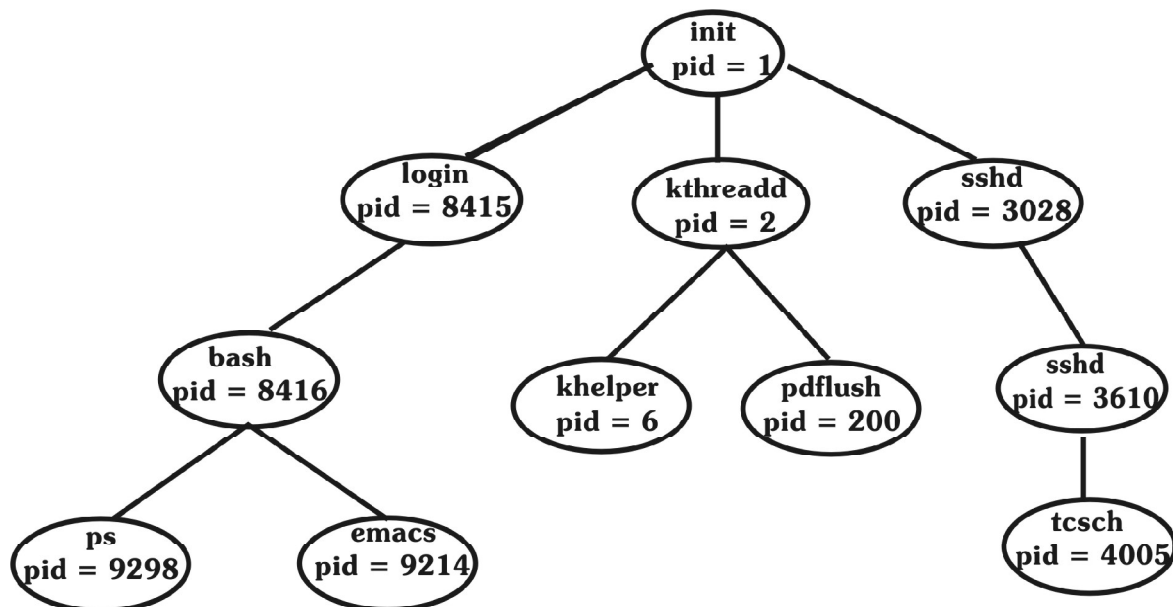
Ans :

i) Process Creation

Through appropriate system calls, such as fork or spawn, processes may create other processes. The process which creates other process, is termed the parent of the other process, while the created sub-process is termed its child.

Each process is given an integer identifier, termed as process identifier, or PID. The parent PID (PPID) is also stored for each process.

A child process may receive some amount of shared resources with its parent depending on system implementation. To prevent runaway children from consuming all of a certain system resource, child processes may or may not be limited to a subset of the resources originally allocated to the parent.



A Tree of Processes on a typical Linux System

There are two options for the parent process after creating the child :

- Wait for the child process to terminate before proceeding. Parent process makes a `wait()` system call, for either a specific child process or for any particular child process, which causes the parent process to block until the `wait()` returns. UNIX shells normally wait for their children to complete before issuing a new prompt.
- Run concurrently with the child, continuing to process without waiting. When a UNIX shell runs a process as a background task, this is the operation seen. It is also possible for the parent to run for a while, and then wait for the child later, which might occur in a sort of a parallel processing operation.

ii) Process Termination

By making the `exit()` system call, typically returning an int, processes may request their own termination. This int is passed along to the parent if it is doing a `wait()`, and is typically zero on successful completion and some non-zero code in the event of any problem.

Processes may also be terminated by the system for a variety of reasons, including :

- The inability of the system to deliver the necessary system resources.
- In response to a KILL command or other unhandled process interrupts.
- A parent may kill its children if the task assigned to them is no longer needed i.e. if the need of having a child terminates.
- If the parent exits, the system may or may not allow the child to continue without a parent (In UNIX systems, orphaned processes are generally inherited by `init`, which then proceeds to kill them.)

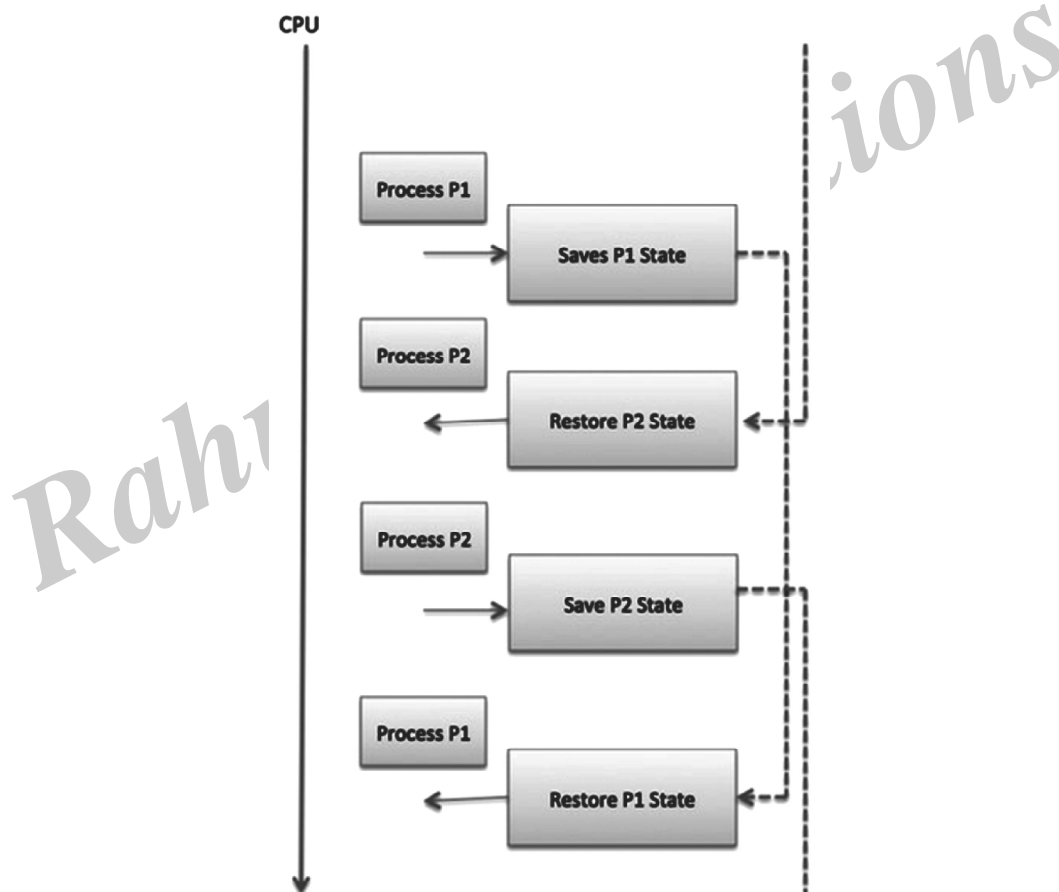
When a process ends, all of its system resources are freed up, open files flushed and closed, etc. The process termination status and execution times are returned to the parent if the parent is waiting for the child to terminate, or eventually returned to init if the process already became an orphan.

The processes which are trying to terminate but cannot do so because their parent is not waiting for them are termed zombies. These are eventually inherited by init as orphans and killed off.

iii) Context Switch

A context switch is the mechanism to store and restore the state or context of a CPU in Process Control block so that a process execution can be resumed from the same point at a later time. Using this technique, a context switcher enables multiple processes to share a single CPU. Context switching is an essential part of a multitasking operating system features.

When the scheduler switches the CPU from executing one process to execute another, the state from the current running process is stored into the process control block. After this, the state for the process to run next is loaded from its own PCB and used to set the PC, registers, etc. At that point, the second process can start executing.



Context switches are computationally intensive since register and memory state must be saved and restored. To avoid the amount of context switching time, some hardware systems employ two or more sets of processor registers. When the process is switched, the following information is stored for later use.

- Program Counter
- Scheduling information

- Base and limit register value
- Currently used register
- Changed State
- I/O State information
- Accounting information

Q22. Explain how the processes will communicate with each other.

Ans :

The OS provides the means for cooperating processes to communicate with each other via an interprocess communication (IPC) facility. IPC provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space. IPC is particularly useful in a distributed environment where the communicating processes may reside on different computers connected with a network e.g. chat program used on the world wide web. IPC is best provided by a message passing system, and the message systems can be defined in many ways.

Message Passing System

Message system allows processes to communicate with one another without the need to resort to shared data. Services are provided as ordinary user processes operate outside the kernel.

Communication among the user processes is accomplished through the passing of messages.

An IPC facility provides at least two operations: send (message) and receive (message).

Messages sent by a process can be of either fixed or variable size.

If processes P and Q want to communicate, they must send messages to send and receive from each other; a communication link must exist between them. There are several methods for logical implementation of a link as follows:

- Direct or indirect communication.
- Symmetric or asymmetric communication.
- Automatic or explicit buffering
- Send by copy or send by reference.
- Fixed-sized or variable-sized message.

(a) Direct Communication

Each process that wants to communicate must explicitly name the recipient or sender of the communication. The send and receive primitives are defined as:

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

A communication link in this scheme has the following properties:

- A link is established automatically between every pair of processes that want to communicate. The processes need to know only each other's identity to communicate.
 - A link is associated with exactly two processes.
 - Exactly one link exists between each pair of processes.
- This scheme exhibits symmetry in addressing; that is, both the sender and the receiver processes must name the other to communicate.

A variant of this scheme employs asymmetry in addressing. Only the sender names the recipient; the recipient is not required to name the sender. In this scheme, the send and receive primitives are as follows:

- send (P, message) – Send a message to process P.
- receive (id, message) – Receive a message from any process; the variable id is set to the name of the process with which communication has taken place.

The disadvantage in both schemes:

Changing the name of a process may necessitate examining all other process definitions. All references to the old name must be found, so that they can be modified to the new name. This situation is not desirable from the viewpoint of separate compilation.

(b) Indirect Communication

The messages are sent to and received from mailboxes, or ports. Each mailbox has a unique identification. Two processes can communicate only

if they share a mailbox. The send and receive primitives are defined as follows:

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

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

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

If processes P1, P2 and P3 all share mailbox A. Process P1 sends a message to A, while P2 and P3 each execute and receive from A.

The process to receive the message depends on one of the scheme that:

- Allows a link to be associated with at most two processes.
- Allows utmost one process at a time to execute a receive operation.
- Allows the system to select arbitrarily which process will receive the message (that is either P2 or P3, but not both, will receive the message). The system may identify the receiver to the sender.

If the mailbox is owned by process (that is, the mailbox is part of the address space of the process), then we distinguish between the owner (who can only receive messages through this mailbox) and the user (who can only send messages to the mailbox). When a process that owns a mailbox terminates, the mailbox disappears. Any process that subsequently sends a message to this mailbox must be notified that the mailbox no longer exists. On the other hand, a mailbox owned by the OS is independent and is not attached to any particular process. The OS then must provide a mechanism that allows a process to do the following:

- Create a new mailbox.
- Send and receive messages through the mailbox.
- Delete a mailbox

Process who create a mailbox is the owner by default and receives messages through this mail box. Ownership can be changed by OS through appropriate system calls to provide multiple receivers for each mailbox.

(c) Synchronization

The send and receive system calls are used to communicate between processes but there are different design options for implementing these calls. Message passing may be either

blocking or non-blocking - also known as synchronous and asynchronous.

➤ Blocking send

The sending process is blocked until the message is received by the receiving process or by the mailbox.

➤ Non-blocking send:

The sending process sends the message and resumes operation.

➤ Blocking receive

The receiver blocks until a message is available.

➤ Non-blocking receive

The receiver retrieves either a valid message or a null. Different combinations of send and receive are possible. When both the send and receive are blocking, we have a rendezvous (to meet) between the sender and receiver.

(d) Buffering

During direct or indirect communication, messages exchanged between communicating processes reside in a temporary queue which are implemented in the following three ways:

➤ Zero capacity

The queue has maximum length 0; thus, the link cannot have any message waiting in it. In this case, the sender must block until the recipient receives the message. This is referred to as no buffering.

- **Bounded capacity**

The queue has finite length n ; thus, at most n messages can reside in it. If the queue is not full when a new message is sent, the latter is placed in the queue (either the message is copied or a pointer to the message is kept), and the sender can continue execution without waiting. If the link is full, the sender must block until space is available in the queue. This is referred to as auto buffering

- **Unbounded capacity**

The queue has potentially infinite length; thus, any number of messages can wait in it. The sender never blocks. This also referred to as auto buffering.

1.2.3 Multithreaded Programming

Q23. What is Thread? Explain about various type of threads.

Ans :

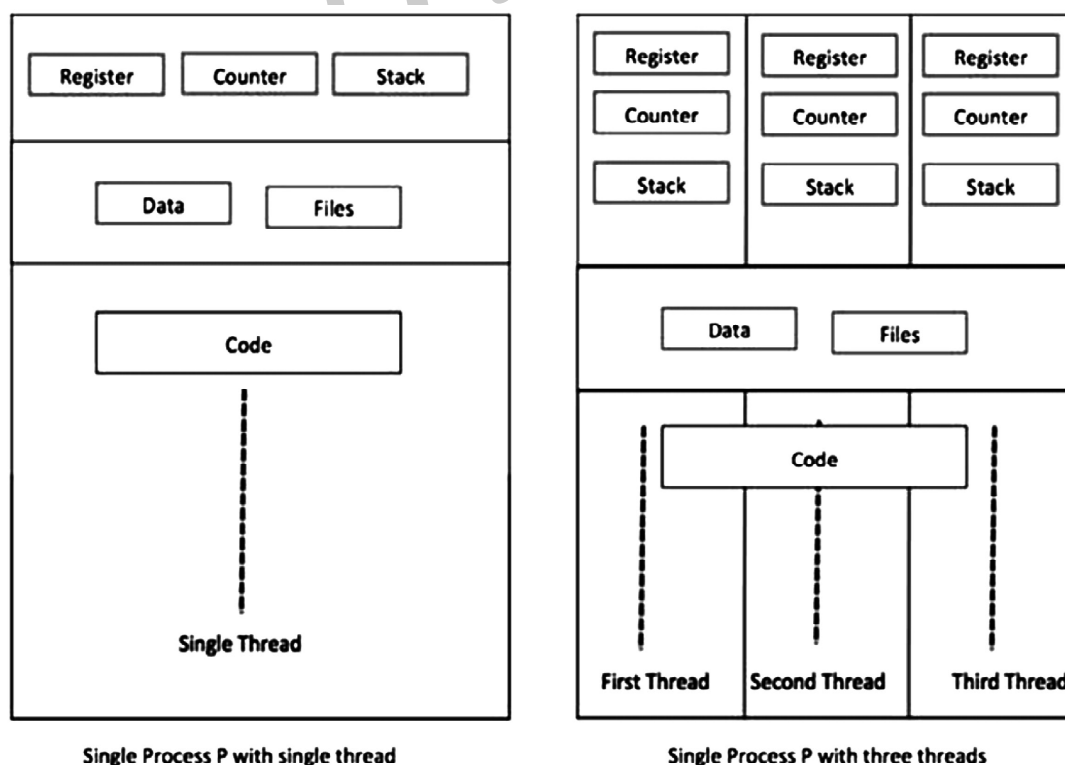
(Imp.)

A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history.

A thread shares with its peer threads few information like code segment, data segment and open files. When one thread alters a code segment memory item, all other threads see that.

A thread is also called a lightweight process. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.

Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control. Threads have been successfully used in implementing network servers and web server. They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors. The following figure shows the working of a single-threaded and a multithreaded process.



Advantages of Thread

- Threads minimize the context switching time.
- Use of threads provides concurrency within a process.
- Efficient communication.
- It is more economical to create and context switch threads.
- Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.

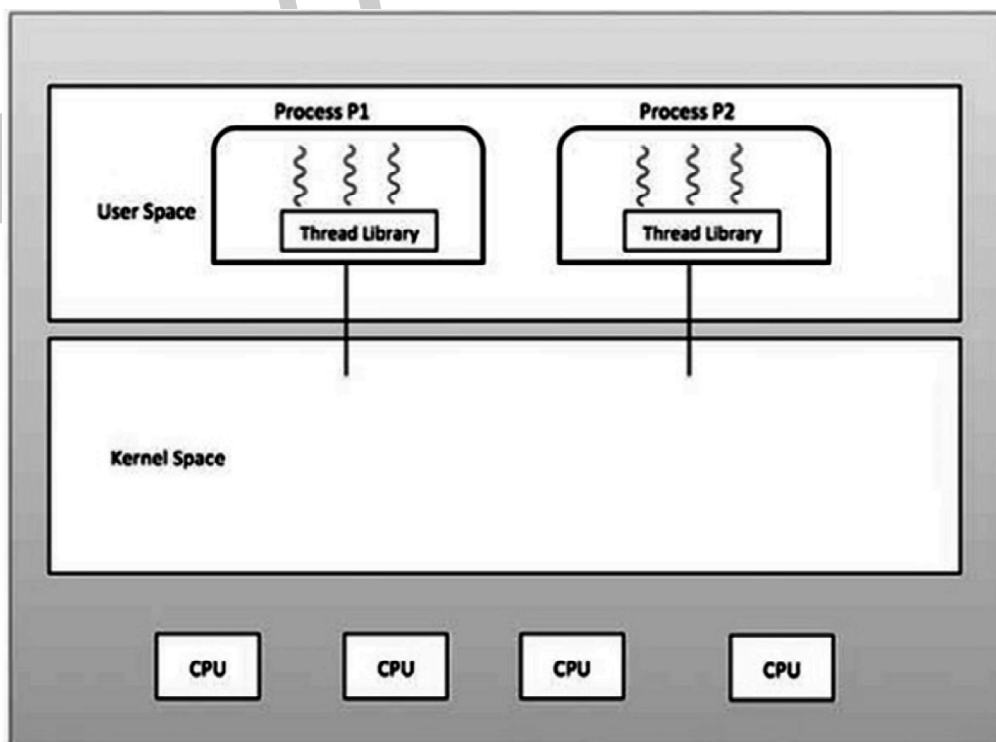
Types of Thread

Threads are implemented in following two ways

1. User Level Threads – User managed threads.
2. Kernel Level Threads – Operating System managed threads acting on kernel, an operating system core.

1. User Level Threads

In this case, the thread management kernel is not aware of the existence of threads. The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts. The application starts with a single thread.



Advantages

- Thread switching does not require Kernel mode privileges.
- User level thread can run on any operating system.
- Scheduling can be application specific in the user level thread.
- User level threads are fast to create and manage.

Disadvantages

- In a typical operating system, most system calls are blocking.
- Multithreaded application cannot take advantage of multiprocessing.

2. Kernel Level Threads

In this case, thread management is done by the Kernel. There is no thread management code in the application area. Kernel threads are supported directly by the operating system. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.

The Kernel maintains context information for the process as a whole and for individuals threads within the process. Scheduling by the Kernel is done on a thread basis. The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.

Advantages

- Kernel can simultaneously schedule multiple threads from the same process on multiple processes.
- If one thread in a process is blocked, the Kernel can schedule another thread of the same process.
- Kernel routines themselves can be multithreaded.

Disadvantages

- Kernel threads are generally slower to create and manage than the user threads.
- Transfer of control from one thread to another within the same process requires a mode switch to the Kernel.

Q24. What are the differences between user level threads and kernel level thread?

Ans :

S.N.	User-Level Threads	Kernel-Level Thread
1	User-level threads are faster to create and manage.	Kernel-level threads are slower to create and manage.
2	Implementation is by a thread library at the user level.	Operating system supports creation of Kernel threads.
3	User-level thread is generic and can run on any operating system.	Kernel-level thread is specific to the operating system.
4	Multi-threaded applications cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.

Q25. Discuss briefly about various multi threading issues.

Ans :

1. Thread Cancellation

Thread cancellation means terminating a thread before it has finished working. There can be two approaches for this, one is Asynchronous cancellation, which terminates the target thread immediately. The other is Deferred cancellation allows the target thread to periodically check if it should be cancelled.

2. Signal Handling

Signals are used in UNIX systems to notify a process that a particular event has occurred. Now in when a Multithreaded process receives a signal, to which thread it must be delivered? It can be delivered to all, or a single thread.

3. fork() System Call

fork() is a system call executed in the kernel through which a process creates a copy of itself. Now the problem in Multithreaded process is, if one thread forks, will the entire process be copied or not?

4. Security Issues

Because of extensive sharing of resources between multiple threads

5. Thread-Local Storage

Most data is shared among threads, and this is one of the major benefits of using threads in the first place. Most major thread libraries (pThreads, Win32, Java) provide support for thread-specific data, known as thread-local storage or TLS. Note that this is more like static data than local variables, because it does not cease to exist when the function ends.

Scheduler Activations

Many implementations of threads provide a virtual processor as an interface between the user thread and the kernel thread, particularly for the many-to-many or two-tier models.

This virtual processor is known as a "Lightweight Process", LWP. There is a one-to-one correspondence between LWPs and kernel threads. The number of kernel threads available may change dynamically. The application (user level thread library) maps user threads onto available LWPs. kernel threads are scheduled onto the real processor(s) by the OS.

The kernel communicates to the user-level thread library when certain events occur (such as a thread about to block) via an upcall, which is handled in the thread library by an upcall handler. The upcall also provides a new LWP for the upcall handler to run on, which it can then use to reschedule the user thread that is about to become blocked. The OS will also issue upcalls when a thread becomes unblocked, so the thread library can make appropriate adjustments. If the kernel thread blocks, then the LWP blocks, which blocks the user thread.

1.2.4 Process Scheduling

Q26. Explain the concept of Process Scheduling.

Ans :

(Imp.)

A Single Process may also contain sub Processes those are also known as the Child Process. So that we can say that a Process which is given to the System is also known as the Parent Process and all the other Parts of the Single Process are known as the Child Process. So that Every Process may also Contains Some Child process.

For Example, if we are giving a Command to Print the File, and if a Single File Contains 8 pages to print. Then there are 8 small or child processes to print; and the Whole Process will be over when all of the 8 pages will be printed.

When a user Request for a Service from the System, then the System Automatically initializes the Process by using the initial State and the System also provides the various types of input and output Resources to the Process, Provides also Some Memory and also Control the Execution and Also Controls the State of the Process. So that in the Execution of Process, we doesn't implement only the Process creation but we also use the various Controlling Mechanism for the Process.

There are Many Types of Operating Systems which executed the Process either Single or Multiple Processes are executed at a Single time. And For Executing the Multiple Processes we must have to use Some Controlling Mechanisms.

There are Two Types of Scheduling

1. Preemptive

In this all the Processes are executed by using some Amount of Time of CPU. The Time of CPU is divided into the Number of Minutes and Time of CPU divided into the Process by using Some Rules. if the time is divided into equal interval than it is called Quantum Time. in the Preemptive Scheduling

Jobs are Executed one by one according to the Scheduling Techniques, But in this when the Higher Priority will Request for a Service. To the CPU, then CPU will transfer the Control to the Request Job, Means the Running job will wait for Some Time.

2. NON-Primitive

In this No Time Scheduling is used and in this CPU will be automatically free after Executing the Whole Process Means When the Execution of the Process will Completed then the CPU will be Free. When two or more Process are given then this will first Complete the Process and after Completing the First Process, this will Automatically start the Second Process.

Non-Preemptive Scheduling means No scheduling then all the Jobs are Executed One by One. And in this when the First Job will be Completed, after that second Job will Started.

1. First Come First Serve

As the name Suggest, the Processes those are Coming first, will be Executed first and Means CPU Will Creates a Queue, means all the Process are Inserted into the Queue and the CPU will Perform all the Process by using their Coming Order.. In this all the Process are arranged by the CPU and After Executing a

Single Process, then this will Automatically Execute second Process by Picking up the next Process.

2. Shortest Job first

In this Scheduling, All the Process are Arranged into their Size Means How Many Time a Process require, of CPU for Executing. CPU Arrange all the Processes according to the Requirement Time. CPU Executes the Processes by Examining the Time Required by Process. CPU Prepare a queue in which all the Processes are arranged by using the Number of Time Units Requires by the Process.

3. Priority Scheduling

When the Process are Given, then Each Process have a Priority means Some Preference issue. Which Job will be executed first, is determined by the CPU. After Examining the Priority of the CPU. Each Process takes different Time of CPU and also the Number of Inputs those are needed by the CPU. So CPU Maintains the Priority Level after Examining the Total time which a Process will consume. All the Processes are Arranged by using Some Priority,. Then CPU Executes the Process by using the Process Priority.

4. Round Robin

In this Scheduling the Time of CPU is divided into the Equal Parts and Assign to various Processes. In this Time of CPU is also known as Quantum Time. In the Round Robin, when the time of First Process has finished, then the CPU will execute the Second Process. But there also be possibility that the Process doesn't End, up to The Time. So that if process doesn't end at the End of Time. Then CPU uses the Context Switching, Means CPU Record the State of Process. After executing the other Processes, he will execute the First Process Again until the Process never ends.

5. Multilevel Queue Scheduling

In this The Time of CPU is divided by using Some Process Categories. In this the Process those are executed on the Foreground or on the Screen, have a higher Priority and the

Process those are running in the Background to fill the Request the user. When we Input the data into the Computer. Then the Data is displayed on the Screen after Processing.

Q27. Explain different types of Schedulers.

Ans :

Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types

- i) Long-Term Scheduler
- ii) Short-Term Scheduler
- iii) Medium-Term Scheduler

i) Long Term Scheduler

It is also called a job scheduler. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.

The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

On some systems, the long-term scheduler may not be available or minimal. Time-sharing operating systems have no long term scheduler. When a process changes the state from new to ready, then there is use of long-term scheduler.

ii) Short Term Scheduler

It is also called as CPU scheduler. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

iii) Medium Term Scheduler

Medium-term scheduling is a part of swapping. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes.

A running process may become suspended if it makes an I/O request. A suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called swapping, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

Q28. Write about scheduling criteria.

Ans :

There are many different criteria's to check when considering the "best" scheduling algorithm :

➤ **CPU utilization**

To make out the best use of CPU and not to waste any CPU cycle, CPU would be working most of the time(Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)

➤ **Throughput**

It is the total number of processes completed per unit time or rather say total amount of work done in a unit of time. This may range from 10/second to 1/hour depending on the specific processes.

➤ **Turnaround time**

It is the amount of time taken to execute a particular process, i.e. The interval from time of submission of the process to the time of completion of the process(Wall clock time).

➤ **Waiting time**

The sum of the periods spent waiting in the

ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU.

➤ **Load average**

It is the average number of processes residing in the ready queue waiting for their turn to get into the CPU.

➤ **Response time**

Amount of time it takes from when a request was submitted until the first response is produced. Remember, it is the time till the first response and not the completion of process execution (final response).

In general CPU utilization and Throughput are maximized and other factors are reduced for proper optimization.

Q29. Explain about various scheduling algorithms.

Ans : (Imp.)

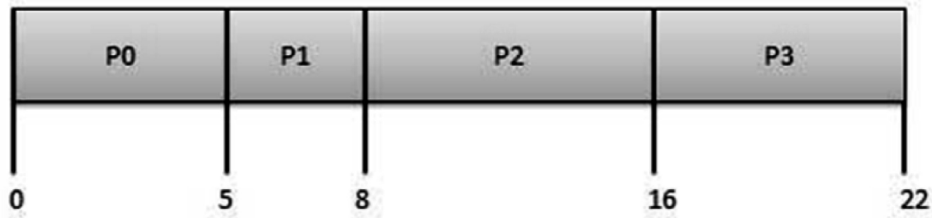
1. First Come First Serve (FCFS) Scheduling
2. Shortest-Job-First (SJF) Scheduling
3. Priority Based Scheduling
4. Shortest Remaining Time Scheduling
5. Round Robin (RR) Scheduling
6. Multilevel Queue Scheduling

These algorithms are either non-preemptive or preemptive. Non-preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

1. First Come First Serve (FCFS)

- Jobs are executed on first come, first serve basis.
- It is a non-preemptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16



Wait time of each process is as follows

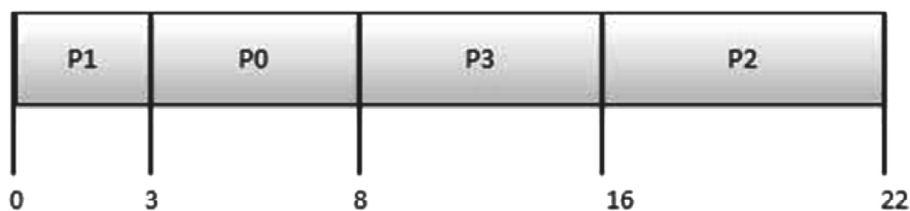
Process	Wait Time : Service Time - Arrival Time
P0	$0 - 0 = 0$
P1	$5 - 1 = 4$
P2	$8 - 2 = 6$
P3	$16 - 3 = 13$

Average Wait Time: $(0 + 4 + 6 + 13) / 4 = 5.75$

2. Shortest Job Next (SJN)

- This is also known as shortest job first, or SJF
- This is a non-preemptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processor should know in advance how much time process will take.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	3
P1	1	3	0
P2	2	8	16
P3	3	6	8



Wait time of each process is as follows

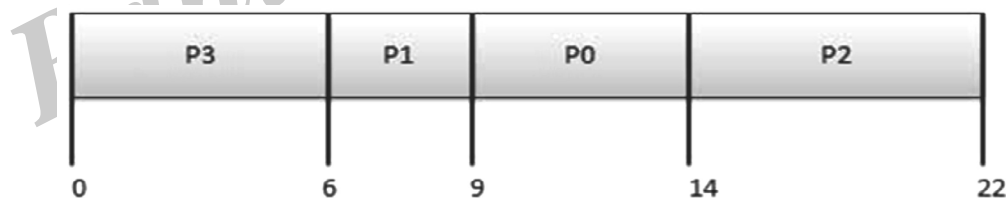
Process	Wait Time : Service Time - Arrival Time
P0	$3 - 0 = 3$
P1	$0 - 0 = 0$
P2	$16 - 2 = 14$
P3	$8 - 3 = 5$

Average Wait Time: $(3+0+14+5) / 4 = 5.50$

3. Priority Based Scheduling

- Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.
- Each process is assigned a priority. Process with highest priority is to be executed first and so on.
- Processes with same priority are executed on first come first served basis.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

Process	Arrival Time	Execute Time	Priority	Service Time
P0	0	5	1	9
P1	1	3	2	6
P2	2	8	1	14
P3	3	6	3	0



Wait time of each process is as follows

Process	Wait Time : Service Time - Arrival Time
P0	$9 - 0 = 9$
P1	$6 - 1 = 5$
P2	$14 - 2 = 12$
P3	$0 - 0 = 0$

Average Wait Time: $(9+5+12+0) / 4 = 6.5$

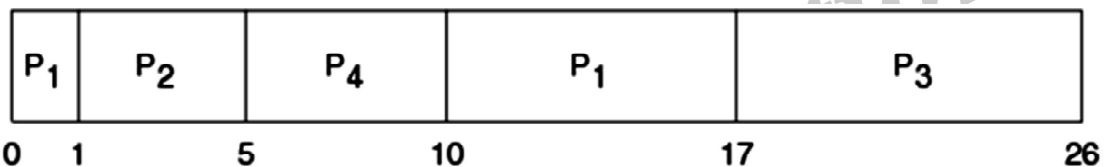
4. Shortest Remaining Time Scheduling

- Shortest remaining time (SRT) is the pre-emptive version of the SJN algorithm.
- The processor is allocated to the job closest to completion but it can be preempted by a newer ready job with shorter time to completion.
- Impossible to implement in interactive systems where required CPU time is not known.
- It is often used in batch environments where short jobs need to give preference.

Example 1:

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

➤ Gantt chart



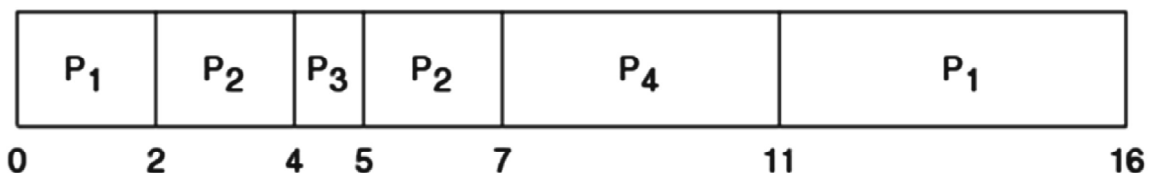
average waiting time: $(9+0+15+2)/4 = 6.5$

average turnaround time: $(17+4+24+7)/4 = 13$

Example 2 :

Process	Burst Time	Arrival	Start	Wait	Finish	TA
1	7	0	0	9	16	16
2	4	2	2	1	7	5
3	1	4	4	0	5	1
4	4	5	7	2	11	6

➤ Gantt chart

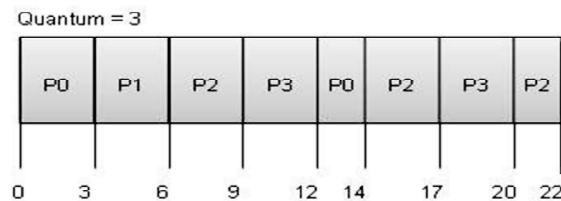


➤ average waiting time: $(9+1+0+2)/4 = 3$

➤ average turnaround time: $(16+5+1+6)/4 = 7$

5. Round Robin Scheduling

- Round Robin is the preemptive process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a quantum.
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes.



Wait time of each process is as follows

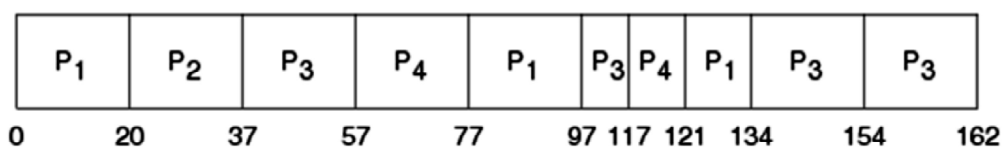
Process	Wait Time : Service Time - Arrival Time
P0	$(0 - 0) + (12 - 3) = 9$
P1	$(3 - 1) = 2$
P2	$(6 - 2) + (14 - 9) + (20 - 17) = 12$
P3	$(9 - 3) + (17 - 12) = 11$

Average Wait Time: $(9 + 2 + 12 + 11) / 4 = 8.5$

Example : (q = 20):

Process	Burst Time	Arrival	Start	Wait	Finish	TA
1	53	0	0	?	134	134
2	17	0	20	?	37	37
3	68	0	37	?	162	162
4	24	0	57	?	121	121

Gantt chart :



waiting times:

$$p_1: (77 - 20) + (121 - 97) = 81$$

$$p_2: (20 - 0) = 20$$

$$p_3: (37-0) + (97-57) + (134-117) = 94$$

$$p_4: (57-0) + (117-77) = 97$$

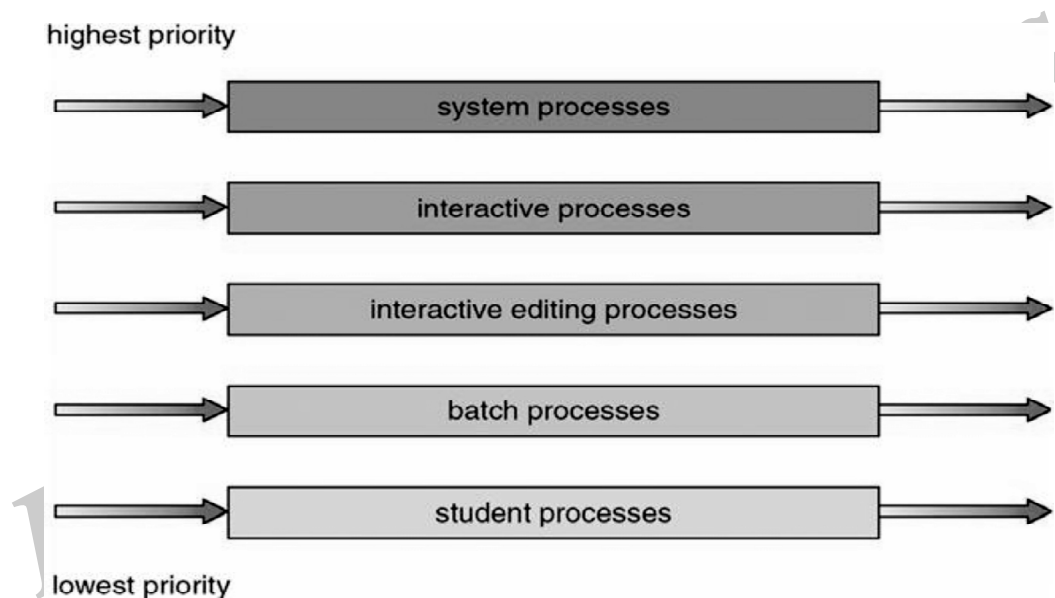
$$\text{average waiting time: } (81+20+94+97)/4 = 73$$

6. Multiple-Level Queues Scheduling

Multiple-level queues are not an independent scheduling algorithm. They make use of other existing algorithms to group and schedule jobs with common characteristics.

- Multiple queues are maintained for processes with common characteristics.
- Each queue can have its own scheduling algorithms.
- Priorities are assigned to each queue.

For example, CPU-bound jobs can be scheduled in one queue and all I/O-bound jobs in another queue. The Process Scheduler then alternately selects jobs from each queue and assigns them to the CPU based on the algorithm assigned to the queue.



Q30. What is multi processor scheduling explain?

Ans :

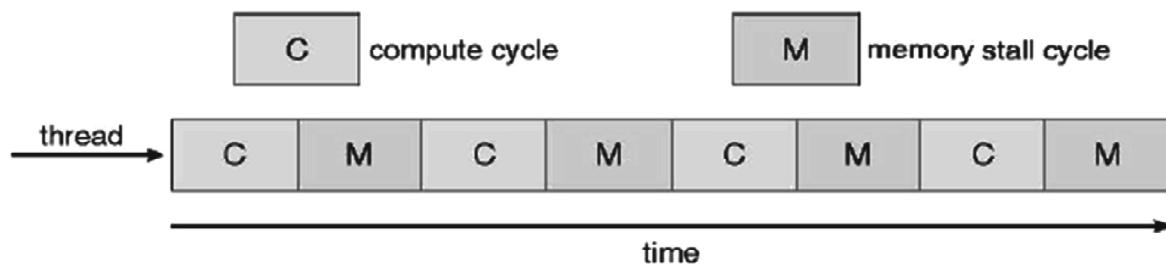
- **CPU scheduling more complex when multiple CPUs are available**
 - Most current general purpose processors are multiprocessors (i.e. multicore processors)
 - No single 'best' solution to multiple-processor scheduling
- A multicore processor typically has two or more homogeneous processor cores
 - Because the cores are all the same, any available processor can be allocated to any process in the system. Approaches to Multiple-Processor Scheduling
- **Asymmetric multiprocessing**
 - All scheduling decisions, I/O processing, and other system activities handled by a single processor
 - Only one processor accesses the system data structures, alleviating the need for data sharing

➤ **Symmetric multiprocessing (SMP)**

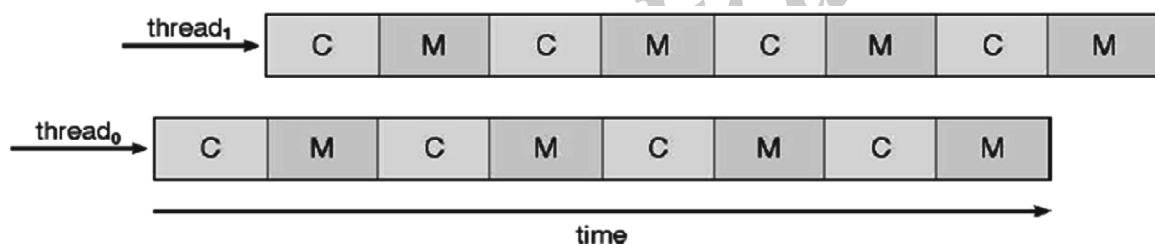
- Each processor is self-scheduling
- All processes may be in a common ready queue, or each processor may have its own private queue of ready processes
- Currently, most common approach to multiple-processor scheduling

➤ **Multithreaded Multicore System**

- A processor core with a single hardware thread doesn't accomplish any work during a memory stall



- A processor core with multiple hardware threads can work on another thread while waiting for a memory read to complete



Scheduling Multithreaded Multicore Systems

Two levels of scheduling must take place

1. Operating system is still scheduling tasks based on its scheduling algorithms
2. Second level of scheduling decides which hardware threads to run
 - Coarse-grained multithreading - a thread executes on a processor until a long-latency event occurs (i.e. reading from memory)
 - Fine-grained multithreading

1.2.5 Process Synchronization

Q31. What is process synchronization? Explain critical section problem with its different solutions.

Ans :

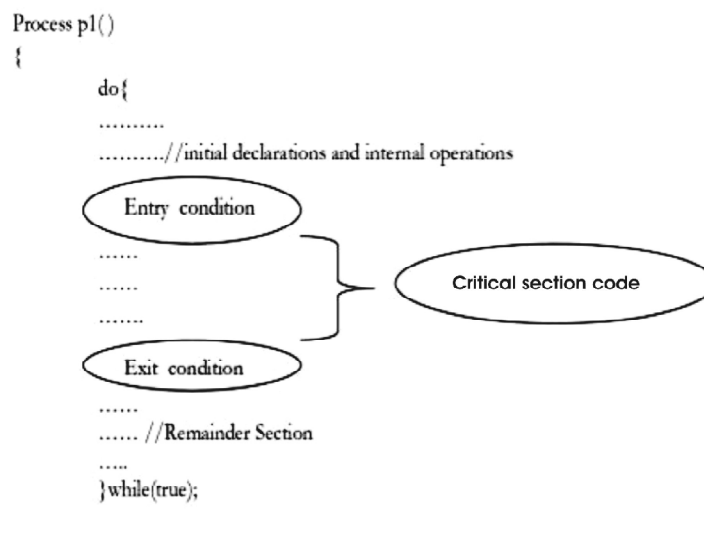
(Imp.)

Process Synchronization means sharing system resources by processes in a such a way that, Concurrent access to shared data is handled thereby minimizing the chance of inconsistent data. Maintaining data consistency demands mechanisms to ensure synchronized execution of cooperating processes.

Process Synchronization was introduced to handle problems that arose while multiple process executions. Some of the problems are discussed below.

Critical Section Problem

- Consider a system consisting of n processes P_0, P_1, \dots, P_n .
- Each process has a segment of code, called a critical section, in which the process may be changing common variables, updating a table, writing a file, and so on. It is section of code that requires access to shared resources.
- An important fact regarding this system is that when a process A is executing in its critical section, no other process except A is allowed to execute in its critical section.
- However many processes may have a critical section code and all of them needs to be executed. So, The Critical Section Problem deals with creating a set of protocols so as to help the process cooperate.
- The code –body shown below depicts how and where the Critical Section code is implemented.



- **Entry Condition:** Process requests for permission to enter the critical section
- **Critical section:** A portion of code within a process that needs access to shared resources and that must not be executed while another process is in its critical section.
- **Exit Condition:** Every critical section must end with an exit condition which alerts the system regarding the exit.

A solution to a critical section problem must satisfy three conditions;

- (a) **Mutual Exclusion:** If a process A is executing in its critical section, then no other processes must execute in its critical section.
- (b) **Progress:** If no process is currently in its critical section, then only those process which are currently not in its remainder section can participate in deciding who will enter the Critical section next.
- (c) **Bounded waiting:** There exists a bound with regards to the number of times other process can enter its critical section between when a process has requested for critical section and when that request is granted

The various solutions to it are:

(a) Petersons solution

Peterson's solution is restricted to two processes that alternate execution between their critical sections and remainder sections. The processes are named P0 and P1. The Petersons solutions asks both the processes to share two data items

```
int turn; //to indicate whose turn is to enter the CS
boolean flag[2]; //to indicate if a process is ready
                //to enter its CS
```

The algorithm for Peterson solution is:

```
do {
    //entry condition
    Flag[i]=true;
    Turn=j;
    while(flag[j] && turn == j)
    CRITICAL SECTION
    //exit condition
    Flag[i]=false;
    //remainder section
} while(true);
```

(b) Hardware based

Many modern computer systems provide a special hardware instruction that allows us either to test and modify the content of a word or to swap the contents of two words atomically—that is, as one uninterruptible unit. It is implemented using either of the two instructions:

`test_and_set()`: this instruction can be executed atomically. Therefore if two such instructions are executed simultaneously on a different PC, then it will be executed in some arbitrary order
`compare_and_swap()`: it also operates atomically as a single uninterruptible unit. It operates on three operands unlike `test_and_set` and also returns the original value of the variable value

(c) Mutex lock

The hardware locks are generally inaccessible to application programmers; instead of it we use software tools-the simplest being mutex. It is short form for mutual exclusion. We use mutex lock to protect critical regions and thus avoid race conditions.

Every process will have to acquire a lock before entering the Critical section (using `acquire()` function) and release the lock when exiting (using the `release()` function). Calls to the two functions mentioned above must be done atomically which is done by the hardware methods described in method (b) above.

(d) Semaphores

Semaphores are similar to the mutex lock except that they are much robust and can provide sophisticated ways for the process to synchronize their activities. A semaphore S is an integer variable and can only be accessed and modified by two functions `wait()` and `signal()`. It can be initialized to a non-negative value. The wait operation decrements the semaphore value while the signal operation increments the semaphore value

Q32. Explain briefly about semaphores.

Ans :

Dijkstra proposed a significant technique for managing concurrent processes for complex mutual exclusion problems. He introduced a new synchronization tool called Semaphore.

Semaphores are of two types

1. Binary semaphore
2. Counting semaphore

Binary semaphore can take the value 0 & 1 only. Counting semaphore can take nonnegative integer values.

Two standard operations, wait and signal are defined on the semaphore. Entry to the critical section is controlled by the wait operation and exit from a critical region is taken care by signal operation. The wait, signal operations are also called P and V operations. The manipulation of semaphore (S) takes place as following:

1. The wait command P(S) decrements the semaphore value by 1. If the resulting value becomes negative then P command is delayed until the condition is satisfied.
2. The V(S) i.e. signals operation increments the semaphore value by 1.

Mutual exclusion on the semaphore is enforced within P(S) and V(S). If a number of processes attempt P(S) simultaneously, only one process will be allowed to proceed & the other processes will be waiting. These operations are defined as under "

P(S) or wait(S):

If $S > 0$ then

Set S to S-1

Else

Block the calling process (i.e. Wait on S)

V(S) or signal(S):

If any processes are waiting on S

Start one of these processes

Else

Set S to S+1

The semaphore operation are implemented as operating system services and so wait and signal are atomic in nature i.e. once started, execution of these operations cannot be interrupted.

Thus semaphore is a simple yet powerful mechanism to ensure mutual exclusion among concurrent processes.

Q33. Explain briefly about Monitors.

Ans :

Monitor is one of the ways to achieve Process synchronization. Monitor is supported by programming languages to achieve mutual exclusion between processes. For example Java Synchronized methods. Java provides wait() and notify() constructs.

1. It is the collection of condition variables and procedures combined together in a special kind of module or a package.
2. The processes running outside the monitor can't access the internal variable of monitor but can call procedures of the monitor.
3. Only one process at a time can execute code inside monitors.

Syntax of Monitor

```
Monitor Demo IIName of Monitor
{
variables;
condition variables;
procedure p1
prodecure p2 {...}
}

Syntax of Monitor
```

Condition Variables

Two different operations are performed on the condition variables of the monitor.

Wait.

signal.

let say we have 2 condition variables

condition x, y; //Declaring variable

Wait operation

x.wait() : Process performing wait operation on any condition variable are suspended. The suspended processes are placed in block queue of that condition variable.

Note: Each condition variable has its unique block queue.

Signal operation

x.signal(): When a process performs signal operation on condition variable, one of the blocked processes is given chance.

If (x block queue empty)

// Ignore signal

else

// Resume a process from block queue.

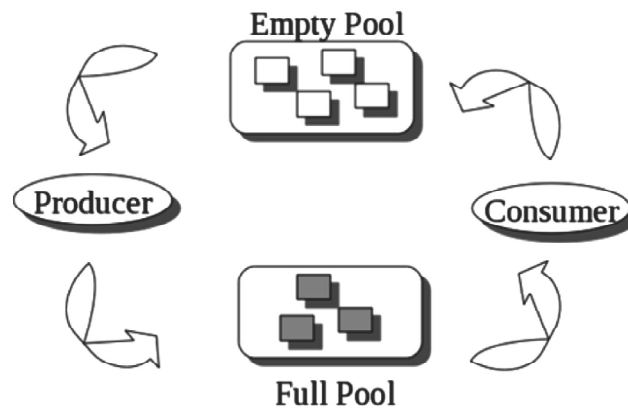
Q34. What is bounded buffer problem? Write about it.

Ans :

This problem is also called the Producers and Consumers problem. A finite supply of containers is available. Producers take an empty container and fill it with a product. Consumers take a full container, consume the product and leave an empty container. The main complexity of this problem is that we must maintain the count for both the number of empty and full containers that are available.

Application : A pipe or other finite queue (buffer), is an example of the bounded buffer problem.

Solution to this problem is, creating two counting semaphores "full" and "empty" to keep track of the current number of full and empty buffers respectively.



Producers produce a product and consumers consume the product, but both use one of the containers each time.

Problem: There is a set of resource buffers shared by producer and consumer threads

- Producer inserts resources into the buffer set
 - Output, disk blocks, memory pages, processes, etc.
- Consumer removes resources from the buffer set
 - Whatever is generated by the producer
- Producer and consumer execute at different rates
 - No serialization of one behind the other
 - Tasks are independent (easier to think about)

The buffer set allows each to run without explicit handoff

- Use three semaphores:
 - mutex – mutual exclusion to shared set of buffers
- Binary semaphore
 - empty – count of empty buffers
- Counting semaphore

- full – count of full buffers
- Counting semaphore

```
Semaphore mutex = 1; // mutual exclusion to shared set of buffers
Semaphore empty = N; // count of empty buffers (all empty to start)
Semaphore full = 0; // count of full buffers (none full to start)
```

```
producer {
while (1) {
Produce new resource;
wait(empty); // wait for empty buffer
wait(mutex); // lock buffer list
Add resource to an empty buffer;
signal(mutex); // unlock buffer list
signal(full); // note a full buffer
}
}
```

```
consumer {
while (1) {
wait(full); // wait for a full buffer
wait(mutex); // lock buffer list
Remove resource from a full buffer;
signal(mutex); // unlock buffer list
signal(empty); // note an empty buffer
Consume resource;
}
}
```

Q35. What is readers writers problem explain about it?

Ans :

The Readers Writers Problem

- In this problem there are some processes (called readers) that only read the shared data, and never change it, and there are other processes (called writers) who may change the data in addition to reading or instead of reading it.
- There are various type of the readers-writers problem, most centred on relative priorities of readers and writers

Application : A message distribution system is an example of Readers and Writers. We must keep track of how many messages are in the queue.



- Writers have mutual exclusion, but multiple readers at the same time is allowed.
A semaphore solution to the readers' priority version (without addressing starvation):
- Writers have mutual exclusion, but multiple readers at the same time is allowed.
A semaphore solution to the readers' priority version (without addressing starvation):


```

// number of readers
int readcount = 0;
// mutual exclusion to readcount
Semaphore mutex = 1;
// exclusive writer or reader
Semaphore w_or_r = 1;

writer {
    wait(w_or_r); // lock out readers
    Write;
    signal(w_or_r); // up for grabs
}

```

```

reader {
    wait(mutex);    // lock readcount
    readcount += 1; // one more reader
    if (readcount == 1)
        wait(w_or_r); // synch w/ writers
    signal(mutex); // unlock readcount
    Read;
    wait(mutex);    // lock readcount
    readcount -= 1; // one less reader
    if (readcount == 0)
        signal(w_or_r); // up for grabs
    signal(mutex); // unlock readcount
}

```

- If there is a writer
 - First reader blocks on w_or_r
 - All other readers block on mutex
- Once a writer exits, all readers can fall through
 - Which reader gets to go first?
- The last reader to exit signals a waiting writer
 - If no writer, then readers can continue
- If readers and writers are waiting on w_or_r, and a writer exits,

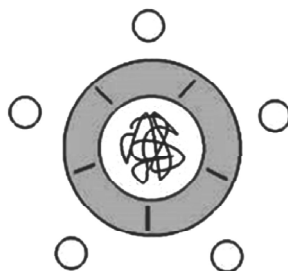
Q36. What is dining philosophers problem? Write about it.

Ans :

(Imp.)

- The dining philosopher's problem involves the allocation of limited resources from a group of processes in a deadlock-free and starvation-free manner.
- There are five philosophers sitting around a table, in which there are five chopsticks kept beside them and a bowl of rice in the centre, When a philosopher wants to eat, he uses two chopsticks - one from their left and one from their right. When a philosopher wants to think, he keeps down both chopsticks at their original place.

Application : The dining philosophers problem represents a situation that can occur in large communities of processes that share a sizeable pool of resources.



A Philosopher needs both the fork on his left and on his right to eat. The forks are shared with the neighbours on either side.

- It can be observed that a straightforward solution, when forks are implemented by semaphores, is exposed to deadlock. There exist two deadlock states when all five philosophers are sitting at the table holding one fork each. One deadlock state is when each philosopher has grabbed the fork left of him, and another is when each has the fork on his right.

A semaphore solution:

```
// represent each chopstick with a semaphore
Semaphore chopstick[] = new Semaphore[5]; // all = 1 initially
processphilosopher_i {
while (true) {
    // pick up left chopstick
    chopstick[i].acquire();
    // pick up right chopstick
    chopstick[(i + 1) % 5].acquire();
    // eat
    // put down left chopstick
    chopstick[i].release();
    // put down right chopstick
    chopstick[(i + 1) % 5].release();

    // think
}
}
```

This solution guarantees no two neighbouring philosophers eat simultaneously, but has the possibility of creating a deadlock.

1.2.6 Deadlocks

Q37. What is Dead Lock? Explain it briefly.

Ans :

In a multi programming environment, several processes may compete for a finite number of resources. A process requests resources, and if the resources are not available at that time, process enters in a waiting state. Sometimes a waiting process is never again able to change state, because the resources it has requested are held by another processes. This situation is called Deadlock.

System Model

A system consists of a finite number of resources to be distributed among a number of competing processes. The resources are partitioned into several types, each consisting of some number of identical

instances. for example if a system has five printer then printer is a resource type and it has five instances.

Identical Resources

If a process requests an instance of a resource type, and allocation of any instance of that resource type will satisfy the request, then the resource instance are identical. Otherwise, the resource type classes have not been defined properly.

Under the normal mode of operation, a process may utilize a resource in only the following sequence:

1. **Request :** If the request can not be granted immediately (for example, if the resource is being used by another process), then the requesting process must wait until it can acquire the resource.
2. **Use :** The process can operate on the resource (for example, if the resource is a printer, the process can print on the printer).
3. **Release :** The process release the resource.

Q38. What are the necessary conditions for deadlocks?

Ans :

A deadlock situation can arise if the following four conditions hold simultaneously in a system.

(i) Mutual Exclusion

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

(ii) Hold and Wait

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

(iii) No Preemption

Resources can not be preempted, that is, a resource can be released only voluntarily by the process holding it, after that process has completed its task.

(iv) Circular Wait

A set $\{P_0, P_1, P_2, \dots, P_n\}$ of waiting processes must exist such that P_0 is waiting for a resource held by P_1 , P_1 is waiting for a resource held by P_2 , ..., P_{n-1} is waiting for a resource held by P_n and P_n is waiting for a resource held by P_0 .

These four conditions must hold simultaneously for a deadlock to occur. The circular-wait condition implies the hold and wait condition, so these four conditions are not completely independent.

Q39. What is the use of resource allocation graphs in OS.

Ans :

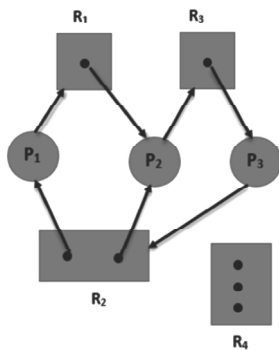
Deadlock can be described more precisely in terms of a directed graph called a system resource allocation graph. This graph consists of a set of vertices V and a set of edges E . The set of vertices is partitioned into two different types of nodes

- $P = \{P_1, P_2, \dots, P_n\}$, the set consisting of all the active processes in the system.
- $R = \{R_1, R_2, \dots, R_n\}$, the set consisting of all resource types in the system.

A directed edge from process P_i to resource type R_j is denoted by $P_i \rightarrow R_j$, it signifies that process P_i has requested an instance of resource type R_j , and is currently waiting for the resource. This type of edge is called a request edge.

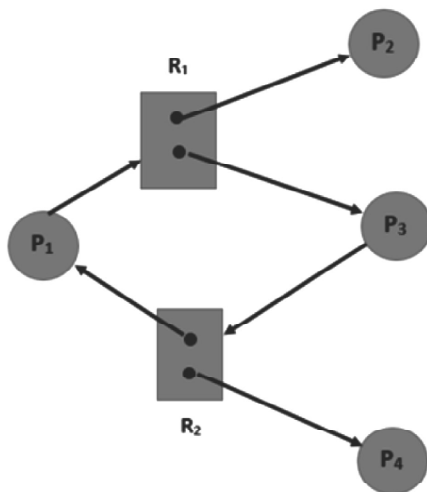
A directed edge from resource type R_j to process P_i is denoted by $R_j \rightarrow P_i$, it signifies that an instance of resource type R_j has been allocated to process P_i . This type of edge is called an assignment edge.

Pictorially, we represent each process P_i as a circle and each resource type R_j as a rectangle. Since resource type R_j may have more than one instance, we represent each such instance as a dot within the rectangle. Note that a request edge points to only the rectangle R_j , whereas an assignment edge must also designate one of the dots in the rectangle.



Resource Allocation Graph

If each resource type has exactly one instance, then a cycle implies that a deadlock has occurred. If the cycle involves only a set of resource type, each of which has only a single instance, then a deadlock has occurred. Each process involved in the cycle is deadlocked. In this case a cycle in the graph is necessary and sufficient condition for the existence of deadlock.



RAG with a Cycle But no Deadlock

If each resource type has several instances, then a cycle does not necessarily imply that a deadlock has occurred. In this case a cycle in the graph is a necessary but not a sufficient condition for existence of deadlock.

Q40. What are the various methods for handling the deadlocks ?

Ans :

we can handle the deadlock in one of the three ways:

- We can use a protocol to prevent or avoid the deadlocks, ensuring that the system will never enter in a deadlock states.
- We can allow the system to enter in a deadlock state, detect it and recover.
- We can ignore the problem altogether and pretend that deadlocks never occur in the system.

The third solution is the one used by most operating system, including UNIX, and WINDOWS, it is then up to the application developer to write programs that handle the deadlocks.

To ensure that deadlocks never occur, the system can use either a deadlock-prevention or deadlock-avoidance scheme.

Deadlock prevention provides a set of methods for ensuring that at least one of the necessary conditions can not hold. These methods prevent deadlocks by constraining how requests for the resources can be made.

Deadlock avoidance requires that the operating system be given in advance additional information concerning which resources a process will request and use during its lifetime. With this additional knowledge, it can decide for each request whether or not the process should wait. To decide whether the current request can be satisfied or must be delayed, the system must consider the resources currently available, the resources currently allocated to each process, and the future requests and releases of each processes.

If a system does not employ either a deadlock prevention or a deadlock avoidance algorithm, then a deadlock situation may arise. In this environment, the system can provide an algorithm that examines the state of the system to determine whether a deadlock has occurred and an algorithm to recover from the deadlock.

If a system neither ensures that a deadlock will never occur nor provides a mechanism for deadlock detection and recovery, then we may arrive at a situation where the system is in a deadlocked state, yet has no way of recognizing what has happened. In this case, the undetected deadlock will result in deterioration of the system's performance, because resources are being held

by processes that can not run and because more and more processes, as they make requests for resources, will enter in a deadlocked state. Eventually, the system will stop functioning and will need to be restarted normally. Although this method may not seem to be viable approach to the deadlock problem, it is nevertheless used in most operating system.

Q41. Write about deadlock prevention techniques.

Ans : (Imp.)

By ensuring that atleast one of these conditions can not hold, we can prevent the occurrence of a deadlock.

i) Mutual Exclusion

The mutual exclusion condition must hold for non sharable resources. For example, a printer can not be simultaneously shared by several processes. Sharable resources, in contrast, do not require mutually exclusive access and thus can not involved in a deadlock. Read only files are a good example of a sharable resource. However, in general, we can not prevent deadlocks by denying the mutual-exclusion condition, because some resources are intrinsically non sharable.

ii) Hold and Wait

To ensure that hold and wait condition never occurs in the system, we must guarantee that, whenever a process requests a resource, it does not hold any other resurces.

One protocol that can be used requires each process to request and be allocated all its resources before it begins execution. We can implement this provision by requiring that system calls requesting resources for a process precede all other system calls.

An alternative protocol allows a process to request resources only when it has none. A process may request some resources and use them. Before it can request any additional resources, however, it must release all the resources that it is currently allocated.

Both these protocols have two main disadvantage.

- Resource utilization may be low, since resources may be allocated but unused for a long period.
- Starvation is possible. A process that needs several popular resources may have to wait indefinitely, because at least one of the resources that it needs is always allocated to some other process.

iii) No preemption

To ensure that no preemption condition does not hold, we can use one of the following two approaches:

- If a process is holding some resources and requests another resource that can not be immediately allocated to it, then all the resources currently being held are preempted. In other words, these resources are implicitly released. The preempted resources are added to the list of resources for which the process is waiting. The process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.
- If a process requests some resources, we first check whether they are available. If they are we allocate them. If they are not, we check whether they are allocated to some other process that is waiting for additional resources. If so we preempt the desired resource from the waiting process and allocate them to the requesting process. If the resources are neither available nor held by waiting process, the requesting process must wait. While it is waiting, some of its resources may be preempted, but only if another process requests them. A process can be restarted only when it is allocated the new resources it is requesting and recovers any resources that were preempted while it is waiting.

These protocol is often applied to resources whose state can be easily saved and restarted later such as CPU registers and memory spaces. It cannot generally be applied to such resources as printers and tape drives.

iv) Circular Wait

To ensure that circular wait condition never holds is to impose a total ordering of all resource types and to require that each process requests resources in an increasing order of enumeration.

To illustrate it, let's assume that $R = \{R_1, R_2, R_3, \dots, R_m\}$ be a set of resource types. We assign to each resource type a unique integer number, which allows us to compare two resources and to determine whether one precedes another in our ordering. Formally we define a one-to-one function $F: R \rightarrow \mathbb{N}$, where \mathbb{N} is the set of the natural numbers. For example,

If the resource type R includes tape-drives, disk-drives and printers then, the function F may be defined as

$$F(\text{tape-drive}) = 1, F(\text{disk-drives}) = 5, F(\text{printer}) = 12$$

We can now consider the following protocol to prevent deadlocks.

- Each process can request resources only in an increasing order of enumeration. That is, a process can initially request any number of instances of a resource type say, R_i , after that the process can request instances of resource type, R_j , if and only if, $F(R_j) \geq F(R_i)$. If several instances of the same resource type are needed a single request for all of them must be issued.
- Whenever, a process requests an instance of resource type R_j it has released any resources R_i such that $F(R_i) \geq F(R_j)$.

If these two protocols are used, then the circular wait condition can not hold.

Q42. Explain deadlock avoidance techniques.

Ans :

Deadlock prevention algorithms, prevent deadlock by restraining how request can be made. for more details read Methods for Handling Deadlocks and Introduction to Deadlocks. An alternative method for avoiding deadlocks is to require additional information about how resources are to be requested. Each request requires that in making this decision the system consider the

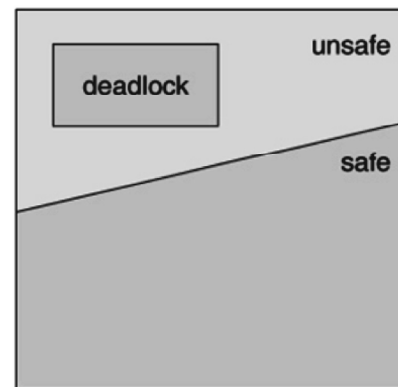
resources currently available, the resources currently allocated to each process, and the future requests and releases of each process.

A deadlock avoidance algorithm dynamically examines the resource allocation state to ensure that a circular wait condition can never exist. The resource allocation state is defined by the number of available and allocated resources and the maximum demands of the processes.

1. Safe State

A state is safe if the system can allocate resources to each process (Up to its maximum) in some order and still avoid a deadlock. More formally, a system is in a safe state only if there exists a safe sequence.

A sequence of processes $\langle P_1, P_2, \dots, P_n \rangle$ is a safe sequence for the current allocation state if, for each P_i , the resource requests that P_i can still make can be satisfied by the currently available resources plus the resources held by all P_j , with $j < i$. If no such sequence exists then the system state is said to be unsafe.



Safe, Unsafe and Deadlock State Spaces

A safe state is never a deadlocked state. Conversely, a deadlocked state is an unsafe state. Not all unsafe states are deadlocks, however, an unsafe state may lead to a deadlock. As long as the state is safe, the operating system can avoid unsafe (and deadlocked) state. In an unsafe state, the operating system can not prevent processes from requesting resources such that a deadlock occurs: The behavior of the processes controls unsafe states.

For example, consider a system with 12 tape drives, allocated as follows. Is this a safe state? What is the safe sequence?

	Maximum Needs	Current Allocation
P0	10	5
P1	4	2
P2	9	2

What happens to the above table if process P2 requests and is granted one more tape drive?

Key to the safe state approach is that when a request is made for resources, the request is granted only if the resulting allocation state is a safe one.

2 Resource-Allocation Graph Algorithm

- If resource categories have only single instances of their resources, then deadlock states can be detected by cycles in the resource-allocation graphs.
- In this case, unsafe states can be recognized and avoided by augmenting the resource-allocation graph with claim edges, noted by dashed lines, which point from a process to a resource that it may request in the future.
- In order for this technique to work, all claim edges must be added to the graph for any particular process before that process is allowed to request any resources.
- When a process makes a request, the claim edge $P_i \rightarrow R_j$ is converted to a request edge. Similarly when a resource is released, the assignment reverts back to a claim edge.
- This approach works by denying requests that would produce cycles in the resource-allocation graph, taking claim edges into effect.

Consider for example what happens when process P2 requests resource R2:

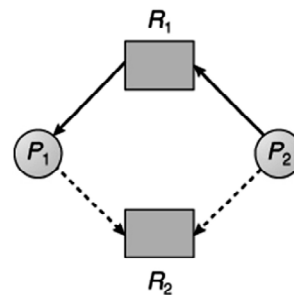


Fig.: Resource allocation graph for deadlock avoidance

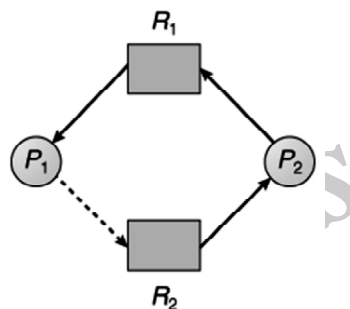


Fig.: An unsafe state in a resource allocation graph

Q43. Explain Banker's Algorithm with an example.

Ans :

Banker's algorithm is a deadlock avoidance algorithm. It is named so because this algorithm is used in banking systems to determine whether a loan can be granted or not.

Consider there are n account holders in a bank and the sum of the money in all of their accounts is S . Everytime a loan has to be granted by the bank, it subtracts the loan amount from the total money the bank has. Then it checks if that difference is greater than S . It is done because, only then, the bank would have enough money even if all the n account holders draw all their money at once.

Banker's algorithm works in a similar way in computers. Whenever a new process is created, it must exactly specify the maximum instances of each resource type that it needs.

Let us assume that there are n processes and m resource types. Some data structures are used to implement the banker's algorithm. They are:

- **Available:** It is an array of length m . It represents the number of available resources of each type. If $Available[j] = k$, then there are k instances available, of resource type R_j .
- **Max:** It is an $n \times m$ matrix which represents the maximum number of instances of each resource that a process can request. If $Max[i][j] = k$, then the process P_i can request at most k instances of resource type R_j .
- **Allocation:** It is an $n \times m$ matrix which represents the number of resources of each type currently allocated to each process. If $Allocation[i][j] = k$, then process P_i is currently allocated k instances of resource type R_j .
- **Need:** It is an $n \times m$ matrix which indicates the remaining resource needs of each process. If $Need[i][j] = k$, then process P_i may need k more instances of resource type R_j to complete its task.

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

Resource Request Algorithm:

This describes the behavior of the system when a process makes a resource request in the form of a request matrix. The steps are:

1. If number of requested instances of each resource is less than the need (which was declared previously by the process), go to step 2.
2. If number of requested instances of each resource type is less than the available resources of each type, go to step 3. If not, the process has to wait because sufficient resources are not available yet.
3. Now, assume that the resources have been allocated. Accordingly do,

$$Available = Available - Request_i$$

$$Allocation_i = Allocation_i + Request_i$$

$$Need_i = Need_i - Request_i$$

This step is done because the system needs to assume that resources have been allocated. So there will be less resources available after allocation.

The number of allocated instances will increase. The need of the resources by the process will reduce. That's what is represented by the above three operations.

After completing the above three steps, check if the system is in safe state by applying the safety algorithm. If it is in safe state, proceed to allocate the requested resources. Else, the process has to wait longer.

Safety Algorithm :

1. Let $Work$ and $Finish$ be vectors of length m and n , respectively. Initially,
2. $Work = Available$
3. $Finish[i] = false$ for $i = 0, 1, \dots, n - 1$.

This means, initially, no process has finished and the number of available resources is represented by the $Available$ array.

4. Find an index i such that both
5. $Finish[i] == false$
6. $Need_i \leq Work$

If there is no such i present, then proceed to step 4.

It means, we need to find an unfinished process whose need can be satisfied by the available resources. If no such process exists, just go to step 4.

7. Perform the following:
8. $Work = Work + Allocation_i$
9. $Finish[i] = true$;

Go to step 2.

When an unfinished process is found, then the resources are allocated and the process is marked finished. And then, the loop is repeated to check the same for all other processes.

10. If $Finish[i] == true$ for all i , then the system is in a safe state.

That means if all processes are finished, then the system is in safe state.

Example:

Considering a system with five processes P_0 through P_4 and three resource types A, B, C. Resource type A has 10 instances, B has 5 instances and type C has 7 instances. Suppose at time t_0 following snapshot of the system has been taken:

Process	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

Q44. Explain briefly about deadlock detection techniques.

Ans :

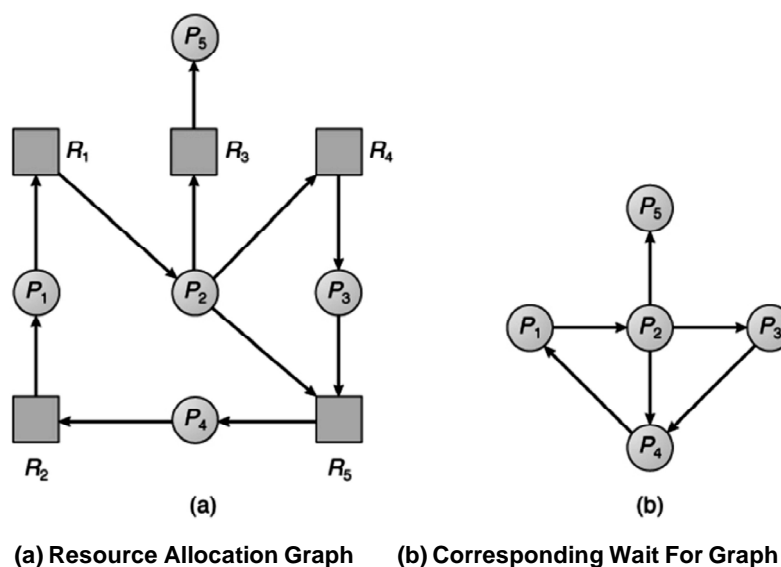
If a system does not employ either a deadlock-prevention or a deadlock-avoidance algorithm, then deadlock situation may occur. In this environment, the system must provide:

- An algorithm that examines the state of the system to determine whether a deadlock has occurred.
- An algorithm to recover from the deadlock.

In the following discussion we elaborate these two requirements as they pertain to system with only a single instances of each type as well as to system with several instances of each resource type. Detection and recovery algorithm require overheads, in addition to the performance hit of constantly checking for deadlocks, a policy / algorithm must be in place for recovering from deadlocks, and there is potential for lost work when processes must be aborted or have their resources preempted.

1. Single Instance of Each Resource Type

If each resource category has a single instance, then we can use a variation of the resource-allocation graph known as a wait-for graph. A wait-for graph can be constructed from a resource-allocation graph by eliminating the resources and collapsing the associated edges, as shown in the figure below.



An arc from P_i to P_j in a wait-for graph indicates that process P_i is waiting for a resource that process P_j is currently holding. As before, cycles in the wait-for graph indicate deadlocks. This algorithm must maintain the wait-for graph, and periodically search it for cycles. An algorithm to detect a cycle in a graph requires an order of n^2 operations, where n is the number of vertices in the graph.

2. Several Instances of a Resource Type

The wait-for graph scheme is not applicable to a resource allocation system with multiple instances of each resource type. Now, we describe deadlock detection algorithm for multiple instances of a resource type. The algorithm employs several time-varying data structures that are similar to those used in the banker's algorithm.

Available: A vector of length m indicates the number of available resources of each type.

Allocation: An $n \times m$ matrix defines the number of resources of each type currently allocated to each process.

Request: An $n \times m$ matrix indicates the current request of each process. If $Request[i][j] = k$, then process P_i is requesting k more instances of resource type R_j .

The algorithm is as follows :

- (i) Let $Work$ and $Finish$ be vector of length m and n , respectively. Initialize
 - $Work = Available$
 - For $i = 0, 1, \dots, n-1$, if $Allocation[i] = 0$, then $Finish[i] = False$; otherwise, $Finish[i] = True$.

- (ii) Find an i such that both
 - $Finish[i] = False$
 - $Request[i] \leq Work$
if no such i exists, go to step 4.
- (iii) Set $Work = Allocation[i]$ and $Finish[i] = True$
Go to Step 2.
- (iv) If $Finish[i] = False$ for some $i, 0 \leq i < n$, then the system is in a deadlocked state. Moreover, if $Finish[i] = False$, then the process P_i is deadlocked.

This algorithm requires an order of $m \times n^2$ operations to detect whether the system is in deadlocked state.

3. Deadlock-Detection Algorithm Usage

When should the deadlock detection be done? Frequently, or infrequently? The answer may depend on how frequently deadlocks are expected to occur, as well as the possible consequences of not catching them immediately (If deadlocks are not removed immediately when they occur, then more and more processes can "back up" behind the deadlock, making the eventual task of unblocking the system more difficult and possibly damaging to more processes).

There are two obvious approaches, each with trade-offs:

1. Do deadlock detection after every resource allocation which cannot be immediately granted. This has the advantage of detecting the deadlock right away, while the minimum number of processes are involved in the deadlock (One might consider that the process

whose request triggered the deadlock condition is the "cause" of the deadlock, but realistically all of the processes in the cycle are equally responsible for the resulting deadlock.) The down side of this approach is the extensive overhead and performance hit caused by checking for deadlocks so frequently.

2. Do deadlock detection only when there is some clue that a deadlock may have occurred, such as when CPU utilization reduces to 40% or some other magic number. The advantage is that deadlock detection is done much less frequently, but the down side is that it becomes impossible to detect the processes involved in the original deadlock, and so deadlock recovery can be more complicated and damaging to more processes.

Q45. Discuss various deadlock recovery techniques.

Ans :

Recovery From Deadlock

There are three basic approaches to recovery from deadlock:

1. Inform the system operator, and allow him/her to take manual intervention.
2. Terminate one or more processes involved in the deadlock (Process Termination).
3. Preempt resources.

Process Termination

Two basic approaches, both of which recover resources allocated to terminated processes:

1. Terminate all processes involved in the deadlock. This definitely solves the deadlock, but at the expense of terminating more processes than would be absolutely necessary.
2. Terminate processes one by one until the deadlock is broken. This is more conservative, but requires doing deadlock detection after each step.

In the latter case there are many factors that can go into deciding which processes to terminate next:

- Process priorities.
- How long the process has been running, and how close it is to finishing.
- How many and what type of resources is the process holding (Are they easy to preempt and restore?).
- How many more resources does the process need to complete.
- How many processes will need to be terminated
- Whether the process is interactive or batch.
- Whether or not the process has made non-restorable changes to any resource.

Resource Preemption

When preempting resources to relieve deadlock, there are three important issues to be addressed:

1. Selecting a victim

Deciding which resources to preempt from which processes involves many of the same decision criteria outlined above.

2. Rollback

Ideally one would like to roll back a preempted process to a safe state prior to the point at which that resource was originally allocated to the process. Unfortunately it can be difficult or impossible to determine what such a safe state is, and so the only safe rollback is to roll back all the way back to the beginning(i.e. abort the process and make it start over).

3. Starvation

How do you guarantee that a process won't starve because its resources are constantly being preempted? One option would be to use a priority system, and increase the priority of a process every time its resources get preempted. Eventually it should get a high enough priority that it won't get preempted any more.

Rahul Publications

UNIT II

Memory management strategies with example architectures:

Swapping, Contiguous allocation, Paging, Segmentation, Segmentation with paging. **Virtual memory management:** Demand paging, Page replacement. Thrashing.

2.1 MEMORY MANAGEMENT STRATEGIES WITH EXAMPLE ARCHITECTURES

2.1.1 Swapping

Q1. What is Memory management? Explain briefly.

Ans :

(Imp.)

There are two types of memories first is the logical memory and second is the physical memory the memory which is temporary such as ram is also known as the temporary memory and the memory which is permanent such as hard disk is also known as the physical memory of system.

When we want to execute any programs then that programs must be brought from the physical memory into the logical memory. So that we use the concept of the memory management. this is the responsibility of the operating system to provide the memory spaces to each and every program. Also manage which process will be executed at that time.

Operating system translates the physical address into the logical address, if he wants to perform the operation, then he must translate the physical address into the logical address. This is also known as binding. Means when a physical address is mapped or converted into the logical address, and then this is called as the binding.

There is also a concept which is also known as the dynamic loading, in this a program doesn't reside into the memory of the computer and we must have to load that process for processing. So that when a process is loaded only when a request has found, then it is called as the loading of the process.

Q2. What is process address space.?

Ans :

Process Address Space

The process address space is the set of logical addresses that a process references in its code.

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.

- **Logical address** – generated by the CPU; also referred to as virtual address
- **Physical address** – address seen by the memory unit
- **Virtual Address** – is a binary number in **virtual** memory that enables a process to use a location in primary storage (main memory) independently of other processes and to use more space than actually exists in primary storage by temporarily relegating some contents to a hard disk or internal flash drive.

Logical and physical addresses are the same in compile-time and load time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme

Virtual and physical addresses are the same in compile-time and load-time address-binding schemes. Virtual and physical addresses differ in execution-time address-binding scheme.

The set of all logical addresses generated by a program is referred to as a **logical address space**. The set of all physical addresses

corresponding to these logical addresses is referred to as a **physical address space**.

The runtime mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device. MMU uses following mechanism to convert virtual address to physical address.

- The value in the base register is added to every address generated by a user process, which is treated as offset at the time it is sent to memory. For example, if the base register value is 10000, then an attempt by the user to use address location 100 will be dynamically reallocated to location 10100.
- The user program deals with virtual addresses; it never sees the real physical addresses.

Q3. What is Swapping? Explain.

Ans :

(Imp.)

Swapping is a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.

Though performance is usually affected by swapping process but it helps in running multiple and big processes in parallel and that's the reason Swapping is also known as a technique for memory compaction.

When there is a situation to perform swapping, then we use the sweeper. The sweeper is used for

1. Selecting which process to be out
2. Selecting which process to be in
3. Providing the memory space to the processes those are newly entered.

In this sweeper will select the suspended process then brings the ready process in memory and after the execution that suspended process will be again entered into the memory by the sweeper.

For swapping the sweeper also uses some address those are also known as the logical and physical address of processes. For providing the logical address into the physical address there the following two approaches used.

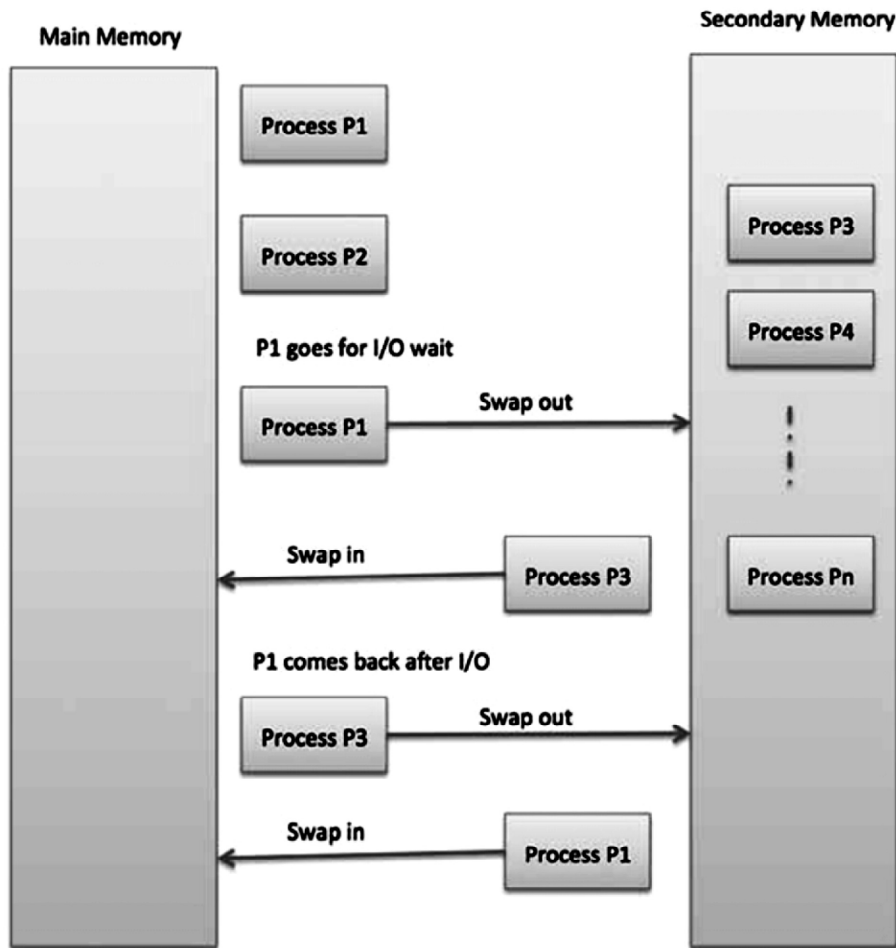
1. Static relocation

In the static relocation swapping never to be performed because the process are always have a memory. Which is not to be changed at the execution time. For example the memory which is provided to the input and output operations and the memory which is provided to the cpu for executing the processes will not be changed so that this is also known as the static relocation of the memory.

2. Dynamic relocation

The dynamic relocation is also known as the memory which is changed and relocates by the processes at the time of execution. So that there must be some mechanism to provide memory those processes those are running. The mapping of the logical address into the physical address will be performed at the time of execution or at run time.

When the memory is provided to the process then the particular address is stored by the cpu which is also have a entry into the partition description table means in which partitions a process is running because when the process needs swapping that particular address must be reloaded.



The total time taken by swapping process includes the time it takes to move the entire process to a secondary disk and then to copy the process back to memory, as well as the time the process takes to regain main memory.

Let us assume that the user process is of size 2048KB and on a standard hard disk where swapping will take place has a data transfer rate around 1 MB per second. The actual transfer of the 1000K process to or from memory will take

$$2048\text{KB} / 1024\text{KB per second}$$

$$= 2 \text{ seconds}$$

$$= 2000 \text{ milliseconds}$$

Now considering in and out time, it will take complete 4000 milliseconds plus other overhead where the process competes to regain main memory.

2.1.2 Contiguous Allocation

Q4. Describe briefly about contiguous allocation.

Ans :

The contiguous allocation means the memory spaces is divided into the small and equal size and the different process uses the various partitions for running their applications process. And when a request has

found then the process will allocate the space. And in this the contiguous spaces is provided to each and every process. Means all the process will reside in the memory of the computer and when a process will request for the memory then this available memory or free memory will be allotted to him.

But there will be problem when the memory which is required by the process is not enough for him or when the size of memory is less which is required for the process. So this problem is also known as the internal fragmentation. The main reason for the internal fragmentation is, all the memory is divided into the fixed and continuous sizes. So that if a process requires large memory then that process will not be fit into the small area.

The second problem is also occurred in the continues allocation. This is also known as the external fragmentation. In this when the memory is not enough after combing the multiple parts of single memory. In this when the required memory is high after combining the various areas of memory then this is called as external fragmentation. So there are the following problems arise when we use the contiguous memory allocation.

1. Wasted memory

The wasted memory is that memory which is unused and which can't be given to the process. When the various process comes which require memory which is not available this is called as the wasted memory.

2. Time complexity

There is also wastage of time for allocating and de-allocating the memory spaces to the process.

3. Memory access

There must be some operations those are performed for providing the memory to the processes.

Q5. How memory is allocated to the process? Explain how to resolve dynamic storage allocation problem?

Ans : (Imp.)

Memory Allocation and Dynamic Storage-Allocation Problem:

Main memory usually has two partitions

- **Low Memory** - Operating system resides in this memory.
- **High Memory** - User processes are held in high memory.

Operating system uses the following memory allocation mechanism.

One method of allocating contiguous memory is to divide all available memory into equal sized partitions, and to assign each process to their own partition. This restricts both the number of simultaneous processes and the maximum size of each process, and is no longer used.

➤ Single-partition allocation

In this type of allocation, relocation-register scheme is used to protect user processes from each other, and from changing operating-system code and data. Relocation register contains value of smallest physical address whereas limit register contains range of logical addresses. Each logical address must be less than the limit register.

➤ Multiple-partition allocation

In this type of allocation, main memory is divided into a number of fixed-sized partitions where each partition should contain only one process. When a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process.

An alternate approach is to keep a list of unused (free) memory blocks (holes), and to find a hole of a suitable size whenever a process needs to be loaded into memory. There are many different strategies for finding the "best" allocation of memory to processes, including the three most commonly discussed:

1. First fit

Search the list of holes until one is found that is big enough to satisfy the request, and assign a portion of that hole to that process. Whatever fraction of the hole not needed by the request is left on the free list as a smaller hole. Subsequent requests may start looking either from the beginning of the list or from the point at which this search ended.

2. Best fit

Allocate the smallest hole that is big enough to satisfy the request. This saves large holes for other process requests that may need them later, but the resulting unused portions of holes may be too small to be of any use, and will therefore be wasted. Keeping the free list sorted can speed up the process of finding the right hole.

3. Worst fit

Allocate the largest hole available, thereby increasing the likelihood that the remaining portion will be usable for satisfying future requests.

Q6. Write a briefly note on fragmentation.

Ans :

As processes are loaded and removed from memory, the free memory space is broken into little pieces. It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused. This problem is known as Fragmentation.

Fragmentation is of two types :

➤ **External fragmentation**

Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.

➤ **Internal fragmentation**

Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.

The following diagram shows how fragmentation can cause waste of memory and a compaction technique can be used to create more free memory out of fragmented memory

Fragmented memory before compaction



Memory after compaction



External fragmentation can be reduced by compaction or shuffle memory contents to place all free memory together in one large block. To make compaction feasible, relocation should be dynamic.

The internal fragmentation can be reduced by effectively assigning the smallest partition but large enough for the process.

2.1.3 Paging Segmentation

Q7. Explain the mechanism of paging segmentation.

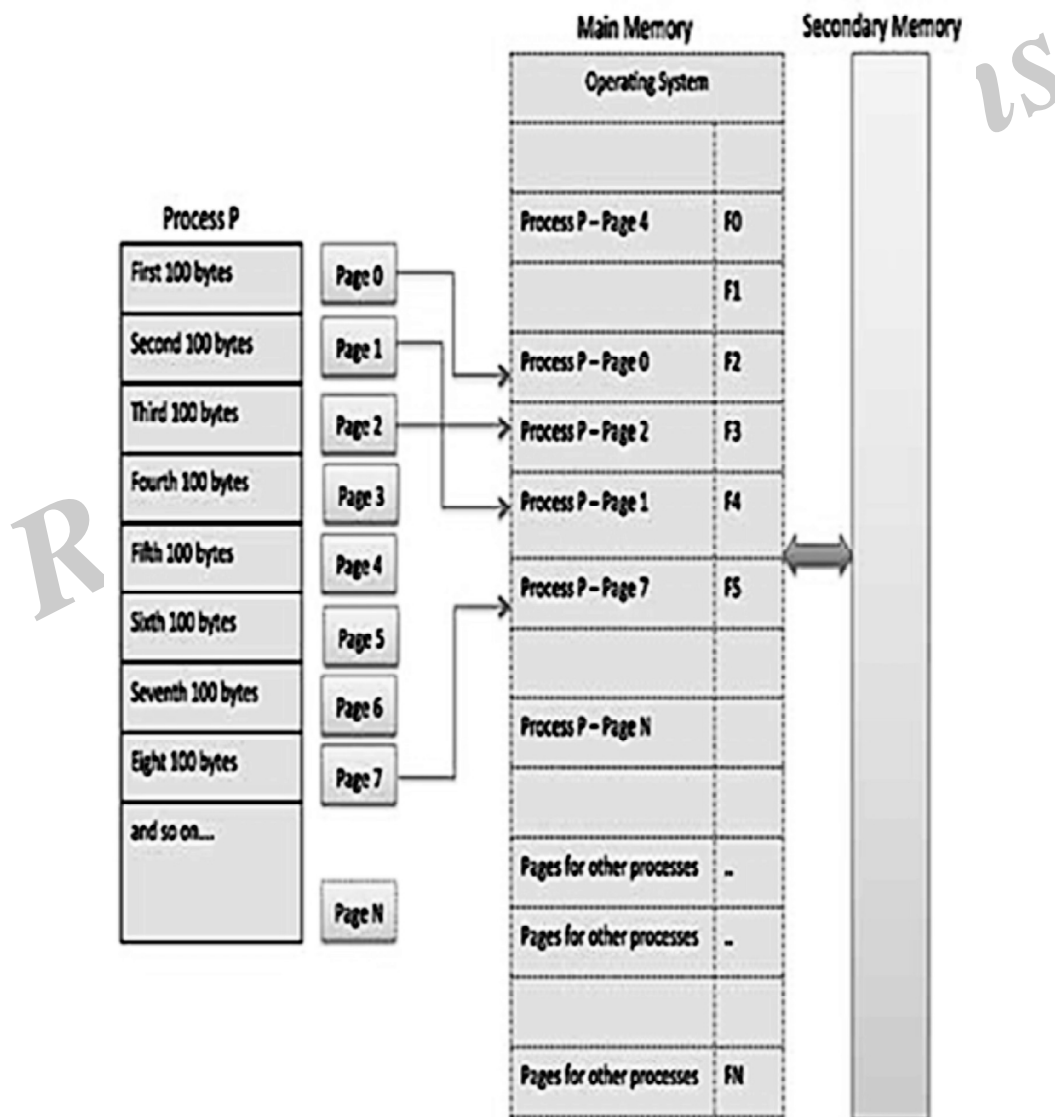
Ans. :

(Imp.)

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 that's set up to emulate the computer's RAM. Paging technique plays an important role in implementing virtual memory.

Paging is a memory management technique in which process address space is broken into blocks of the same size called pages (size is power of 2, between 512 bytes and 8192 bytes). The size of the process is measured in the number of pages.

Similarly, main memory is divided into small fixed-sized blocks of (physical) memory called frames and the size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to void external fragmentation.



Address Translation

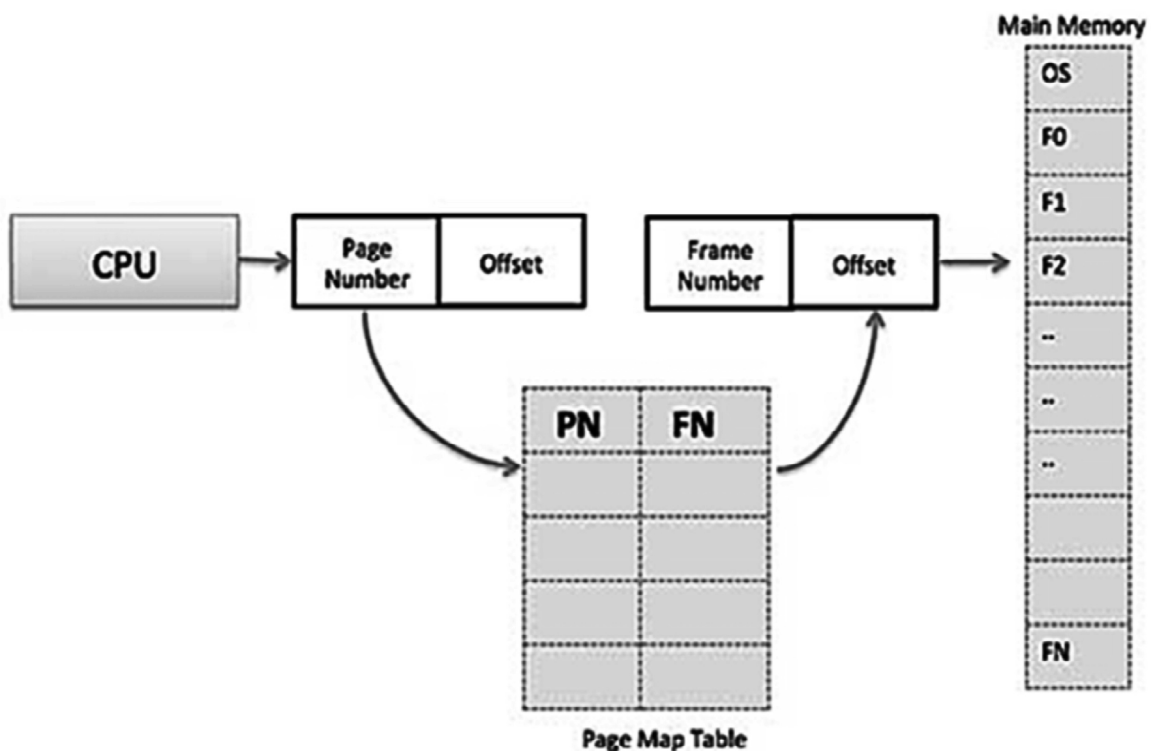
Page address is called logical address and represented by page number and the offset.

Logical Address = Page Number + Page Offset

Frame address is called physical address and represented by a frame number and the offset.

Physical Address = Frame Number + Page Offset

A data structure called page map table is used to keep track of the relation between a page of a process to a frame in physical memory.



When the system allocates a frame to any page, it translates this logical address into a physical address and create entry into the page table to be used throughout execution of the program.

When a process is to be executed, its corresponding pages are loaded into any available memory frames. Suppose you have a program of 8Kb but your memory can accommodate only 5Kb at a given point in time, then the paging concept will come into picture.

When a computer runs out of RAM, the operating system (OS) will move idle or unwanted pages of memory to secondary memory to free up RAM for other processes and brings them back when needed by the program.

This process continues during the whole execution of the program where the OS keeps removing idle pages from the main memory and write them onto the secondary memory and bring them back when required by the program.

Advantages and Disadvantages of Paging

Here is a list of advantages and disadvantages of paging :

- Paging reduces external fragmentation, but still suffer from internal fragmentation.

- Paging is simple to implement and assumed as an efficient memory management technique.
- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not be good for a system having small RAM.

Q8. Explain about the characteristics and process of segmentation.

Ans :

Segmentation

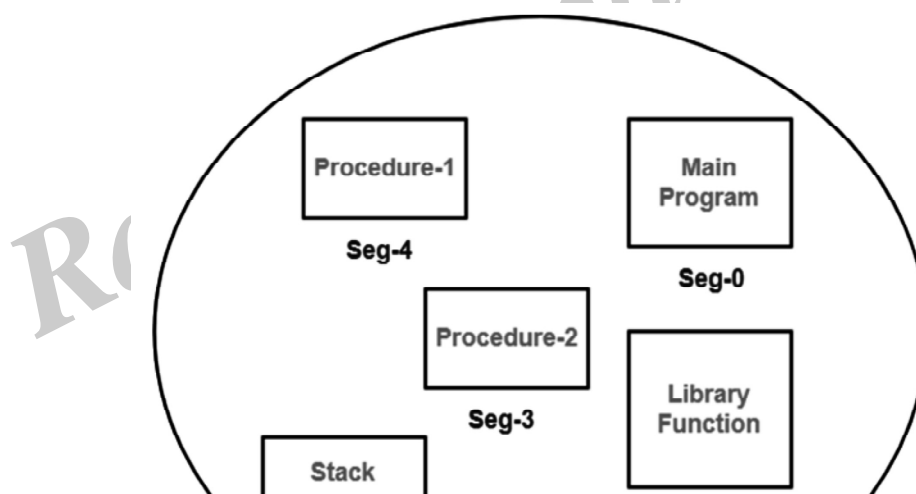
- Like Paging, Segmentation is another non-contiguous memory allocation technique.
- In segmentation, process is not divided blindly into fixed size pages.
- Rather, the process is divided into modules for better visualization.

Characteristics

- Segmentation is a variable size partitioning scheme.
- In segmentation, secondary memory and main memory are divided into partitions of unequal size.
- The size of partitions depend on the length of modules.
- The partitions of secondary memory are called as **segments**.

Example:

Consider a program is divided into 5 segments as



Segment Table

- Segment table is a table that stores the information about each segment of the process.
- It has two columns.
- First column stores the size or length of the segment.
- Second column stores the base address or starting address of the segment in the main memory.
- Segment table is stored as a separate segment in the main memory.
- Segment table base register (STBR) stores the base address of the segment table.

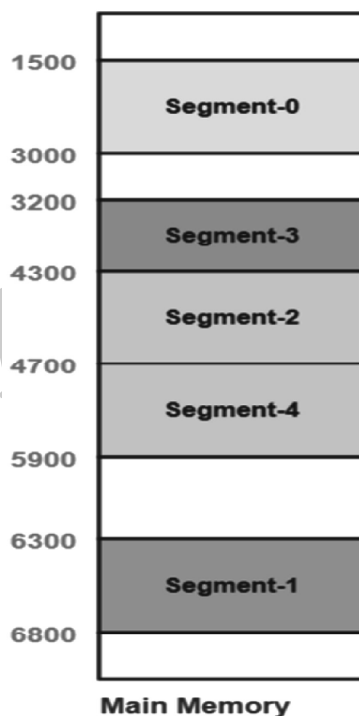
For the above illustration, consider the segment table is -

	Limit	Base
Seg-0	1500	1500
Seg-1	500	6300
Seg-2	400	4300
Seg-3	1100	3200
Seg-4	1200	4700

Segment Table

Here,

- Limit indicates the length or size of the segment.
 - Base indicates the base address or starting address of the segment in the main memory.
- In accordance to the above segment table, the segments are stored in the main memory as -



Translating Logical Address into Physical Address -

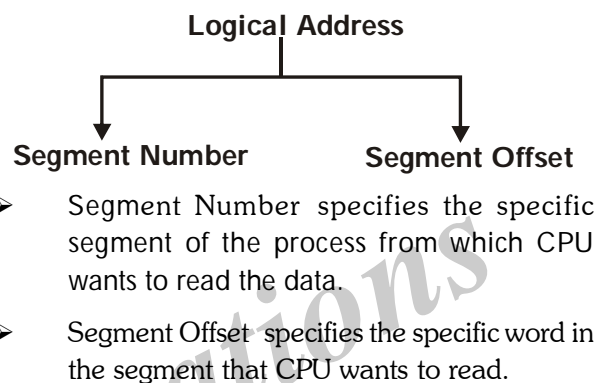
- CPU always generates a logical address.
- A physical address is needed to access the main memory.

Following steps are followed to translate logical address into physical address

Step-01:

CPU generates a logical address consisting of two parts -

1. Segment Number
2. Segment Offset



Step-02:

- For the generated segment number, corresponding entry is located in the segment table.
- Then, segment offset is compared with the limit (size) of the segment.

Now, two cases are possible-

Case-01: Segment Offset \geq Limit

- If segment offset is found to be greater than or equal to the limit, a trap is generated.

Case-02: Segment Offset $<$ Limit

If segment offset is found to be smaller than the limit, then request is treated as a valid request.

- The segment offset must always lie in the range $[0, \text{limit}-1]$,
- Then, segment offset is added with the base address of the segment.
- The result obtained after addition is the address of the memory location storing the required word.

The following diagram illustrates the above steps of translating logical address into physical address

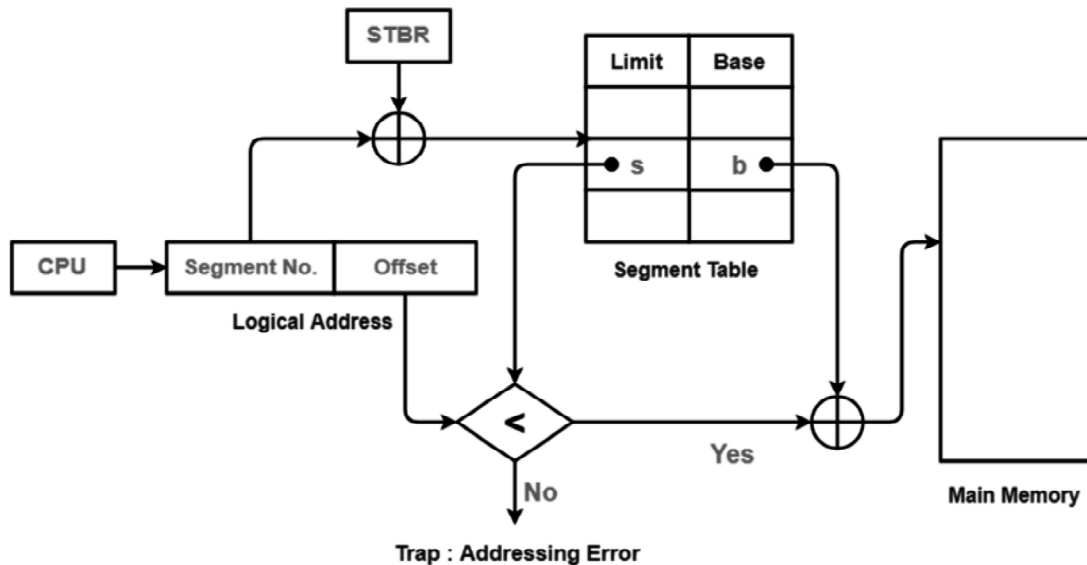


Fig. : Translating Logical Address into Physical Address

Advantages

The advantages of segmentation are :

- It allows to divide the program into modules which provides better visualization.
- Segment table consumes less space as compared to **Page Table** in paging.
- It solves the problem of internal fragmentation.

Disadvantages

The disadvantages of segmentation are :

- There is an overhead of maintaining a segment table for each process.
- The time taken to fetch the instruction increases since now two memory accesses are required.
- Segments of unequal size are not suited for swapping.
- It suffers from external fragmentation as the free space gets broken down into smaller pieces with the processes being loaded and removed from the main memory.

2.1.4 Segmentation With Paging

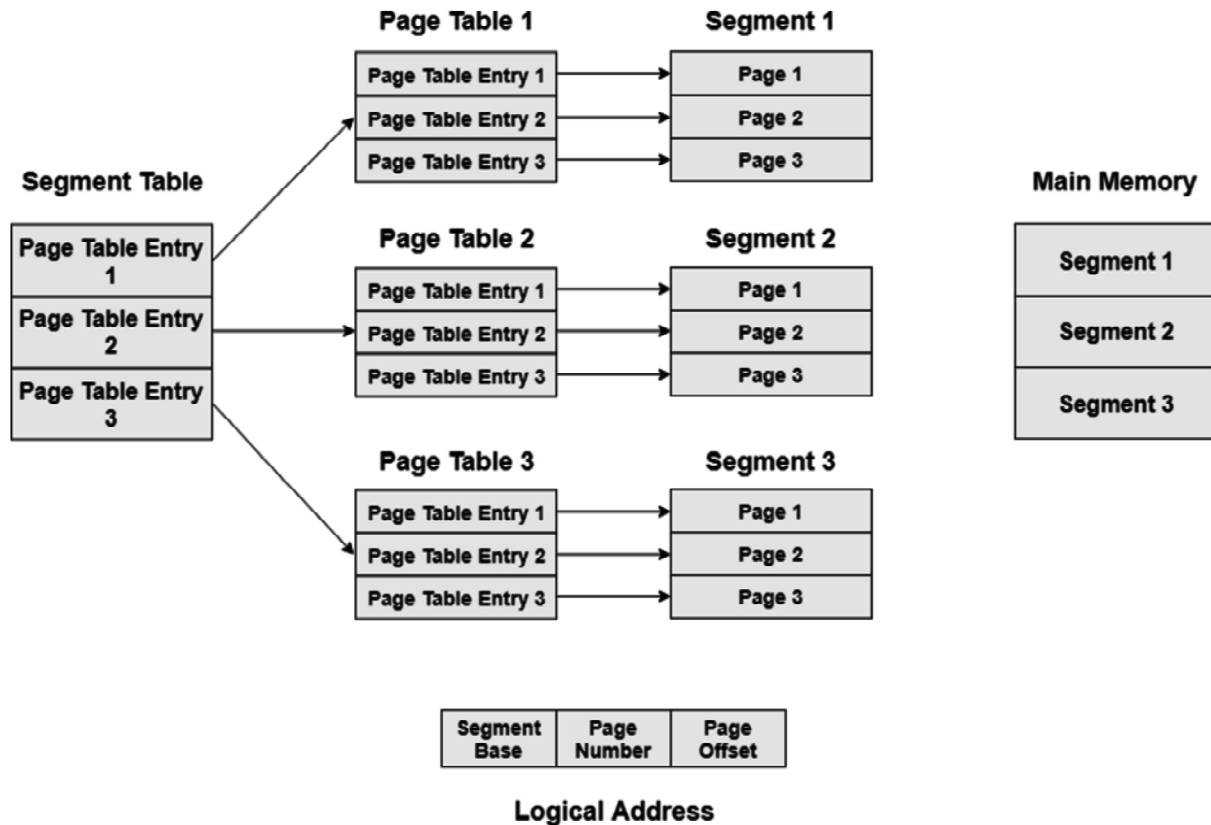
Q9. Explain about Segmented Paging.

Ans.:

(Imp.)

- Process is first divided into segments and then each segment is divided into pages.
- These pages are then stored in the frames of main memory.
- A page table exists for each segment that keeps track of the frames storing the pages of that segment.
- Each page table occupies one frame in the main memory.
- Number of entries in the page table of a segment = Number of pages that segment is divided.

- A segment table exists that keeps track of the frames storing the page tables of segments.
- Number of entries in the segment table of a process = Number of segments that process is divided.
- The base address of the segment table is stored in the segment table base register.
- Each Page table contains the various information about every page of the segment. The Segment Table contains the information about every segment. Each segment table entry points to a page table entry and every page table entry is mapped to one of the page within a segment.



Translation of logical address to physical address

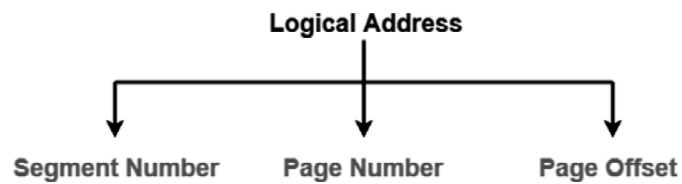
The CPU generates a logical address which is divided into two parts: Segment Number and Segment Offset. The Segment Offset must be less than the segment limit. Offset is further divided into Page number and Page Offset. To map the exact page number in the page table, the page number is added into the page table base.

Following steps are followed to translate logical address into physical address

Step-01:

CPU generates a logical address consisting of three parts :

1. Segment Number
2. Page Number
3. Page Offset



- Segment Number specifies the specific segment from which CPU wants to read the data.
- Page Number specifies the specific page of that segment from which CPU wants to read the data.
- Page Offset specifies the specific word on that page that CPU wants to read.

Step-02:

- For the generated segment number, corresponding entry is located in the segment table.
- Segment table provides the frame number of the frame storing the page table of the referred segment.
- The frame containing the page table is located.

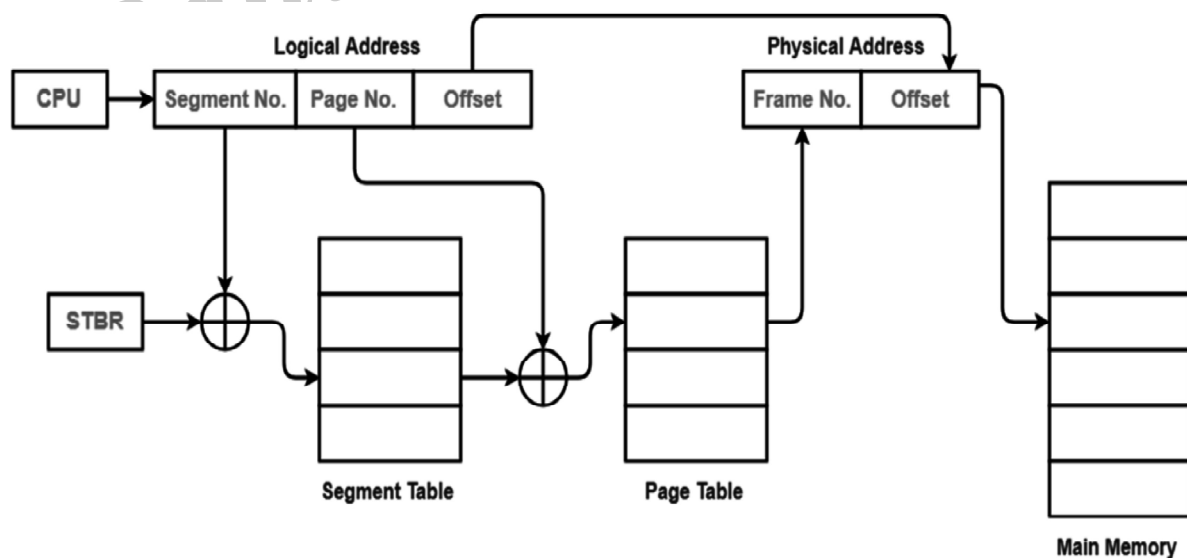
Step-03:

- For the generated page number, corresponding entry is located in the page table.
- Page table provides the frame number of the frame storing the required page of the referred segment.
- The frame containing the required page is located.

Step-04:

- The frame number combined with the page offset forms the required physical address.
- For the generated page offset, corresponding word is located in the page and read.

The following diagram illustrates the above steps of translating logical address into physical address



Translating Logical Address into Physical Address

Advantages

The advantages of segmented paging are

- Segment table contains only one entry corresponding to each segment.
- It reduces memory usage.
- The size of Page Table is limited by the segment size.
- It solves the problem of external fragmentation.

Disadvantages

The disadvantages of segmented paging are

- Segmented paging suffers from internal fragmentation.
- The complexity level is much higher as compared to paging.

2.2 VIRTUAL MEMORY MANAGEMENT**2.2.1 Demand Paging****Q10. What is Demand Paging?**

Ans :

Demand Paging is the Concept in which a Process is Copied into the Logical Memory from the Physical Memory when we need them. A Process can load either Entire, Copied into the Main Memory or the part of single Process is copied into the Memory so that is only the single Part of the Process is copied into the Memory then this is also called as the Lazy Swapping.

For Swapping the Process from the Main Memory or from the Physical Memory, a Page Table must be used. The Page Table is used for Storing the Entries which Contains the Page or Process Number and also the offset Number which indicates the address of the Process where a Process is Stored and there will also be the Special or Extra Bit which is also Known as the Flag Bit which indicates whether the Page is Stored into the Physical Memory.

The Page Table Contains two Entries those are used as valid and invalid means whether the Process is Stored into the Page Table. Or Whether the Demand Program is Stored into the Physical Memory So that they can be easily swapped. If the

Requested Program is not stored into the Page Table then the Page Table must Contains the Entries as v and I means valid and invalid along the Page Number.

When a user Request for any Operation then the Operating System perform the following instructions

1. First of all this will fetch all the instructions from the Physical Memory into the Logical Memory.
2. Decode all the instructions means this will find out which Operation has to be performed on the instructions.
3. Perform Requested Operation.
4. Stores the Result into the Logical Memory and if needed the Results will be Stored into the Physical Memory.

2.2.2 Page Replacement**Q11. What is the use of page replacement algorithm? Explain about various page replacement algorithms.**

Ans :

(Imp.)

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 time of the algorithm itself. There are many different page replacement algorithms.

Reference String

The string of memory references is called reference string. Reference strings are generated artificially or by tracing a given system and recording the address of each memory reference. For a given page size, we need to consider only the page number, not the entire address.

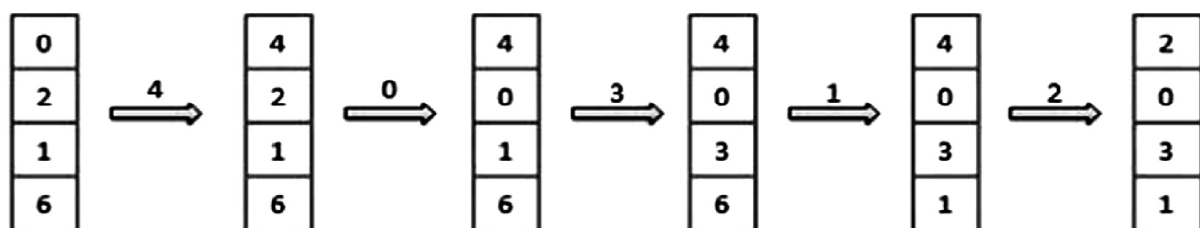
- If we have a reference to a page p , then any immediately following references to page p will never cause a page fault. Page p will be in memory after the first reference; the immediately following references will not fault.
- For example, consider the following sequence of addresses " 23,215,600,1234,76,96
- If page size is 100, then the reference string is 1,2,6,12,0,0

1. First In First Out (FIFO) algorithm

- Oldest page in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages from the tail and add new pages at the head.

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

Misses : x x x x x x x x x



Fault Rate = 9 / 12 = 0.75

Drawback

- FIFO page replacement algorithm =s performance is not always good.
- To illustrate this, consider the following example:

Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

If No.of available frames = 3 then the no.of page faults =9

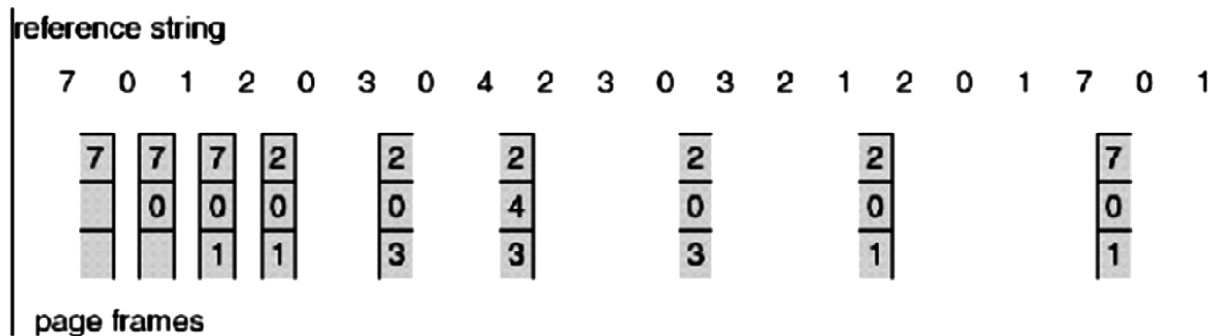
If No.of available frames =4 then the no.of page faults =10

Here the no. of page faults increases when the no.of frames increases .This is called as Belady's Anomaly.

2. Optimal page replacement algorithm

Replace the page that will not be used for the longest period of time.

Example:



Reference string: 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1

No. of available frames = 3

Drawback

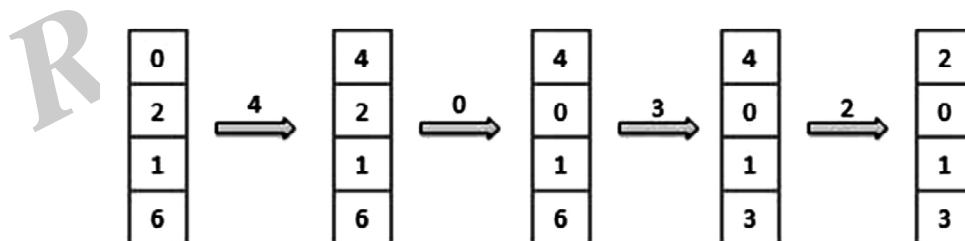
- It is difficult to implement as it requires future knowledge of the reference string.

3. 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 = $8 / 12 = 0.67$

LRU page replacement can be implemented using

(i) Counters

- Every page table entry has a time-of-use field and a clock or counter is associated with the CPU.
- The counter or clock is incremented for every memory reference.
- Each time a page is referenced, copy the counter into the time-of-use field.
- When a page needs to be replaced, replace the page with the smallest counter value.

(ii) Stack

Keep a stack of page numbers. Whenever a page is referenced, remove the page from the stack and put it on top of the stack.

When a page needs to be replaced, replace the page that is at the bottom of the stack. (LRU page).

Use of A Stack to Record The Most Recent Page References

4. 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 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 free pool.

5. LRU Approximation Page Replacement

With each page associate a reference bit, initially set to 0. When page is referenced, the bit is set to 1

When a page needs to be replaced, replace the page whose reference bit is 0. The order of use is not known, but we know which pages were used and which were not used.

(i) Additional Reference Bits Algorithm

Keep an 8-bit byte for each page in a table in memory. At regular intervals, a timer interrupt transfers control to OS. The OS shifts reference bit for each page into higher-order bit shifting the other bits right 1 bit and discarding the lower-order bit.

Example:

If reference bit is 00000000 then the page has not been used for 8 time periods.

If reference bit is 11111111 then the page has been used atleast once each time period.

If the reference bit of page 1 is 11000100 and page 2 is 01110111 then page 2 is the LRU page.

(ii) Second Chance Algorithm

Basic algorithm is FIFO

When a page has been selected, check its reference bit.

If 0 proceed to replace the page

If 1 give the page a second chance and move on to the next FIFO page.

When a page gets a second chance, its reference bit is cleared and arrival time is reset to current time.

Hence a second chance page will not be replaced until all other pages are replaced.

(iii) Enhanced Second Chance Algorithm

Consider both reference bit and modify bit

There are four possible classes

(iv) 1.(0,0) – neither recently used nor modified Best page to replace

(0,1) – not recently used but modified page has to be written out before replacement.

(1,0) – recently used but not modified page may be used again

(1,1) – recently used and modified page may be used again and page has to be written to disk.

6. Counting-Based Page Replacement

Keep a counter of the number of references that have been made to each page

Least Frequently Used (LFU) Algorithm: replaces page with smallest count

Most Frequently Used (MFU) Algorithm: replaces page with largest count

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

Q12. Consider a main memory with five page frames and the following sequence of page references: 3, 8, 2, 3, 9, 1, 6, 3, 8, 9, 3, 6, 2, 1, 3. which one of the following is true with respect to page replacement policies First-In-First-out (FIFO) and Least Recently Used (LRU)?

- A. Both incur the same number of page faults
- B. FIFO incurs 2 more page faults than LRU
- C. LRU incurs 2 more page faults than FIFO
- D. FIFO incurs 1 more page faults than LRU

Ans :

Number of frames = 5

FIFO

According to FIFO, the page which first comes in the memory will first goes out.

Request	3	8	2	3	9	1	6	3	8	9	3	6	2	1	3
Frame 5						1	1	1	1	1	1	1	1	1	1
Frame 4					9	9	9	9	9	9	9	9	2	2	2
Frame 3			2	2	2	2	2	2	8	8	8	8	8	8	8
Frame 2		8	8	8	8	8	8	3	3	3	3	3	3	3	3
Frame 1	3	3	3	3	3	3	6	6	6	6	6	6	6	6	6
Miss/Hit	Miss	Miss	Miss	Hit	Miss	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Hit	Hit

Number of Page Faults = 9

Number of hits = 6

LRU

According to LRU, the page which has not been requested for a long time will get replaced with the new one.

Request	3	8	2	3	9	1	6	3	8	9	3	6	2	1	3
Frame 5						1	1	1	1	1	1	1	2	2	2
Frame 4					9	9	9	9	9	9	9	9	9	9	9
Frame 3			2	2	2	2	2	2	8	8	8	8	8	1	1
Frame 2		8	8	8	8	8	6	6	6	6	6	6	6	6	6
Frame 1	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
Miss/Hit	Miss	Miss	Miss	Hit	Miss	Miss	Hit	Hit	Miss	Hit	Miss	Hit	Miss	Miss	Hit

Number of Page Faults = 9

Number of Hits = 6

The Number of page faults in both the cases is equal therefore the Answer is option (A).

Q13. Explain Belady's Anomaly with an example.

Ans :

(Imp.)

In the case of LRU and optimal page replacement algorithms, it is seen that the number of page faults will be reduced if we increase the number of frames. However, Balady found that, In FIFO page replacement algorithm, the number of page faults will get increased with the increment in number of frames.

This is the strange behavior shown by FIFO algorithm in some of the cases. This is an Anomaly called as Belady's Anomaly.

Let's examine such example :

The reference String is given as 0 1 5 3 0 1 4 0 1 5 3 4. Let's analyze the behavior of FIFO algorithm in two cases.

Case 1: Number of frames = 3

Request	0	1	5	3	0	1	4	0	1	5	3	4
Frame3			5	5	5	1	1	1	1	1	3	3
Frame2		1	1	1	0	0	0	0	0	5	5	5
Frame1	0	0	0	3	3	3	4	4	4	4	4	4
Miss/Hit	Miss	Miss	Miss	Miss	Miss	Miss	Miss	Hit	Hit	Miss	Miss	Hit

Number of Page Faults = 9

Case 2: Number of frames = 4

Request	0	1	5	3	0	1	4	0	1	5	3	4
Frame4				3	3	3	3	3	3	5	5	5
Frame3			5	5	5	5	5	5	1	1	1	1
Frame2		1	1	1	1	1	1	0	0	0	0	4
Frame1	0	0	0	0	0	0	4	4	4	4	3	3
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Miss	Miss	Miss	Miss	Miss	Miss

Number of Page Faults = 10

Therefore, in this example, the number of page faults is increasing by increasing the number of frames hence this suffers from Belady's Anomaly.

2.2.3 Thrashing

Q14. Define thrashing. Explain the techniques of thrashing.

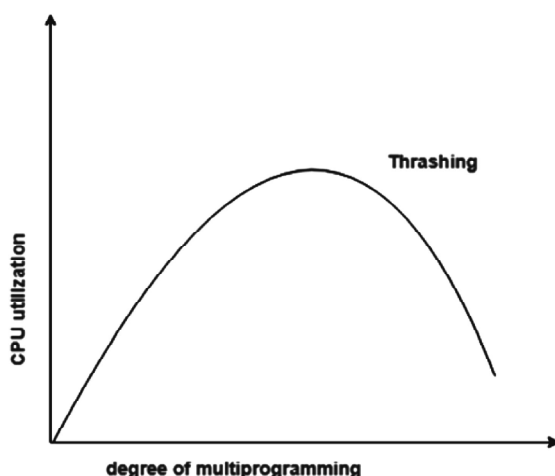
Ans :

(Imp.)

When a program needs space larger than RAM or it needs space when RAM is full, the Operating System will try to allocate space from secondary memory and behaves like it has that much amount of memory by serving to that program. This concept is called virtual memory. To know about thrashing we first need to know what is page fault and swapping.

Page fault and swapping: We know every program is divided into some pages. When a program needs a page which is not in RAM, that is called a page fault. Whenever a page fault happens, the operating system will try to fetch that page from secondary memory and try to swap it with one of the pages in RAM. This is called swapping.

If this page fault and then swapping happen very frequently at a higher rate, then the operating system has to spend more time to swap these pages. This state is called thrashing. Because of this, CPU utilization is going to be reduced.



Effect of Thrashing

Whenever thrashing starts, the operating system tries to apply either Global page replacement Algorithm or Local page replacement algorithm.

Global Page Replacement

Since global page replacement can access to bring any page, it tries to bring more pages whenever thrashing is found. But what actually will happen is, due to this, no process gets enough frames and by result thrashing will increase more and more. So global page replacement algorithm is not suitable when thrashing happens.

Local Page Replacement

Unlike global page replacement algorithm, local page replacement will select pages which only belong to that process. So there is a chance to reduce the thrashing. But it is proven that there are many disadvantages if we use local page replacement. So local page replacement is just an alternative than global page replacement in a thrashing scenario.

Techniques to Handle Thrashing

Working Set Model

This model is based on the above-stated concept of the Locality Model.

The basic principle states that if we allocate enough frames to a process to accommodate its current locality, it will only fault whenever it moves to some new locality. But if the allocated frames are lesser than the size of the current locality, the process is bound to thrash.

According to this model, based on a parameter A , the working set is defined as the set of pages in the most recent ' A ' page references. Hence, all the actively used pages would always end up being a part of the working set.

The accuracy of the working set is dependant on the value of parameter A . If A is too large, then working sets may overlap. On the other hand, for smaller values of A , the locality might not be covered entirely.

If D is the total demand for frames and W_i is the working set size for a process i ,

Now, if 'm' is the number of frames available in the memory, there are 2 possibilities:

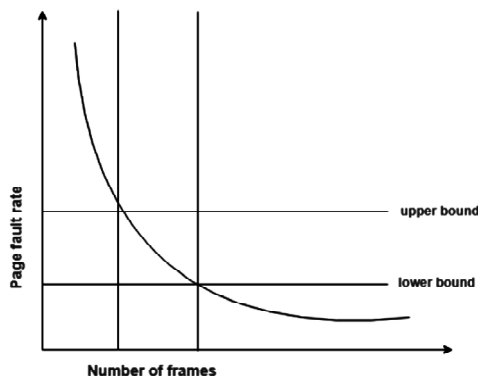
- (i) $D > m$ i.e. total demand exceeds the number of frames, then thrashing will occur as some processes would not get enough frames.
- (ii) $D \leq m$, then there would be no thrashing.

If there are enough extra frames, then some more processes can be loaded in the memory. On the other hand, if the summation of working set sizes exceeds the availability of frames, then some of the processes have to be suspended (swapped out of memory).

1. This technique prevents thrashing along with ensuring the highest degree of multi programming possible. Thus, it optimizes CPU utilisation.

2. Page Fault Frequency

A more direct approach to handle thrashing is the one that uses Page-Fault Frequency concept.



The problem associated with Thrashing is the high page fault rate and thus, the concept here is to control the page fault rate.

If the page fault rate is too high, it indicates that the process has too few frames allocated to it. On the contrary, a low page fault rate indicates that the process has too many frames.

Upper and lower limits can be established on the desired page fault rate as shown in the diagram.

If the page fault rate falls below the lower limit, frames can be removed from the process. Similarly, if the page fault rate exceeds the upper limit, more number of frames can be allocated to the process.

In other words, the graphical state of the system should be kept limited to the rectangular region formed in the given diagram.

Here too, if the page fault rate is high with no free frames, then some of the processes can be suspended and frames allocated to them can be reallocated to other processes. The suspended processes can then be restarted later.

UNIT III

File system interface: File concepts, Access methods and protection.

File system implementation: File system structure, Allocation methods.
Directory implementation of file systems, Mass storage structures, I/O systems

3.1 FILE SYSTEM INTERFACE

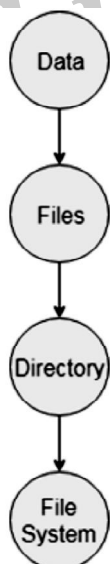
3.1.1 File Concepts

Q1. What is file? Explain about the structure of the file and its attributes.

Ans : (Imp.)

A file can be defined as a data structure which stores the sequence of records. Files are stored in a file system, which may exist on a disk or in the main memory. Files can be simple (plain text) or complex (specially-formatted).

The collection of files is known as Directory. The collection of directories at the different levels, is known as File System.



File Structure

A file has various kinds of structure. Some of them can be :

➤ **Simple Record Structure** with lines of fixed or variable lengths.

➤ **Complex Structures** like formatted document or reloadable load files.

➤ **No Definite Structure** like sequence of words and bytes etc.

Attributes of a File

Following are some of the attributes of a file :

1. Name

Every file carries a name by which the file is recognized in the file system. One directory cannot have two files with the same name.

2. Identifier

Along with the name, Each File has its own extension which identifies the type of the file. For example, a text file has the extension .txt, A video file can have the extension .mp4.

3. Type

In a File System, the Files are classified in different types such as video files, audio files, text files, executable files, etc.

4. Location

In the File System, there are several locations on which, the files can be stored. Each file carries its location as its attribute.

5. Size

The Size of the File is one of its most important attribute. By size of the file, we mean the number of bytes acquired by the file in the memory.

6. Protection

The Admin of the computer may want the different protections for the different files. Therefore each file carries its own set of permissions to the different group of Users.

7. Time and Date

Every file carries a time stamp which contains the time and date on which the file is last modified.

Q2. Write about Different Types of Files.

Ans :

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 program.
- 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 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 device 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.

Operations on the File

There are various operations which can be implemented on a file. We will see all of them in detail.

1. Create

Creation of the file is the most important operation on the file. Different types of files are created by different methods for example text editors are used to create a text file, word processors are used to create a word file and Image editors are used to create the image files.

2. Write

Writing the file is different from creating the file. The OS maintains a write pointer for every file which points to the position in the file from which, the data needs to be written.

3. Read

Every file is opened in three different modes Read, Write and append. A Read pointer is maintained by the OS, pointing to the position up to which, the data has been read.

4. Re-position

Re-positioning is simply moving the file pointers forward or backward depending upon the user's requirement. It is also called as seeking.

5. Delete

Deleting the file will not only delete all the data stored inside the file, it also deletes all the attributes of the file. The space which is allocated to the file will now become available and can be allocated to the other files.

6. Truncate

Truncating is simply deleting the file except deleting attributes. The file is not completely deleted although the information stored inside the file get replaced.

3.1.2 Access Methods And Protection**Q3. What are the various File Access Mechanisms ?**

Ans :

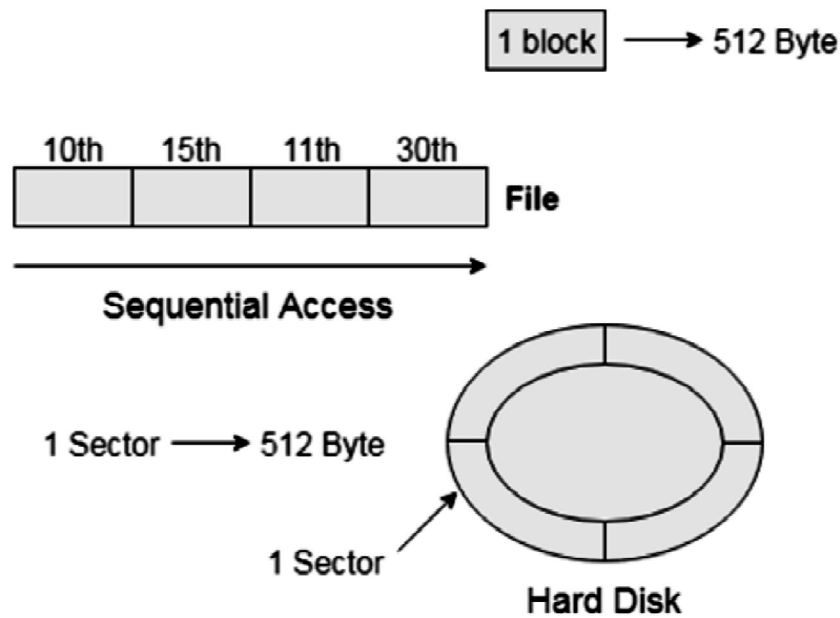
(Imp.)

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 -

1. Sequential access
2. Direct/Random access
3. Indexed sequential access

1. Sequential Access



Most of the operating systems access the file sequentially. In other words, we can say that most of the files need to be accessed sequentially by the operating system.

In sequential access, the OS reads the file word by word. A pointer is maintained which initially points to the base address of the file. If the user wants to read the first word of the file, then the pointer provides that word to the user and increases its value by 1 word. This process continues till the end of the file.

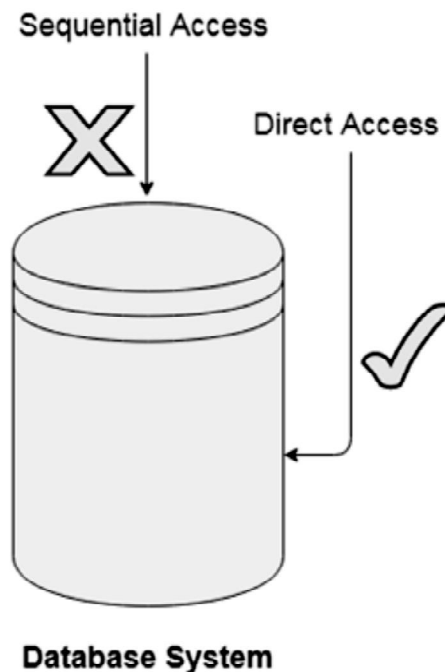
Modern word systems do provide the concept of direct access and indexed access but the most used method is sequential access due to the fact that most of the files such as text files, audio files, video files, etc. need to be sequentially accessed.

2. Direct Random Access

The Direct Access is mostly required in the case of database systems. In most of the cases, we need filtered information from the database. The sequential access can be very slow and inefficient in such cases.

Suppose every block of the storage stores 4 records and we know that the record we needed is stored in the 10th block. In that case, the sequential access will not be implemented because it will traverse all the blocks in order to access the needed record.

Direct access will give the required result despite of the fact that the operating system has to perform some complex tasks such as determining the desired block number. However, that is generally implemented in database applications.



iii) Indexed Sequential Access

If a file can be sorted on any of the filed then an index can be assigned to a group of certain records. However, A particular record can be accessed by its index. The index is nothing but the address of a record in the file.

In index accessing, searching in a large database became very quick and easy but we need to have some extra space in the memory to store the index value.

Q4. What is file protection? Explain various types of file access control techniques

Ans : (Imp.)

Protection

- Files must be kept safe for reliability (against accidental damage), and protection (against deliberate malicious access.) The former is usually managed with backup copies. This section discusses the latter.
- One simple protection scheme is to remove all access to a file. However this makes the file unusable, so some sort of controlled access must be arranged.

Types of Access

- The following low-level operations are often controlled:
 - Read - View the contents of the file
 - Write - Change the contents of the file.
 - Execute - Load the file onto the CPU and follow the instructions contained therein.
 - Append - Add to the end of an existing file.
 - Delete - Remove a file from the system.
 - List -View the name and other attributes of files on the system.

- Higher-level operations, such as copy, can generally be performed through combinations of the above.

Access Control

- One approach is to have complicated **Access Control Lists, ACL**, which specify exactly what access is allowed or denied for specific users or groups.
 - The AFS uses this system for distributed access.
 - Control is very finely adjustable, but may be complicated, particularly when the specific users involved are unknown. (AFS allows some wild cards, so for example all users on a certain remote system may be trusted, or a given username may be trusted when accessing from any remote system.)
- UNIX uses a set of 9 access control bits, in three groups of three. These correspond to R, W, and X permissions for each of the Owner, Group, and Others. The RWX bits control the following privileges for ordinary files and directories:

Bit	Files	Directories
R	Read (view) file contents.	Read directory contents. Required to get a listing of the directory.
W	Write (change) file contents.	Change directory contents. Required to create or delete files.
X	Execute file contents as a program.	Access detailed directory information. Required to get a long listing, or to access any specific file in the directory. Note that if a user has X but not R permissions on a directory, they can still access specific files, but only if they already know the name of the file they are trying to access.

- In addition there are some special bits that can also be applied:
 - The set user ID (SUID) bit and/or the set group ID (SGID) bits applied to executable files temporarily change the identity of whoever runs the program to match that of the owner / group of the executable program.
 - The sticky bit on a directory modifies write permission, allowing users to only delete files for which they are the owner.
 - The SUID, SGID, and sticky bits are indicated with an S, S, and T in the positions for execute permission for the user, group, and others, respectively. If the letter is lower case, (s, s, t), then the corresponding execute permission is not also given. If it is upper case, (S, S, T), then the corresponding execute permission IS given.
 - The numeric form of chmod is needed to set these advanced bits.

```

-rw-rw-r-- 1 pbg staff 31200 Sep 3 08:30 intro.ps
drwx----- 5 pbg staff 512 Jul 8 09:33 private/
drwxrwxr-x 2 pbg staff 512 Jul 8 09:35 doc/
drwxrwx--- 2 jwg student 512 Aug 3 14:13 student-proj/
-rw-r--r-- 1 pbg staff 9423 Feb 24 2012 program.c
-rwxr-xr-x 1 pbg staff 20471 Feb 24 2012 program
drwx--x--x 4 tag faculty 512 Jul 31 10:31 lib/
drwx----- 3 pbg staff 1024 Aug 29 06:52 mail/
drwxrwxrwx 3 pbg staff 512 Jul 8 09:35 test/

```

Sample permissions in a UNIX system.

3.2 FILE SYSTEM IMPLEMENTATION

3.2.1 File System Structure

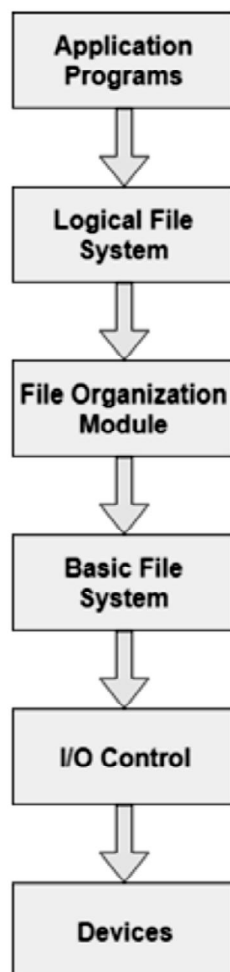
Q5. Explain the concept of file system structure.

Ans : (Imp.)

File System provide efficient access to the disk by allowing data to be stored, located and retrieved in a convenient way. A file System must be able to store the file, locate the file and retrieve the file.

Most of the Operating Systems use layering approach for every task including file systems. Every layer of the file system is responsible for some activities.

The image shown below, elaborates how the file system is divided in different layers, and also the functionality of each layer.



➤ When an application program asks for a file, the first request is directed to the logical file system. The logical file system contains the Meta data of the file and directory structure. If the application program doesn't have the required permissions of the file then this layer will throw an error. Logical file systems also verify the path to the file.

➤ Generally, files are divided into various logical blocks. Files are to be stored in the hard disk and to be retrieved from the hard disk. Hard disk is divided into various tracks and sectors. Therefore, in order to store and retrieve the files, the logical blocks need to be mapped to physical blocks. This mapping is done by File organization module. It is also responsible for free space management.

➤ Once File organization module decided which physical block the application program needs, it passes this information to basic file system. The basic file system is responsible for issuing the commands to I/O control in order to fetch those blocks.

➤ I/O controls contain the codes by using which it can access hard disk. These codes are known as device drivers. I/O controls are also responsible for handling interrupts.

Q6. Explain about file system implementation techniques

Ans :

➤ File systems store several important data structures on the disk

- A **boot-control block**, (per volume) a.k.a. the **boot block** in UNIX or the **partition boot sector** in Windows contains information about how to boot the system off of this disk. This will generally be the first sector of the volume if there is a bootable system loaded on that volume, or the block will be left vacant otherwise.
- A **volume control block**, (per volume) a.k.a. the **master file table** in UNIX or the **superblock** in Windows, which contains information such as the

partition table, number of blocks on each filesystem, and pointers to free blocks and free FCB blocks.

- A directory structure (per file system), containing file names and pointers to corresponding FCBs. UNIX uses inode numbers, and NTFS uses a **master file table**.
 - The **File Control Block, FCB**, (per file) containing details about ownership, size, permissions, dates, etc. UNIX stores this information in inodes, and NTFS in the master file table as a relational database structure.
- There are also several key data structures stored in memory:
- An in-memory mount table.
 - An in-memory directory cache of recently accessed directory information.
 - **A system-wide open file table**, containing a copy of the FCB for every currently open file in the system, as well as some other related information.
 - **A per-process open file table**, containing a pointer to the system open file table as well as some other information.
 - When a new file is created, a new FCB is allocated and filled out with important information regarding the new file. The appropriate directory is modified with the new file name and FCB information.
 - When a file is accessed during a program, the `open()` system call reads in the FCB information from disk, and stores it in the system-wide open file table. An entry is added to the per-process open file table referencing the system-wide table, and an index into the per-process table is returned by the `open()` system call. UNIX refers to this index as a **file descriptor**, and Windows refers to it as a **file handle**.
 - If another process already has a file open when a new request comes in for

the same file, and it is sharable, then a counter in the system-wide table is incremented and the per-process table is adjusted to point to the existing entry in the system-wide table.

- When a file is closed, the per-process table entry is freed, and the counter in the system-wide table is decremented. If that counter reaches zero, then the system wide table is also freed. Any data currently stored in memory cache for this file is written out to disk if necessary.

Partitions and Mounting

- Physical disks are commonly divided into smaller units called partitions. They can also be combined into larger units, but that is most commonly done for RAID installations and is left for later chapters.
- Partitions can either be used as raw devices or they can be formatted to hold a file system . Raw partitions are generally used for swap space, and may also be used for certain programs such as databases that choose to manage their own disk storage system.
- The boot block is accessed as part of a raw partition, by the boot program prior to any operating system being loaded. Modern boot programs understand multiple OSes and filesystem formats, and can give the user a choice of which of several available systems to boot.
- The **root partition** contains the OS kernel and at least the key portions of the OS needed to complete the boot process. At boot time the root partition is mounted, and control is transferred from the boot program to the kernel found there.
- Continuing with the boot process, additional filesystems get mounted, adding their information into the appropriate mount table structure..Filesystems may be mounted either automatically or manually. In UNIX a mount point is indicated by setting a flag in the in-memory copy of the inode, so all future references to that inode get re-directed to the root directory of the mounted filesystem.

Virtual File Systems

- **Virtual File Systems, VFS**, provide a common interface to multiple different filesystem types. In addition, it provides for a unique identifier (vnode) for files across the entire space, including across all filesystems of different types. (UNIXinodes are unique only across a single filesystem, and certainly do not carry across networked file systems.)
- The VFS in Linux is based upon four key object types:
 - The **inode** object, representing an individual file
 - The **file** object, representing an open file.
 - The **superblock** object, representing a filesystem.
 - The **dentry** object, representing a directory entry.
- Linux VFS provides a set of common functionalities for each filesystem, using function pointers accessed through a table.

3.2.2 Allocation Methods

Q7. Explain different file allocation methods

Ans :

There are various methods which can be used to allocate disk space to the files. Selection of an appropriate allocation method will significantly affect the performance and efficiency of the system. Allocation method provides a way in which the disk will be utilized and the files will be accessed.

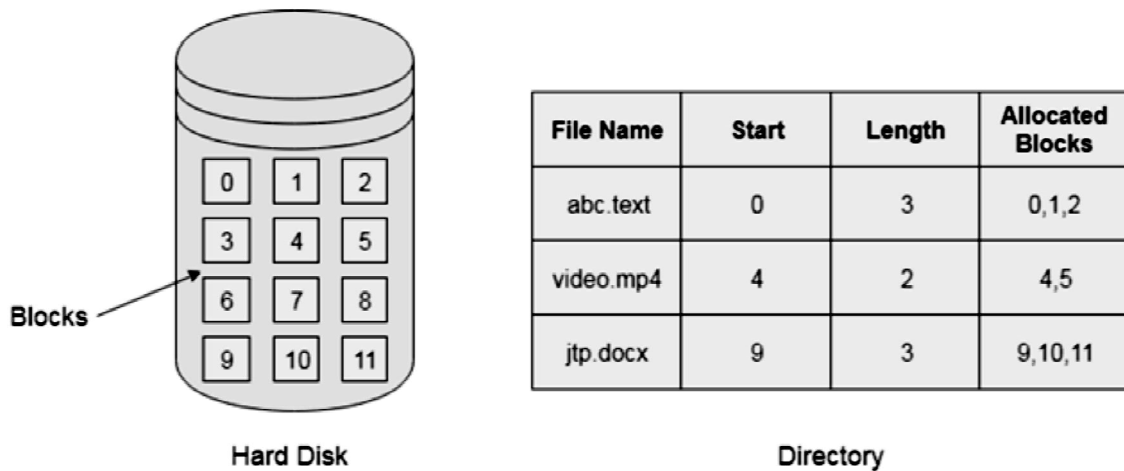
There are following methods which can be used for allocation.

1. Contiguous Allocation.
2. Linked List Allocation
3. FAT
4. Indexed Allocation Scheme
5. Linked Indexed Allocation
6. Multilevel Indexed Allocation
7. Inode

1. Contiguous Allocation

If the blocks are allocated to the file in such a way that all the logical blocks of the file get the contiguous physical block in the hard disk then such allocation scheme is known as contiguous allocation.

In the image shown below, there are three files in the directory. The starting block and the length of each file are mentioned in the table. We can check in the table that the contiguous blocks are assigned to each file as per its need.

**Advantages**

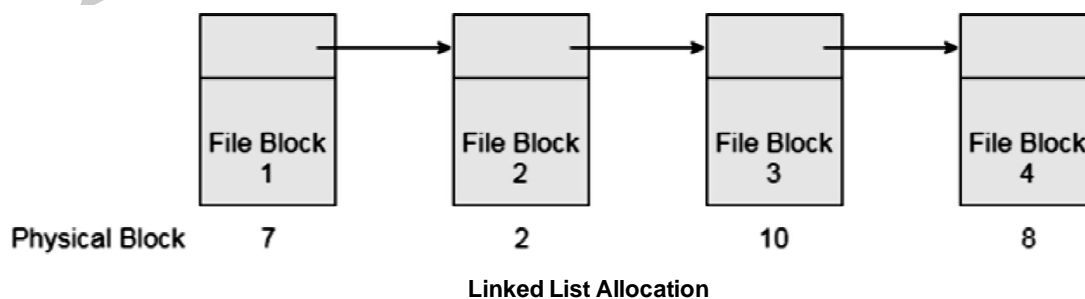
1. It is simple to implement.
2. We will get Excellent read performance.
3. Supports Random Access into files.

Disadvantages

1. The disk will become fragmented.
2. It may be difficult to have a file grow.

2. Linked List Allocation

Linked List allocation solves all problems of contiguous allocation. In linked list allocation, each file is considered as the linked list of disk blocks. However, the disks blocks allocated to a particular file need not to be contiguous on the disk. Each disk block allocated to a file contains a pointer which points to the next disk block allocated to the same file.

**Advantages**

1. There is no external fragmentation with linked allocation.
2. Any free block can be utilized in order to satisfy the file block requests.
3. File can continue to grow as long as the free blocks are available.
4. Directory entry will only contain the starting block address.

Disadvantages

1. Random Access is not provided.
2. Pointers require some space in the disk blocks.
3. Any of the pointers in the linked list must not be broken otherwise the file will get corrupted.
4. Need to traverse each block.

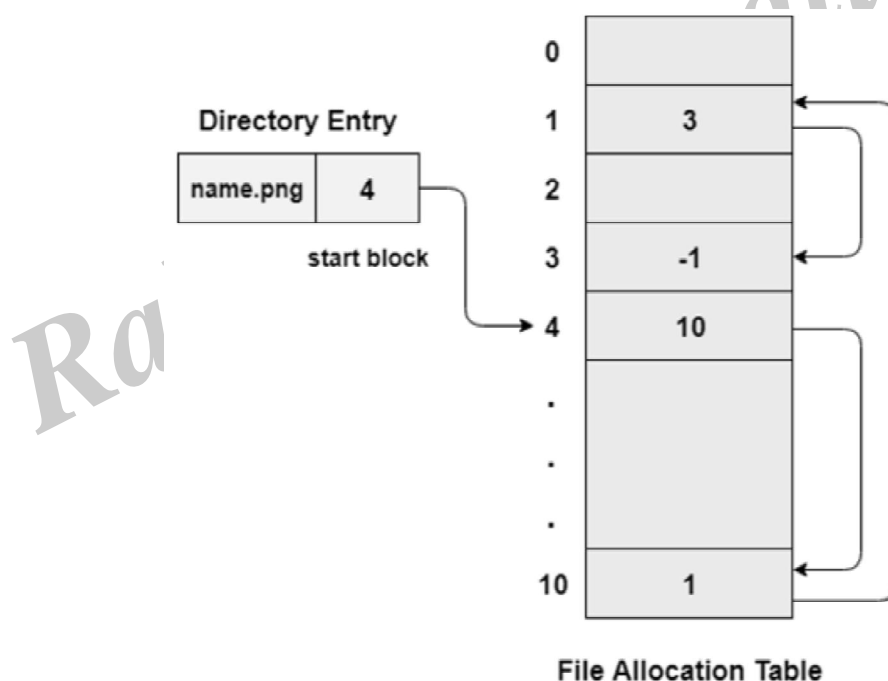
3. File Allocation Table (FAT)

The main disadvantage of linked list allocation is that the Random access to a particular block is not provided. In order to access a block, we need to access all its previous blocks.

File Allocation Table overcomes this drawback of linked list allocation. In this scheme, a file allocation table is maintained, which gathers all the disk block links. The table has one entry for each disk block and is indexed by block number.

File allocation table needs to be cached in order to reduce the number of head seeks. Now the head doesn't need to traverse all the disk blocks in order to access one successive block.

It simply accesses the file allocation table, read the desired block entry from there and access that block. This is the way by which the random access is accomplished by using FAT. It is used by MS-DOS and pre-NT Windows versions.

**Advantages**

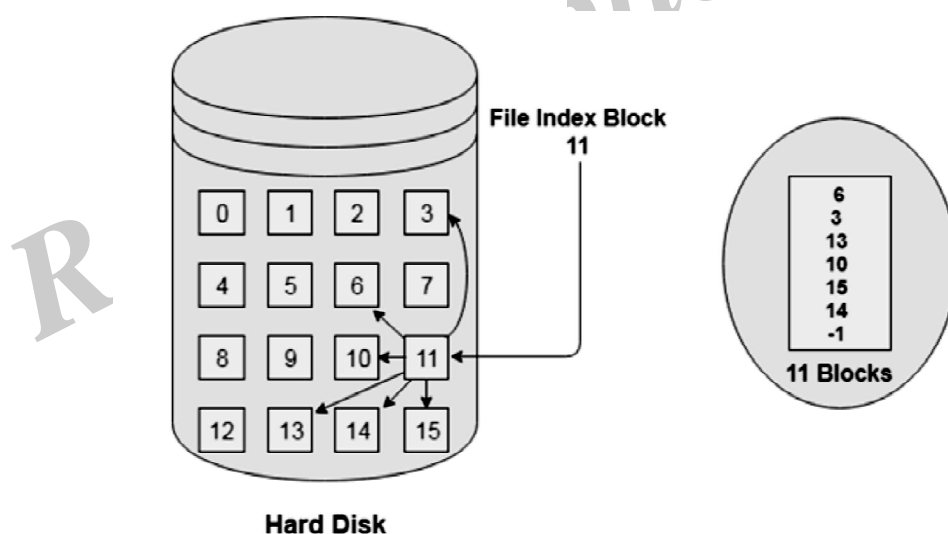
1. Uses the whole disk block for data.
2. A bad disk block doesn't cause all successive blocks lost.
3. Random access is provided although its not too fast.
4. Only FAT needs to be traversed in each file operation.

Disadvantages

1. Each Disk block needs a FAT entry.
2. FAT size may be very big depending upon the number of FAT entries.
3. Number of FAT entries can be reduced by increasing the block size but it will also increase Internal Fragmentation.
4. File allocation table tries to solve as many problems as possible but leads to a drawback. The more the number of blocks, the more will be the size of FAT.
5. Therefore, we need to allocate more space to a file allocation table. Since, file allocation table needs to be cached therefore it is impossible to have as many space in cache. Here we need a new technology which can solve such problems.

4. Indexed Allocation Scheme

Instead of maintaining a file allocation table of all the disk pointers, Indexed allocation scheme stores all the disk pointers in one of the blocks called as indexed block. Indexed block doesn't hold the file data, but it holds the pointers to all the disk blocks allocated to that particular file. Directory entry will only contain the index block address.

**Advantages**

1. Supports direct access
2. A bad data block causes the lost of only that block.

Disadvantages

1. A bad index block could cause the lost of entire file.
2. Size of a file depends upon the number of pointers, a index block can hold.

3. Having an index block for a small file is totally wastage.
4. More pointer overhead

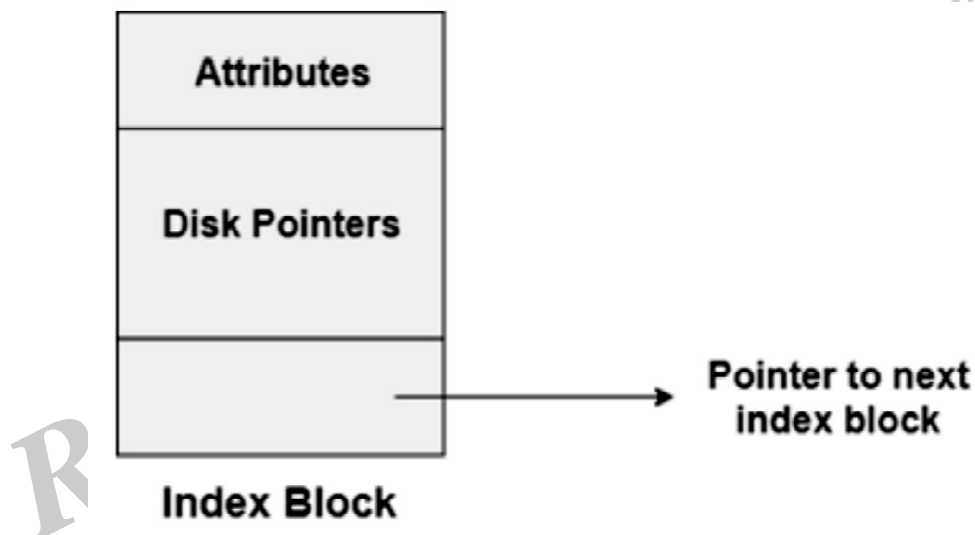
5. Linked Indexed Allocation

- Single level linked Index Allocation

In index allocation, the file size depends on the size of a disk block. To allow large files, we have to link several index blocks together. In linked index allocation,

- Small header giving the name of the file
- Set of the first 100 block addresses
- Pointer to another index block

For the larger files, the last entry of the index block is a pointer which points to another index block. This is also called as linked schema.



Advantage: It removes file size limitations

Disadvantage: Random Access becomes a bit harder

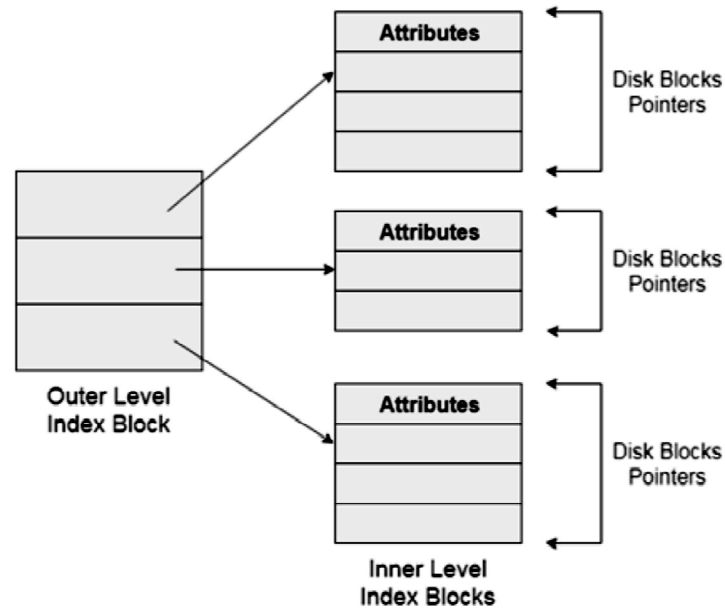
6. Multilevel Index Allocation

In Multilevel index allocation, we have various levels of indices. There are outer level index blocks which contain the pointers to the inner level index blocks and the inner level index blocks contain the pointers to the file data.

- The outer level index is used to find the inner level index.
- The inner level index is used to find the desired data block.

Advantage: Random Access becomes better and efficient.

Disadvantage: Access time for a file will be higher.

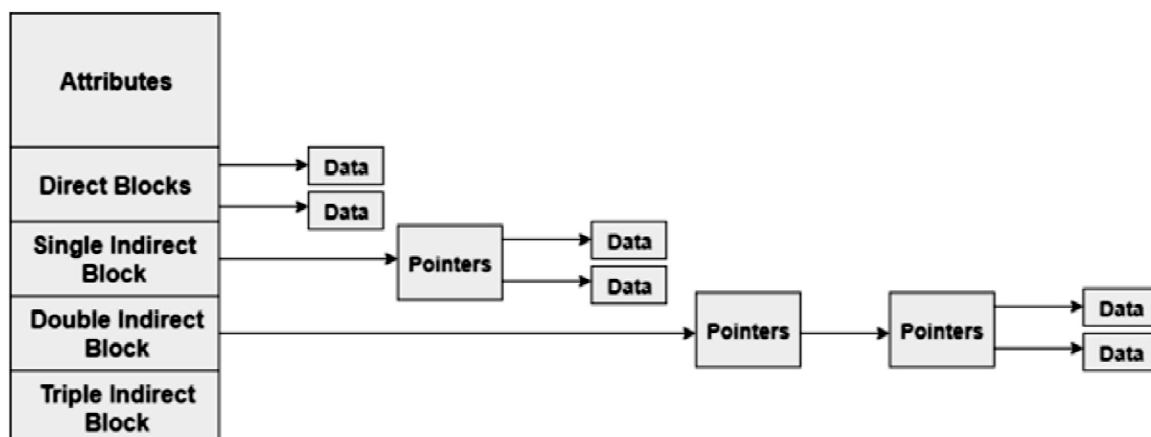


7. Inode

In UNIX based operating systems, each file is indexed by an Inode. Inode are the special disk block which is created with the creation of the file system. The number of files or directories in a file system depends on the number of Inodes in the file system.

An Inode includes the following information

1. Attributes (permissions, time stamp, ownership details, etc) of the file
2. A number of direct blocks which contains the pointers to first 12 blocks of the file.
3. A single indirect pointer which points to an index block. If the file cannot be indexed entirely by the direct blocks then the single indirect pointer is used.
4. A double indirect pointer which points to a disk block that is a collection of the pointers to the disk blocks which are index blocks. Double index pointer is used if the file is too big to be indexed entirely by the direct blocks as well as the single indirect pointer.
5. A triple index pointer that points to a disk block that is a collection of pointers. Each of the pointers is separately pointing to a disk block which also contains a collection of pointers which are separately pointing to an index block that contains the pointers to the file blocks.



3.2.3 Directory Implementation of File Systems

Q8. Write about Directory structure of file system.

Ans :

(Imp.)

Storage Structure

- A disk can be used in its entirety for a file system.
- Alternatively a physical disk can be broken up into multiple **partitions, slices, or mini-disks**, each of which becomes a virtual disk and can have its own file system.
- Or, multiple physical disks can be combined into one **volume**, i.e. a larger virtual disk, with its own file system spanning the physical disks.

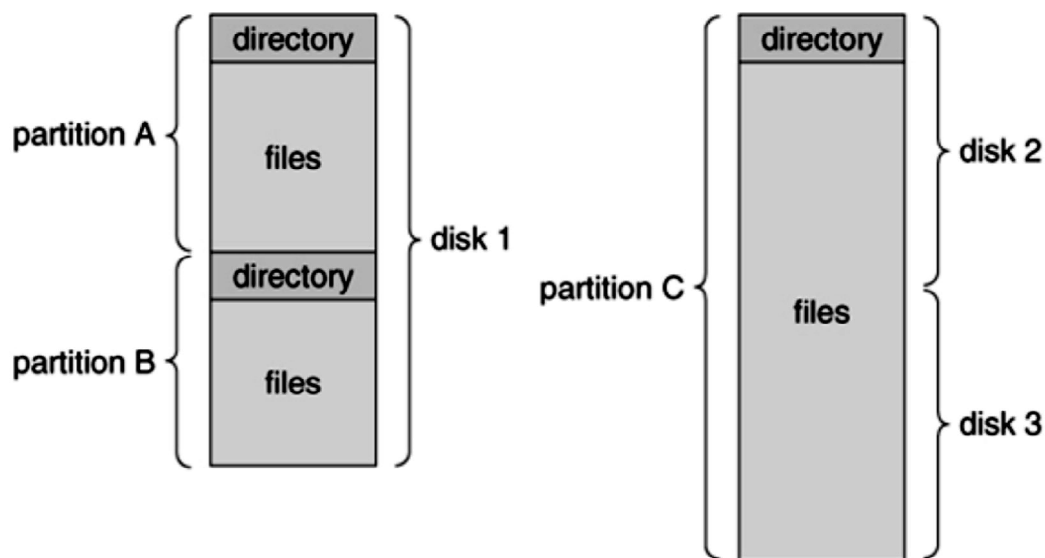


Fig . A typical file-system organization.

Directory Overview

- Directory operations to be supported include:
 - Search for a file
 - Create a file - add to the directory
 - Delete a file - erase from the directory
 - List a directory - possibly ordered in different ways.
 - Rename a file - may change sorting order
 - Traverse the file system.

Single-Level Directory

- Simple to implement, but each file must have a unique name.

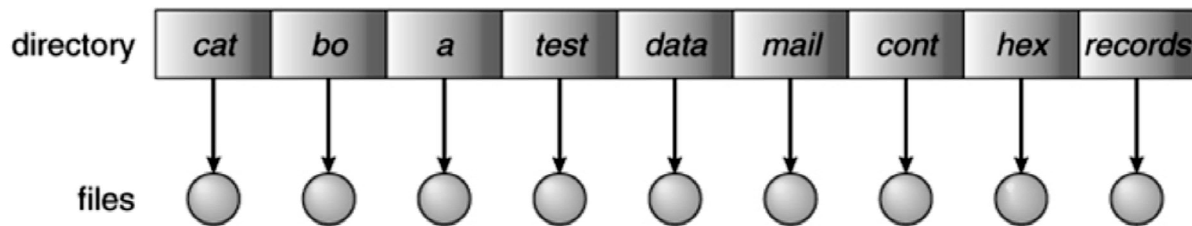


Fig . Single-level directory

Two-Level Directory

- Each user gets their own directory space.
- File names only need to be unique within a given user's directory.
- A master file directory is used to keep track of each users directory, and must be maintained when users are added to or removed from the system.
- A separate directory is generally needed for system (executable) files.
- Systems may or may not allow users to access other directories besides their own
 - If access to other directories is allowed, then provision must be made to specify the directory being accessed.
 - If access is denied, then special consideration must be made for users to run programs located in system directories. A **search path** is the list of directories in which to search for executable programs, and can be set uniquely for each user.

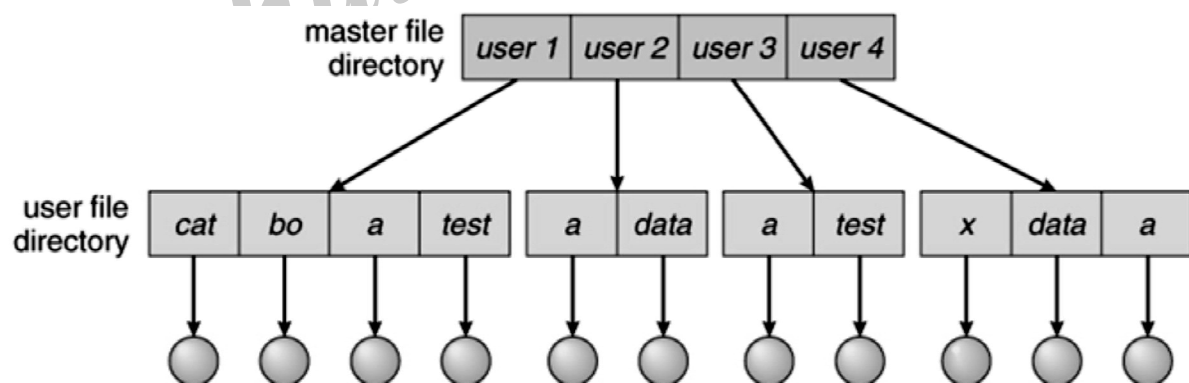


Fig . Two-level directory structure.

Tree-Structured Directories

- An obvious extension to the two-tiered directory structure, and the one with which we are all most familiar.
- Each user / process has the concept of a **current directory** from which all (relative) searches take place.

- Files may be accessed using either absolute pathnames .
- Directories are stored the same as any other file in the system, except there is a bit that identifies them as directories, and they have some special structure that the OS understands.
- One question for consideration is whether or not to allow the removal of directories that are not empty - Windows requires that directories be emptied first, and UNIX provides an option for deleting entire sub-trees.

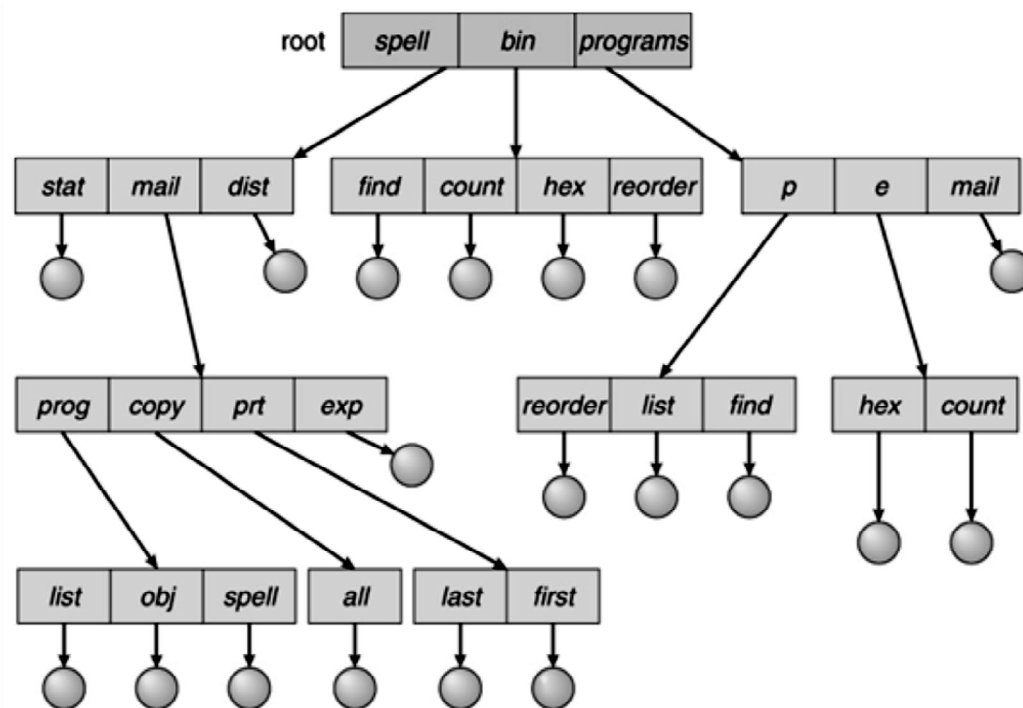


Fig . Tree-structured directory structure.

Acyclic-Graph Directories

- When the same files need to be accessed in more than one place in the directory structure.
- UNIX provides two types of **links** for implementing the acyclic-graph structure.
 - A **hard link** (usually just called a link) involves multiple directory entries that both refer to the same file. Hard links are only valid for ordinary files in the same filesystem.
 - A **symbolic link**, that involves a special file, containing information about where to find the linked file. Symbolic links may be used to link directories and/or files in other filesystems, as well as ordinary files in the current filesystem.
- Windows only supports symbolic links, termed **shortcuts**.
- Hard links require a **reference count**, or **link count** for each file, keeping track of how many directory entries are currently referring to this file. Whenever one of the references is removed the link count is reduced, and when it reaches zero, the disk space can be reclaimed.
- For symbolic links there is some question as to what to do with the symbolic links when the original file is moved or deleted:

- One option is to find all the symbolic links and adjust them also.
- Another is to leave the symbolic links dangling, and discover that they are no longer valid the next time they are used.
- What if the original file is removed, and replaced with another file having the same name before the symbolic link is next used?

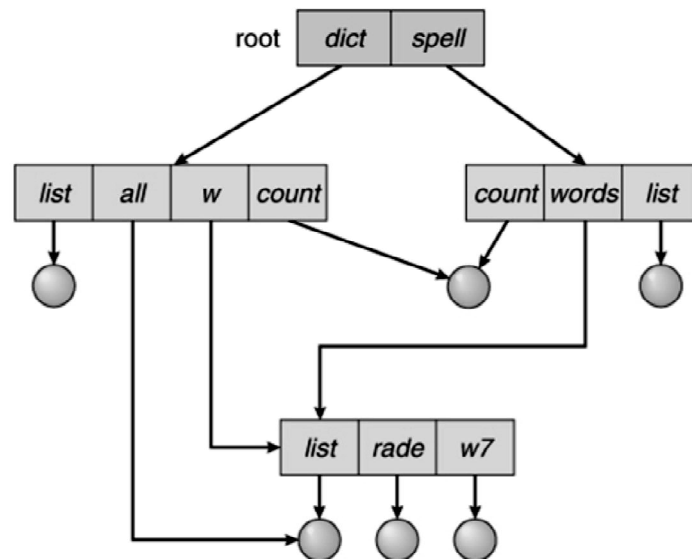


Fig. Acyclic-graph directory structure.

General Graph Directory

- If cycles are allowed in the graphs, then several problems can arise:
- Search algorithms can go into infinite loops. One solution is to not follow links in search algorithms.
 - Sub-trees can become disconnected from the rest of the tree and still not have their reference counts reduced to zero. Periodic garbage collection is required to detect and resolve this problem

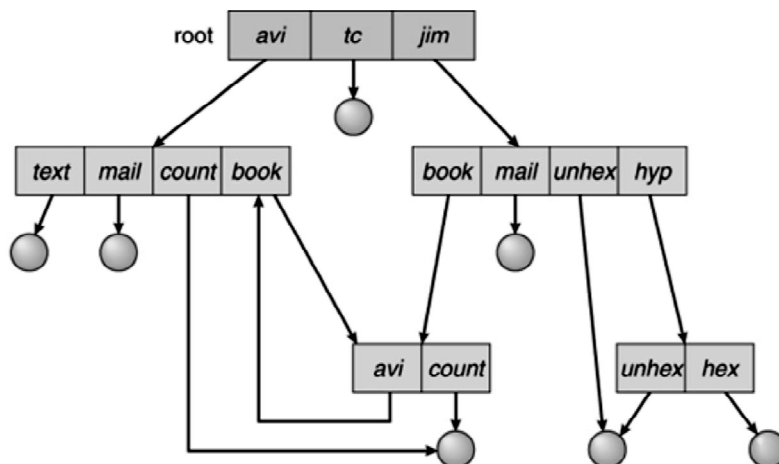


Fig . General graph directory.

Q9. What is File System Mounting ?

Ans .:

- The basic idea behind mounting file systems is to combine multiple file systems into one large tree structure.
- The mount command is given a filesystem to mount and a **mount point** (directory) on which to attach it.
- Once a file system is mounted onto a mount point, any further references to that directory actually refer to the root of the mounted file system.
- Any files (or sub-directories) that had been stored in the mount point directory prior to mounting the new filesystem are now hidden by the mounted filesystem, and are no longer available. For this reason some systems only allow mounting onto empty directories.
- Filesystems can only be mounted by root, unless root has previously configured certain filesystems to be mountable onto certain pre-determined mount points. (E.g. root may allow users to mount floppy filesystems to /mnt or something like it.) Anyone can run the mount command to see what filesystems are currently mounted.
- Filesystems may be mounted read-only, or have other restrictions imposed.

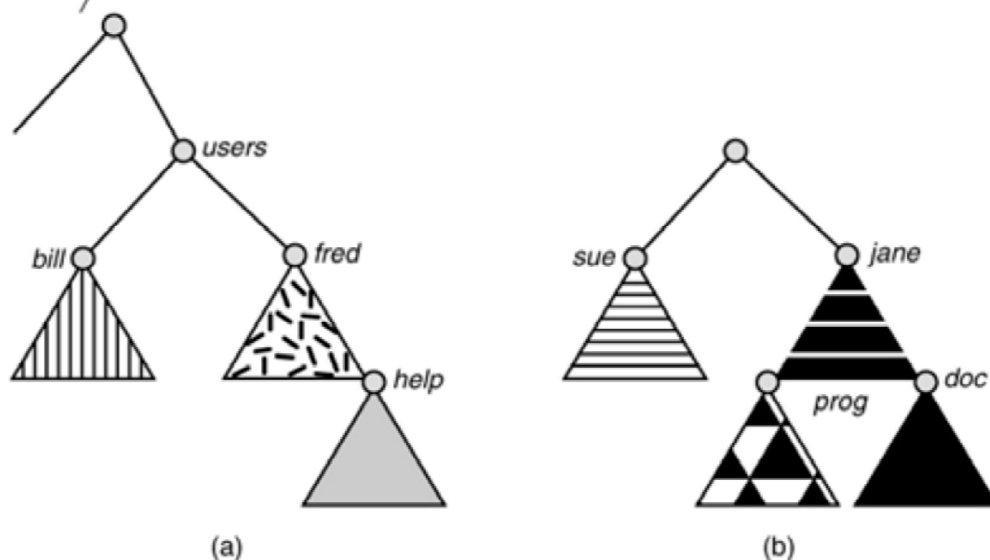


Figure - File system. (a) Existing system. (b) Unmounted volume.

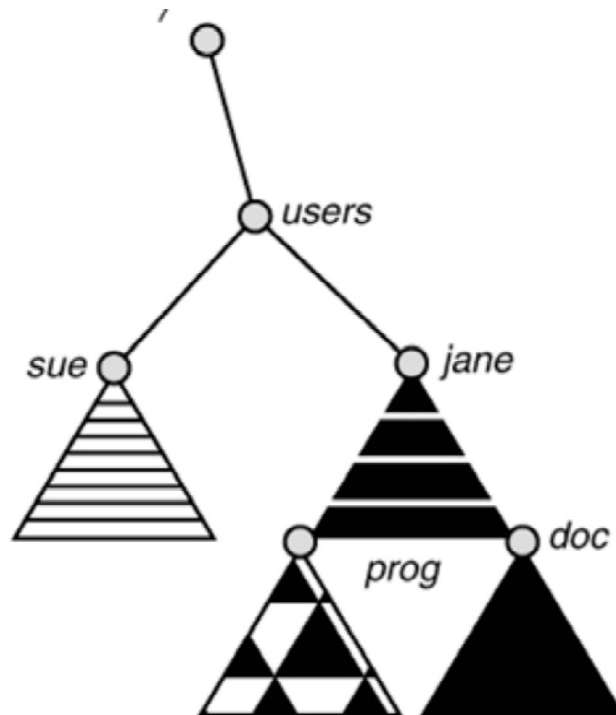


Fig . Mount point

- The traditional Windows OS runs an extended two-tier directory structure, where the first tier of the structure separates volumes by drive letters, and a tree structure is implemented below that level.
- Macintosh runs a similar system, where each new volume that is found is automatically mounted and added to the desktop when it is found.
- More recent Windows systems allow file systems to be mounted to any directory in the file system, much like UNIX.

Q10. Explain about directory implementation methods

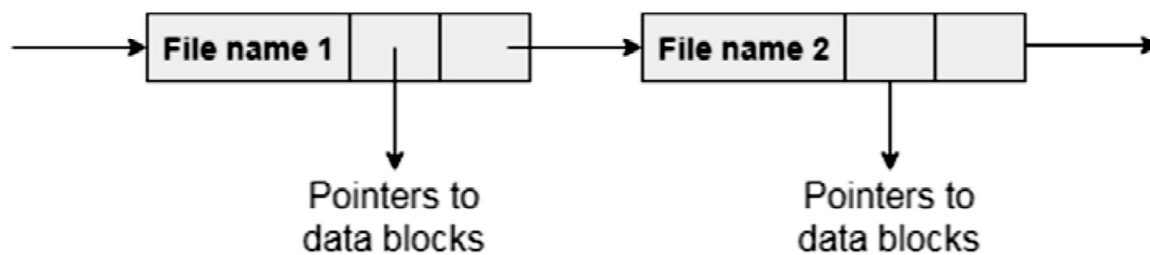
Ans :

Directories need to be fast to search, insert, and delete, with a minimum of wasted disk space.

1. Linear List

Characteristics

- When a new file is created, then the entire list is checked whether the new file name is matching to a existing file name or not. In case, it doesn't exist, the file can be created at the beginning or at the end. Therefore, searching for a unique name is a big concern because traversing the whole list takes time.
- The list needs to be traversed in case of every operation (creation, deletion, updating, etc) on the files therefore the systems become inefficient.



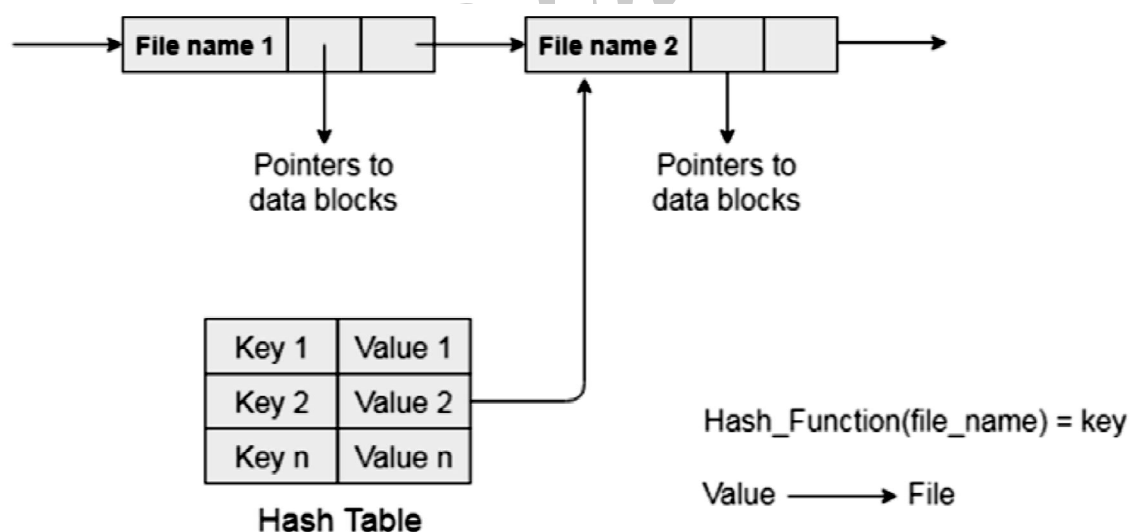
Linear List

2. Hash Table

To overcome the drawbacks of singly linked list implementation of directories, there is an alternative approach that is hash table. This approach suggests to use hash table along with the linked lists.

A key-value pair for each file in the directory gets generated and stored in the hash table. The key can be determined by applying the hash function on the file name while the key points to the corresponding file stored in the directory.

Now, searching becomes efficient due to the fact that now, entire list will not be searched on every operating. Only hash table entries are checked using the key and if an entry found then the corresponding file will be fetched using the value.



3.2.4 Mass Storage Structures

Q11. What are Mass (Secondary) Storage Devices? Explain.

Ans :

(Imp.)

Secondary storage devices are those devices whose memory is non volatile, meaning, the stored data will be intact even if the system is turned off. Here are a few things worth noting about secondary storage.

- Secondary storage is also called auxiliary storage.
- Secondary storage is less expensive when compared to primary memory like RAMs.
- The speed of the secondary storage is also lesser than that of primary storage.
- Hence, the data which is less frequently accessed is kept in the secondary storage.
- A few examples are magnetic disks, magnetic tapes, removable thumb drives etc.

Magnetic Disk Structure

In modern computers, most of the secondary storage is in the form of magnetic disks. Hence, knowing the structure of a magnetic disk is necessary to understand how the data in the disk is accessed by the computer.

Structure of a magnetic disk

A magnetic disk contains several **platters**. Each platter is divided into circular shaped **tracks**. The length of the tracks near the centre is less than the length of the tracks farther from the centre. Each track is further divided into **sectors**, as shown in the figure.

Tracks of the same distance from centre form a cylinder. A read-write head is used to read data from a sector of the magnetic disk.

The speed of the disk is measured as two parts:

- **Transfer rate:** This is the rate at which the data moves from disk to the computer.
- **Random access time:** It is the sum of the seek time and rotational latency.

Seek time is the time taken by the arm to move to the required track. **Rotational latency** is defined as the time taken by the arm to reach the required sector in the track.

Even though the disk is arranged as sectors and tracks physically, the data is logically arranged and addressed as an array of blocks of fixed size. The size of a block can be **512** or **1024** bytes. Each logical block is mapped with a sector on the disk, sequentially. In this way, each sector in the disk will have a logical address.

Solid-State Disks - New

- As technologies improve and economics change, old technologies are often used in different ways. One example of this is the increasing use of **solid state disks, or SSDs**.
- SSDs use memory technology as a small fast hard disk. Specific implementations may use either flash memory or DRAM chips protected by a battery to sustain the information through power cycles.
- Because SSDs have no moving parts they are much faster than traditional hard drives, and certain problems such as the scheduling of disk accesses simply do not apply.
- However SSDs also have their weaknesses: They are more expensive than hard drives, generally not as large, and may have shorter life spans.
- SSDs are especially useful as a high-speed cache of hard-disk information that must be accessed quickly. One example is to store filesystem meta-data, e.g. directory and inode information, that must be accessed quickly and often. Another variation is a boot disk containing the OS and some application executables, but no vital user data. SSDs are also used in laptops to make them smaller, faster, and lighter.
- Because SSDs are so much faster than traditional hard disks, the throughput of the bus can become a limiting factor, causing some SSDs to be connected directly to the system PCI bus for example.

Magnetic Tapes

- Magnetic tapes were once used for common secondary storage before the days of hard disk drives, but today are used primarily for backups.
- Accessing a particular spot on a magnetic tape can be slow, but once reading or writing commences, access speeds are comparable to disk drives.

- Capacities of tape drives can range from 20 to 200 GB, and compression can double that capacity.

Q12. Write about the disk structure

Ans :

- The traditional head-sector-cylinder, HSC numbers are mapped to linear block addresses by numbering the first sector on the first head on the outermost track as sector 0. Numbering proceeds with the rest of the sectors on that same track, and then the rest of the tracks on the same cylinder before proceeding through the rest of the cylinders to the center of the disk. In modern practice these linear block addresses are used in place of the HSC numbers for a variety of reasons:
 1. The linear length of tracks near the outer edge of the disk is much longer than for those tracks located near the center, and therefore it is possible to squeeze many more sectors onto outer tracks than onto inner ones.
 2. All disks have some bad sectors, and therefore disks maintain a few spare sectors that can be used in place of the bad ones. The mapping of spare sectors to bad sectors is managed internally to the disk controller.
 3. Modern hard drives can have thousands of cylinders, and hundreds of sectors per track on their outermost tracks. These numbers exceed the range of HSC numbers for many (older) operating systems, and therefore disks can be configured for any convenient combination of HSC values that falls within the total number of sectors physically on the drive.
- There is a limit to how closely packed individual bits can be placed on a physical media, but that limit is growing increasingly more packed as technological advances are made.

- Modern disks pack many more sectors into outer cylinders than inner ones, using one of two approaches:

- With **Constant Linear Velocity, CLV**, the density of bits is uniform from cylinder to cylinder. Because there are more sectors in outer cylinders, the disk spins slower when reading those cylinders, causing the rate of bits passing under the read-write head to remain constant. This is the approach used by modern CDs and DVDs.
- With **Constant Angular Velocity, CAV**, the disk rotates at a constant angular speed, with the bit density decreasing on outer cylinders. (These disks would have a constant number of sectors per track on all cylinders.)

Q13. Write about Disk Attachment.

Ans :

Disk drives can be attached either directly to a particular host (a local disk) or to a network.

1. Host-Attached Storage

- Local disks are accessed through I/O Ports as described earlier.
- The most common interfaces are IDE or ATA, each of which allow up to two drives per host controller.
- SATA is similar with simpler cabling.
- High end workstations or other systems in need of larger number of disks typically use SCSI disks:
 - ▶ The SCSI standard supports up to 16 **targets** on each SCSI bus, one of which is generally the host adapter and the other 15 of which can be disk or tape drives.
 - ▶ A SCSI target is usually a single drive, but the standard also supports up to 8 **units** within each target. These would generally be used for accessing individual disks within a RAID array.

- ▶ The SCSI standard also supports multiple host adapters in a single computer, i.e. multiple SCSI busses.
- ▶ Modern advancements in SCSI include “fast” and “wide” versions, as well as SCSI-2.
- ▶ SCSI cables may be either 50 or 68 conductors. SCSI devices may be external as well as internal.
- ▶ FC is a high-speed serial architecture that can operate over optical fiber or four-conductor copper wires, and has two variants:
 - ▶ A large switched fabric having a 24-bit address space. This variant allows for multiple devices and multiple hosts to interconnect, forming the basis for the **storage-area networks, SANs**.
 - ▶ The **arbitrated loop, FC-AL**, that can address up to 126 devices.

2. Network-Attached Storage

- Network attached storage connects storage devices to computers using a remote procedure call, RPC, interface, typically with something like NFS file system mounts.
- NAS can be implemented using SCSI cabling, or **iSCSI** uses Internet protocols and standard network connections, allowing long-distance remote access to shared files.
- NAS allows computers to easily share data storage, but tends to be less efficient than standard host-attached storage.

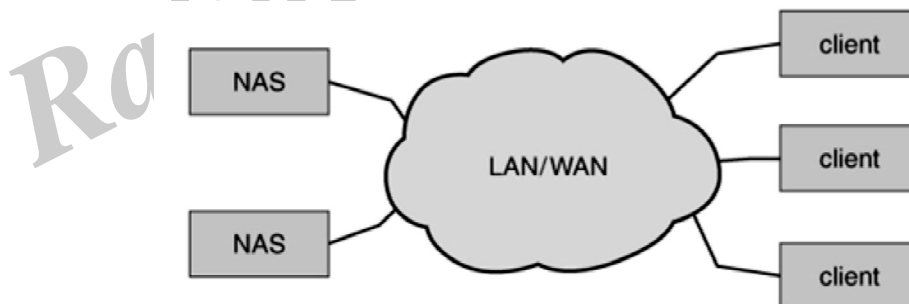


Fig . Network-attached storage

3. Storage-Area Network

- A **Storage-Area Network, SAN**, connects computers and storage devices in a network, using storage protocols instead of network protocols.
- One advantage of this is that storage access does not tie up regular networking bandwidth.
- SAN is very flexible and dynamic, allowing hosts and devices to attach and detach on the fly.
- SAN is also controllable, allowing restricted access to certain hosts and devices.

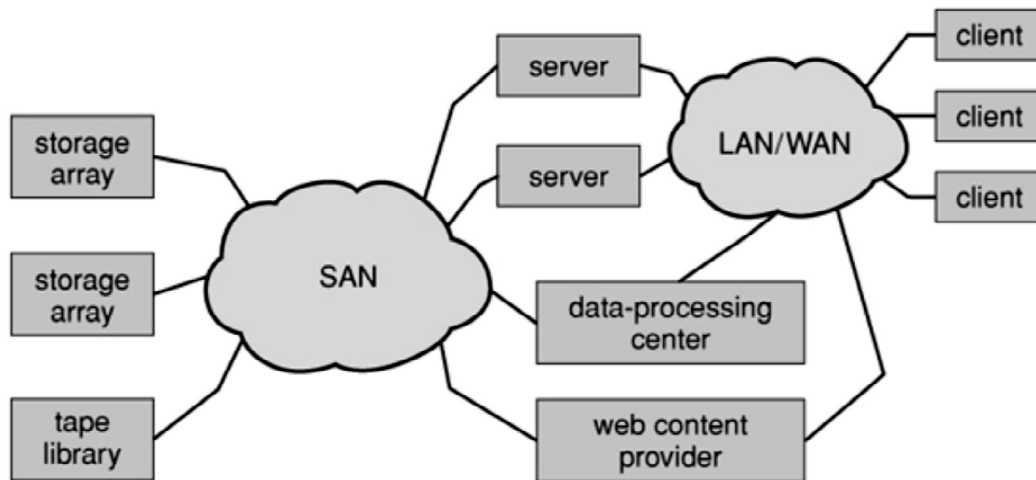


Fig . Storage-area network

Q14. What is Disk Scheduling? Explain various Disk Scheduling Algorithms

Ans :

- As mentioned earlier, disk transfer speeds are limited primarily by **seek times** and **rotational latency**. When multiple requests are to be processed there is also some inherent delay in waiting for other requests to be processed.
- **Bandwidth** is measured by the amount of data transferred divided by the total amount of time from the first request being made to the last transfer being completed, (for a series of disk requests.)
- Both bandwidth and access time can be improved by processing requests in a good order.
- Disk requests include the disk address, memory address, number of sectors to transfer, and whether the request is for reading or writing.

Various disk scheduling algorithms are-

1. FCFS Algorithm
2. SSTF Algorithm
3. SCAN Algorithm
4. C-SCAN Algorithm
5. LOOK Algorithm
6. C-LOOK Algorithm

1. FCFS Disk Scheduling Algorithm-

- As the name suggests, this algorithm entertains requests in the order they arrive in the disk queue.
- It is the simplest disk scheduling algorithm.

Advantages-

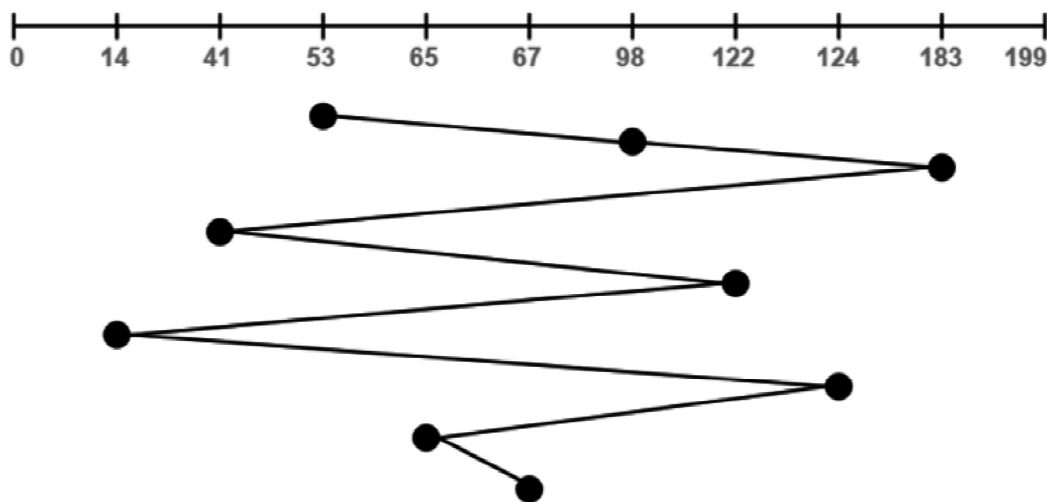
- It is simple, easy to understand and implement.
- It does not cause starvation to any request.

Disadvantages -

- It results in increased total seek time.
- It is inefficient.

Example

Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The FCFS scheduling algorithm is used. The head is initially at cylinder number 53. The cylinders are numbered from 0 to 199. The total head movement (in number of cylinders) incurred while servicing these requests is _____.

Solution :

Total head movements incurred while servicing these requests

$$\begin{aligned}
 &= (98 - 53) + (183 - 98) + (183 - 41) + (122 - 41) + (122 - 14) \\
 &\quad + (124 - 14) + (124 - 65) + (67 - 65) \\
 &= 45 + 85 + 142 + 81 + 108 + 110 + 59 + 2 = 632
 \end{aligned}$$

2. SSTF Disk Scheduling Algorithm-

- SSTF stands for **Shortest Seek Time First**.
- This algorithm services that request next which requires least number of head movements from its current position regardless of the direction.
- It breaks the tie in the direction of head movement.

Advantages-

- It reduces the total seek time as compared to **FCFS**.
- It provides increased throughput.
- It provides less average response time and waiting time.

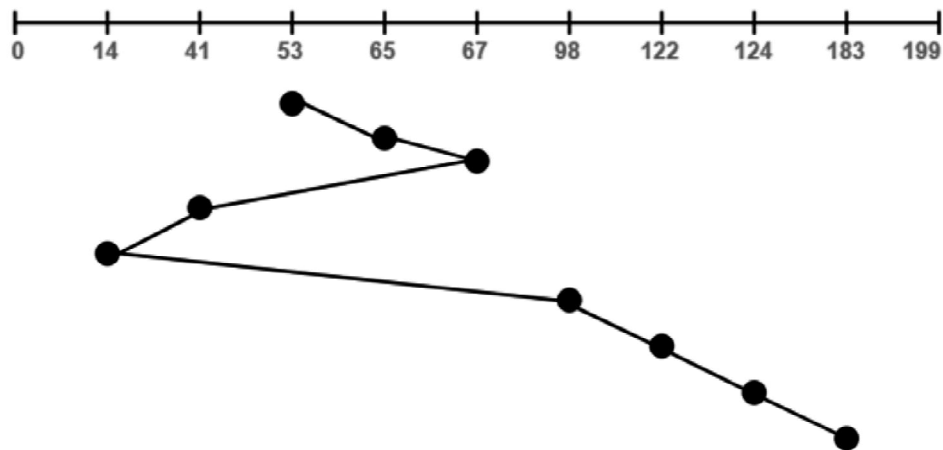
Disadvantages-

- There is an overhead of finding out the closest request.
- The requests which are far from the head might starve for the CPU.
- It provides high variance in response time and waiting time.
- Switching the direction of head frequently slows down the algorithm.

Example 1:

Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The SSTF scheduling algorithm is used. The head is initially at cylinder number 53 moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199. The total head movement (in number of cylinders) incurred while servicing these requests is _____.

Solution :



Total head movements incurred while servicing these requests

$$\begin{aligned}
 &= (65 - 53) + (67 - 65) + (67 - 41) + (41 - 14) + (98 - 14) + (122 - 98) + (124 - 122) \\
 &\quad + (183 - 124) \\
 &= 12 + 2 + 26 + 27 + 84 + 24 + 2 + 59 \\
 &= 236
 \end{aligned}$$

Example 2 :

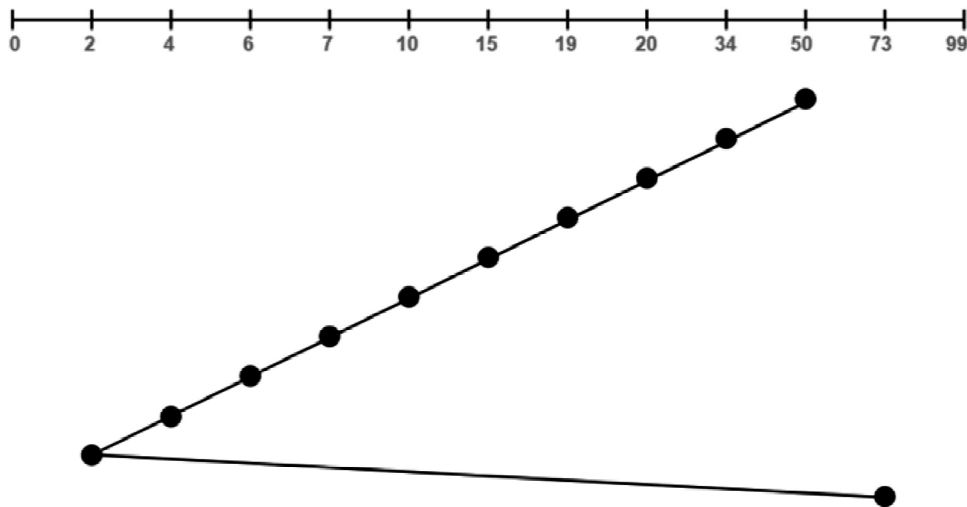
Consider a disk system with 100 cylinders. The requests to access the cylinders occur in following sequence-

4, 34, 10, 7, 19, 73, 2, 15, 6, 20

Assuming that the head is currently at cylinder 50, what is the time taken to satisfy all requests if it takes 1 ms to move from one cylinder to adjacent one and shortest seek time first policy is used?

1. 95 ms
2. 119 ms
3. 233 ms
4. 276 ms

Solution :



Total head movements incurred while servicing these requests

$$\begin{aligned}
 &= (50 - 34) + (34 - 20) + (20 - 19) + (19 - 15) + (15 - 10) + (10 - 7) + (7 - 6) \\
 &\quad + (6 - 4) + (4 - 2) + (73 - 2) \\
 &= 16 + 14 + 1 + 4 + 5 + 3 + 1 + 2 + 2 + 71 \\
 &= 119
 \end{aligned}$$

Time taken for one head movement = 1 msec. So,

Time taken for 119 head movements

$$\begin{aligned}
 &= 119 \times 1 \text{ msec} \\
 &= 119 \text{ msec}
 \end{aligned}$$

3. SCAN Disk Scheduling Algorithm

- As the name suggests, this algorithm scans all the cylinders of the disk back and forth.
- Head starts from one end of the disk and move towards the other end servicing all the requests in between.
- After reaching the other end, head reverses its direction and move towards the starting end servicing all the requests in between.
- The same process repeats.

NOTE - SCAN Algorithm is also called as Elevator Algorithm. This is because its working resembles the working of an elevator.

Advantages-

- It is simple, easy to understand and implement.
- It does not lead to starvation.
- It provides low variance in response time and waiting time.

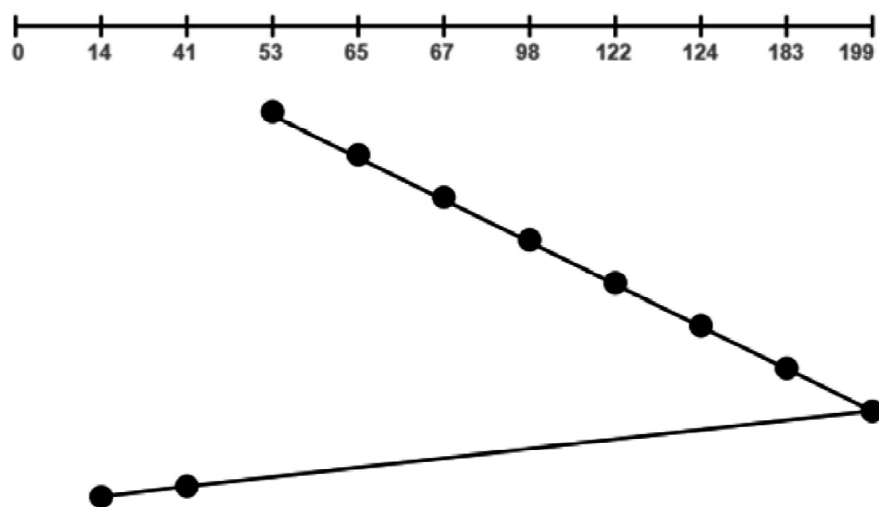
Disadvantages-

- It causes long waiting time for the cylinders just visited by the head.
- It causes the head to move till the end of the disk even if there are no requests to be serviced.

Example

Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The SCAN scheduling algorithm is used. The head is initially at cylinder number 53 moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199. The total head movement (in number of cylinders) incurred while servicing these requests is _____.

Solution :



Total head movements incurred while servicing these requests

$$\begin{aligned}
 &= (65 - 53) + (67 - 65) + (98 - 67) + (122 - 98) + (124 - 122) + (183 - 124) \\
 &\quad + (199 - 183) + (199 - 41) + (41 - 14) \\
 &= 12 + 2 + 31 + 24 + 2 + 59 + 16 + 158 + 27 \\
 &= 331
 \end{aligned}$$

Alternatively,

Total head movements incurred while servicing these requests

$$\begin{aligned}
 &= (199 - 53) + (199 - 14) \\
 &= 146 + 185 \\
 &= 331
 \end{aligned}$$

4. C-SCAN Disk Scheduling Algorithm-

- Circular-SCAN Algorithm is an improved version of the **SCAN Algorithm**.
- Head starts from one end of the disk and move towards the other end servicing all the requests in between.
- After reaching the other end, head reverses its direction.

- It then returns to the starting end without servicing any request in between.
- The same process repeats.

Advantages-

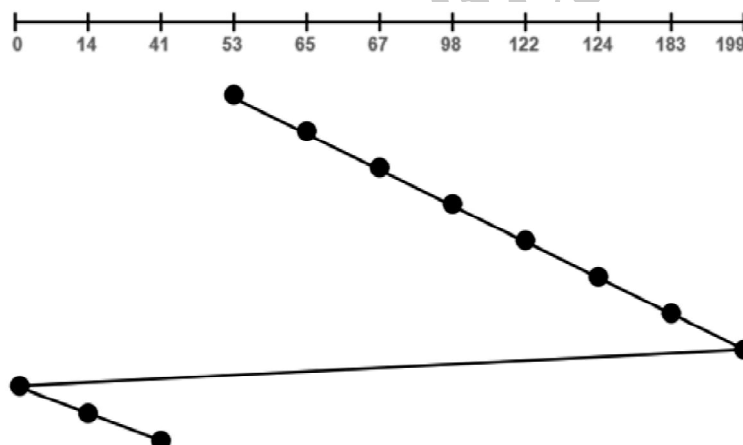
- The waiting time for the cylinders just visited by the head is reduced as compared to the SCAN Algorithm.
- It provides uniform waiting time.
- It provides better response time.

Disadvantages-

- It causes more seek movements as compared to SCAN Algorithm.
- It causes the head to move till the end of the disk even if there are no requests to be serviced.

Example-

Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The C-SCAN scheduling algorithm is used. The head is initially at cylinder number 53 moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199. The total head movement (in number of cylinders) incurred while servicing these requests is _____.

Solution :

Total head movements incurred while servicing these requests

$$\begin{aligned}
 &= (65 - 53) + (67 - 65) + (98 - 67) + (122 - 98) + (124 - 122) + (183 - 124) \\
 &\quad + (199 - 183) + (199 - 0) + (14 - 0) + (41 - 14) \\
 &= 12 + 2 + 31 + 24 + 2 + 59 + 16 + 199 + 14 + 27 \\
 &= 386
 \end{aligned}$$

Alternatively,

Total head movements incurred while servicing these requests

$$\begin{aligned}
 &= (199 - 53) + (199 - 0) + (41 - 0) \\
 &= 146 + 199 + 41 \\
 &= 386.
 \end{aligned}$$

5. LOOK Disk Scheduling Algorithm-

- LOOK Algorithm is an improved version of the **SCAN Algorithm**.
- Head starts from the first request at one end of the disk and moves towards the last request at the other end servicing all the requests in between.
- After reaching the last request at the other end, head reverses its direction.
- It then returns to the first request at the starting end servicing all the requests in between.
- The same process repeats.

NOTE

The main difference between SCAN Algorithm and LOOK Algorithm is-

- SCAN Algorithm scans all the cylinders of the disk starting from one end to the other end even if there are no requests at the ends.
- LOOK Algorithm scans all the cylinders of the disk starting from the first request at one end to the last request at the other end.

Advantages

- It does not causes the head to move till the ends of the disk when there are no requests to be serviced.
- It provides better performance as compared to SCAN Algorithm.
- It does not lead to starvation.
- It provides low variance in response time and waiting time.

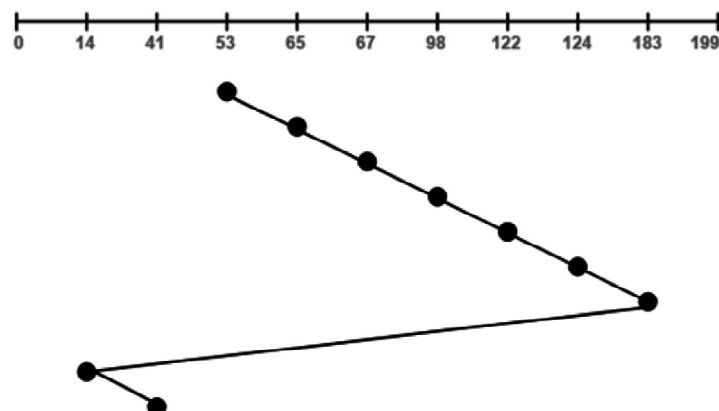
Disadvantages

- There is an overhead of finding the end requests.
- It causes long waiting time for the cylinders just visited by the head.

Example

Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The LOOK scheduling algorithm is used. The head is initially at cylinder number 53 moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199. The total head movement (in number of cylinders) incurred while servicing these requests is _____.

Solution:



Total head movements incurred while servicing these requests

$$\begin{aligned} &= (65 - 53) + (67 - 65) + (98 - 67) + (122 - 98) + (124 - 122) + (183 - 124) \\ &\quad + (183 - 41) + (41 - 14) \\ &= 12 + 2 + 31 + 24 + 2 + 59 + 142 + 27 \\ &= 299 \end{aligned}$$

Alternatively,

Total head movements incurred while servicing these requests

$$\begin{aligned} &= (183 - 53) + (183 - 14) \\ &= 130 + 169 \\ &= 299 \end{aligned}$$

6. C-LOOK Disk Scheduling Algorithm-

- Circular-LOOK Algorithm is an improved version of the **LOOK Algorithm**.
- Head starts from the first request at one end of the disk and moves towards the last request at the other end servicing all the requests in between.
- After reaching the last request at the other end, head reverses its direction.
- It then returns to the first request at the starting end without servicing any request in between.
- The same process repeats.

Advantages-

- It does not causes the head to move till the ends of the disk when there are no requests to be serviced.
- It reduces the waiting time for the cylinders just visited by the head.
- It provides better performance as compared to LOOK Algorithm.
- It does not lead to starvation.
- It provides low variance in response time and waiting time.

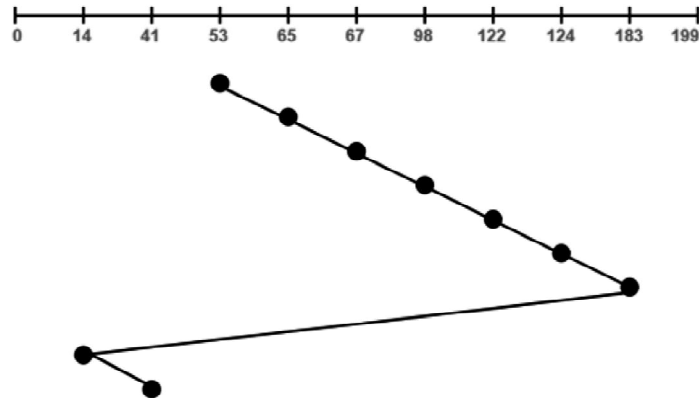
Disadvantages-

- There is an overhead of finding the end requests.

Example 1:

Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The C-LOOK scheduling algorithm is used. The head is initially at cylinder number 53 moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199. The total head movement (in number of cylinders) incurred while servicing these requests is _____.

Solution:



Total head movements incurred while servicing these requests

$$\begin{aligned}
 &= (65 - 53) + (67 - 65) + (98 - 67) + (122 - 98) + (124 - 122) + (183 - 124) \\
 &\quad + (183 - 14) + (41 - 14) \\
 &= 12 + 2 + 31 + 24 + 2 + 59 + 169 + 27 = 326
 \end{aligned}$$

Alternatively,

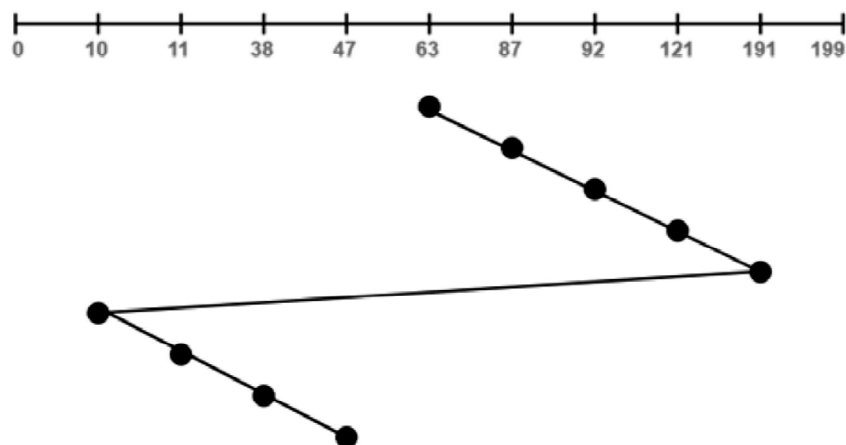
Total head movements incurred while servicing these requests

$$\begin{aligned}
 &= (183 - 53) + (183 - 14) + (41 - 14) \\
 &= 130 + 169 + 27 = 326
 \end{aligned}$$

Example 2 :

Consider a disk queue with requests for I/O to blocks on cylinders 47, 38, 121, 191, 87, 11, 92, 10. The C-LOOK scheduling algorithm is used. The head is initially at cylinder number 63 moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199. The total head movement (in number of cylinders) incurred while servicing these requests is _____.

Solution :



Total head movements incurred while servicing these requests

$$\begin{aligned} &= (87 - 63) + (92 - 87) + (121 - 92) + (191 - 121) + (191 - 10) + (11 - 10) + (38 - 11) + \\ &\quad (47 - 38) \\ &= 24 + 5 + 29 + 70 + 181 + 1 + 27 + 9 \\ &= 346 \end{aligned}$$

Alternatively,

Total head movements incurred while servicing these requests

$$\begin{aligned} &= (191 - 63) + (191 - 10) + (47 - 10) \\ &= 128 + 181 + 37 \\ &= 346 \end{aligned}$$

3.2.5 I/O Systems

Q15. What is I/O Hardware?

Ans :

One of the important jobs of an Operating System is to manage various I/O devices including mouse, keyboards, touch pad, disk drives, display adapters, USB devices, Bit-mapped screen, LED, Analog-to-digital converter, On/off switch, network connections, audio I/O, printers etc.

An I/O system is required to take an application I/O request and send it to the physical device, then take whatever response comes back from the device and send it to the application. I/O devices can be divided into two categories “

- **Block devices** - A block device is one with which the driver communicates by sending entire blocks of data. For example, Hard disks, USB cameras, Disk-On-Key etc.
- **Character devices** - A character device is one with which the driver communicates by sending and receiving single characters (bytes, octets). For example, serial ports, parallel ports, sounds cards etc

Q16. Write a Short Note on Device Controllers.

Ans :

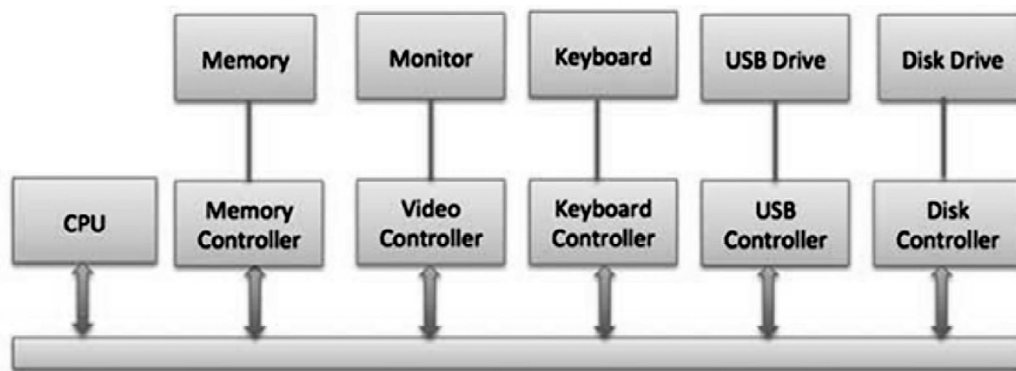
Device Controllers

Device drivers are software modules that can be plugged into an OS to handle a particular device. Operating System takes help from device drivers to handle all I/O devices.

The Device Controller works like an interface between a device and a device driver. I/O units (Keyboard, mouse, printer, etc.) typically consist of a mechanical component and an electronic component where electronic component is called the device controller.

There is always a device controller and a device driver for each device to communicate with the Operating Systems. A device controller may be able to handle multiple devices. As an interface its main task is to convert serial bit stream to block of bytes, perform error correction as necessary.

Any device connected to the computer is connected by a plug and socket, and the socket is connected to a device controller. Following is a model for connecting the CPU, memory, controllers, and I/O devices where CPU and device controllers all use a common bus for communication.



Synchronous vs asynchronous I/O

- **Synchronous I/O** - In this scheme CPU execution waits while I/O proceeds
- **Asynchronous I/O** - I/O proceeds concurrently with CPU execution

Q17. Write about various communication I/O devices.

Ans :

(Imp.)

The CPU must have a way to pass information to and from an I/O device. There are three approaches available to communicate with the CPU and Device.

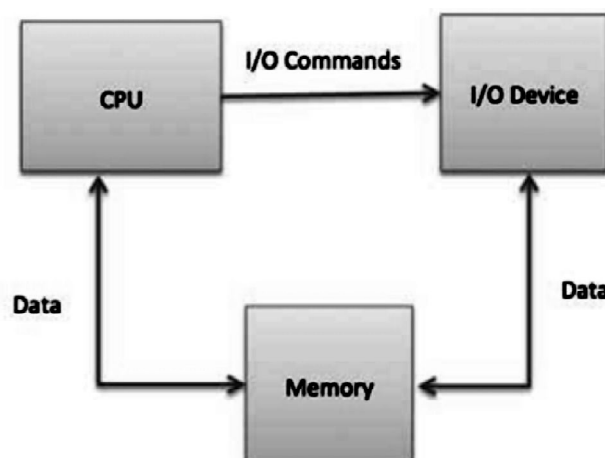
1. Special Instruction I/O
2. Memory-mapped I/O
3. Direct memory access (DMA)

1. Special Instruction I/O

This uses CPU instructions that are specifically made for controlling I/O devices. These instructions typically allow data to be sent to an I/O device or read from an I/O device.

2. Memory-mapped I/O

When using memory-mapped I/O, the same address space is shared by memory and I/O devices. The device is connected directly to certain main memory locations so that I/O device can transfer block of data to/from memory without going through CPU.



While using memory mapped IO, OS allocates 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.

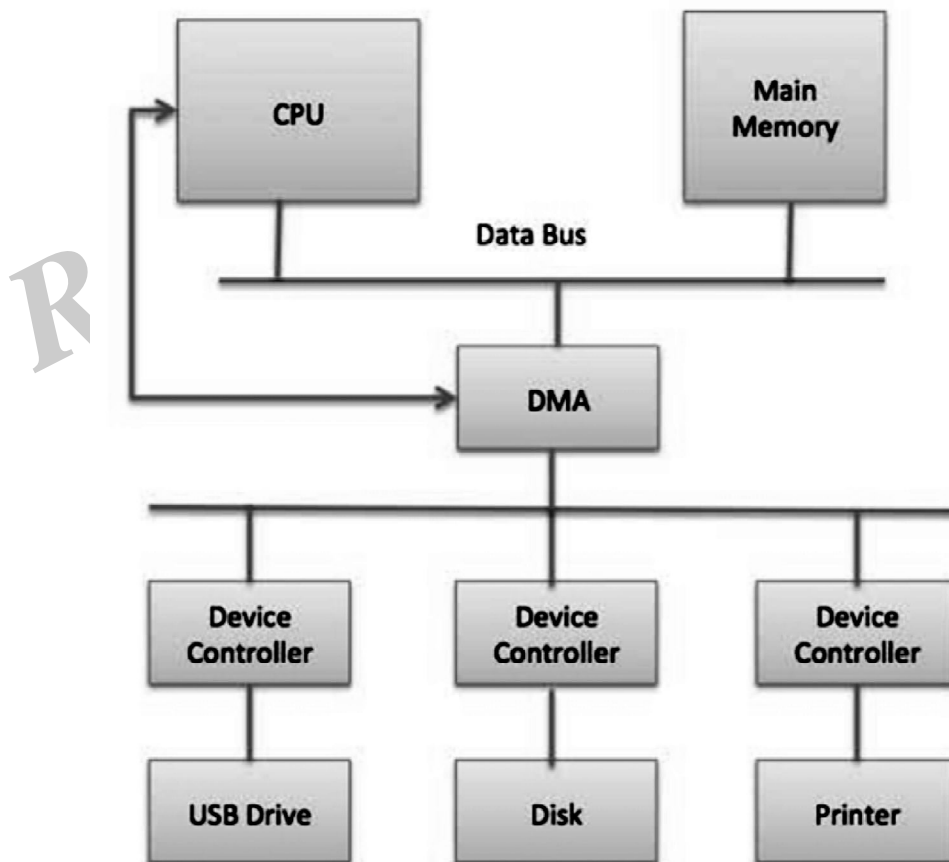
The advantage to this method is that every instruction which can access memory can be used to manipulate an I/O device. Memory mapped IO is used for most high-speed I/O devices like disks, communication interfaces.

3. Direct Memory Access (DMA)

Slow devices like keyboards will generate an interrupt to the main CPU after each byte is transferred. If a fast device such as a disk generated an interrupt for each byte, the operating system would spend most of its time handling these interrupts. So a typical computer uses direct memory access (DMA) hardware to reduce this overhead.

Direct Memory Access (DMA) means CPU grants I/O module authority to read from or write to memory without involvement. DMA module itself controls exchange of data between main memory and the I/O device. CPU is only involved at the beginning and end of the transfer and interrupted only after entire block has been transferred.

Direct Memory Access needs a special hardware called DMA controller (DMAC) that manages the data transfers and arbitrates access to the system bus. The controllers are programmed with source and destination pointers (where to read/write the data), counters to track the number of transferred bytes, and settings, which includes I/O and memory types, interrupts and states for the CPU cycles.



The operating system uses the DMA hardware as follows -

Step	Description
1	Device driver is instructed to transfer disk data to a buffer address X.
2	Device driver then instruct disk controller to transfer data to buffer.
3	Disk controller starts DMA transfer.
4	Disk controller sends each byte to DMA controller.
5	DMA controller transfers bytes to buffer, increases the memory address, decreases the counter C until C becomes zero.
6	When C becomes zero, DMA interrupts CPU to signal transfer completion.

Q18. Write a Short Note on Polling and Interrupts I/O.

Ans :

Polling vs Interrupts I/O

A computer must have a way of detecting the arrival of any type of input. There are two ways that this can happen, known as **polling** and **interrupts**. Both of these techniques allow the processor to deal with events that can happen at any time and that are not related to the process it is currently running.

Polling I/O

Polling is the simplest way for an I/O device to communicate with the processor. The process of periodically checking status of the device to see if it is time for the next I/O operation, is called polling. The I/O device simply puts the information in a Status register, and the processor must come and get the information.

Most of the time, devices will not require attention and when one does it will have to wait until it is next interrogated by the polling program. This is an inefficient method and much of the processors time is wasted on unnecessary polls.

Compare this method to a teacher continually asking every student in a class, one after another, if they need help. Obviously the more efficient method would be for a student to inform the teacher whenever they require assistance.

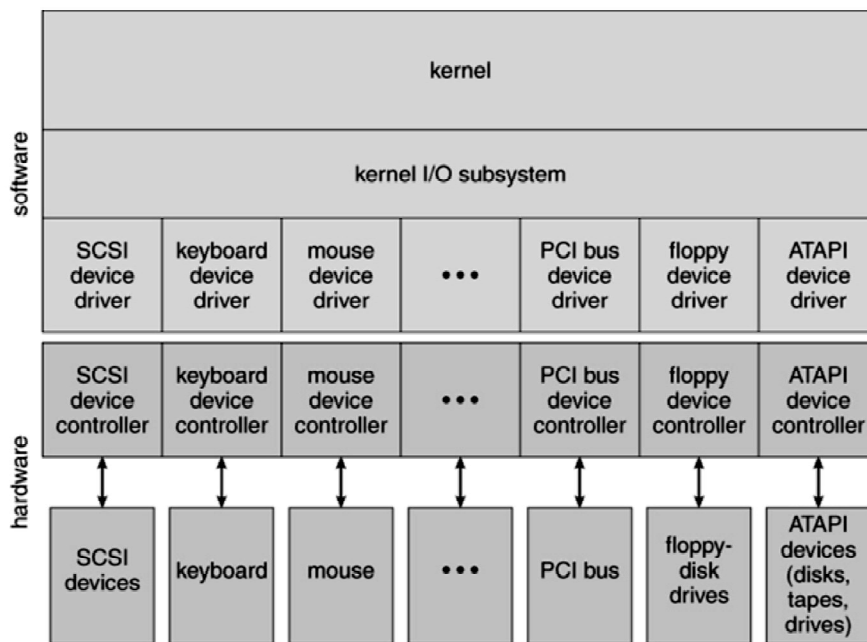
Interrupts I/O

An alternative scheme for dealing with I/O is the interrupt-driven method. An interrupt is a signal to the microprocessor from a device that requires attention.

A device controller puts an interrupt signal on the bus when it needs CPU's attention when CPU receives an interrupt, It saves its current state and invokes the appropriate interrupt handler using the interrupt vector (addresses of OS routines to handle various events). When the interrupting device has been dealt with, the CPU continues with its original task as if it had never been interrupted.

Q19. What is application I/O Interface? Describe various application I/O interfaces*Ans :***(Imp.)**

User application access to a wide variety of different devices is accomplished through layering, and through encapsulating all of the device-specific code into device drivers, while application layers are presented with a common interface for all devices.

**Fig . A kernel I/O structure.**

- Devices differ on many different dimensions, as outlined in Figure.

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read-write	CD-ROM graphics controller disk

Fig . Characteristics of I/O devices.

- Most devices can be characterized as either block I/O, character I/O, memory mapped file access, or network sockets. A few devices are special, such as time-of-day clock and the system timer.

1. Block and Character Devices

- **Block devices** are accessed a block at a time, and are indicated by a “b” as the first character in a long listing on UNIX systems. Operations supported include read(), write(), and seek().
 - Accessing blocks on a hard drive directly (without going through the filesystem structure) is called **raw I/O**, and can speed up certain operations by bypassing the buffering and locking normally conducted by the OS. (It then becomes the application’s responsibility to manage those issues.)
 - A new alternative is **direct I/O**, which uses the normal filesystem access, but which disables buffering and locking operations.
- Memory-mapped file I/O can be layered on top of block-device drivers.
 - Rather than reading in the entire file, it is mapped to a range of memory addresses, and then paged into memory as needed using the virtual memory system.
 - Access to the file is then accomplished through normal memory accesses, rather than through read() and write() system calls. This approach is commonly used for executable program code.
- **Character devices** are accessed one byte at a time, and are indicated by a “c” in UNIX long listings. Supported operations include get() and put(), with more advanced functionality such as reading an entire line supported by higher-level library routines.

2. Network Devices

- Because network access is inherently different from local disk access, most systems provide a separate interface for network devices.
- One common and popular interface is the **socket** interface, which acts like a cable or pipeline connecting two networked entities. Data can be put into the socket at one end, and read out sequentially at the other end. Sockets are normally full-duplex, allowing for bi-directional data transfer.
- The select() system call allows servers (or other applications) to identify sockets which have data waiting, without having to poll all available sockets.

3. Clocks and Timers

- Three types of time services are commonly needed in modern systems:

- o Get the current time of day.
 - o Get the elapsed time (system or wall clock) since a previous event.
 - o Set a timer to trigger event X at time T.
- Unfortunately time operations are not standard across all systems.
- A **programmable interrupt timer, PIT** can be used to trigger operations and to measure elapsed time. It can be set to trigger an interrupt at a specific future time, or to trigger interrupts periodically on a regular basis.
- The scheduler uses a PIT to trigger interrupts for ending time slices.
 - The disk system may use a PIT to schedule periodic maintenance cleanup, such as flushing buffers to disk.
 - Networks use PIT to abort or repeat operations that are taking too long to complete. I.e. resending packets if an acknowledgement is not received before the timer goes off.
 - More timers than actually exist can be simulated by maintaining an ordered list of timer events, and setting the physical timer to go off when the next scheduled event should occur.
- On most systems the system clock is implemented by counting interrupts generated by the PIT. Unfortunately this is limited in its resolution to the interrupt frequency of the PIT, and may be subject to some drift over time. An alternate approach is to provide direct access to a high frequency hardware counter, which provides much higher resolution and accuracy, but which does not support interrupts.
4. Blocking and Non-blocking I/O
- With **blocking I/O** a process is moved to the wait queue when an I/O request is made, and moved back to the ready queue when the request completes, allowing other processes to run in the meantime.
- With **non-blocking I/O** the I/O request returns immediately, whether the requested I/O operation has (completely) occurred or not. This allows the process to check for available data without getting hung completely if it is not there.
- One approach for programmers to implement non-blocking I/O is to have a multi-threaded application, in which one thread makes blocking I/O calls (say to read a keyboard or mouse), while other threads continue to update the screen or perform other tasks.

A subtle variation of the non-blocking I/O is the **asynchronous I/O**, in which the I/O request returns immediately allowing the process to continue on with other tasks, and then the process is notified when the I/O operation has completed and the data is available for use.

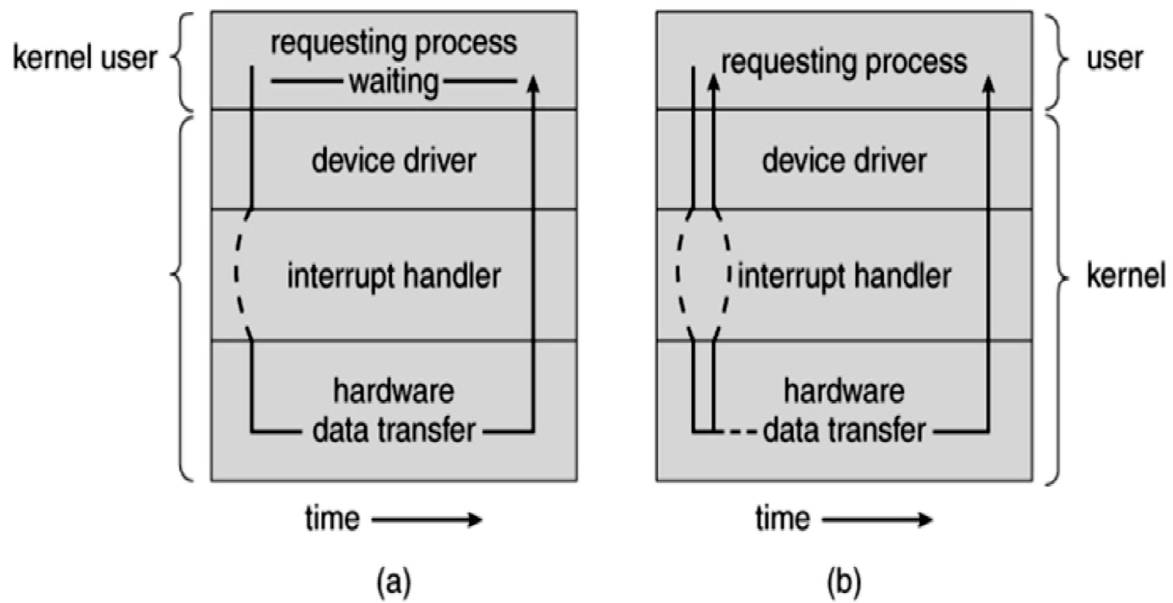


Fig. Two I/O methods: (a) synchronous and (b) asynchronous

UNIT IV

System Protection : Principles and Domain, Access Matrix and implementation, Access control and access rights, Capability based systems.

Language based Protection, System Security : Problem, Program threats, cryptography, user authentication, implementing security defenses, Firewalling, Computer security Classification

4.1 SYSTEM PROTECTION

4.1.1 Principles and Domain

Q1. What are the Goals and Principles of Protection?

Ans : (Imp.)

Goals

- Obviously to prevent malicious misuse of the system by users or programs.
- To ensure that each shared resource is used only in accordance with system policies, which may be set either by system designers or by system administrators.
- To ensure that errant programs cause the minimal amount of damage possible.

Principles

- The **principle of least privilege** dictates that programs, users, and systems be given just enough privileges to perform their tasks.
- This ensures that failures do the least amount of harm and allow the least of harm to be done.
- For example, if a program needs special privileges to perform a task, it is better to make it a SGID program with group ownership of "network" or "backup" or some other pseudo group, rather than SUID with root ownership. This limits the amount of damage that can occur if something goes wrong.
- Typically each user is given their own account, and has only enough privilege to modify their own files.

- The root account should not be used for normal day to day activities - The System Administrator should also have an ordinary account, and reserve use of the root account for only those tasks which need the root privileges

4.1.2 Domain of Protection

Q2. Write about the Domain of Protection.

Ans :

- A computer can be viewed as a collection of processes and objects.
- The **need to know principle** states that a process should only have access to those objects it needs to accomplish its task, and furthermore only in the modes for which it needs access and only during the time frame when it needs access.
- The modes available for a particular object may depend upon its type.

Domain Structure

- A **protection domain** specifies the resources that a process may access.
- Each domain defines a set of objects and the types of operations that may be invoked on each object.
- An **access right** is the ability to execute an operation on an object.
- A domain is defined as a set of < object, { access right set } > pairs, as shown below. Note that some domains may be disjoint while others overlap.



Fig. System with three protection domains.

- The association between a process and a domain may be static or dynamic.
 - If the association is static, then the need-to-know principle requires a way of changing the contents of the domain dynamically.
 - If the association is dynamic, then there needs to be a mechanism for **domain switching**.
- Domains may be realized in different fashions - as users, or as processes, or as procedures. E.g. if each user corresponds to a domain, then that domain defines the access of that user, and changing domains involves changing user ID.

An Example: UNIX

- UNIX associates domains with users.
- Certain programs operate with the SUID bit set, which effectively changes the user ID, and therefore the access domain, while the program is running. (and similarly for the SGID bit.) Unfortunately this has some potential for abuse.
- An alternative used on some systems is to place privileged programs in special directories, so that they attain the identity of the directory owner when they run. This prevents crackers from placing SUID programs in random directories around the system.
- Yet another alternative is to not allow the changing of ID at all. Instead, special privileged daemons are launched at boot time, and user processes send messages to these daemons when they need special tasks performed.

4.1.3 Access Matrix And Implementation

Q3. Explain briefly about access matrix implementation.

Ans :

- The model of protection that we have been discussing can be viewed as an **access matrix**, in which columns represent different system resources and rows represent different protection domains. Entries within the matrix indicate what access that domain has to that resource.

object domain	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Fig . Access matrix

- Domain switching can be easily supported under this model, simply by providing “switch” access to other domains:

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Fig . Access matrix of above figure with domains as objects

- The ability to copy rights is denoted by an asterisk, indicating that processes in that domain have the right to copy that access within the same column, i.e. for the same object. There are two important variations:
- If the asterisk is removed from the original access right, then the right is transferred, rather than being copied. This may be termed a transfer right as opposed to a copy right.
 - If only the right and not the asterisk is copied, then the access right is added to the new domain, but it may not be propagated further. That is the new domain does not also receive the right to copy the access. This may be termed a limited copy right, as shown in Figure below:

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)

Figure - Access matrix with copy rights.

- The **owner** right adds the privilege of adding new rights or removing existing ones:

object domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

(a)

object domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)

Fig. Access matrix with owner rights.

- Copy and owner rights only allow the modification of rights within a column. The addition of **control rights**, which only apply to domain objects, allow a process operating in one domain to affect the rights available in other domains. For example in the table below, a process operating in domain D_2 has the right to control any of the rights in domain D_4 .

object domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

Fig . Modified access matrix of above Figure

Q4. Discuss, how to implement access matrix.*Ans :*

Implementation of Access Matrix

1. Global Table

- The simplest approach is one big global table with < domain, object, rights > entries.
- Unfortunately this table is very large (even if sparse) and so cannot be kept in memory (without invoking virtual memory techniques.)
- There is also no good way to specify groupings - If everyone has access to some resource, then it still needs a separate entry for every domain.

2. Access Lists for Objects

- Each column of the table can be kept as a list of the access rights for that particular object, discarding blank entries.
- For efficiency a separate list of default access rights can also be kept, and checked first.

3. Capability Lists for Domains

- In a similar fashion, each row of the table can be kept as a list of the capabilities of that domain.
- Capability lists are associated with each domain, but not directly accessible by the domain or any user process.
- Capability lists are themselves protected resources, distinguished from other data in one of two ways:
 - A tag, possibly hardware implemented, distinguishing this special type of data.
 - The address space for a program may be split into multiple segments.

4. A Lock-Key Mechanism

- Each resource has a list of unique bit patterns, termed locks.
- Each domain has its own list of unique bit patterns, termed keys.

- Access is granted if one of the domain's keys fits one of the resource's locks.

- Again, a process is not allowed to modify its own keys.

5. Comparison

- Each of the methods here has certain advantages or disadvantages, depending on the particular situation and task at hand.
- Many systems employ some combination of the listed methods.

4.1.4 Access Control And Access Rights**Q5. Write about Access Control Mechanism.***Ans :*

The access control is just another name for compartmentalization of resources. It is useful to group general problems involved in making certain that files are not read or modified by unauthorized personnel or by less privileged programs running on the operating system. There are two major aspects of access control:

1. Access control policies

Access control policy defined "which data is to be protected from whom". In simplest form this is a matrix in which rows define users and columns define files or directories. If value of a cell in this matrix is zero the access to particular file/directory by a particular user is prohibited. Higher value define what type of access is allowed (read-only, read-write, execute, etc). Of course this simple model is unrealistic in filesystems with thousand or millions files and directories. In this case inheritance mechanisms are usually defined to simplify it.

2. Access control mechanisms

The manner by which the operating system enforces the access control policy. Among them the following are the most important

- ▶ Accounts security mechanisms
- ▶ Root Security mechanisms
- **Role-Based Access Control, RBAC,** assigns privileges to users, programs, or roles

as appropriate, where “privileges” refer to the right to call certain system calls, or to use certain parameters with those calls.

- RBAC supports the principle of least privilege, and reduces the susceptibility to abuse as opposed to SUID or SGID programs.

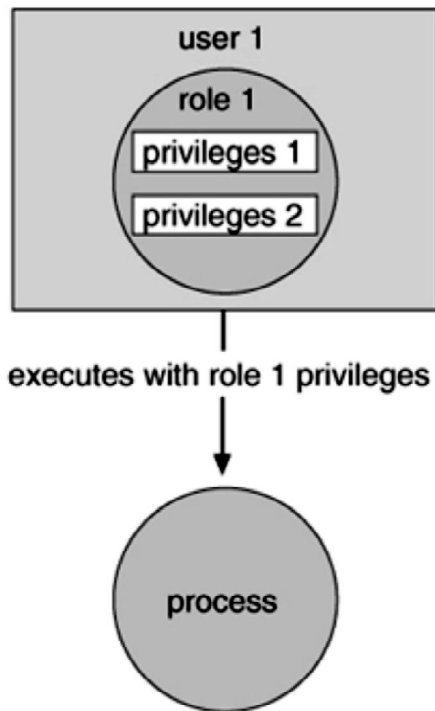


Fig . Role-based access control in Solaris 10.

Q6. How to Revoke Access Rights? Explain.

Ans .:

Revocation of Access Rights

- The need to revoke access rights dynamically raises several questions:
 - **Immediate versus delayed** - If delayed, can we determine when the revocation will take place?
 - **Selective versus general** - Does revocation of an access right to an object affect all users who have that right, or only some users?
 - **Partial versus total** - Can a subset of rights for an object be revoked, or are all rights revoked at once?

- Temporary versus permanent - If rights are revoked, is there a mechanism for processes to re-acquire some or all of the revoked rights?

- With an access list scheme revocation is easy, immediate, and can be selective, general, partial, total, temporary, or permanent, as desired.

- With capabilities lists the problem is more complicated, because access rights are distributed throughout the system. A few schemes that have been developed include:

- **Reacquisition** - Capabilities are periodically revoked from each domain, which must then re-acquire them.
- **Back-pointers** - A list of pointers is maintained from each object to each capability which is held for that object.
- **Indirection** - Capabilities point to an entry in a global table rather than to the object. Access rights can be revoked by changing or invalidating the table entry, which may affect multiple processes, which must then re-acquire access rights to continue.
- **Keys** - A unique bit pattern is associated with each capability when created, which can be neither inspected nor modified by the process.
 - ▶ A master key is associated with each object.
 - ▶ When a capability is created, its key is set to the object's master key.
 - ▶ As long as the capability's key matches the object's key, then the capabilities remain valid.
 - ▶ The object master key can be changed with the set-key command, thereby invalidating all current capabilities.
 - ▶ More flexibility can be added to this scheme by implementing a list of keys for each object, possibly in a global table.

4.1.5 Capability Based Systems

Q7. Explain briefly about Capability Based Systems.

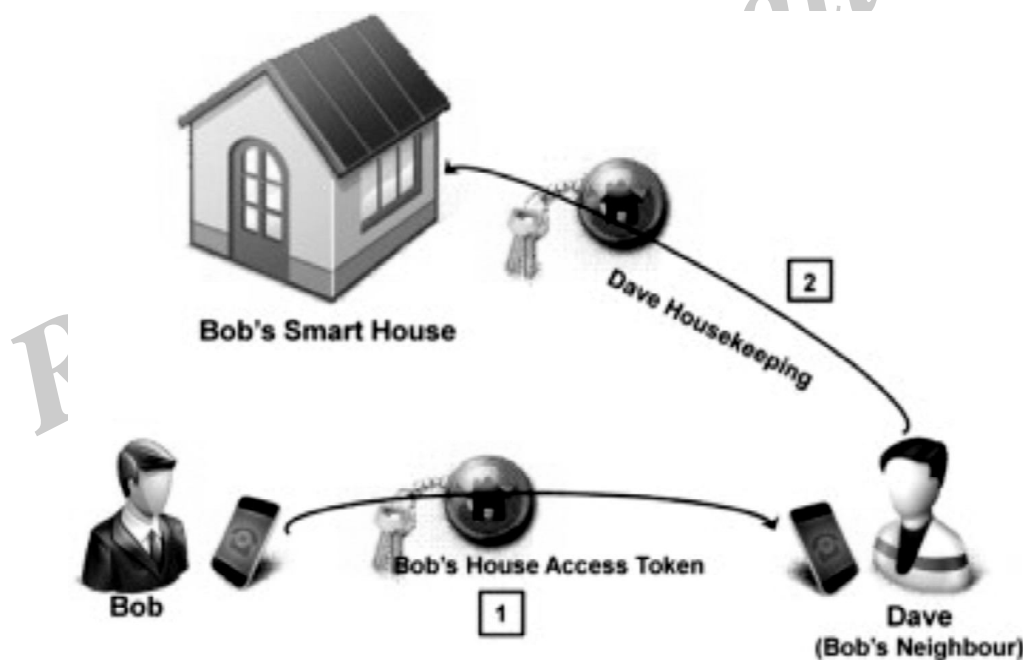
Ans. :

There are two capability-based protection systems. These systems vary in their complexity and in the types of policies that can be implemented on them. Neither system is widely used, but they are interesting proving grounds for protection theories.

An Example: Hydra Hydra is a capability-based protection system that provides considerable flexibility. A fixed set of possible access rights is known to and interpreted by the system. These rights include such basic forms of access as the right to read, write, or execute a memory segment. In addition, a user (of the protection system) can declare other rights.

The interpretation of user-defined rights is performed solely by the user's program, but the system provides access protection for the use of these rights, as well as for the use of system-defined rights. These facilities constitute a significant development in protection technology. Operations on objects are defined procedurally.

The procedures that implement such operations are themselves a form of object, and they are accessed indirectly by capabilities. The names of user-defined procedures must be identified to the protection system if it is to deal with objects of the userdefined type.



When the definition of an object is made known to Hydra, the type becomes auxiliary rights. Auxiliary rights can be described in a capability for an instance of the type. For a process to perform an operation on a typed object, the capability it holds for that object must contain the name of the operation being invoked among its auxiliary rights. This restriction enables discrimination of access rights to be made on an instance-by-instance and process-by-process basis.

Hydra also provides rights amplification. This scheme allows a procedure to be certified as trustworthy to act on a formal parameter of a specified type on behalf of any process that holds a right to execute the procedure. The rights held by a trustworthy procedure are independent of, and may exceed, the rights

held by the calling process. However, such a procedure must not be regarded as universally trustworthy (the procedure is not allowed to act on other types, for instance), and the trustworthiness must not be extended to any other procedures or program segments that might be executed by a process.

Amplification allows implementation procedures access to the representation variables of an abstract data type. If a process holds a capability to a typed object A, for instance, this capability may include an auxiliary right to invoke some operation P but would not include any of the so-called kernel rights, such as read, write, or execute, on the segment that represents A. Such a capability gives a process a means of indirect access (through the operation P) to the representation of A, but only for specific purposes.

When a process invokes the operation P on an object A, however, the capability for access to A may be amplified as control passes to the code body of P. This amplification may be necessary to allow P the right to access the storage segment representing A so as to implement the operation that P defines on the abstract data type.

4.1.6 Language Based Protection

Q8. What is Language Based Protection? Explain.

Ans :

- As systems have developed, protection systems have become more powerful, and also more specific and specialized.
- To refine protection even further requires putting protection capabilities into the hands of individual programmers, so that protection policies can be implemented on the application level, i.e. to protect resources in ways that are known to the specific applications but not to the more general operating system.

Compiler-Based Enforcement

- In a compiler-based approach to protection enforcement, programmers directly specify the protection needed for different resources at the time the resources are declared.

➤ This approach has several advantages:

1. Protection needs are simply declared, as opposed to a complex series of procedure calls.
2. Protection requirements can be stated independently of the support provided by a particular OS.
3. The means of enforcement need not be provided directly by the developer.
4. Declarative notation is natural, because access privileges are closely related to the concept of data types.

➤ Regardless of the means of implementation, compiler-based protection relies upon the underlying protection mechanisms provided by the underlying OS, such as the Cambridge CAP or Hydra systems.

➤ Even if the underlying OS does not provide advanced protection mechanisms, the compiler can still offer some protection, such as treating memory accesses differently in code versus data segments. (E.g. code segments can't be modified, data segments can't be executed.)

➤ There are several areas in which compiler-based protection can be compared to kernel-enforced protection:

- **Security.** Security provided by the kernel offers better protection than that provided by a compiler. The security of the compiler-based enforcement is dependent upon the integrity of the compiler itself, as well as requiring that files not be modified after they are compiled. The kernel is in a better position to protect itself from modification, as well as protecting access to specific files. Where hardware support of individual memory accesses is available, the protection is stronger still.
- **Flexibility.** A kernel-based protection system is not as flexible to provide the specific protection needed by an individual programmer, though it may provide support which the programmer

may make use of. Compilers are more easily changed and updated when necessary to change the protection services offered or their implementation.

- **Efficiency.** The most efficient protection mechanism is one supported by hardware and microcode. Insofar as software based protection is concerned, compiler-based systems have the advantage that many checks can be made off-line, at compile time, rather than during execution.

The concept of incorporating protection mechanisms into programming languages is in its infancy, and still remains to be fully developed. However the general goal is to provide mechanisms for three functions:

1. Distributing capabilities safely and efficiently among customer processes. In particular a user process should only be able to access resources for which it was issued capabilities.
2. Specifying the type of operations a process may execute on a resource, such as reading or writing.
3. Specifying the order in which operations are performed on the resource, such as opening before reading.

Protection in Java

- Java was designed from the very beginning to operate in a distributed environment, where code would be executed from a variety of trusted and untrusted sources. As a result the Java Virtual Machine, JVM incorporates many protection mechanisms
- When a Java program runs, it loads up classes dynamically, in response to requests to instantiate objects of particular types. These classes may come from a variety of different sources, some trusted and some not, which requires that the protection mechanism be implemented at the resolution of individual classes, something not supported by the basic operating system.

- As each class is loaded, it is placed into a separate protection domain. The capabilities of each domain depend upon whether the source URL is trusted or not, the presence or absence of any digital signatures on the class (Chapter 15), and a configurable policy file indicating which servers a particular user trusts, etc.

- When a request is made to access a restricted resource in Java, (e.g. open a local file), some process on the current call stack must specifically assert a privilege to perform the operation. In essence this method assumes responsibility for the restricted access. Naturally the method must be part of a class which resides in a protection domain that includes the capability for the requested operation. This approach is termed stack inspection, and works like this:

- When a caller may not be trusted, a method executes an access request within a `doPrivileged()` block, which is noted on the calling stack.
- When access to a protected resource is requested, `checkPermissions()` inspects the call stack to see if a method has asserted the privilege to access the protected resource.
 - ▶ If a suitable `doPrivileged` block is encountered on the stack before a domain in which the privilege is disallowed, then the request is granted.
 - ▶ If a domain in which the request is disallowed is encountered first, then the access is denied and an `AccessControlException` is thrown.
 - ▶ If neither is encountered, then the response is implementation dependent.

- In the example below the untrusted applet's call to `get()` succeeds, because the trusted URL loader asserts the privilege of opening the specific URL `lucent.com`. However when

the applet tries to make a direct call to `open()` it fails, because it does not have privilege to access any sockets.

protection domain:	untrusted applet	URL loader	networking
socket permission:	none	*.lucent.com:80, connect	any
class:	gui: ... get(url); open(addr); ...	get(URL u): ... doPrivileged { open('proxy.lucnet.com:80'); } <request u from proxy> ...	open(Addr a): ... checkPermission (a, connect); connect (a); ...

Fig - Stack inspection

4.2 SYSTEM SECURITY

4.2.1 Problem

Q9. What is the Security Problem? Explain about the various types of Security Violations.

Ans :

(Imp.)

Security refers to providing a protection system to computer system resources such as CPU, memory, disk, software programs and most importantly data/information stored in the computer system. If a computer program is run by an unauthorized user, then he/she may cause severe damage to computer or data stored in it. So a computer system must be protected against unauthorized access, malicious access to system memory, viruses, worms etc.

➤ **Some of the most common types of violations include:**

- **Breach of Confidentiality** - Theft of private or confidential information, such as credit-card numbers, trade secrets, patents, secret formulas, manufacturing procedures, medical information, financial information, etc.
- **Breach of Integrity** - Unauthorized modification of data, which may have serious indirect consequences. For example a popular game or other program's source code could be modified to open up security holes on users systems before being released to the public.
- **Breach of Availability** - Unauthorized destruction of data, often just for the "fun" of causing havoc and for bragging rites. Vandalism of web sites is a common form of this violation.
- **Theft of Service** - Unauthorized use of resources, such as theft of CPU cycles, installation of daemons running an unauthorized file server, or tapping into the target's telephone or networking services.
- **Denial of Service, DOS** - Preventing legitimate users from using the system, often by overloading and overwhelming the system with an excess of requests for service.

- One common attack is masquerading, in which the attacker pretends to be a trusted third party. A variation of this is the man-in-the-middle, in which the attacker masquerades as both ends of the conversation to two targets.
- A **replay attack** involves repeating a valid transmission. Sometimes this can be the entire attack, (such as repeating a request for a money transfer) or other times the content of the original message is replaced with malicious content.

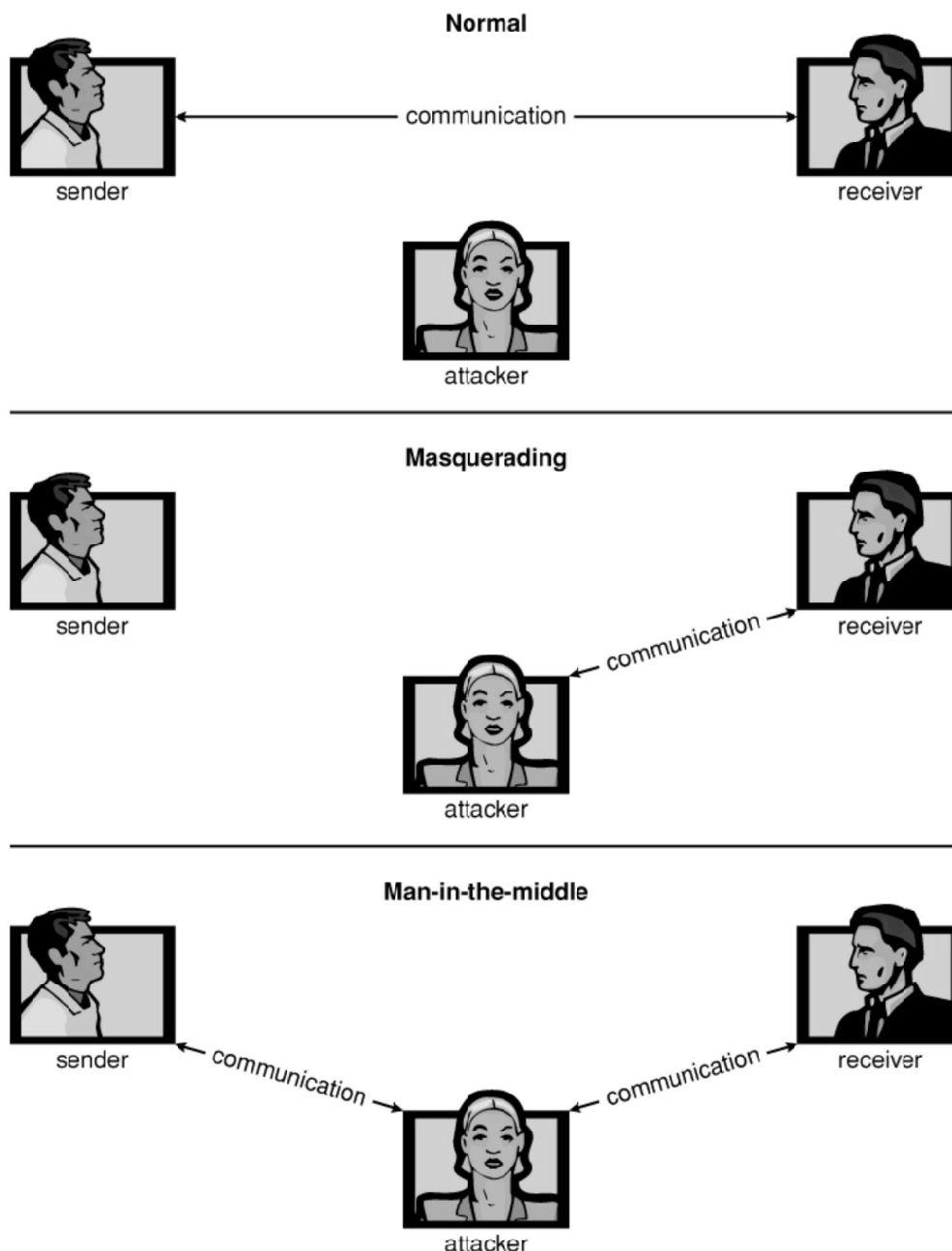


Fig. Standard security attacks.

➤ There are four levels at which a system must be protected:

1. **Physical** - The easiest way to steal data is to pocket the backup tapes. Also, access to the root console will often give the user special privileges, such as rebooting the system as root from removable media. Even general access to terminals in a computer room offers some opportunities for an attacker, although today's modern high-speed networking environment provides more and more opportunities for remote attacks.

2. **Human** - There is some concern that the humans who are allowed access to a system be trustworthy, and that they cannot be coerced into breaching security. However more and more attacks today are made via **social engineering**, which basically means fooling trustworthy people into accidentally breaching security.

➤ **Phishing** involves sending an innocent-looking e-mail or web site designed to fool people into revealing confidential information. E.g. spam e-mails pretending to be from e-Bay, PayPal, or any of a number of banks or credit-card companies.

➤ **Dumpster Diving** involves searching the trash or other locations for passwords that are written down. (Note: Passwords that are too hard to remember, or which must be changed frequently are more likely to be written down somewhere close to the user's station.)

➤ **Password Cracking** involves divining users passwords, either by watching them type in their passwords, knowing something about them like their pet's names, or simply trying all words in common dictionaries. (Note: "Good" passwords should involve a minimum number of characters, include non-alphabetical characters, and not appear in any dictionary (in any language), and should be changed frequently. Note also that it is proper etiquette to look away from the keyboard while someone else is entering their password.)

3. **Operating System** - The OS must protect itself from security breaches, such as runaway processes (denial of service), memory-access violations, stack overflow violations, the launching of programs with excessive privileges, and many others.

4. **Network** - As network communications become ever more important and pervasive in modern computing environments, it becomes ever more important to protect this area of the system. (Both protecting the network itself from attack, and protecting the local system from attacks coming in through the network.) This is a growing area of concern as wireless communications and portable devices become more and more prevalent.

4.2.2 Program Threats

Q10. Explain about various types of program threats.

Ans :

Operating system's processes and kernel do the designated task as instructed. If a user program made these process do malicious tasks, then it is known as **Program Threats**. One of the common example of program threat is a program installed in a computer which can store and send user credentials via network to some hacker. Following is the list of some well-known program threats.

Types of Program Threats –

1. Virus

An infamous threat, known most widely. It is a self-replicating and a malicious thread which attaches itself to a system file and then rapidly replicates itself, modifying and destroying essential files leading to a system breakdown.

Further, Types of computer viruses can be described briefly as follows:

- File/parasitic – appends itself to a file
- Boot/memory – infects the boot sector
- Macro – written in a high-level language like VB and affects MS Office files
- Source code – searches and modifies source codes

- Polymorphic – changes in copying each time
- Encrypted – encrypted virus + decrypting code
- Stealth – avoids detection by modifying parts of the system that can be used to detect it, like the read system call
- Tunneling – installs itself in the interrupt service routines and device drivers
- Multipartite – infects multiple parts of the system

2. Trojan Horse

A code segment that misuses its environment is called a Trojan Horse. They seem to be attractive and harmless cover program but are a really harmful hidden program which can be used as the virus carrier. In one of the versions of Trojan, User is fooled to enter its confidential login details on an application. Those details are stolen by a login emulator and can be further used as a way of information breaches.

Another variance is Spyware, Spyware accompanies a program that the user has chosen to install and downloads ads to display on the user's system, thereby creating pop-up browser windows and when certain sites are visited by the user, it captures essential information and sends it over to the remote server. Such attacks are also known as Covert Channels.

3. Trap Door

The designer of a program or system might leave a hole in the software that only he is capable of using, the Trap Door works on the similar principles. Trap Doors are quite difficult to detect as to analyze them, one needs to go through the source code of all the components of the system.

4. Logic Bomb

A Logic Bomb is code that is not designed to cause havoc all the time, but only when a certain set of circumstances occurs, such as when a particular date or time is reached or some other noticeable event.

A classic example is the Dead-Man Switch, which is designed to check whether a certain person (e.g. the author) is logging in every day, and if they don't log in for a long time (presumably because they've been fired), then the logic bomb goes off and either opens up security holes or causes other problems.

Stack and Buffer Overflow

This is a classic method of attack, which exploits bugs in system code that allows buffers to overflow. Consider what happens in the following code, for example, if `argv[1]` exceeds 256 characters:

- The `strcpy` command will overflow the buffer, overwriting adjacent areas of memory.
- So how does overflowing the buffer cause a security breach? Well the first step is to understand the structure of the stack in memory:
 - The "bottom" of the stack is actually at a high memory address, and the stack grows towards lower addresses.
 - However the address of an array is the lowest address of the array, and higher array elements extend to higher addresses. (I.e. an array "grows" towards the bottom of the stack.
 - In particular, writing past the top of an array, as occurs when a buffer overflows with too much input data, can eventually overwrite the return address, effectively changing where the program jumps to when it returns.

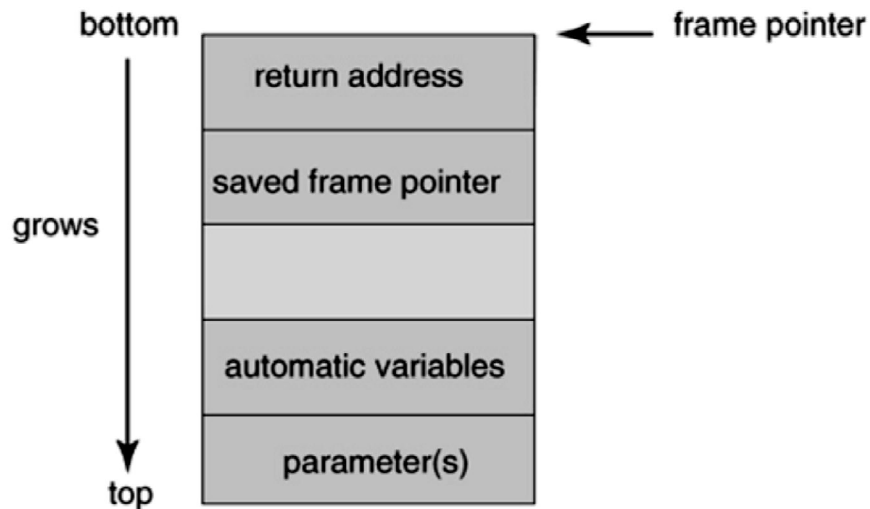


Fig . The layout for a typical stack frame

- Now that we know how to change where the program returns to by overflowing the buffer, the second step is to insert some nefarious code, and then get the program to jump to our inserted code.
- Our only opportunity to enter code is via the input into the buffer, which means there isn't room for very much. One of the simplest and most obvious approaches is to insert the code for "exec(/bin/ sh)". To do this requires compiling a program that contains this instruction, and then using an assembler or debugging tool to extract the minimum extent that includes the necessary instructions.
- The bad code is then padded with as many extra bytes as are needed to overflow the buffer to the correct extent, and the address of the buffer inserted into the return address location. (Note, however, that neither the bad code or the padding can contain null bytes, which would terminate the strcpy.)
- The resulting block of information is provided as "input", copied into the buffer by the original program, and then the return statement causes control to jump to the location of the buffer and start executing the code to launch a shell.

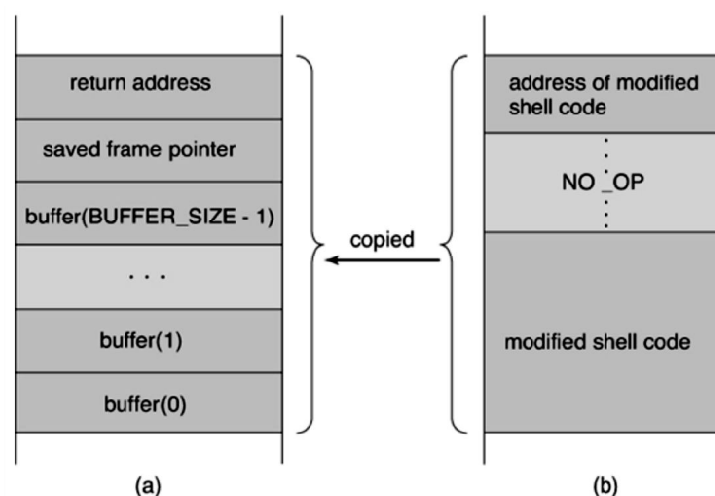


Fig. Hypothetical stack frame for Figure (a) before and (b) after.

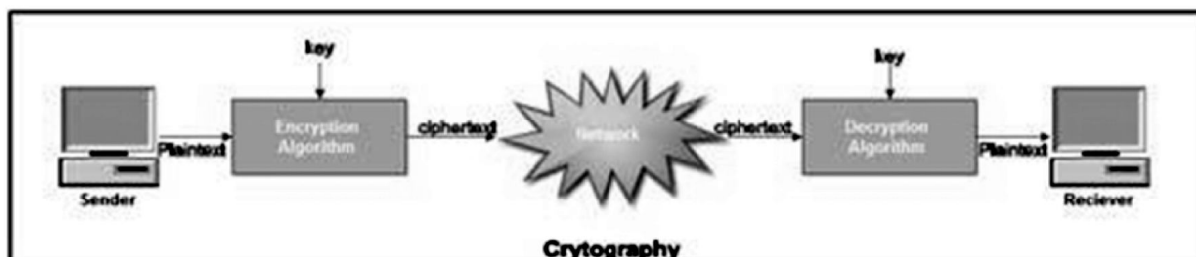
4.2.3 Cryptography

Q11. What is Cryptography? Write about it.

Ans :

(Imp.)

- Cryptography is a technique to provide message confidentiality.
- The term **cryptography** is a Greek word which means "secret writing".
- It is an art and science of transforming messages so as to make them secure and immune to attacks.
- Cryptography involves the process of encryption and decryption. This process is depicted.



➤ The terminology used in cryptography is given below:

1. **Plaintext.** The original message or data that is fed into the algorithm as input is called plaintext.
2. **Encryption algorithm.** The encryption algorithm is the algorithm that performs various substitutions and transformations on the plaintext. Encryption is the process of changing plaintext into cipher text.
3. **Ciphertext.** Ciphertext is the encrypted form the message. It is the scrambled message produced as output. It depends upon the plaintext and the key.
4. **Decryption algorithm.** The process of changing Ciphertext into plain text is known as decryption. Decryption algorithm is essentially the encryption algorithm run in reverse. It takes the Ciphertext and the key and produces the original plaintext.
5. **Key.** It also acts as input to the encryption algorithm. The exact substitutions and transformations performed by the algorithm depend on the key. Thus a key is a number or a set of number that the algorithm uses to perform encryption and decryption.

➤ There are two different approaches to attack an encryption scheme:

1. Cryptanalysis
2. Brute-force attack

1. Cryptanalysis

- The process of attempting to discover the plaintext or key IS known as cryptanalysis.
- The strategy used by cryptanalyst depends on the nature of the encryption scheme and the information available to the cryptanalyst.
- Cryptanalyst can do any or all of six different things:

- Attempt to break a single message.
- Attempt to recognize patterns in encrypted messages, to be able to break subsequent ones by applying a straight forward decryption algorithm.
- Attempt to infer some meaning without even breaking the encryption, such as noticing an unusual-frequency of communication or determining something by whether the communication was short or long.
- Attempt to deduce the key, in order to break subsequent messages easily.
- Attempt to find weaknesses in the implementation or environment of use encryption.
- Attempt to find general weaknesses in an encryption algorithm without necessarily having intercepted any messages.

2. Brute-force attack

- This method tries every possible key on a piece of Ciphertext until an intelligible translation into plaintext is obtained.

Q12. What is Encryption? Write about Symmetric and Asymmetric Encryption.

Ans :

(Imp.)

- The basic idea of encryption is to encode a message so that only the desired recipient can decode and read it. Encryption has been around since before the days of Caesar, and is an entire field of study in itself. Only some of the more significant computer encryption schemes will be covered here.
- The basic process of encryption is shown in Figure and will form the basis of most of our discussion on encryption. The steps in the procedure and some of the key terminology are as follows:
 1. The **sender** first creates a **message, m** in plaintext.
 2. The message is then entered into an **encryption algorithm, E**, along with the **encryption key, Ke**.
 3. The encryption algorithm generates the **ciphertext, c**, $= E(Ke)(m)$. For any key k, $E(k)$ is an algorithm for generating ciphertext from a message, and both E and $E(k)$ should be efficiently computable functions.
 4. The ciphertext can then be sent over an unsecure network, where it may be received by **attackers**.
 5. The **recipient** enters the ciphertext into a **decryption algorithm, D**, along with the **decryption key, Kd**.
 6. The decryption algorithm re-generates the plaintext message, m , $= D(Kd)(c)$. For any key k, $D(k)$ is an algorithm for generating a clear text message from a ciphertext, and both D and $D(k)$ should be efficiently computable functions.
 7. The algorithms described here must have this important property: Given a ciphertext c, a computer can only compute a message m such that $c = E(k)(m)$ if it possesses $D(k)$. (In other words, the messages can't be decoded unless you have the decryption algorithm and the decryption key.)

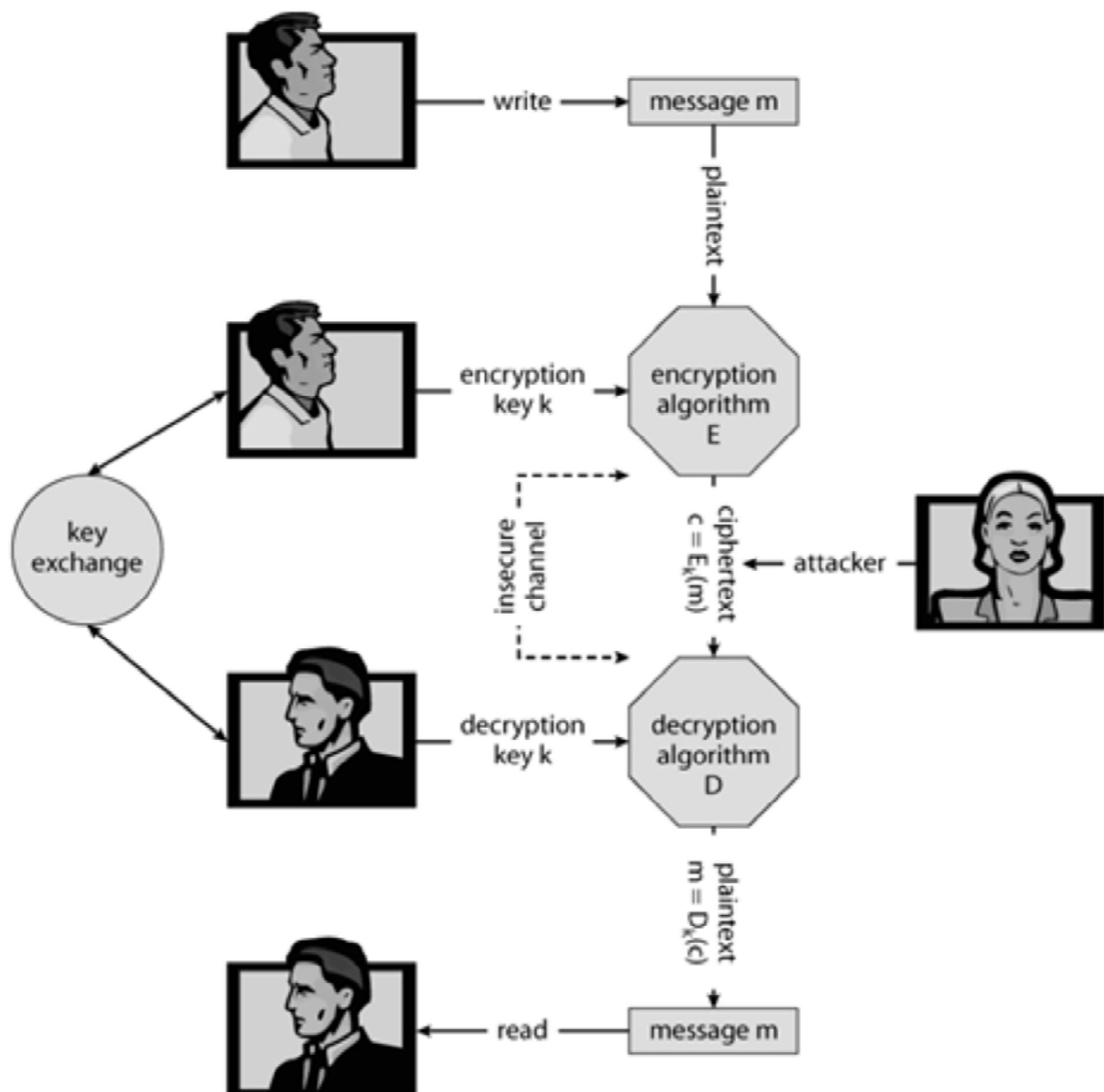


Fig. A secure communication over an insecure medium.

A) Symmetric Encryption

- With **symmetric encryption** the same key is used for both encryption and decryption, and must be safely guarded. There are a number of well-known symmetric encryption algorithms that have been used for computer security:
 - The **Data-Encryption Standard, DES**, developed by the National Institute of Standards, NIST, has been a standard civilian encryption standard for over 20 years. Messages are broken down into 64-bit chunks, each of which are encrypted using a 56-bit key through a series of substitutions and transformations. Some of the transformations are hidden (black boxes), and are classified by the U.S. government.

- DES is known as a **block cipher**, because it works on blocks of data at a time. Unfortunately this is a vulnerability if the same key is used for an extended amount of data. Therefore an enhancement is to not only encrypt each block, but also to XOR it with the previous block, in a technique known as **cipher-block chaining**.
- As modern computers become faster and faster, the security of DES has decreased, to where it is now considered insecure because its keys can be exhaustively searched within a reasonable amount of computer time. An enhancement called **triple DES** encrypts the data three times using three separate keys (actually two encryptions and one decryption) for an effective key length of 168 bits. Triple DES is in widespread use today.
- The **Advanced Encryption Standard, AES**, developed by NIST in 2001 to replace DES uses key lengths of 128, 192, or 256 bits, and encrypts in blocks of 128 bits using 10 to 14 rounds of transformations on a matrix formed from the block.
- The **twofish algorithm**, uses variable key lengths up to 256 bits and works on 128 bit blocks.
- **RC5** can vary in key length, block size, and the number of transformations, and runs on a wide variety of CPUs using only basic computations.
- **RC4** is a **stream cipher**, meaning it acts on a stream of data rather than blocks. The key is used to seed a pseudo-random number generator, which generates a **keystream** of keys. RC4 is used in **WEP**, but has been found to be breakable in a reasonable amount of computer time.

B) Asymmetric Encryption

- With **asymmetric encryption**, the decryption key, K_d , is not the same as the encryption key, K_e , and more importantly cannot be derived from it, which means the encryption key can be made publicly available, and only the decryption key needs to be kept secret. (or vice-versa, depending on the application.)
- One of the most widely used asymmetric encryption algorithms is **RSA**, named after its developers - Rivest, Shamir, and Adleman.
- RSA is based on two large prime numbers, p and q , (on the order of 512 bits each), and their product N .
 - K_e and K_d must satisfy the relationship:

$$(K_e * K_d) \% [(p - 1) * (q - 1)] = 1$$
 - The encryption algorithm is:

$$c = E(K_e)(m) = m^{K_e} \% N$$
 - The decryption algorithm is:

$$m = D(K_d)(c) = c^{K_d} \% N$$
- An example using small numbers:
 - $p = 7$
 - $q = 13$
 - $N = 7 * 13 = 91$
 - $(p - 1) * (q - 1) = 6 * 12 = 72$
 - Select $K_e < 72$ and relatively prime to 72, say 5

- Now select K_d , such that $(K_e * K_d) \% 72 = 1$, say 29
- The public key is now $(5, 91)$ and the private key is $(29, 91)$
- Let the message, $m = 42$
- Encrypt: $c = 42^5 \% 91 = 35$
- Decrypt: $m = 35^{29} \% 91 = 42$

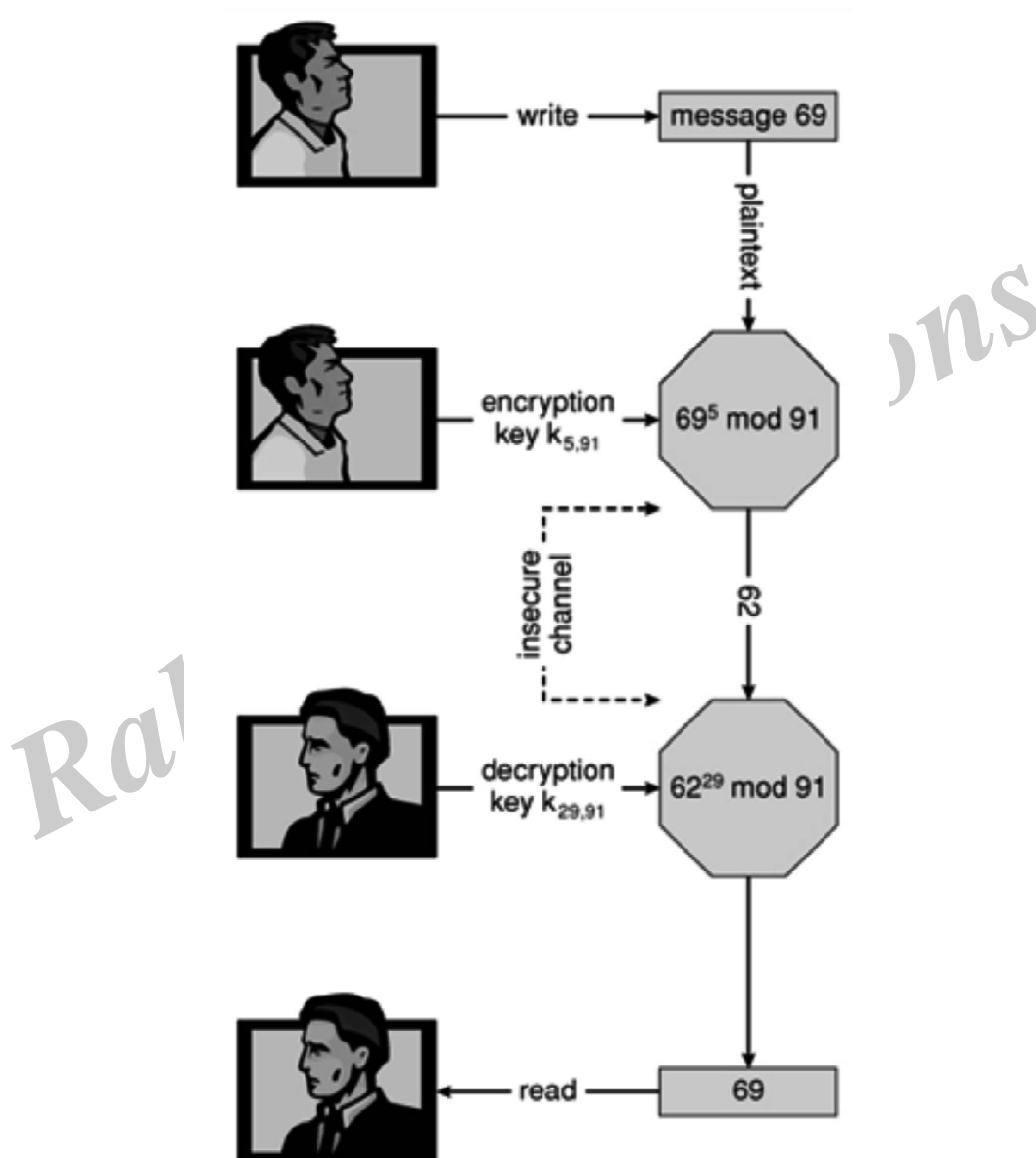


Fig. Encryption and decryption using RSA asymmetric cryptography

- Note that asymmetric encryption is much more computationally expensive than symmetric encryption, and as such it is not normally used for large transmissions. Asymmetric encryption is suitable for small messages, authentication, and key distribution, as covered in the following sections.

Q13. Write a note on authentication and key distribution.

Ans :

Authentication

- Authentication involves verifying the identity of the entity who transmitted a message.
- For example, if $D(K_d)(c)$ produces a valid message, then we know the sender was in possession of $E(K_e)$.
- This form of authentication can also be used to verify that a message has not been modified
- Authentication revolves around two functions, used for **signatures** (or **signing**), and **verification**:
 - A signing function, **$S(K_s)$** that produces an **authenticator, A**, from any given message m.
 - A Verification function, **$V(K_v, m, A)$** that produces a value of "true" if A was created from m, and "false" otherwise.
 - Obviously S and V must both be computationally efficient.
 - More importantly, it must not be possible to generate a valid authenticator, A, without having possession of $S(K_s)$.
 - Furthermore, it must not be possible to derive $S(K_s)$ from the combination of (m and A), since both are sent visibly across networks.
- Understanding authenticators begins with an understanding of hash functions, which is the first step:
 - **Hash functions, $H(m)$** generate a small fixed-size block of data known as a **message digest**, or **hash value** from any given input data.
 - For authentication purposes, the hash function must be **collision resistant on m**. That is it should not be reasonably possible to find an alternate message m' such that $H(m') = H(m)$.

- Popular hash functions are **MD5**, which generates a 128-bit message digest, and **SHA-1**, which generates a 160-bit digest.

- Message digests are useful for detecting (accidentally) changed messages, but are not useful as authenticators, because if the hash function is known, then someone could easily change the message and then generate a new hash value for the modified message. Therefore authenticators take things one step further by encrypting the message digest.
- A **message-authentication code, MAC**, uses symmetric encryption and decryption of the message digest, which means that anyone capable of verifying an incoming message could also generate a new message.
- An asymmetric approach is the **digital-signature algorithm**, which produces authenticators called **digital signatures**. In this case K_s and K_v are separate, K_v is the public key, and it is not practical to determine $S(K_s)$ from public information. In practice the sender of a message signs it (produces a digital signature using $S(K_s)$), and the receiver uses $V(K_v)$ to verify that it did indeed come from a trusted source, and that it has not been modified.
- There are three good reasons for having separate algorithms for encryption of messages and authentication of messages:
 1. Authentication algorithms typically require fewer calculations, making verification a faster operation than encryption.
 2. Authenticators are almost always smaller than the messages, improving space efficiency.
 3. Sometimes we want authentication only, and not confidentiality, such as when a vendor issues a new software patch.

Another use of authentication is **non-repudiation**, in which a person filling out an electronic form cannot deny that they were the ones who did so.

Key Distribution

- Key distribution with symmetric cryptography is a major problem, because all keys must be kept secret, and they obviously can't be transmitted over unsecure channels. One option is to send them out-of-band, say via paper or a confidential conversation.
- Another problem with symmetric keys, is that a separate key must be maintained and used for each correspondent with whom one wishes to exchange confidential information.
- Asymmetric encryption solves some of these problems, because the public key can be freely transmitted through any channel, and the private key doesn't need to be transmitted anywhere. Recipients only need to maintain one private key for all incoming messages, though senders must maintain a separate public key for each recipient to which they might wish to send a message. Fortunately the public keys are not confidential, so this key-ring can be easily stored and managed.
- Unfortunately there are still some security concerns regarding the public keys used in asymmetric encryption. Consider for example the following man-in-the-middle attack involving phony public keys:
- One solution to the above problem involves digital certificates, which are public keys that have been digitally signed by a trusted third party. But wait a minute - How do we trust that third party, and how do we know they are really who they say they are? Certain certificate authorities have their public keys included within web browsers and other certificate consumers before they are distributed.

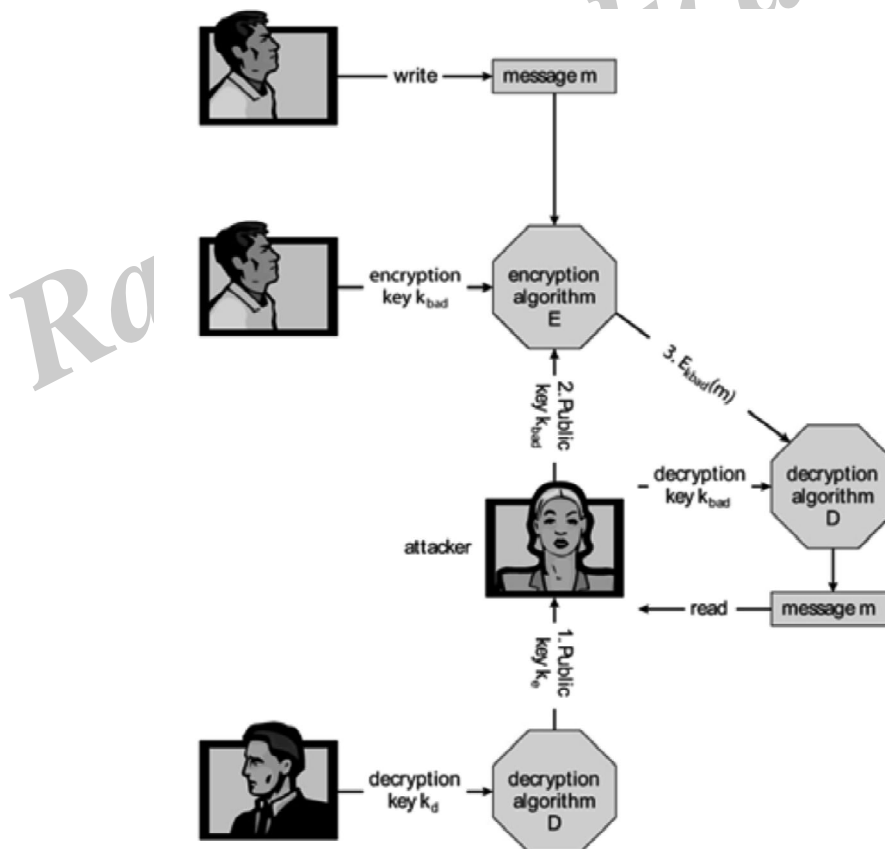


Fig. A man-in-the-middle attack on asymmetric cryptography

Q14. Write about the implementation of cryptography in secure socket layers (SSL).

Ans : (Imp.)

Implementation of Cryptography

- Network communications are implemented in multiple layers - Physical, Data Link, Network, Transport, and Application being the most common breakdown.
- Encryption and security can be implemented at any layer in the stack, with pros and cons to each choice:
 - Because packets at lower levels contain the contents of higher layers, encryption at lower layers automatically encrypts higher layer information at the same time.
 - However security and authorization may be important to higher levels independent of the underlying transport mechanism or route taken.
- At the network layer the most common standard is **IPSec**, a secure form of the IP layer, which is used to set up **Virtual Private Networks, VPNs**.
- At the transport layer the most common implementation is SSL, described below.

An Example: SSL

- SSL (Secure Sockets Layer) 3.0 was first developed by Netscape, and has now evolved into the industry-standard TLS protocol. It is used by web browsers to communicate securely with web servers, making it perhaps the most widely used security protocol on the Internet today.
- SSL is quite complex with many variations, only a simple case of which is shown here.
- The heart of SSL is **session keys**, which are used once for symmetric encryption and then discarded, requiring the generation of new keys for each new session. The big challenge is how to safely create such keys while avoiding man-in-the-middle and replay attacks.

➤ Prior to commencing the transaction, the server obtains a **certificate** from a **certification authority, CA**, containing:

- Server attributes such as unique and common names.
- Identity of the public encryption algorithm, $E()$, for the server.
- The public key, k_e for the server.
- The validity interval within which the certificate is valid.
- A digital signature on the above issued by the CA:
 - ▶ $a = S(K_{CA})(\text{attrs}, E(k_e), \text{interval})$

➤ In addition, the client will have obtained a public **verification algorithm**, $V(K_{CA})$, for the certifying authority. Today's modern browsers include these built-in by the browser vendor for a number of trusted certificate authorities.

➤ The procedure for establishing secure communications is as follows:

1. The client, c , connects to the server, s , and sends a random 28-byte number, n_c .
2. The server replies with its own random value, n_s , along with its certificate of authority.
3. The client uses its verification algorithm to confirm the identity of the sender, and if all checks out, then the client generates a 46 byte random **premaster secret**, **pms**, and sends an encrypted version of it as $cpms = E(k_s)(pms)$
4. The server recovers **pms** as $D(k_s)(cpms)$.
5. Now both the client and the server can compute a shared 48-byte **master secret**, **ms**, $= f(pms, n_s, n_c)$
6. Next, both client and server generate the following from **ms**:

- ▶ Symmetric encryption keys k_{sc_crypt} and k_{cs_crypt} for encrypting messages from the server to the client and vice-versa respectively.
 - ▶ MAC generation keys k_{sc_mac} and k_{cs_mac} for generating authenticators on messages from server to client and client to server respectively.
7. To send a message to the server, the client sends:
- ▶ $c = E(k_{cs_crypt})(m, S(k_{cs_mac})(m))$
8. Upon receiving c , the server recovers:
- ▶ $(m, a) = D(k_{cs_crypt})(c)$
 - ▶ and accepts it if $V(k_{sc_mac})(m, a)$ is true.

This approach enables both the server and client to verify the authenticity of every incoming message, and to ensure that outgoing messages are only readable by the process that originally participated in the key generation.

SSL is the basis of many secure protocols, including **Virtual Private Networks, VPNs**, in which private data is distributed over the insecure public internet structure in an encrypted fashion that emulates a privately owned network.

4.2.4 User Authentication

Q15. Write about various user authentication processes.

Ans :

(Imp.)

Authentication is the process of determining whether someone or something is, in fact, who or what it declares itself to be. Authentication technology provides access control for systems by checking to see if a user's credentials match the credentials in a database of authorized users or in a data authentication server.

Passwords

- Passwords are the most common form of user authentication. If the user is in possession of

the correct password, then they are considered to have identified themselves.

- In theory separate passwords could be implemented for separate activities, such as reading this file, writing that file, etc. In practice most systems use one password to confirm user identity, and then authorization is based upon that identification. This is a result of the classic trade-off between security and convenience.

Password Vulnerabilities

- Passwords can be guessed.
 - Intelligent guessing requires knowing something about the intended target in specific, or about people and commonly used passwords in general.
 - Brute-force guessing involves trying every word in the dictionary, or every valid combination of characters. For this reason good passwords should not be in any dictionary (in any language), should be reasonably lengthy, and should use the full range of allowable characters by including upper and lower case characters, numbers, and special symbols.
- "Shoulder surfing" involves looking over people's shoulders while they are typing in their password.
 - Even if the lurker does not get the entire password, they may get enough clues to narrow it down, especially if they watch on repeated occasions.
 - Common courtesy dictates that you look away from the keyboard while someone is typing their password.
 - Passwords echoed as stars or dots still give clues, because an observer can determine how many characters are in the password.
- "Packet sniffing" involves putting a monitor on a network connection and reading data contained in those packets.

- SSH encrypts all packets, reducing the effectiveness of packet sniffing.
 - However you should still never e-mail a password, particularly not with the word "password" in the same message or worse yet the subject header.
 - Beware of any system that transmits passwords in clear text. ("Thank you for signing up for XYZ. Your new account and password information are shown below".) You probably want to have a spare throw-away password to give these entities, instead of using the same high-security password that you use for banking or other confidential uses.
- Long hard to remember passwords are often written down, particularly if they are used seldomly or must be changed frequently. Hence a security trade-off of passwords that are easily divined versus those that get written down.
- Passwords can be given away to friends or co-workers, destroying the integrity of the entire user-identification system.
- Most systems have configurable parameters controlling password generation and what constitutes acceptable passwords.
- They may be user chosen or machine generated.
 - They may have minimum and/or maximum length requirements.
 - They may need to be changed with a given frequency. (In extreme cases for every session.)
 - A variable length history can prevent repeating passwords.
 - More or less stringent checks can be made against password dictionaries.
- Rather they are encrypted and stored in that form. When a user enters their password, that too is encrypted, and if the encrypted version match, then user authentication passes.
- The encryption scheme was once considered safe enough that the encrypted versions were stored in the publicly readable file `"/etc/passwd"`.
- They always encrypted to a 13 character string, so an account could be disabled by putting a string of any other length into the password field.
 - Modern computers can try every possible password combination in a reasonably short time, so now the encrypted passwords are stored in files that are only readable by the super user. Any password-related programs run as `setuid root` to get access to these files. (`/etc/shadow`)
 - A random seed is included as part of the password generation process, and stored as part of the encrypted password. This ensures that if two accounts have the same plain-text password that they will not have the same encrypted password. However cutting and pasting encrypted passwords from one account to another will give them the same plain-text passwords.

Encrypted Passwords

- Modern systems do not store passwords in clear-text form, and hence there is no mechanism to look up an existing password.

One-Time Passwords

- One-time passwords resist shoulder surfing and other attacks where an observer is able to capture a password typed in by a user.
- These are often based on a **challenge** and a **response**. Because the challenge is different each time, the old response will not be valid for future challenges.
 - ▶ For example, The user may be in possession of a secret function $f(x)$. The system challenges with some given value for x , and the user responds with $f(x)$, which the system can then verify. Since the

challenger gives a different (random) x each time, the answer is constantly changing.

- ▶ A variation uses a map (e.g. a road map) as the key. Today's question might be "On what corner is SEO located?", and tomorrow's question might be "How far is it from Navy Pier to Wrigley Field?" Obviously "Taylor and Morgan" would not be accepted as a valid answer for the second question!
- Another option is to have some sort of electronic card with a series of constantly changing numbers, based on the current time. The user enters the current number on the card, which will only be valid for a few seconds. A **two-factor authorization** also requires a traditional password in addition to the number on the card, so others may not use it if it were ever lost or stolen.
- A third variation is a **code book**, or **one-time pad**. In this scheme a long list of passwords is generated, and each one is crossed off and cancelled as it is used. Obviously it is important to keep the pad secure.

Biometrics

- Biometrics involve a physical characteristic of the user that is not easily forged or duplicated and not likely to be identical between multiple users.
 - Fingerprint scanners are getting faster, more accurate, and more economical.
 - Palm readers can check thermal properties, finger length, etc.
 - Retinal scanners examine the back of the users' eyes.
 - Voiceprint analyzers distinguish particular voices.
 - Difficulties may arise in the event of colds, injuries, or other physiological changes.

4.2.5 Implementing Security Defenses

Q16. What is security policy? Write various security policies.

Ans :

(Imp.)

Security Policy

- A security policy should be well thought-out, agreed upon, and contained in a living document that everyone adheres to and is updated as needed.
- Examples of contents include how often port scans are run, password requirements, virus detectors, etc.

Vulnerability Assessment

- Periodically examine the system to detect vulnerabilities.
 - Port scanning.
 - Check for bad passwords.
 - Look for suid programs.
 - Unauthorized programs in system directories.
 - Incorrect permission bits set.
 - Program checksums / digital signatures which have changed.
 - Unexpected or hidden network daemons.
 - New entries in startup scripts, shutdown scripts, cron tables, or other system scripts or configuration files.
 - New unauthorized accounts.
- The government considers a system to be only as secure as its most far-reaching component. Any system connected to the Internet is inherently less secure than one that is in a sealed room with no external communications.
- Some administrators advocate "security through obscurity", aiming to keep as much information about their systems hidden as possible, and not announcing any security concerns they come across. Others announce

security concerns from the rooftops, under the theory that the hackers are going to find out anyway, and the only one kept in the dark by obscurity are honest administrators who need to get the word.

Intrusion Detection

- Intrusion detection attempts to detect attacks, both successful and unsuccessful attempts. Different techniques vary along several axes:
 - The time that detection occurs, either during the attack or after the fact.
 - The types of information examined to detect the attack(s). Some attacks can only be detected by analyzing multiple sources of information.
 - The response to the attack, which may range from alerting an administrator to automatically stopping the attack (e.g. killing an offending process), to tracing back the attack in order to identify the attacker.
 - ▶ Another approach is to divert the attacker to a **honeypot**, on a **honeynet**. The idea behind a honeypot is a computer running normal services, but which no one uses to do any real work. Such a system should not see any network traffic under normal conditions, so any traffic going to or from such a system is by definition suspicious. Honeypots are normally kept on a honeynet protected by a **reverse firewall**, which will let potential attackers in to the honeypot, but will not allow any outgoing traffic. (So that if the honeypot is compromised, the attacker cannot use it as a base of operations for attacking other systems.) Honeypots are closely watched, and any suspicious activity carefully logged and investigated.
- Intrusion Detection Systems, IDSs, raise the alarm when they detect an intrusion. Intrusion Detection and Prevention Systems,

IDPs, act as filtering routers, shutting down suspicious traffic when it is detected.

- There are two major approaches to detecting problems:

- **Signature-Based Detection** scans network packets, system files, etc. looking for recognizable characteristics of known attacks, such as text strings for messages or the binary code for “exec / bin/sh”. The problem with this is that it can only detect previously encountered problems for which the signature is known, requiring the frequent update of signature lists.
- **Anomaly Detection** looks for “unusual” patterns of traffic or operation, such as unusually heavy load or an unusual number of logins late at night.
 - ▶ The benefit of this approach is that it can detect previously unknown attacks, so called **zero-day attacks**.
 - ▶ One problem with this method is characterizing what is “normal” for a given system. One approach is to benchmark the system, but if the attacker is already present when the benchmarks are made, then the “unusual” activity is recorded as “the norm.”
 - ▶ Another problem is that not all changes in system performance are the result of security attacks. If the system is bogged down and really slow late on a Thursday night, does that mean that a hacker has gotten in and is using the system to send out SPAM, or does it simply mean that a CS 385 assignment is due on Friday? :-)
 - ▶ To be effective, anomaly detectors must have a very low **false alarm (false positive)** rate, lest the warnings get ignored, as well as a low **false negative** rate in which attacks are missed.

Virus Protection

- Modern anti-virus programs are basically signature-based detection systems, which also have the ability (in some cases) of **disinfecting** the affected files and returning them back to their original condition.
- Both viruses and anti-virus programs are rapidly evolving. For example viruses now commonly mutate every time they propagate, and so anti-virus programs look for families of related signatures rather than specific ones.
- Some antivirus programs look for anomalies, such as an executable program being opened for writing (other than by a compiler.)
- Avoiding bootleg, free, and shared software can help reduce the chance of catching a virus, but even shrink-wrapped official software has on occasion been infected by disgruntled factory workers.
- Some virus detectors will run suspicious programs in a **sandbox**, an isolated and secure area of the system which mimics the real system.
- **Rich Text Format, RTF**, files cannot carry macros, and hence cannot carry Word macro viruses.
- Known safe programs (e.g. right after a fresh install or after a thorough examination) can be digitally signed, and periodically the files can be re-verified against the stored digital signatures. (Which should be kept secure, such as on off-line write-only medium.)

Auditing, Accounting, and Logging

- Auditing, accounting, and logging records can also be used to detect anomalous behavior.

- Some of the kinds of things that can be logged include authentication failures and successes, logins, running of suid or sgid programs, network accesses, system calls, etc. In extreme cases almost every keystroke and electron that moves can be logged for future analysis.

- "The Cuckoo's Egg" tells the story of how Cliff Stoll detected one of the early UNIX break ins when he noticed anomalies in the accounting records on a computer system being used by physics researchers.

Tripwire Filesystem (New Sidebar)

- The tripwire filesystem monitors files and directories for changes, on the assumption that most intrusions eventually result in some sort of undesired or unexpected file changes.
- The tw.config file indicates what directories are to be monitored, as well as what properties of each file are to be recorded. (E.g. one may choose to monitor permission and content changes, but not worry about read access times.)
- When first run, the selected properties for all monitored files are recorded in a database. Hash codes are used to monitor file contents for changes.
- Subsequent runs report any changes to the recorded data, including hash code changes, and any newly created or missing files in the monitored directories.
- For full security it is necessary to also protect the tripwire system itself, most importantly the database of recorded file properties. This could be saved on some external or write-

only location, but that makes it harder to change the database when legitimate changes are made.

- It is difficult to monitor files that are supposed to change, such as log files. The best tripwire can do in this case is to watch for anomalies, such as a log file that shrinks in size.

4.2.6 Firewalling

Q17. Write about the use of firewalling in Operating system.

Ans. :

(Imp.)

- Firewalls are devices (or sometimes software) that sit on the border between two security domains and monitor/log activity between them, sometimes restricting the traffic that can pass between them based on certain criteria.
- For example a firewall router may allow HTTP: requests to pass through to a web server inside a company domain while not allowing telnet, ssh, or other traffic to pass through.
- A common architecture is to establish a de-militarized zone, DMZ, which sort of sits "between" the company domain and the outside world, as shown below. Company computers can reach either the DMZ or the outside world, but outside computers can only reach the DMZ. Perhaps most importantly, the DMZ cannot reach any of the other company computers, so even if the DMZ is breached, the attacker cannot get to the rest of the company network.

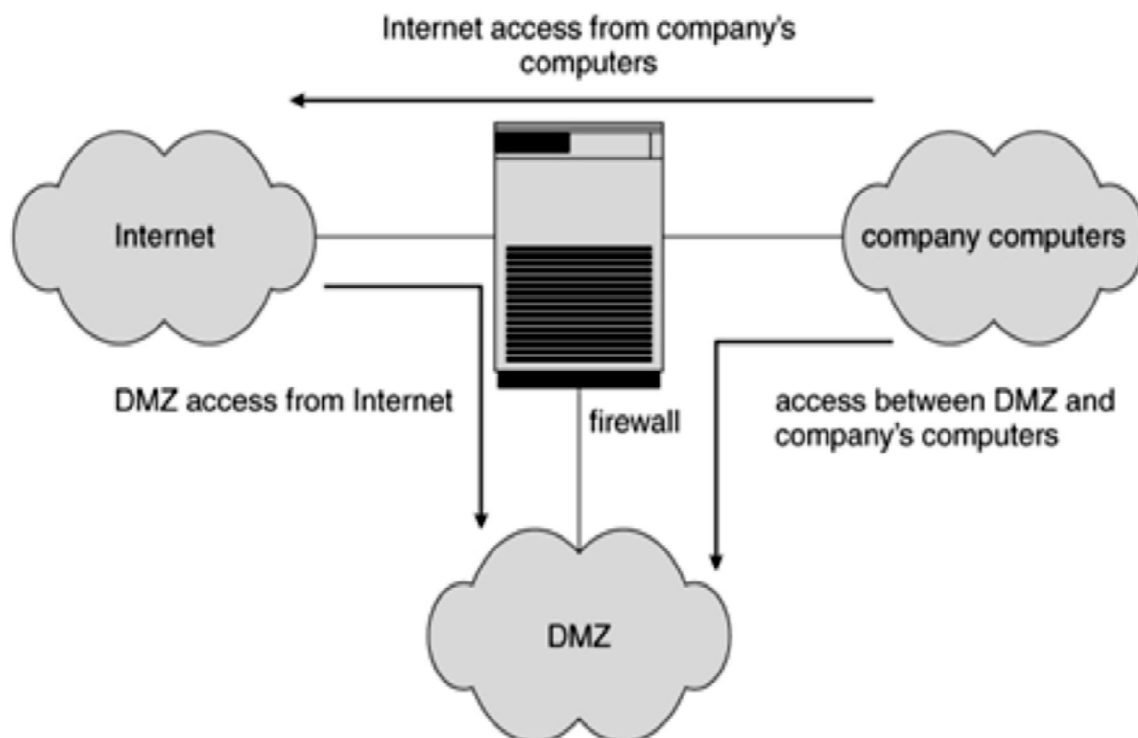


Fig. Domain separation via firewall

- Firewalls themselves need to be resistant to attacks, and unfortunately have several vulnerabilities:
- **Tunneling**, which involves encapsulating forbidden traffic inside of packets that are allowed.
 - Denial of service attacks addressed at the firewall itself.
- Spoofing, in which an unauthorized host sends packets to the firewall with the return address of an authorized host.
- In addition to the common firewalls protecting a company internal network from the outside world, there are also some specialized forms of firewalls that have been recently developed:
- A **personal firewall** is a software layer that protects an individual computer. It may be a part of the operating system or a separate software package.
 - An **application proxy firewall** understands the protocols of a particular service and acts as a stand-in (and relay) for the particular service. For example, and SMTP proxy firewall would accept SMTP requests from the outside world, examine them for security concerns, and forward only the "safe" ones on to the real SMTP server behind the firewall.
 - **XML firewalls** examine XML packets only, and reject ill-formed packets. Similar firewalls exist for other specific protocols.
 - **System call firewalls** guard the boundary between user mode and system mode, and reject any system calls that violate security policies.

4.2.7 Computer Security Classification

Q18. Write about the Levels of Computer Security Classification.

Ans :

(Imp.)

The U.S. Department of Defense Trusted Computer System Evaluation Criteria specify four security classifications in systems: A, B, C, and D. This specification is widely used to determine the security of a facility and to model security solutions, so we explore it here.

The lowest-level classification is division D, or minimal protection. Division D includes only one class and is used for systems that have failed to meet the requirements of any of the other security classes. For instance, MS-DOS and Windows 3.1 are in division D. Division C, the next level of security, provides discretionary protection and accountability of users and their actions through the use of audit capabilities.

Division C has two levels

CI and C2. A CI-class system incorporates some form of controls that allow users to protect private information and to keep other users from accidentally reading or destroying their data. A CI environment is one in which cooperating users access data at the same levels of sensitivity. Most versions of UNIX are CI class.

S.N.	Classification Type & Description
1.	Type A Highest Level. Uses formal design specifications and verification techniques. Grants a high degree of assurance of process security.
2.	Type B Provides mandatory protection system. Have all the properties of a class C2 system. Attaches a sensitivity label to each object. It is of three types. <ul style="list-style-type: none"> • B1 – Maintains the security label of each object in the system. Label is used for making decisions to access control. • B2 – Extends the sensitivity labels to each system resource, such as storage objects, supports covert channels and auditing of events. • B3 – Allows creating lists or user groups for access-control to grant access or revoke access to a given named object.
3.	Type C Provides protection and user accountability using audit capabilities. It is of two types. <ul style="list-style-type: none"> • C1 – Incorporates controls so that users can protect their private information and keep other users from accidentally reading / deleting their data. UNIX versions are mostly C1 class. • C2 – Adds an individual-level access control to the capabilities of a C1 level system.
4.	Type D Lowest level. Minimum protection. MS-DOS, Window 3.1 fall in this category.

UNIT V	<p>Case Studies : The Linux System-Design principles. Kernel modules, Process management.</p> <p>Scheduling, Memory management : File systems, Input and Output, Inter process communication.</p> <p>Windows 7 -Design principles : System components, Terminal services and fast user switching File systems, Networking, Programmer interface.</p>
-----------	---

5.1 CASE STUDIES

5.1.1 The Linux System

Q1. What are the various components of Linux System ?

Ans :

(Imp.)

system-management programs	user processes	user utility programs	compilers
system shared libraries			
Linux kernel			
loadable kernel modules			

- Like most UNIX implementations, Linux is composed of three main bodies of code; the most important distinction between the kernel and all other components.
 - The **kernel** is responsible for maintaining the important abstractions of the operating system
 - Kernel code executes in kernel mode with full access to all the physical resources of the computer

All kernel code and data structures are kept in the same single address space
 - The **system libraries** define a standard set of functions through which applications interact with the kernel, and which implement much of the operating-system functionality that does not need the full privileges of kernel code
 - The **system utilities** perform individual specialized management tasks
- User-mode programs rich and varied, including multiple **shells** like the **bourne-again (bash)**

5.1.2 Design Principles

Q2. Write about the history of linux system and its design principles.

Ans :

History

- Linux uses many tools developed as part of Berkeley's BSD operating system, MIT's X Window System, and the Free Software Foundation's GNU project
- The main system libraries were started by the GNU project, with improvements provided by the Linux community
- Linux networking-administration tools were derived from 4.3BSD code; recent BSD derivatives such as Free BSD have borrowed code from Linux in return
- The Linux system is maintained by a loose network of developers collaborating over the Internet, with a small number of public ftp sites acting as de facto standard repositories
- **File System Hierarchy Standard** document maintained by the Linux community to ensure compatibility across the various system components

Specifies overall layout of a standard Linux file system, determines under which directory names configuration files, libraries, system binaries, and run-time data files should be stored

Design Principles

- Linux is a multiuser, multitasking system with a full set of UNIX-compatible tools
- Its file system adheres to traditional UNIX semantics, and it fully implements the standard UNIX networking model
- Main design goals are speed, efficiency, and standardization
- Linux is designed to be compliant with the relevant POSIX documents; at least two Linux distributions have achieved official POSIX certification
 - Supports Pthreads and a subset of POSIX real-time process control

The Linux programming interface adheres to the SVR4 UNIX semantics, rather than to BSD behavior

5.1.3 Kernel Modules

Q3. Discuss about the kernel modules of Linux Systems.

Ans :

(Imp.)

- Sections of kernel code that can be compiled, loaded, and unloaded independent of the rest of the kernel.
- A kernel module may typically implement a device driver, a file system, or a networking protocol
- The module interface allows third parties to write and distribute, on their own terms, device drivers or file systems that could not be distributed under the GPL.
- Kernel modules allow a Linux system to be set up with a standard, minimal kernel, without any extra device drivers built in.
- Four components to Linux module support:
 - module-management system
 - module loader and unloader
 - driver-registration system
 - conflict-resolution mechanism

Module Management

- Supports loading modules into memory and letting them talk to the rest of the kernel
- Module loading is split into two separate sections:
 - Managing sections of module code in kernel memory
 - Handling symbols that modules are allowed to reference

The module requestor manages loading requested, but currently unloaded, modules; it also regularly queries the kernel to see whether a dynamically loaded module is still in use, and will unload it when it is no longer actively needed

Driver Registration

- Allows modules to tell the rest of the kernel that a new driver has become available

- The kernel maintains dynamic tables of all known drivers, and provides a set of routines to allow drivers to be added to or removed from these tables at any time
- Registration tables include the following items:
 - Device drivers
 - File systems
 - Network protocols
 - Binary format

Conflict Resolution

- A mechanism that allows different device drivers to reserve hardware resources and to protect those resources from accidental use by another driver.
- The conflict resolution module aims to:
 - Prevent modules from clashing over access to hardware resources
 - Prevent autoprobes from interfering with existing device drivers
 - Resolve conflicts with multiple drivers trying to access the same hardware:
 - Kernel maintains list of allocated HW resources
 - Driver reserves resources with kernel database first
 - Reservation request rejected if resource not available

5.1.4 Process Management

Q4. Write about the process management of Linux System.

Ans : (Imp.)

- UNIX process management separates the creation of processes and the running of a new program into two distinct operations.
 - The fork() system call creates a new process
 - A new program is run after a call to exec()
- Under UNIX, a process encompasses all the information that the operating system must

maintain to track the context of a single execution of a single program

- Under Linux, process properties fall into three groups: the process's identity, environment, and context

Process Identity

- **Process ID (PID)** - The unique identifier for the process; used to specify processes to the operating system when an application makes a system call to signal, modify, or wait for another process
- **Credentials** - Each process must have an associated user ID and one or more group IDs that determine the process's rights to access system resources and files
- **Personality** - Not traditionally found on UNIX systems, but under Linux each process has an associated personality identifier that can slightly modify the semantics of certain system calls
 - Used primarily by emulation libraries to request that system calls be compatible with certain specific flavors of UNIX
- **Namespace** - Specific view of file system hierarchy
 - Most processes share common namespace and operate on a shared file-system hierarchy
 - But each can have unique file-system hierarchy with its own root directory and set of mounted file systems

Process Context

- The (constantly changing) state of a running program at any point in time
- The **scheduling context** is the most important part of the process context; it is the information that the scheduler needs to suspend and restart the process
- The kernel maintains **accounting** information about the resources currently being consumed by each process, and the total resources consumed by the process in its lifetime so far

- The **file table** is an array of pointers to kernel file structures
When making file I/O system calls, processes refer to files by their index into this table, the **file descriptor (fd)**
- Whereas the file table lists the existing open files, the **file-system context** applies to requests to open new files
 - The current root and default directories to be used for new file searches are stored here
- The **signal-handler table** defines the routine in the process's address space to be called when specific signals arrive
- The **virtual-memory context** of a process describes the full contents of the its private address space

5.1.5 Scheduling

Q5. Discuss how scheduling can happen in Linux System.

Ans : (Imp.)

- The job of allocating CPU time to different tasks within an operating system
- While scheduling is normally thought of as the running and interrupting of processes, in Linux, scheduling also includes the running of the various kernel tasks
- Running kernel tasks encompasses both tasks that are requested by a running process and tasks that execute internally on behalf of a device driver
- As of 2.5, new scheduling algorithm – preemptive, priority-based, known as O(1)
 - Real-time range
 - nice value
 - Had challenges with interactive performance

Completely Fair Scheduler (CFS)

- Eliminates traditional, common idea of time slice

- Instead all tasks allocated portion of processor's time
- CFS calculates how long a process should run as a function of total number of tasks
- N runnable tasks means each gets 1/N of processor's time
- Then weights each task with its nice value
- Smaller nice value -> higher weight (higher priority)
- Then each task run with for time proportional to task's weight divided by total weight of all runnable tasks
- Configurable variable target latency is desired interval during which each task should run at least once
 - Consider simple case of 2 runnable tasks with equal weight and target latency of 10ms – each then runs for 5ms
 - If 10 runnable tasks, each runs for 1ms
- Minimum granularity ensures each run has reasonable amount of time (which actually violates fairness idea)

5.1.6 Memory Management

Q6. Write a note on memory management of Linux system.

Ans :

- Linux's physical memory-management system deals with allocating and freeing pages, groups of pages, and small blocks of memory
- It has additional mechanisms for handling virtual memory, memory mapped into the address space of running processes
- Splits memory into four different zones due to hardware characteristics

Managing Physical Memory

- The page allocator allocates and frees all physical pages; it can allocate ranges of physically-contiguous pages on request
- The allocator uses a buddy-heap algorithm to keep track of available physical pages

- Each allocatable memory region is paired with an adjacent partner
- Whenever two allocated partner regions are both freed up they are combined to form a larger region
- If a small memory request cannot be satisfied by allocating an existing small free region, then a larger free region will be subdivided into two partners to satisfy the request
- Memory allocations in the Linux kernel occur either statically (drivers reserve a contiguous area of memory during system boot time) or dynamically (via the page allocator)
- Also uses **slab allocator** for kernel memory
- **Page cache** and virtual memory system also manage physical memory
 - Page cache is kernel's main cache for files and main mechanism for I/O to block devices
 - Page cache stores entire pages of file contents for local and network file I/O
- Memory allocations in the Linux kernel occur either statically (drivers reserve a contiguous area of memory during system boot time) or dynamically (via the page allocator)
- Also uses **slab allocator** for kernel memory
- **Page cache** and virtual memory system also manage physical memory
 - Page cache is kernel's main cache for files and main mechanism for I/O to block devices
 - Page cache stores entire pages of file contents for local and network file I/O

5.1.7 File Systems

Q7. Write about the file system of Linux System.

Ans :

- To the user, Linux's file system appears as a hierarchical directory tree obeying UNIX semantics

- Internally, the kernel hides implementation details and manages the multiple different file systems via an abstraction layer, that is, the virtual file system (VFS)
- The Linux VFS is designed around object-oriented principles and is composed of four components:
 - A set of definitions that define what a file object is allowed to look like
 - ▶ The **inode object** structure represent an individual file
 - ▶ The **file object** represents an open file
 - ▶ The **superblock object** represents an entire file system

A **dentry object** represents an individual directory entry

- To the user, Linux's file system appears as a hierarchical directory tree obeying UNIX semantics
- Internally, the kernel hides implementation details and manages the multiple different file systems via an abstraction layer, that is, the virtual file system (VFS)
- The Linux VFS is designed around object-oriented principles and layer of software to manipulate those objects with a set of operations on the objects

The Linux ext3 File System

- **ext3** is standard on disk file system for Linux
 - Uses a mechanism similar to that of BSD Fast File System (FFS) for locating data blocks belonging to a specific file
 - Supersedes older **extfs**, **ext2** file systems
 - Work underway on ext4 adding features like extents of course, many other file system choices with Linux distros
- The main differences between ext2fs and FFS concern their disk allocation policies
- In ffs, the disk is allocated to files in blocks of 8Kb, with blocks being subdivided into

fragments of 1Kb to store small files or partially filled blocks at the end of a file

- ext3 does not use fragments; it performs its allocations in smaller units
- The default block size on ext3 varies as a function of total size of file system with support for 1, 2, 4 and 8 KB blocks
- ext3 uses cluster allocation policies designed to place logically adjacent blocks of a file into physically adjacent blocks on disk, so that it can submit an I/O request for several disk blocks as a single operation on a **block group**
- Maintains bit map of free blocks in a block group, searches for free byte to allocate at least 8 blocks at a time

5.1.8 Input and Output

Q8. Describe Linux Input and Output system.

Ans :

- The Linux device-oriented file system accesses disk storage through two caches:
 - Data is cached in the page cache, which is unified with the virtual memory system
 - Metadata is cached in the buffer cache, a separate cache indexed by the physical disk block
- Linux splits all devices into three classes:
 - **block devices** allow random access to completely independent, fixed size blocks of data
 - **character devices** include most other devices; they don't need to support the functionality of regular files
 - **network devices** are interfaced via the kernel's networking subsystem

Block Devices

- Provide the main interface to all disk devices in a system
- The block buffer cache serves two main purposes:

- it acts as a pool of buffers for active I/O
- it serves as a cache for completed I/O

- The **request manager** manages the reading and writing of buffer contents to and from a block device driver

- Kernel 2.6 introduced **Completely Fair Queueing(CFQ)**

- Now the default scheduler
- Fundamentally different from elevator algorithms
- Maintains set of lists, one for each process by default
- Uses C-SCAN algorithm, with round robin between all outstanding I/O from all processes
- Four blocks from each process put on at once

5.1.9 Inter Process Communication

Q9. Explain about the inter process communication in LINUX.

Ans :

(Imp.)

- Like UNIX, Linux informs processes that an event has occurred via **signals**
- There is a limited number of signals, and they cannot carry information: Only the fact that a signal occurred is available to a process
- The Linux kernel does not use signals to communicate with processes which are running in kernel mode, rather, communication within the kernel is accomplished via scheduling states and **wait_queue** structures
- Also implements System V Unix semaphores
 - Process can wait for a signal or a semaphore
 - Semaphores scale better
 - Operations on multiple semaphores can be atomic Passing Data Between Processes

- The **pipe** mechanism allows a child process to inherit a communication channel to its parent, data written to one end of the pipe can be read at the other
- Shared memory offers an extremely fast way of communicating; any data written by one process to a shared memory region can be read immediately by any other process that has mapped that region into its address space
- To obtain synchronization, however, shared memory must be used in conjunction with another Interprocess-communication mechanism

Q10. Discuss the Network Structure of LINUX

Ans :

- Networking is a key area of functionality for Linux
 - It supports the standard Internet protocols for UNIX to UNIX communications
 - It also implements protocols native to non-UNIX operating systems, in particular, protocols used on PC networks, such as Appletalk and IPX
- Internally, networking in the Linux kernel is implemented by three layers of software:
 - The socket interface
 - Protocol drivers
 - Network device drivers
- Most important set of protocols in the Linux networking system is the internet protocol suite
 - It implements routing between different hosts anywhere on the network
 - On top of the routing protocol are built the UDP, TCP and ICMP protocols
 - Packets also pass to **firewall management** for filtering based on **firewall chains** of rules

5.2 WINDOWS 7 –

5.2.1 Design Principles

Q11. Describe briefly about Windows Architecture.

Ans :

(Imp.)

Windows Architecture Layered system of modules.

Protected mode — hardware abstraction layer (HAL) , kernel, executive. Executive includes file systems, network stack, and device drivers.

User mode — collection of subsystems, services, DLLs, and the GUI Environmental subsystems emulate different operating systems Protection subsystems provide security functions Windows services provide facilities for networking, device interfaces, background execution, and extension of the system Rich shared libraries with thousands of APIs are implemented using DLLs to allow code sharing and simplify updates A graphical user interface is built into Win32 and used by most programs that interact directly with the us.

- The Win32 environment subsystem can run 32-bit Windows applications. It contains the console as well as text window support, shutdown and hard-error handling for all other environment subsystems. It also supports Virtual DOS Machines (VDMs), which allow MS-DOS and 16-bit Windows (Win16) applications to run on Windows NT. There is a specific MS-DOS VDM that runs in its own address space and which emulates an Intel 80486 running MS-DOS 5.0. Win16 programs, however, run in a Win16 VDM. Each program, by default, runs in the same process, thus using the same address space, and the Win16 VDM gives each program its own thread on which to run. However, Windows NT does allow users to run a Win16 program in a separate Win16 VDM, which allows the program to be preemptively multitasked, as Windows NT will pre-empt the whole VDM process, which only contains one running application. The Win32 environment subsystem process (csrss.exe) also includes the window management functionality, sometimes called a "window

manager". It handles input events (such as from the keyboard and mouse), then passes messages to the applications that need to receive this input. Each application is responsible for drawing or refreshing its own windows and menus, in response to these messages.

- The OS/2 environment subsystem supports 16-bit character-based OS/2 applications and emulates OS/2 1.x, but not 32-bit or graphical OS/2 applications as used with OS/2 2.x or later, on x86 machines only.^[3] To run graphical OS/2 1.x programs, the Windows NT Add-On Subsystem for Presentation Manager must be installed.^[3] The last version of Windows NT to have an OS/2 subsystem was Windows 2000; it was removed as of Windows XP.
- The POSIX environment subsystem supports applications that are strictly written to either the POSIX.1 standard or the related ISO/IEC standards. This subsystem has been replaced by Interix, which is a part of Windows Services for UNIX.^[4] This was in turn replaced by the Windows Subsystem for Linux.

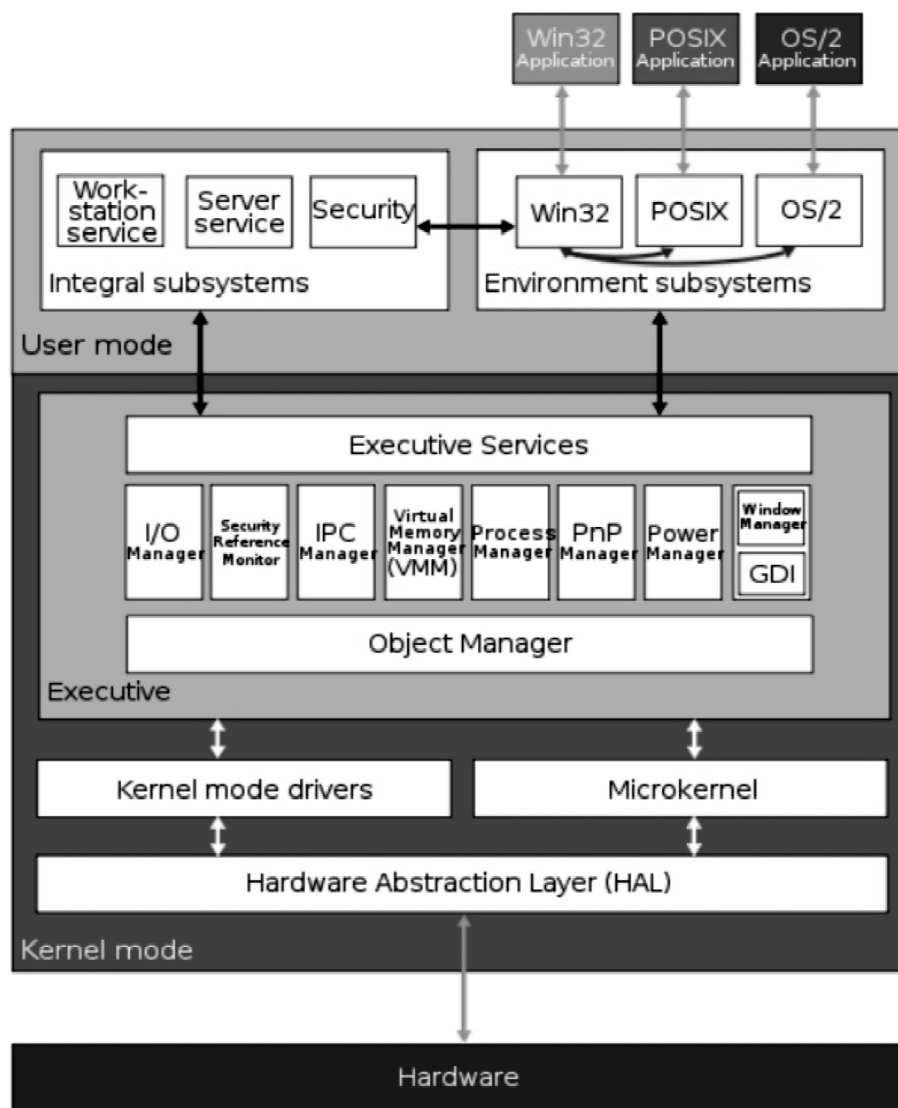


Fig.: Win 32 Architecture

Q12. Write about the design principles of windows.

Ans :

Design Principles

Design Principles Extensibility layered architecture Kernel layer runs in protected mode and provides access to the CPU by supporting threads, interrupts, and traps. Executive runs in protected mode above the Kernel layer and, provides the basic system services On top of the executive, environmental subsystems operate in user mode providing different OS APIs (as with Mach) Modular structure allows additional environmental subsystems to be added without affecting the executive Portability Windows 7 can be moved from one hardware platform to another with relatively few changes Written in C and C++ Platform-dependent code is isolated in a dynamic link library (DLL) called the "hardware abstraction layer" (HAL)

- Reliability – Windows uses hardware protection for virtual memory, and software protection mechanisms for operating system resources
- Compatibility – applications that follow the IEEE 1003.1 (POSIX) standard can be compiled to run on Windows without changing the source code. Applications created for previous versions of Windows run using various virtual machine techniques This is deprecated in Windows 8.
- Performance – Windows subsystems can communicate with one another via high-performance message passing Preemption of low priority threads enables the system to respond quickly to external events Designed for symmetrical multiprocessing,
- scaling to 100s of cores International support – supports different locales via the national language support (NLS) API, use of UNICODE throughout, and providing facilities for differences in date formats, currency, etc.

5.2.2 System Components

Q13. Write about Windows Kernel.

Ans :

System Components — Kernel

System Components — Kernel Foundation for the executive and the subsystems Never paged out of memory; execution is never preempted Four main responsibilities: thread scheduling interrupt and exception handling low-level processor synchronization recovery after a power failure Kernel is object-oriented, uses two sets of objects dispatcher objects control dispatching and synchronization (events, mutexes, semaphores, threads and timers) control objects (asynchronous procedure calls, interrupts, power notify, process and profile objects)

Kernel — Process and Threads

Kernel — Process and Threads The process has a virtual memory address space, information (such as a base priority), and an affinity for one or more processors. Threads are the unit of execution scheduled by the kernel's dispatcher. Each thread has its own state, including a priority, processor affinity, and accounting information. A thread can be one of six states: ready, standby, running, waiting, transition, and terminated.

Kernel — Scheduling

Kernel — Scheduling Windows scheduler: Pre-emptive (since Windows NT) Multilevel feedback queue The dispatcher uses a 32-level priority scheme to determine the order of thread execution. Priorities are divided into two classes The real-time class contains threads with priorities ranging from 16 to 31 The variable class contains threads having priorities from 0 to 15 Characteristics of Windows 7's priority strategy Gives very good response times to interactive threads that are using the mouse and windows Enables I/O-bound threads to keep the I/O devices busy Compute-bound threads soak up the spare CPU cycles in the background.

Scheduling can occur when a thread enters the ready or wait state, when a thread terminates, or when an application changes a thread's priority or processor affinity. Real-time threads are given preferential access to the CPU; but Windows 7 does not guarantee that a real-time thread will start to execute within any particular time limit. This is known as soft real-time.

5.2.3 Terminal Services And Fast User Switching File Systems

Q14. Write about Windows File System.

Ans : (Imp.)

File System

1. The fundamental structure of the Windows 7 file system (NTFS) is a *volume*
 - a. Created by the Windows 7 disk administrator utility
 - b. Based on a logical disk partition
 - c. May occupy a portions of a disk, an entire disk, or span across several disks
2. All *metadata*, such as information about the volume, is stored in a regular file
3. NTFS uses *clusters* as the underlying unit of disk allocation
 - a. A cluster is a number of disk sectors that is a power of two

Because the cluster size is smaller than for the 16-bit FAT file system, the amount of internal fragmentation is reduced

Internal Layout

1. NTFS uses logical cluster numbers (LCNs) as disk addresses
2. A file in NTFS is not a simple byte stream, as in MS-DOS or UNIX, rather, it is a structured object consisting of attributes
3. Every file in NTFS is described by one or more records in an array stored in a special file called the Master File Table (MFT)
4. Each file on an NTFS volume has a unique ID called a file reference.
 - a. 64-bit quantity that consists of a 48-bit file number and a 16-bit sequence number
 - b. Can be used to perform internal consistency checks
5. The NTFS name space is organized by a hierarchy of directories; the index root contains the top level of the B+ tree

Recovery

1. All file system data structure updates are performed inside transactions that are logged.
 - a. Before a data structure is altered, the transaction writes a log record that contains redo and undo information.
 - b. After the data structure has been changed, a commit record is written to the log to signify that the transaction succeeded.
2. After a crash, the file system data structures can be restored to a consistent state by processing the log records.
3. This scheme does not guarantee that all the user file data can be recovered after a crash, just that the file system data structures (the metadata files) are undamaged and reflect some consistent state prior to the crash.
4. The log is stored in the third metadata file at the beginning of the volume.

The logging functionality is provided by the Windows 7 log file service.

Security

1. Security of an NTFS volume is derived from the Windows 7 object model.
2. Each file object has a security descriptor attribute stored in this MFT record.

This attribute contains the access token of the owner of the file, and an access control list that states the access privileges that are granted to each user that has access to the file.

File System — Compression

1. To compress a file, NTFS divides the file's data into compression units, which are blocks of 16 contiguous clusters.
2. For sparse files, NTFS uses another technique to save space.
 - a. Clusters that contain all zeros are not actually allocated or stored on disk.
 - b. Instead, gaps are left in the sequence of virtual cluster numbers stored in the MFT entry for the file.

When reading a file, if a gap in the virtual cluster numbers is found, NTFS just zero-fills that portion of the caller's buffer.

File System — Reparse Points

1. A reparse point returns an error code when accessed. The reparse data tells the I/O manager what to do next.
2. Reparse points can be used to provide the functionality of UNIX mounts.

Reparse points can also be used to access files that have been moved to offline storage.

5.2.4 Networking

Q15. Explain, how networking will be done in windows 7.

Ans :

- The server message block (SMB) protocol is used to send I/O requests over the network. It has four message types:
 1. Session control
 2. File
 3. Printer
 4. Message
- The network basic Input/Output system (NetBIOS) is a hardware abstraction interface for networks
 5. Used to:
 - ▶ Establish logical names on the network
 - ▶ Establish logical connections of sessions between two logical names on the network
 - ▶ Support reliable data transfer for a session via NetBIOS requests or SMBs
- Windows 7 uses the TCP/IP Internet protocol version 4 and version 6 to connect to a wide variety of operating systems and hardware platforms.
- PPTP (Point-to-Point Tunneling Protocol) is used to communicate between Remote

Access Server modules running on Windows 7 machines that are connected over the Internet.

The Data Link Control protocol (DLC) is used to access IBM mainframes and HP printers that are directly connected to the network (possible on 32-bit only versions using unsigned drivers).

Distributed Processing Mechanisms

- Windows 7 supports distributed applications via named NetBIOS, named pipes and mailslots, Windows Sockets, Remote Procedure Calls (RPC), and Network Dynamic Data Exchange (NetDDE).
- NetBIOS applications can communicate over the network using TCP/IP.
- Named pipes are connection-oriented messaging mechanism that are named via the uniform naming convention (UNC).
- Mailslots are a connectionless messaging mechanism that are used for broadcast applications, such as for finding components on the network.
- Winsock, the windows sockets API, is a session-layer interface that provides a standardized interface to many transport protocols that may have different addressing schemes.
- The Windows 7 RPC mechanism follows the widely-used Distributed Computing Environment standard for RPC messages, so programs written to use Windows 7 RPCs are very portable.
 - RPC messages are sent using NetBIOS, or Winsock on TCP/IP networks, or named pipes on LAN Manager networks.
- Windows 7 provides the Microsoft Interface Definition Language to describe the remote procedure names, arguments, and results.

Redirectors and Servers

- In Windows 7, an application can use the Windows 7 I/O API to access files from a remote computer as if they were local, provided that the remote computer is running an MS-NET server.

- A redirector is the client-side object that forwards I/O requests to remote files, where they are satisfied by a server.

For performance and security, the redirectors and servers run in kernel mode

Access to a Remote File

- The application calls the I/O manager to request that a file be opened (we assume that the file name is in the standard UNC format).
- The I/O manager builds an I/O request packet.
- The I/O manager recognizes that the access is for a remote file, and calls a driver called a Multiple Universal Naming Convention Provider (MUP).
- The MUP sends the I/O request packet asynchronously to all registered redirectors.
- A redirector that can satisfy the request responds to the MUP

To avoid asking all the redirectors the same question in the future, the MUP uses a cache to remember with redirector can handle this file.

- The redirector sends the network request to the remote system.
- The remote system network drivers receive the request and pass it to the server driver.
- The server driver hands the request to the proper local file system driver.
- The proper device driver is called to access the data.

The results are returned to the server driver, which sends the data back to the requesting redirector

Domains

- NT uses the concept of a domain to manage global access rights within groups.
- A domain is a group of machines running NT server that share a common security policy and user database.
- Windows 7 provides three models of setting up trust relationships

- One way, A trusts B
- Two way, transitive, A trusts B, B trusts C so A, B, C trust each other

Crosslink – allows authentication to bypass hierarchy to cut down on authentication traffic.

5.2.5 Programmer Interface

Q16. Write about programmer interface of Windows 7.

Ans : (Imp.)

Programmer Interface - Access to Kernel Obj.

- A process gains access to a kernel object named XXX by calling the CreateXXX function to open a handle to XXX; the handle is unique to that process.
- A handle can be closed by calling the CloseHandle function; the system may delete the object if the count of processes using the object drops to 0.
- Windows 7 provides three ways to share objects between processes
 - A child process inherits a handle to the object
 - One process gives the object a name when it is created and the second process opens that name
 - DuplicateHandle function

Given a handle to process and the handle's value a second process can get a handle to the same object, and thus share it

Process Management

- Process is started via the CreateProcess routine which loads any dynamic link libraries that are used by the process, and creates a primary thread.
- Additional threads can be created by the CreateThread function.

Every dynamic link library or executable file that is loaded into the address space of a process is identified by an instance handle.

Scheduling in Win32 utilizes four priority classes:

1. IDLE_PRIORITY_CLASS (priority level 4)
2. NORMAL_PRIORITY_CLASS (level 8 — typical for most processes)
3. HIGH_PRIORITY_CLASS (level 13)
4. REALTIME_PRIORITY_CLASS (level 24)
 - To provide performance levels needed for interactive programs, 7 has a special scheduling rule for processes in the NORMAL_PRIORITY_CLASS
5. 7 distinguishes between the foreground process that is currently selected on the screen, and the background processes that are not currently selected.

When a process moves into the foreground, 7 increases the scheduling quantum by some factor, typically 3.

Programmer Interface - Interprocess Communication

- Win32 applications can have interprocess communication by sharing kernel objects.
- An alternate means of interprocess communications is message passing, which is particularly popular for Windows GUI applications
- One thread sends a message to another thread or to a window.
- A thread can also send data with the message.
- Every Win32 thread has its own input queue from which the thread receives messages.

This is more reliable than the shared input queue of 16-bit windows, because with separate queues, one stuck application cannot block input to the other applications

Programmer Interface — Memory Management

- Virtual memory:
 - VirtualAlloc reserves or commits virtual memory
 - VirtualFree decommits or releases the memory
 - These functions enable the application to determine the virtual address at which the memory is allocated
- An application can use memory by memory mapping a file into its address space
 - Multistage process
- Two processes share memory by mapping the same file into their virtual memory
- A heap in the Win32 environment is a region of reserved address space
 - A Win 32 process is created with a 1 MB default heap
 - Access is synchronized to protect the heap's space allocation data structures from damage by concurrent updates by multiple threads
- Because functions that rely on global or static data typically fail to work properly in a multithreaded environment, the thread-local storage mechanism allocates global storage on a per-thread basis

The mechanism provides both dynamic and static methods of creating thread-local storage.

Lab Practicals

1. Unix Shell Commands

Ans :

(a) File handling commands

1. Files Listing

To perform Files listings or to list files and directories `ls` command is used

`$ls`

```
manav@manav-MSI: /  
manav@manav-MSI: $ ls  
bin    dev    initrd.img  lib32  lost+found  opt    run    srv    tmp    vmlinuz  
boot   etc    initrd.img.old  lib64  media      proc   sbin   swapfile  usr    vmlinuz.old  
cdrom  home   lib         libx32  mnt        root   soap   sys      var  
manav@manav-MSI: $
```

All your files and directories in the current directory would be listed and each type of file would be displayed with a different color. Like in the output directories are displayed with dark blue color.

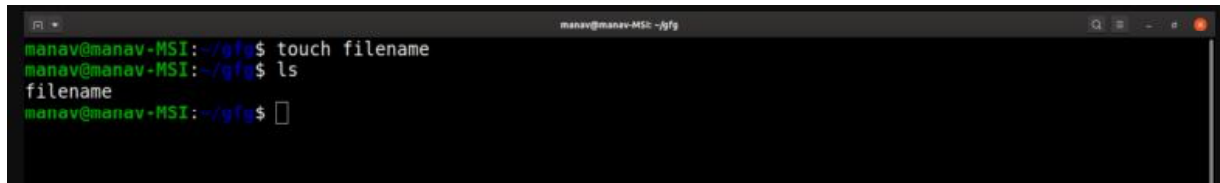
`$ls -l`

```
manav@manav-MSI: $ ls -l  
total 2097272  
drwxr-xr-x 1 root root      7 Nov 18 23:58 bin -> usr/bin  
drwxr-xr-x 4 root root    4096 Jan 24 20:13 boot  
drwxr-xr-x 2 root root    4096 Nov 18 23:59 cdrom  
drwxr-xr-x 17 root root   4420 Jan 26 2020 dev  
drwxr-xr-x 154 root root 12288 Jan 25 21:04 etc  
drwxr-xr-x 3 root root    4096 Nov 18 23:59 home  
lrwxrwxrwx 1 root root      32 Dec 4 01:33 initrd.img -> boot/initrd.img-5.0.0-37-generic  
lrwxrwxrwx 1 root root      32 Dec 4 01:33 initrd.img.old -> boot/initrd.img-5.0.0-36-generic  
lrwxrwxrwx 1 root root      7 Nov 18 23:58 lib -> usr/lib  
lrwxrwxrwx 1 root root     10 Nov 18 23:58 lib32 -> usr/lib32  
lrwxrwxrwx 1 root root     10 Nov 18 23:58 lib64 -> usr/lib64  
lrwxrwxrwx 1 root root     10 Nov 18 23:58 libx32 -> usr/libx32  
drwx----- 2 root root 16384 Nov 18 23:57 lost+found  
drwxr-xr-x 3 root root    4096 Nov 21 12:48 media  
drwxr-xr-x 2 root root    4096 Apr 17 2019 mnt  
drwxr-xr-x 3 root root    4096 Nov 21 18:12 mnt  
dr-xr-xr-x 341 root root    0 Jan 26 2020 proc  
drwx----- 10 root root    4096 Jan 22 10:51 root  
drwxr-xr-x 37 root root   1080 Jan 26 13:06 run  
lrwxrwxrwx 1 root root      8 Nov 18 23:58 sbin -> usr/sbin  
drwxr-xr-x 15 root root    4096 Dec 16 19:46 soap  
drwxr-xr-x 2 root root    4096 Apr 17 2019 srv  
-rw----- 1 root root 2147483648 Nov 22 22:36 swapfile  
dr-xr-xr-x 13 root root    0 Jan 26 2020 sys  
drwxrwxrwt 23 root root   40960 Jan 26 14:20 tmp  
drwxr-xr-x 14 root root    4096 Apr 17 2019 usr  
drwxr-xr-x 15 root root    4096 Dec 26 02:59 var  
lrwxrwxrwx 1 root root      29 Dec 4 01:33 vmlinuz -> boot/vmlinuz-5.0.0-37-generic  
lrwxrwxrwx 1 root root      29 Dec 4 01:33 vmlinuz.old -> boot/vmlinuz-5.0.0-36-generic  
manav@manav-MSI: $
```

It returns the detailed listing of the files and directories in the current directory. The command gives as the owner of the file and even which file could be managed by which user or group and which user/group has the right to access or execute which file.

2. Creating Files

`touch` command can be used to create a new file. It will create and open a new blank file if the file with a filename does not exist. And in case the file already exists then the file will not be affected.

\$touch filename

```
manav@manav-MSI:~/gfg$ touch filename
manav@manav-MSI:~/gfg$ ls
filename
manav@manav-MSI:~/gfg$
```

3. Displaying File Contents

cat command can be used to display the contents of a file. This command will display the contents of the 'filename' file. And if the output is very large then we could use more or less to fit the output on the terminal screen otherwise the content of the whole file is displayed at once.

\$cat filename

```
manav@manav-MSI:~/gfg$ cat filename
This is the content of file.
manav@manav-MSI:~/gfg$
```

4. Copying a File

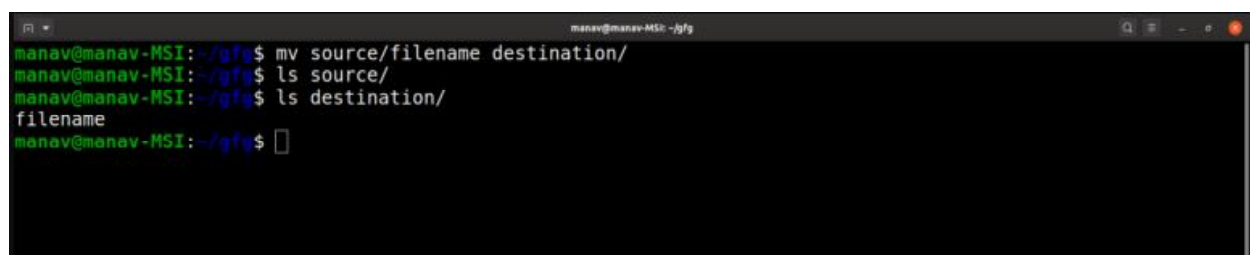
cp command could be used to create the copy of a file. It will create the new file in destination with the same name and content as that of the file 'filename'.

\$cp source/filename destination/

```
manav@manav-MSI:~/gfg$ cp source/filename destination/
manav@manav-MSI:~/gfg$ ls destination/
filename
manav@manav-MSI:~/gfg$ ls source/
filename
manav@manav-MSI:~/gfg$
```

5. Moving a File

mv command could be used to move a file from source to destination. It will remove the file filename from the source folder and would be creating a file with the same name and content in the destination folder.

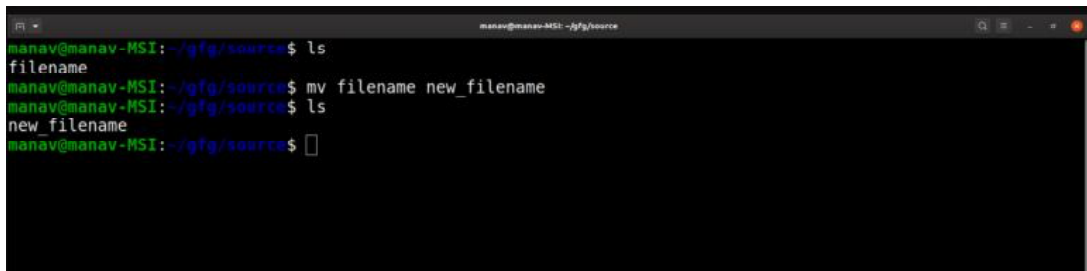
\$mv source/filename destination/

```
manav@manav-MSI:~/gfg$ mv source/filename destination/
manav@manav-MSI:~/gfg$ ls source/
manav@manav-MSI:~/gfg$ ls destination/
filename
manav@manav-MSI:~/gfg$
```

6. Renaming a File

mv command could be used to rename a file. It will rename the filename to new_filename or in other words, it will remove the filename file and would be creating a new file with the new_filename with the same content and name as that of the filename file.

\$mv filename new_filename



```
manav@manav-MSI: ~/gfg/source$ ls
filename
manav@manav-MSI: ~/gfg/source$ mv filename new_filename
manav@manav-MSI: ~/gfg/source$ ls
new_filename
manav@manav-MSI: ~/gfg/source$
```

7. Deleting a File

rm command could be used to delete a file. It will remove the filename file from the directory.

\$rm filename



```
manav@manav-MSI: ~/gfg/source$ ls
filename
manav@manav-MSI: ~/gfg/source$ rm filename
manav@manav-MSI: ~/gfg/source$ ls
manav@manav-MSI: ~/gfg/source$
```

1. **chmod**: change file access permissions

- **description:** This command is used to change the file permissions. These permissions are read, write and execute permission for the owner, group, and others.
- **syntax (symbolic mode):**
chmod [ugoa][[+ -=][mode]] file
- The first optional parameter indicates who – this can be (u)ser, (g)roup, (o)thers or (a)ll
- The second optional parameter indicates opcode – this can be for adding (+), removing (-) or assigning (=) permission.
- The third optional parameter indicates the mode – this can be (r)ead, (w)rite, or e(x)ecute.

Example: Add write permission for user, group, and others for file1

```
$ ls -l
-rw-r--r-- 1 user staff 39 Jun 21 15:37 file1
-rw-r--r-- 1 user staff 35 Jun 21 15:32 file2
$ chmod ugo+w file1
$ ls -l
-rw-rw-r-- 1 user staff 39 Jun 21 15:37 file1
-rw-r--r-- 1 user staff 35 Jun 21 15:32 file2
$ chmod o-w file1
$ ls -l
-rw-rw-r-- 1 user staff 39 Jun 21 15:37 file1
-rw-r--r-- 1 user staff 35 Jun 21 15:32 file2
```

- **syntax (numeric mode):**
chmod [mode] file
- The mode is a combination of three digits – the first digit indicates the permission for the user, the second digit for the group, and the third digit for others.
- Each digit is computed by adding the associated permissions. Read permission is '4', write permission is '2' and execute permission is '1'.

- **Example:** Give read/write/execute permission to the user, read/execute permission to the group, and execute permission to others.

```
$ ls -l
-rw-r--r-- 1 user staff 39 Jun 21 15:37 file1
-rw-r--r-- 1 user staff 35 Jun 21 15:32 file2
$ chmod 777 file1
$ ls -l
-rwxrwxrwx 1 user staff 39 Jun 21 15:37 file1
-rw-r--r-- 1 user staff 35 Jun 21 15:32 file2
```

2. **chown:** change ownership of the file.

- **description:** Only the owner of the file has the right to change the file ownership.

- **syntax:** chown [owner] [file]

- **Example:** Change the owner of file1 to user2 assuming that it is currently owned by the current user

```
$ chown user2 file1
```

3. **chgrp:** change the group ownership of the file.

- **description:** Only the owner of the file has the right to change the file ownership.

- **syntax:** chgrp [group] [file]

- **Example:** Change group of file1 to group2 assuming it is currently owned by the current user.

```
$ chgrp group2 file1
```

While creating a new file, Unix sets the default file permissions. Unix uses the value stored in a variable called umask to decide the default permissions. The umask value tells Unix which of the three sets of permissions need to be disabled.

The flag consists of three octal digits, each representing the permissions masks for the user, the group, and others. The default permissions are determined by subtracting the umask value from '777' for directories and '666' for files. The default value of the umask is '022'.

4. **umask:** change default access permissions

- **description:** This command is used to set the default file permissions. These permissions are read, write and execute permission for the owner, group, and others.

- **syntax:** umask [mode]

- The mode is a combination of three digits – the first digit indicates the permission for the user, the second digit for the group, and the third digit for others.

- Each digit is computed by adding the associated permissions. Read permission is '4', write permission is '2' and execute permission is '1'.

Example: Give read/write/execute permission to the user, and no permissions to group or others. i.e. the permission for files will be 600, and for directories will be 700.

```
$ umask 077
```

- **Example:** Give read/write/execute permission to the user, read/execute permissions to group or others for directories and read-only permission to group or others for other files. i.e. the permission for files will be 644, and for directories will be 755.

```
$ umask 022
```

(b) **Directory handling commands**

```
pwd
```

This command displays the present working directory where you are currently in.

In the following example I am inside yusufshakeel directory which is inside the home directory.

```
$ pwd
/home/yusufshakeel
ls
```

This command will list the content of a directory.

In the following example we are listing the content of a directory.

```
$ ls
happy helloworld.txt super
ls -la
```

This command will list all the content of a directory including the hidden files and directories.

In the following example we are listing all the content of a directory.

```
$ ls -la
total 0
drwxr-xr-x 5 yusufshakeel yusufshakeel 160 Sep 6 02:53 .
drwx-----+ 8 yusufshakeel yusufshakeel 256 Sep 6 02:53 ..
drwxr-xr-x 2 yusufshakeel yusufshakeel 64 Sep 6 02:53 happy
-rw-r--r-- 1 yusufshakeel yusufshakeel 0 Sep 6 02:53 helloworld.txt
```

```
drwxr-xr-x 2 yusufshakeel yusufshakeel 64 Sep
6 02:53 super
```

```
mkdir
```

This command will create a new directory, provided it doesn't exist.

In the following example we are creating a new directory `example`.

```
$ mkdir example
```

```
mkdir -p
```

This command will create nested directories.

In the following example we are creating `world` directory which is inside the `hello` directory which is inside the `example` directory.

```
$ mkdir -p example/hello/world
```

```
rmdir
```

This command will remove/delete an existing directory, provided it is empty.

In the following example we are removing/deleting an existing directory `example`.

```
$ rmdir example
```

```
cd
```

This command is used to change directory.

In the following command we are moving to root directory.

```
$ cd /
```

In the following command we are moving to `/var/www/html` directory.

```
$ cd /var/www/html
```

```
cd ..
```

This command will take us one level up the directory tree.

```
$ cd ..
```

Example: If we are inside `world` directory which is inside the `hello` directory i.e. `/hello/world` then, `cd ..` will take us one level up to the `hello` directory.

(c) General purpose commands

1) **cal:** Displays the calendar.

➤ Syntax: `cal [[month] year]`

➤ Example: display the calendar for April 2018

➤ `$ cal 4 2018`

2) **date:** Displays the system date and time.

➤ Syntax: `date [+format]`

➤ Example: Display the date in `dd/mm/yy` format

➤ `$ date +%d/%m/%y`

3) **banner:** Prints a large banner on the standard output.

➤ Syntax: `banner message`

➤ Example: Print "Unix" as the banner

➤ `$ banner Unix`

4) **who:** Displays the list of users currently logged in

➤ Syntax: `who [option] ... [file][arg1]`

➤ Example: List all currently logged-in users

➤ `$ who`

5) **whoami:** Displays the user id of the currently logged-in user.

➤ Syntax: `whoami [option]`

➤ Example: List currently logged-in user

➤ `$ whoami`

6) **ls:** List directory contents

➤ Syntax: `ls [OPTION] [FILE]`

➤ Example: list all (including hidden files) directory contents, in long format, sorted by time,

➤ `$ ls -alt`

7) **which:** Locate a command

➤ Syntax: `which [-a] filename`

➤ Example: List all paths from where 'cat' can run

➤ `$ which -a cat`

8) **man:** Interface for working with the online reference manuals.

➤ Syntax: `man [-s section] item`

➤ Example: Show the manual page for the 'cat' command

➤ `$ man cat`

9) **su:** Change user-id or become super-user.

➤ Syntax: `su [options] [username]`

➤ Example: Change user-id to 'user1' (if it exists)

➤ `$ su user1`

10) **sudo:** Execute a command as some other user or super-user

➤ Syntax: `sudo [options] [command]`

➤ Example: Get a file listing of an unlisted directory

➤ `$ sudo ls /usr/local/protected`

11) **find:** Used to search for files and directories as mentioned in the 'expression'

➤ **Syntax:** `find [starting-point] [expression]`

➤ **Example:** In '/usr' folder, find character device files, of the name 'backup'

- `$ find /usr -type c -name backup`
- 12) **du:** Estimate disk usage in blocks
- Syntax: `du [options] [file]`
- Example: Show the number of blocks occupied by files in the current directory
- `$ du`
- 13) **df:** Show the number of free blocks for the mounted file system
- Syntax: `df [options] [file]`
- Example: Show the number of free blocks in local file systems
- `$ df -l`

2. Unix Shell Scripts

Ans :

- (a) Print Multiplication table of a given no. using all loops
- ```
echo "Enter Number to Generate Multiplication Table"
read -p "Enter the number : " number
echo "*****"
i=1
while [$i -le 10]
do
echo " $number * $i = `expr $number * $i` "
i=`expr $i + 1`
done
echo "*****"
```

**Input: 5**

**Output:** 5 \* 1 = 5

```
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

- (b) **Perform all arithmetic operations**

```
clear
sum=0
i="y"
echo " Enter one no."
```

```
read n1
echo "Enter second no."
read n2
while [$i = "y"]
do
echo "1.Addition"
echo "2.Subtraction"
echo "3.Multiplication"
echo "4.Division"
echo "Enter your choice"
read ch
case $ch in
```

- 1) `sum=`expr $n1 + $n2``  
`echo "Sum = "$sum;;`
  - 2) `sum=`expr $n1 - $n2``  
`echo "Sub = "$sum;;`
  - 3) `sum=`expr $n1 \* $n2``  
`echo "Mul = "$sum;;`
  - 4) `sum=`expr $n1 / $n2``  
`echo "Div = "$sum;;`  
`*)echo "Invalid choice";;`
- ```
esac
echo "Do u want to continue?"
read i
if [ $i != "y" ]
then
exit
fi
done
```

Output

```
*****
[04mca58@LINTEL 04mca58]$ sh calculator.sh
Enter any no.
121
Enter one no.
21
Enter second no.
58
1. Addition
2. Subtraction
3. Multiplication
4. Division
```

```
Enter your choice
1
Sum = 79
Do u want to continue ?
y
1. Addition
2. Subtraction
3. Multiplication
4. Division
Enter your choice
2
Sub = -37
Do u want to continue ?
y
1. Addition
2. Subtraction
3. Multiplication
4. Division
Enter your choice
3
Mul = 1218
Do u want to continue ?
y
1. Addition
2. Subtraction
3. Multiplication
4. Division
Enter your choice
4
Div = 0
Do u want to continue ?
n
```

(c) Print the type of a file

```
#!/bin/bash
read -p "Enter file name : " filename
while read line
do
echo $line
done < $filename
```

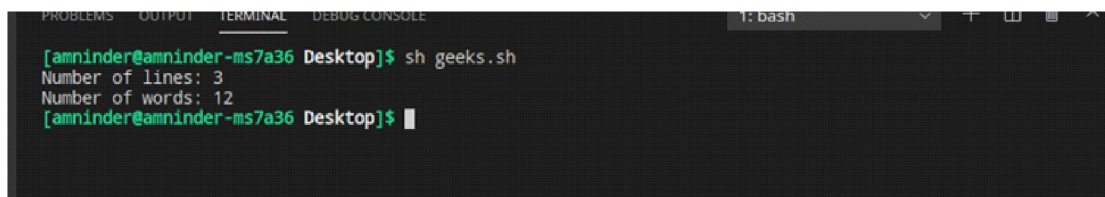
```
bahar56@bahar56-VirtualBox:~/Desktop$ chmod +x read_file.sh
bahar56@bahar56-VirtualBox:~/Desktop$ ./read_file.sh
Enter file name : geeks.txt
With so much being published on education right now,
from tweets to blogs,academic journals to practical handbooks for teaching,
the pressure to keep up can be immense.
bahar56@bahar56-VirtualBox:~/Desktop$
```

(d) Rename all files whose names end with .c as .old

```
#!/bin/bash
for f in *.txt;
do mv — "$f" "${f%.c}.old"
done
```

(e) Display the no. of lines in each of text file in a given dir

```
#!/usr/bin/bash
# path to the file
file_path="/home/amninder/Desktop/demo.txt"
# using wc command to count number of lines
number_of_lines='wc —lines < $file_path'
# using wc command to count number of words
number_of_words='wc —word < $file_path'
# Displaying number of lines and number of words
echo "Number of lines: $number_of_lines"
echo "Number of words: $number_of_words"
```

Output:


```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
1: bash
[amninder@amninder-ms7a36 Desktop]$ sh geeks.sh
Number of lines: 3
Number of words: 12
[amninder@amninder-ms7a36 Desktop]$
```

3. Simulate the following CPU scheduling algorithms.*Ans :***(a) FCFS**

```
include <stdio.h>
int main()
{
    int n, bt[20], wt[20], tat[20], avwt=0, avtat=0, i, j;
    printf("Enter total number of processes(maximum 20):");
    scanf("%d", &n);
    printf("\nEnter Process Burst Time\n");
    for(i=0; i<n; i++)
    {
        printf("P[%d]:", i+1);
        scanf("%d", &bt[i]);
    }
    wt[0]=0; //waiting time for first process is 0
    //calculating waiting time
    for(i=1; i<n; i++)
    {
```

```

        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i] += bt[j];
    }
    printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time");
    //calculating turnaround time
    for(i=0;i<n;i++)
    {
        Video Player is loading.
        Pause
        Unmute
        Loaded: 2.91%
        Remaining Time -3:20
        Auto(360p)
        ShareFullscreen
        FIFO Coding in STL - first in first out PLC Logic - Siemens Tia Portal
        tat[i]=bt[i]+wt[i];
        avwt += wt[i];
        avtat += tat[i];
        printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);
    }
    avwt/=i;
    avtat/=i;
    printf("\n\nAverage Waiting Time:%d",avwt);
    printf("\n\nAverage Turnaround Time:%d",avtat);
    return 0;
}

```

Output:

```

C:\Users\admin\Desktop\Untitled1.exe
Enter total number of processes(maximum 20):3
Enter Process Burst Time
P[1]:24
P[2]:3
P[3]:3

Process      Burst Time      Waiting Time      Turnaround Time
P[1]         24              0                24
P[2]         3              24              27
P[3]         3              27              30

Average Waiting Time:17
Average Turnaround Time:27
Process returned 0 (0x0)   execution time : 7.661 s
Press any key to continue.

```

(b) SJF

```
#include <stdio.h>
void main()
{
    int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
    float avg_wt,avg_tat;
    printf("Enter number of process:");
    scanf("%d",&n);
    printf("\nEnter Burst Time:\n");
    for(i=0;i<n;i++)
    {
        printf("p%d:",i+1);
        scanf("%d",&bt
[i]);
        p[i]=i+1; //contains process number
    }
    //sorting burst time in ascending order using selection sort
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(bt[j]<bt[pos])
                pos=j;
        }
        temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;
        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
    }
    wt[0]=0; //waiting time for first process will be zero
    //calculate waiting time
    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];
        total+=wt[i];
    }
    avg_wt=(float)total/n; //average waiting time
    total=0;
    printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i]; //calculate turnaround time
```

```

        total += tat[i];
        printf("\np%d\t\t %d\t\t %d\t\t\t %d", p[i], bt[i], wt[i], tat[i]);
    }
    avg_tat = (float)total/n; //average turnaround time
    printf("\n\nAverage Waiting Time = %f", avg_wt);
    printf("\n\nAverage Turnaround Time = %f\n", avg_tat);
}

```

```

C:\Users\admin\Desktop\Untitled1.exe
Enter number of process:4
Enter Burst Time:
p1:4
p2:8
p3:3
p4:7
Process      Burst Time      Waiting Time      Turnaround Time
p3           3              0                3
p1           4              3                7
p4           7              7               14
p2           8             14               22
Average Waiting Time=6.000000
Average Turnaround Time=11.500000
Process returned 35 (0x23)   execution time : 5.567 s
Press any key to continue.

```

(c) Round Robin

```

#include <stdio.h>
int main()
{
    int count, j, n, time, remain, flag = 0, time_quantum;
    int wait_time = 0, turnaround_time = 0, at[10], bt[10], rt[10];
    printf("Enter Total Process:\t ");
    scanf("%d", &n);
    remain = n;
    for(count = 0; count < n; count++)
    {
        printf("Enter Arrival Time and Burst Time for Process Process Number %d :", count + 1);
        scanf("%d", &at[count]);
        scanf("%d", &bt[count]);
        rt[count] = bt[count];
    }
    printf("Enter Time Quantum:\t");
    scanf("%d", &time_quantum);
    printf("\n\nProcess\t | Turnaround Time | Waiting Time\n\n");
    for(time = 0, count = 0; remain != 0;)
    {
        if(rt[count] <= time_quantum && rt[count] > 0)

```

```

{
    time += rt[count];
    rt[count] = 0;
    flag = 1;
}
else if(rt[count] > 0)
{
    rt[count] -= time_quantum;
    time += time_quantum;
}
if(rt[count] == 0 && flag == 1)
{
    remain--;
    printf("P[%d]\t|\t%d\t|\t%d\n", count+1, time-at[count], time-at[count]-bt[count]);
    wait_time += time-at[count]-bt[count];
    turnaround_time += time-at[count];
    flag = 0;
}
if(count == n-1)
    count = 0;
else if(at[count+1] <= time)
    count++;
else
    count = 0;
}
printf("\nAverage Waiting Time = %f\n", wait_time*1.0/n);
printf("Avg Turnaround Time = %f", turnaround_time*1.0/n);
return 0;
}

```

```

tusharsoni@tusharsoni-Lenovo-G50-70: ~/Desktop
Average Waiting Time= 5.250000
Avg Turnaround Time = 9.500000tusharsoni@tusharsoni-Lenovo-G50-70:~/Desktop$ ./a
.out
Enter Total Process:      4
Enter Arrival Time and Burst Time for Process Process Number 1 :0
9
Enter Arrival Time and Burst Time for Process Process Number 2 :1
5
Enter Arrival Time and Burst Time for Process Process Number 3 :2
3
Enter Arrival Time and Burst Time for Process Process Number 4 :3
4
Enter Time Quantum:      5

Process |Turnaround time|waiting time
P[2]    |      9      |      4
P[3]    |     11      |      8
P[4]    |     14      |     10
P[1]    |     21      |     12

Average Waiting Time= 8.500000
Avg Turnaround Time = 13.750000tusharsoni@tusharsoni-Lenovo-G50-70:~/Desktop$

```

```
(d) Priority.
    # include <stdio.h>
    int main()
{
    int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;
    printf("Enter Total Number of Process:");
    scanf("%d",&n);
    printf("\nEnter Burst Time and Priority\n");
    for(i=0;i<n;i++)
    {
        printf("\nP[%d]\n",i+1);
        printf("Burst Time:");
        scanf("%d",&bt[i]);
        printf("Priority:");
        scanf("%d",&pr[i]);
        p[i]=i+1; //contains process number
    }
    //sorting burst time, priority and process number in ascending order using selection sort
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(pr[j]<pr[pos])
                pos=j;
        }
        temp=pr[i];
        pr[i]=pr[pos];
        pr[pos]=temp;
        temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;
        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
    }
    wt[0]=0; //waiting time for first process is zero
    //calculate waiting time
    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
```



```

        wt[i] += bt[j];
        total += wt[i];
    }
    avg_wt = total/n; //average waiting time
    total = 0;
    printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");
    for(i=0; i<n; i++)
    {
        tat[i] = bt[i] + wt[i]; //calculate turnaround time
        total += tat[i];
        printf("\nP[%d]\t\t %d\t\t %d\t\t\t%d", p[i], bt[i], wt[i], tat[i]);
    }
    avg_tat = total/n; //average turnaround time
    printf("\n\nAverage Waiting Time=%d", avg_wt);
    printf("\nAverage Turnaround Time=%d\n", avg_tat);
    return 0;
}

```

The screenshot shows a Windows command prompt window titled "C:\Users\admin\Desktop\Untitled1.exe". The program prompts the user to "Enter Total Number of Process:4" and "Enter Burst Time and Priority". It then lists four processes: P[1] (Burst Time:6, Priority:3), P[2] (Burst Time:2, Priority:2), P[3] (Burst Time:14, Priority:1), and P[4] (Burst Time:6, Priority:4). Below this, a table displays the scheduling results:

Process	Burst Time	Waiting Time	Turnaround Time
P[3]	14	0	14
P[2]	2	14	16
P[1]	6	16	22
P[4]	6	22	28

At the bottom, the program outputs "Average Waiting Time=13" and "Average Turnaround Time=20".

4. Write a C program to simulate producer-consumer problem using Semaphores

Ans :

```

#include <stdio.h>
int mutex=1,full=0,empty=3,x=0;
main()
{
    int n;
    void producer();
}

```

```

void consumer();
int wait(int);
int signal(int);
printf("\n 1.Producer \n 2.Consumer \n 3.Exit");
while(1)
{
    printf("\n Enter your choice:");
    scanf("%d",&n);
    switch(n)
    {
        case 1:
            if((mutex==1)&&(empty!=0))
                producer();
            else
                printf("Buffer is full");
            break;
        case 2:
            if((mutex==1)&&(full!=0))
                consumer();
            else
                printf("Buffer is empty");
            break;
        case 3:
            exit(0);
            break;
    }
}
int wait(int s)
{
    return (--s);
}
int signal(int s)
{
    return(++s);
}
void producer()
{
    mutex=wait(mutex);
    full=signal(full);
    empty=wait(empty);
    x++;

```

```

printf("\n Producer produces the item %d",x);
mutex=signal(mutex);
}
void consumer()
{
    mutex=wait(mutex);
    full=wait(full);
    empty=signal(empty);
    printf("\n Consumer consumes item %d",x);
    x--;
    mutex=signal(mutex);
}

```

Output:

[examuser35@localhost Jebastin]\$ cc pc.c

```

1.Producer
2.Consumer
3.Exit
Enter your choice:1
Producer produces the item 1
Enter your choice:1
Producer produces the item 2
Enter your choice:1
Producer produces the item 3
Enter your choice:1
Buffer is full
Enter your choice:2
Consumer consumes item 3
Enter your choice:2
Consumer consumes item 2
Enter your choice:2
Consumer consumes item 1
Enter your choice: 2
Buffer is empty
Enter your choice: 3

```

5. Write a C program to simulate the concept of Dining-philosophers problem.

Ans :

```

#include<stdio.h>
#define n 4
int completedPhilo = 0,i;
struct fork{
    int taken;

```

```

    }ForkAvil[n];
    struct philosp{
    int left;
    int right;
    }Philostatus[n];
    void goForDinner(int phillID){ //same like threads concept here cases implemented
    if(Philostatus[phillID].left == 10 && Philostatus[phillID].right == 10)
        printf("Philosopher %d completed his dinner\n",phillID+1);
    //if already completed dinner
    else if(Philostatus[phillID].left == 1 && Philostatus[phillID].right == 1){
        //if just taken two forks
        printf("Philosopher %d completed his dinner\n",phillID+1);
        Philostatus[phillID].left = Philostatus[phillID].right = 10; //remembering that he completed dinner by
        assigning value 10
        int otherFork = phillID-1;
        if(otherFork == -1)
            otherFork=(n-1);
        ForkAvil[phillID].taken = ForkAvil[otherFork].taken = 0; //releasing forks
        printf("Philosopher %d released fork %d and fork %d\n",phillID+1,phillID+1,otherFork+1);
        compltdPhilo++;
    }
    else if(Philostatus[phillID].left == 1 && Philostatus[phillID].right == 0){ //left already taken, trying for right
    fork
        if(phillID==(n-1)){
            if(ForkAvil[phillID].taken == 0){ //KEY POINT OF THIS PROBLEM, THAT LAST PHILOSOPHER
            TRYING IN reverse DIRECTION
                ForkAvil[phillID].taken = Philostatus[phillID].right = 1;
                printf("Fork %d taken by philosopher %d\n",phillID+1,phillID+1);
            }else{
                printf("Philosopher %d is waiting for fork %d\n",phillID+1,phillID+1);
            }
        }else{ //except last philosopher case
            int dupphillID = phillID;
            phillID-=1;
            if(phillID == -1)
                phillID=(n-1);
            if(ForkAvil[phillID].taken == 0){
                ForkAvil[phillID].taken = Philostatus[dupphillID].right = 1;
                printf("Fork %d taken by Philosopher %d\n",phillID+1,dupphillID+1);
            }else{
                printf("Philosopher %d is waiting for Fork %d\n",dupphillID+1,phillID+1);
            }
        }
    }
}

```

```

    }
    else if(Philostatus[phillID].left==0){ //nothing taken yet
        if(phillID==(n-1)){
            if(ForkAvil[phillID-1].taken==0){ //KEY POINT OF THIS PROBLEM, THAT LAST PHILOSOPHER
TRYING IN reverse DIRECTION
                ForkAvil[phillID-1].taken = Philostatus[phillID].left = 1;
                printf("Fork %d taken by philosopher %d\n",phillID,phillID+1);
            }else{
                printf("Philosopher %d is waiting for fork %d\n",phillID+1,phillID);
            }
        }else{ //except last philosopher case
            if(ForkAvil[phillID].taken == 0){
                ForkAvil[phillID].taken = Philostatus[phillID].left = 1;
                printf("Fork %d taken by Philosopher %d\n",phillID+1,phillID+1);
            }else{
                printf("Philosopher %d is waiting for Fork %d\n",phillID+1,phillID+1);
            }
        }
    }else{}
}
int main(){
for(i=0;i<n;i++)
    ForkAvil[i].taken=Philostatus[i].left=Philostatus[i].right=0;
while(compltedPhilo<n){
/* Observe here carefully, while loop will run until all philosophers complete dinner
Actually problem of deadlock occur only thy try to take at same time
This for loop will say that they are trying at same time. And remaining status will print by go for dinner
function
*/
for(i=0;i<n;i++)
    goForDinner(i);
printf("\nTill now num of philosophers completed dinner are %d\n\n",compltedPhilo);
}
return 0;
}

#include<stdio.h>
#define n 4
int compltedPhilo = 0,i;
struct fork{
int taken;
}ForkAvil[n];
struct philosp{
int left;

```

```

int right;
}Philostatus[n];
void goForDinner(int phillID){ //same like threads concept here cases implemented
if(Philostatus[phillID].left == 10 && Philostatus[phillID].right == 10)
    printf("Philosopher %d completed his dinner\n", phillID+1);
//if already completed dinner
else if(Philostatus[phillID].left == 1 && Philostatus[phillID].right == 1){
    //if just taken two forks
    printf("Philosopher %d completed his dinner\n", phillID+1);
    Philostatus[phillID].left = Philostatus[phillID].right = 10; //remembering that he completed dinner by assigning
    value 10

    int otherFork = phillID-1;
    if(otherFork == -1)
        otherFork = (n-1);
    ForkAvil[phillID].taken = ForkAvil[otherFork].taken = 0; //releasing forks
    printf("Philosopher %d released fork %d and fork %d\n", phillID+1, phillID+1, otherFork+1);
    compltedPhilo++;
}
else if(Philostatus[phillID].left == 1 && Philostatus[phillID].right == 0){ //left already taken, trying for right fork
    if(phillID == (n-1)){
        if(ForkAvil[phillID].taken == 0){ //KEY POINT OF THIS PROBLEM, THAT LAST PHILOSOPHER TRYING
        IN reverse DIRECTION
            ForkAvil[phillID].taken = Philostatus[phillID].right = 1;
            printf("Fork %d taken by philosopher %d\n", phillID+1, phillID+1);
        }else{
            printf("Philosopher %d is waiting for fork %d\n", phillID+1, phillID+1);
        }
    }else{ //except last philosopher case
        int dupphillID = phillID;
        phillID--;
        if(phillID == -1)
            phillID = (n-1);
        if(ForkAvil[phillID].taken == 0){
            ForkAvil[phillID].taken = Philostatus[dupphillID].right = 1;
            printf("Fork %d taken by Philosopher %d\n", phillID+1, dupphillID+1);
        }else{
            printf("Philosopher %d is waiting for Fork %d\n", dupphillID+1, phillID+1);
        }
    }
}
else if(Philostatus[phillID].left == 0){ //nothing taken yet
    if(phillID == (n-1)){
        if(ForkAvil[phillID-1].taken == 0){ //KEY POINT OF THIS PROBLEM, THAT LAST PHILOSOPHER
        TRYING IN reverse DIRECTION

```

```

ForkAvil[philID-1].taken = PhiloStatus[philID].left = 1;
printf("Fork %d taken by philosopher %d\n",philID,philID + 1);
    }else{
        printf("Philosopher %d is waiting for fork %d\n",philID+1,philID);
    }
    }else{ //except last philosopher case
        if(ForkAvil[philID].taken == 0){
            ForkAvil[philID].taken = PhiloStatus[philID].left = 1;
            printf("Fork %d taken by Philosopher %d\n",philID+1,philID+1);
        }else{
            printf("Philosopher %d is waiting for Fork %d\n",philID+1,philID+1);
        }
    }
}
}
int main(){
for(i=0;i<n;i++)
    ForkAvil[i].taken=PhiloStatus[i].left=PhiloStatus[i].right=0;
while(compltedPhilo<n){
/* Observe here carefully, while loop will run until all philosophers complete dinner
Actually problem of deadlock occur only thy try to take at same time
This for loop will say that they are trying at same time. And remaining status will print by go for dinner function
*/
for(i=0;i<n;i++)
    goForDinner(i);
printf("\nTill now num of philosophers completed dinner are %d\n\n",compltedPhilo);
}
return 0;
}

```

Till now num of philosophers completed dinner are 0
 Fork 4 taken by Philosopher 1
 Philosopher 2 is waiting for Fork 1
 Philosopher 3 is waiting for Fork 2
 Philosopher 4 is waiting for fork 3
 Till now num of philosophers completed dinner are 0
 Philosopher 1 completed his dinner
 Philosopher 1 released fork 1 and fork 4
 Fork 1 taken by Philosopher 2
 Philosopher 3 is waiting for Fork 2
 Philosopher 4 is waiting for fork 3
 Till now num of philosophers completed dinner are 1
 Philosopher 1 completed his dinner
 Philosopher 2 completed his dinner
 Philosopher 2 released fork 2 and fork 1
 Fork 2 taken by Philosopher 3
 Philosopher 4 is waiting for fork 3

6. Simulate MVT and MFT.*Ans .:*

```

#include<stdio.h>
#include<conio.h>
main()
{
    int ms, bs, nob, ef,n, mp[10],tif=0;
    int i,p=0;
    clrscr();
    printf("Enter the total memory available (in Bytes) — ");
    scanf("%d",&ms);
    printf("Enter the block size (in Bytes) — ");
    scanf("%d", &bs);
    nob=ms/bs;
    ef=ms - nob*bs;
    printf("\nEnter the number of processes — ");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("Enter memory required for process %d (in Bytes)— ",i+1);
        scanf("%d",&mp[i]);
    }

    printf("\nNo. of Blocks available in memory — %d",nob);
    printf("\n\nPROCESS\tMEMORY REQUIRED\tALLOCATED\tINTERNAL\nFRAGMENTATION");
    for(i=0;i<n && p<nob;i++)
    {
        printf("\n %d\t\t%d",i+1,mp[i]);
        if(mp[i] > bs)
            printf("\t\tNO\t\t—");
        else
        {
            printf("\t\tYES\t\t%d",bs-mp[i]);
            tif = tif + bs-mp[i];
            p++;
        }
    }
}

```

```

int ms, bs, nob, ef, n, mp[10], tif = 0;
int i, p = 0;
clrscr();
printf("Enter the total memory available (in Bytes) — ");
scanf("%d", &ms);
printf("Enter the block size (in Bytes) — ");
scanf("%d", &bs);
nob = ms / bs;
ef = ms - nob * bs;
printf("\nEnter the number of processes — ");
scanf("%d", &n);
for(i = 0; i < n; i++)
{
    printf("Enter memory required for process %d (in Bytes) — ", i + 1);
    scanf("%d", &mp[i]);
}

printf("\nNo. of Blocks available in memory — %d", nob);
printf("\n\nPROCESS\tMEMORY REQUIRED\tALLOCATED\tINTERNAL\nFRAGMENTATION");
for(i = 0; i < n && p < nob; i++)
{
    printf("\n %d\t\t%d", i + 1, mp[i]);
    if(mp[i] > bs)
        printf("\t\tNO\t\t—");
    else
    {
        printf("\t\tYES\t\t%d", bs - mp[i]);
        tif = tif + bs - mp[i];
        p++;
    }
}

if(i < n)
PROGRAM
MFT MEMORY MANAGEMENT TECHNIQUE
#include <stdio.h>
#include <conio.h>
main()
{
    int ms, bs, nob, ef, n, mp[10], tif = 0;
    int i, p = 0;
    clrscr();
    printf("Enter the total memory available (in Bytes) — ");

```



```

scanf("%d",&ms);
printf("Enter the block size (in Bytes) — ");
scanf("%d", &bs);
nob=ms/bs;
ef=ms - nob*bs;
printf("\nEnter the number of processes — ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
    printf("Enter memory required for process %d (in Bytes)— ",i+1);
    scanf("%d",&mp[i]);
}

printf("\nNo. of Blocks available in memory — %d",nob);
printf("\n\nPROCESS\tMEMORY REQUIRED\tALLOCATED\tINTERNAL FRAGMENTATION");
for(i=0;i<n && p<nob;i++)
{
    printf("\n %d\t\t%d",i+1,mp[i]);
    if(mp[i] > bs)
        printf("\t\tNO\t\t—");
    else
    {
        printf("\t\tYES\t\t%d",bs-mp[i]);
        tif = tif + bs-mp[i];
        p++;
    }
}

if(i<n)
printf("\nMemory is Full, Remaining Processes cannot be accomodated");
printf("\n\nTotal Internal Fragmentation is %d",tif);
printf("\nTotal External Fragmentation is %d",ef);
getch();
}

```

Input

Enter the total memory available (in Bytes) — 1000
 Enter the block size (in Bytes)— 300
 Enter the number of processes – 5
 Enter memory required for process 1 (in Bytes) — 275
 Enter memory required for process 2 (in Bytes) — 400
 Enter memory required for process 3 (in Bytes) — 290
 Enter memory required for process 4 (in Bytes) — 293
 Enter memory required for process 5 (in Bytes) — 100
 No. of Blocks available in memory — 3

Output**PROCESS MEMORY-REQUIRED ALLOCATED INTERNAL-FRAGMENTATION**

1	275	YES	25
2	400	NO	-----
3	290	YES	10
4	293	YES	7

Memory is Full, Remaining Processes cannot be accommodated

Total Internal Fragmentation is 42

Total External Fragmentation is 100

MVT MEMORY MANAGEMENT TECHNIQUE

```
#include <stdio.h>
#include <conio.h>
main()
{
    int ms, mp[10], i, temp, n = 0;
    char ch = 'y';
    clrscr();
    printf("\nEnter the total memory available (in Bytes) — ");
    scanf("%d", &ms);
    temp = ms;
    for(i = 0; ch == 'y'; i++, n++)
    {
        printf("\nEnter memory required for process %d (in Bytes) — ", i + 1);
        scanf("%d", &mp[i]);
        if(mp[i] <= temp)
        {
            printf("\nMemory is allocated for Process %d ", i + 1);
            temp = temp - mp[i];
        }
        else
        {
            printf("\nMemory is Full");
            break;
        }
        printf("\nDo you want to continue(y/n) — ");
        scanf(" %c", &ch);
    }

    printf("\n\nTotal Memory Available — %d", ms);
    printf("\n\n\t\tPROCESS\t\tMEMORY ALLOCATED ");
    for(i = 0; i < n; i++)
        printf("\n\t\t%d\t\t%d", i + 1, mp[i]);
    printf("\n\nTotal Memory Allocated is %d", ms - temp);
}
```

```
printf("\nTotal External Fragmentation is %d",temp);
getch();
```

```
}
```

Input

```
Enter the total memory available (in Bytes) — 1000
Enter memory required for process 1 (in Bytes) — 400
Memory is allocated for Process 1
Do you want to continue(y/n) — y
Enter memory required for process 2 (in Bytes) — 275
Memory is allocated for Process 2
Do you want to continue(y/n) — y
Enter memory required for process 3 (in Bytes) — 550
```

Output

```
Memory is Full
Total Memory Available — 1000
Process Memory - allocated
1      400
2      275
Total Memory Allocated is 675
Total External Fragmentation is 325
```

7. Write a C program to simulate the following contiguous memory allocation techniques.*Ans :*

(a) WORST-FIT

```
#include <stdio.h>
#include <conio.h>
#define max 25
void main()
{
    int frag[max],b[max],f[max],i,j,nb,nf,temp;
    static int bf[max],ff[max];
    clrscr();
    printf("\n\tMemory Management Scheme - First Fit");
    printf("\nEnter the number of blocks:");
    scanf("%d",&nb);
    printf("Enter the number of files:");
    scanf("%d",&nf);
    printf("\nEnter the size of the blocks:-\n");
    for(i=1;i<=nb;i++)
    {
        printf("Block %d:",i);
        scanf("%d",&b[i]);
    }
}
```

```

    printf("Enter the size of the files :-\n");
    for(i=1;i<=nf;i++)
    {
        printf("File %d:",i);
        scanf("%d",&f[i]);
    }

    for(i=1;i<=nf;i++)
    {
        for(j=1;j<=nb;j++)
        {
            if(bf[j]!=1)
            {
                temp=b[j]-f[i];
                if(temp>=0)
            {
                ff[i]=j;
                break;
            }
        }
        frag[i]=temp;
        bf[ff[i]]=1;
    }

    printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragement");
    for(i=1;i<=nf;i++)
    printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i,f[i],ff[i],b[ff[i]], frag[i]);
    getch();
}

```

Input

Enter the number of blocks: 3
Enter the number of files: 2
Enter the size of the blocks:-
Block 1: 5
Block 2: 2
Block 3: 7
Enter the size of the files:-
File 1: 1
File 2: 4

Output

File No	File Size	Block No	Block Size	Fragement
1	1	3	7	6
2	4	1	5	1

(b) Best-fit

```
#include <stdio.h>
#include <conio.h>
#define max 25
void main()
{
    int frag[max], b[max], f[max], i, j, nb, nf, temp, lowest = 10000;
    static int bf[max], ff[max];
    clrscr();
    printf("\nEnter the number of blocks:");
    scanf("%d", &nb);
    printf("Enter the number of files:");
    scanf("%d", &nf);
    printf("\nEnter the size of the blocks:-\n");
    for(i = 1; i <= nb; i++)
    {
        printf("Block %d:", i);
        scanf("%d", &b[i]);
    }
    printf("Enter the size of the files :-\n");
    for(i = 1; i <= nf; i++)
    {
        printf("File %d:", i);
        scanf("%d", &f[i]);
    }
    for(i = 1; i <= nf; i++)
    {
        for(j = 1; j <= nb; j++)
        {
            if(bf[j] != 1)
            {
                temp = b[j] - f[i];
                if(temp >= 0)
                if(lowest > temp)
                {
                    ff[i] = j;
                    lowest = temp;
                }
            }
        }
        frag[i] = lowest;
        bf[ff[i]] = 1;
        lowest = 10000;
    }
    printf("\nFile No\tFile Size\tBlock No\tBlock Size\tFragment");
    for(i = 1; i <= nf && ff[i] != 0; i++)
```

```
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
getch();
```

```
}
```

Input

Enter the number of blocks: 3

Enter the number of files: 2

Enter the size of the blocks:-

Block 1: 5

Block 2: 2

Block 3: 7

Enter the size of the files:-

File 1: 1

File 2: 4

Output

File No.	File Size	Block No	Block Size	Fragment
1	1	3	7	6
2	4	1	5	1

(c) First-fit

```
#include <stdio.h>
```

```
#include <conio.h>
```

```
#define max 25
```

```
void main()
```

```
{
```

```
int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;
```

```
static int bf[max],ff[max];
```

```
clrscr();
```

```
printf("\n\tMemory Management Scheme - Worst Fit");
```

```
printf("\nEnter the number of blocks:");
```

```
scanf("%d",&nb);
```

```
printf("Enter the number of files:");
```

```
scanf("%d",&nf);
```

```
printf("\nEnter the size of the blocks:-\n");
```

```
for(i=1;i<=nb;i++)
```

```
{
```

```
printf("Block %d:",i);
```

```
scanf("%d",&b[i]);
```

```
}
```

```
printf("Enter the size of the files :-\n");
```

```
for(i=1;i<=nf;i++)
```

```
{
```

```
printf("File %d:",i);
```

```
scanf("%d",&f[i]);
```

```
}
```

```
for(i=1;i<=nf;i++)
```

```
{
```

```
for(j=1;j<=nb;j++)
```

```

{
    if(bf[j]!=1) //if bf[j] is not allocated
    {
        temp=b[j]-f[i];
        if(temp>=0)
            if(highest<temp)
            {
                ff[i]=j;
                highest=temp;
            }
    }
    frag[i]=highest;
    bf[ff[i]]=1;
    highest=0;
}

printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
for(i=1;i<=nf;i++)
    printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
getch();
}

```

Input

Enter the number of blocks: 3
Enter the number of files: 2
Enter the size of the blocks:-
Block 1: 5
Block 2: 2
Block 3: 7
Enter the size of the files:-
File 1: 1
File 2: 4

Output

File No	File Size	Block No	Block Size	Fragment
1	1	3	7	6
2	4	1	5	1

8. Simulate following page replacement algorithms.*Ans :*

(a) FIFO

```

#include<stdio.h>
int main()

```

```

{
    int i,j,n,a[50],frame[10],no,k,avail,count=0;

```

```

        printf("\n ENTER THE NUMBER OF PAGES:\n");
scanf("%d",&n);
        printf("\n ENTER THE PAGE NUMBER :\n");
        for(i=1;i<=n;i++)
            scanf("%d",&a[i]);
        printf("\n ENTER THE NUMBER OF FRAMES :");
        scanf("%d",&no);
        for(i=0;i<no;i++)
            frame[i]= -1;
            j=0;
            printf("\tref string\t page frames\n");
        for(i=1;i<=n;i++)
        {
            printf("%d\t",a[i]);
            avail=0;
            for(k=0;k<no;k++)

if(frame[k]==a[i])

                avail=1;
            if (avail==0)
            {
                frame[j]=a[i];
                j=(j+1)%no;
                count++;
                for(k=0;k<no;k++)
                    printf("%d\t",frame[k]);
            }
            printf("\n");
        }

        printf("Page Fault Is %d",count);
        return 0;
    }

```

Output:

```

ENTER THE NUMBER OF PAGES: 20
ENTER THE PAGE NUMBER : 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
ENTER THE NUMBER OF FRAMES :3

```

ref string	page frames
7	7
0	7
1	7
2	2
0	
3	2
0	2
4	4
2	4
3	4

0	0	2	3
3			
2			
1	0	1	3
2	0	1	2
0			
1			
7	7	1	2
0	7	0	2
1	7	0	1

Page Fault Is 15

(b) LRU

```
#include <stdio.h>
```

```
main()
```

```
{
```

```
int q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20];
```

```
printf("Enter no of pages:");
```

```
scanf("%d",&n);
```

```
printf("Enter the reference string:");
```

```
for(i=0;i<n;i++)
```

```
scanf("%d",&p[i]);
```

```
printf("Enter no of frames:");
```

```
scanf("%d",&f);
```

```
q[k]=p[k];
```

```
printf("\n\t%d\n",q[k]);
```

```
c++;
```

```
k++;
```

```
for(i=1;i<n;i++)
```

```
{
```

```
c1=0;
```

```
for(j=0;j<f;j++)
```

```
{
```

```
if(p[i]!=q[j])
```

```
c1++;
```

```
}
```

```
if(c1==f)
```

```
{
```

```
c++;
```

```
if(k<f)
```

```
{
```

```
q[k]=p[i];
```

```
k++;
```

```
for(j=0;j<k;j++)
```

```
printf("\t%d",q[j]);
```

```
printf("\n");
```

```
}
```

```
else
```

```

    {
        for(r=0;r<f;r++)
        {
            c2[r]=0;
            for(j=i-1;j<n;j--)
            {
                if(q[r]!=p[j])
                    c2[r]++;
                else
                    break;
            }
        }
        for(r=0;r<f;r++)
            b[r]=c2[r];
        for(r=0;r<f;r++)
        {
            for(j=r;j<f;j++)
            {
                if(b[r]<b[j])
                {
                    t=b[r];
                    b[r]=b[j];
                    b[j]=t;
                }
            }
        }
        for(r=0;r<f;r++)
        {
            if(c2[r]==b[0])
                q[r]=p[i];
            printf("\t%d",q[r]);
        }
        printf("\n");
    }
}

printf("\nThe no of page faults is %d",c);
}

```

Output:

Enter no of pages:10

Enter the reference string:7 5 9 4 3 7 9 6 2 1

Enter no of frames:3

```

7
7      5
7      5      9
4      5      9
4      3      9
4      3      7
9      3      7
9      6      7
9      6      2
1      6      2

```

The no of page faults is 10

(c) OPTIMAL

```

#include <stdio.h>
int main()
{
    int n,pg[30],fr[10]; int count[10], i,j,k, fault, f,
    flag, temp,current,c,dist,
    max,m,
    cnt,p,x;
    fault=0;
    dist=0;
    k=0;
    printf("Enter the total no pages:\t");
    scanf("%d",&n);
    printf("Enter the sequence:");
    for(i=0;i<n;i++)
        scanf("%d",&pg[i]);
    printf("\nEnter frame size:");
    scanf("%d",&f);
    for(i=0;i<f;i++)
    {
        count[i]=0;
        fr[i]=-1;
    }
    for(i=0;i<n;i++)
    {
        flag=0;
        temp=pg[i];
        for(j=0;j<f;j++)
        {
            if(temp==fr[j])

```

```

        flag=1;
        break;
    }
}

if((flag==0)&&(k<f))
{
    fault++;
    fr[k]=temp;
    k++;
}

else if((flag==0)&&(k==f))
{
    fault++;
    for(cnt=0;cnt<f;cnt++)
    {
        current=fr[cnt];
        for(c=i;c<n;c++)
        {
            if(current!=pg[c])
                count[cnt]++;
            else
                break;
        }
    }

    max=0;
    for(m=0;m<f;m++)
    {
        if(count[m]>max)
        {
            max=count[m];
            p=m;
        }
    }

    fr[p]=temp;

    printf("\npage %d frame\t",pg[i]);
    for(x=0;x<f;x++)
    {
        printf("%d\t",fr[x]);
    }
}

printf("\nTotal number of faults= %d",fault);
return 0;
}

```

Output:

```

Enter the total no pages:      10
Enter the sequence:0
1
2
3
0
1
2
3
0
1
Enter frame size:3
page 0 frame 0      -1      -1
page 1 frame 0      1      -1
page 2 frame 0      1      2
page 3 frame 0      1      3
page 0 frame 0      1      3
page 1 frame 0      1      3
page 2 frame 0      2      3
page 3 frame 0      2      3
page 0 frame 0      2      3
page 1 frame 0      1      3
Total number of faults=6_

```

9. Simulate following File Organization Techniques*Ans :***(a) Single level directory b. Two level directory**

Single Level Directory Organization

#include <stdio.h>

struct

{

char dname[10],fname[10][10];

int fcnt;

}dir;

void main()

{

int i,ch;

char f[30];

clrscr();

dir.fcnt = 0;

printf("\nEnter name of directory — ");

scanf("%s", dir.dname);

while(1)

{

```

printf("\n\n 1. Create File\t2. Delete File\t3. Search
File \n 4. Display Files\t5. Exit\nEnter your choice — ");

```

scanf("%d",&ch);

switch(ch)

{

case 1: printf("\nEnter the name of the file — ");

scanf("%s",dir.fname[dir.fcnt]);

dir.fcnt++;

break;

case 2: printf("\nEnter the name of the file — ");

scanf("%s",f);

for(i=0;i<dir.fcnt;i++)

{

if(strcmp(f, dir.fname[i])==0)

{

printf("File %s is deleted ",f);

strcpy(dir.fname[i],dir.fname[dir.fcnt-1]);

break;

}

}

if(i==dir.fcnt)

printf("File %s not found",f);

else

dir.fcnt--;

break;

case 3: printf("\nEnter the name of the file — ");

scanf("%s",f);

for(i=0;i<dir.fcnt;i++)

{

if(strcmp(f, dir.fname[i])==0)

{

printf("File %s is found ", f);

break;

}

}

if(i==dir.fcnt)

printf("File %s not found",f);

break;

case 4: if(dir.fcnt==0)

printf("\n Directory Empty");

else

{

printf("\n The Files are — ");

for(i=0;i<dir.fcnt;i++)

printf("\t%s",dir.fname[i]);

}

break;

default: exit(0);

}

```

}
    getch();
}

```

Output:

```

Enter name of directory — CSE
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit
Enter your choice — 1
Enter the name of the file — A
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit
Enter your choice — 1
Enter the name of the file — B
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit
Enter your choice — 1
Enter the name of the file — C
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit
Enter your choice — 4
The Files are — A B C
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit
Enter your choice — 3
Enter the name of the file — ABC
File ABC not found
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit
Enter your choice — 2
Enter the name of the file — B
File B is deleted
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit
Enter your choice — 5

```

2. Two Level Directory Organization

```

#include <stdio.h>
struct
{
    char dname[10], fname[10][10];
    int fcnt;
} dir[10];
void main()

```

```

{
    int i, ch, dcnt, k;
    char f[30], d[30];
    clrscr();
    dcnt = 0;
    while(1)
    {
        printf("\n\n 1. Create Directory\t 2. Create File\t
3. Delete File");
        printf("\n 4. Search File \t \t 5. Display \t 6. Exit \t
Enter your choice — ");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1: printf("\n Enter name of directory — ");
                scanf("%s", dir[dcnt].dname);
                dir[dcnt].fcnt = 0;
                dcnt++;
                printf("Directory created");
                break;
            case 2: printf("\n Enter name of the directory — ");
                scanf("%s", d);
                for(i = 0; i < dcnt; i++)
                    if(strcmp(d, dir[i].dname) == 0)
                {
                    printf("Enter name of the file — ");
                    scanf("%s", dir[i].fname[dir[i].fcnt]);
                    dir[i].fcnt++;
                    printf("File created");
                    break;
                }
            if(i == dcnt)
                printf("Directory %s not found", d);
                break;
            case 3: printf("\nEnter name of the directory — ");
                scanf("%s", d);
                for(i = 0; i < dcnt; i++)
                {
                    if(strcmp(d, dir[i].dname) == 0)
                {
                    printf("Enter name of the file — ");
                    scanf("%s", f);
                    for(k = 0; k < dir[i].fcnt; k++)

```

```

{
    if(strcmp(f, dir[i].fname[k]) == 0)
    {
        printf("File %s is deleted ", f);
        dir[i].fcnt--;
        strcpy(dir[i].fname[k], dir[i].fname[dir[i].fcnt]);
        goto jmp;
    }
}

printf("File %s not found", f);
goto jmp;

}
}

printf("Directory %s not found", d);
jmp : break;
case 4: printf("\nEnter name of the directory — ");
scanf("%s", d);
for(i=0; i<dcnt; i++)
{
    if(strcmp(d, dir[i].dname) == 0)
    {
        printf("Enter the name of the file — ");
        scanf("%s", f);
        for(k=0; k<dir[i].fcnt; k++)
        {
            if(strcmp(f, dir[i].fname[k]) == 0)
            {
                printf("File %s is found ", f);
                goto jmp1;
            }
        }

        printf("File %s not found", f);
        goto jmp1;
    }
}

printf("Directory %s not found", d);
jmp1: break;
case 5: if(dcnt == 0)
    printf("\nNo Directory's ");
else
{
    printf("\nDirectory\tFiles");
    for(i=0; i<dcnt; i++)
{

```

```

        printf("\n%s\t\t", dir[i].dname);
        for(k=0; k<dir[i].fcnt; k++)
            printf("\t%s", dir[i].fname[k]);
    }
}

break;
default: exit(0);

}
}

getch();
}

```

Output:

```

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit
Enter your choice — 1
Enter name of directory — DIR1
Directory created
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit
Enter your choice — 1
Enter name of directory — DIR2

```

Directory created

```

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit
Enter your choice — 2
Enter name of the directory – DIR1
Enter name of the file — A1
File created
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit
Enter your choice — 2
Enter name of the directory – DIR1
Enter name of the file — A2
File created
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit
Enter your choice — 2
Enter name of the directory – DIR2
Enter name of the file — B1
File created
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit

```

Enter your choice — 5
 Directory Files
 DIR1 A1 A2
 DIR2 B1
 1. Create Directory 2. Create File 3. Delete File
 4. Search File 5. Display 6. Exit
 Enter your choice — 4
 Enter name of the directory – DIR
 Directory not found
 1. Create Directory 2. Create File 3. Delete File
 4. Search File 5. Display 6. Exit
 Enter your choice — 3
 Enter name of the directory – DIR1
 Enter name of the file — A2
 File A2 is deleted
 1. Create Directory 2. Create File 3. Delete File
 4. Search File 5. Display 6. Exit
 Enter your choice – 6

10. Simulate following file allocation strategies

Ans :

(a) Sequential

/* Program to simulate sequential file allocation strategy */

Program Code:

```
#include <stdio.h>
```

```
#include <conio.h>
```

```
void main()
```

```
{
    int f[50], i, st, len, j, c, k, count = 0;
    clrscr();
    for(i=0; i<50; i++)
        f[i]=0;
    printf("Files Allocated are : \n");
    x: count=0;
    printf("Enter starting block and length of files: ");
    scanf("%d%d", &st, &len);
    for(k=st; k<(st+len); k++)
        if (f[k]==0)
            count++;
    if(len==count)
    {
        for(j=st; j<(st+len); j++)
```

```
        if(f[j]==0)
        {
            f[j]=1;
            printf("%d\t%d\n", j, f[j]);
        }
        if (j!=(st+len-1))
            printf(" The file is allocated to disk\n");
    }
    else
        printf(" The file is not allocated \n");
    printf("Do you want to enter more file (Yes - 1/No - 0)");
    scanf("%d", &c);
    if(c==1)
        goto x;
    else
        exit();
    getch();
}
```

Output:

Files Allocated are :

Enter starting block and length of files: 14 3

14 1

15 1

16 1

The file is allocated to disk

Do you want to enter more file(Yes - 1/No - 0)1

Enter starting block and length of files: 14 1

The file is not allocated

Do you want to enter more file(Yes - 1/No - 0)1

Enter starting block and length of files: 14 4

The file is not allocated

Do you want to enter more file(Yes - 1/No - 0)0

(b) Indexed

/* Program to simulate indexed file allocation strategy */

Program Code:

```
#include <stdio.h>
```

```
#include <conio.h>
```

```
#include <stdlib.h>
```

```
void main()
```

```
{
    int f[50], index[50], i, n, st, len, j, c, k, ind,
    count=0;
```

```

clrscr();
for(i=0;i<50;i++)
f[i]=0;
x:printf("Enter the index block: ");
scanf("%d",&ind);
if(f[ind]!=1)
{
printf("Enter no of blocks needed and no of files
for the index %d on the disk : \n", ind);
scanf("%d",&n);
}
else
{
printf("%d index is already allocated \n",ind);
goto x;
}
y: count=0;
for(i=0;i<n;i++)
{
scanf("%d", &index[i]);
if(f[index[i]]==0)
count++;
}
if(count==n)
{
for(j=0;j<n;j++)
f[index[j]]=1;
printf("Allocated\n");
printf("File Indexed\n");
for(k=0;k<n;k++)
printf("%d----->%d :
%d\n",ind,index[k],f[index[k]]);
}
else
{
printf("File in the index is already allocated \n");
printf("Enter another file indexed");
goto y;
}
printf("Do you want to enter more file
(Yes - 1/No - 0)");
scanf("%d", &c);

```

```

if(c==1)
goto x;
else
exit(0);
getch();
}

```

Output:

```

Enter the index block: 5
Enter no of blocks needed and no of files for the
index 5 on the disk :
4
1 2 3 4
Allocated
File Indexed
5 -----> 1 : 1
5 -----> 2 : 1
5 -----> 3 : 1
5 -----> 4 : 1
Do you want to enter more file(Yes - 1/No - 0)1
Enter the index block: 4
4 index is already allocated
Enter the index block: 6
Enter no of blocks needed and no of files for the
index 6 on the disk :
2
7 8
A5llocated
File Indexed
6 -----> 7 : 1
6 -----> 8 : 1
Do you want to enter more file(Yes - 1/No - 0)0

```

(c) Linked.

```

/* Program to simulate linked file allocation
strategy */
Program Code:
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
int f[50], p, i, st, len, j, c, k, a;
clrscr();
for(i=0;i<50;i++)

```



```

    f[i]=0;
printf("Enter how many blocks already allocated: ");
scanf("%d",&p);
printf("Enter blocks already allocated: ");
for(i=0;i<p;i++)
{
    scanf("%d",&a);
    f[a]=1;
}
x: printf("Enter index starting block and length: ");
scanf("%d%d",&st,&len);
k=len;
if(f[st]==0)
{
    for(j=st;j<(st+k);j++)
    {
        if(f[j]==0)
        {
            f[j]=1;
            printf("%d————>%d\n",j,f[j]);
        }
        else
        {
            printf("%d Block is already allocated\n",j);
            k++;
        }
    }
}
else
printf("%d starting block is already allocated\n",st);
printf("Do you want to enter more file (Yes - 1/No - 0)");
scanf("%d",&c);
if(c==1)
goto x;
else
exit(0);
getch();
}

```

Output:

Enter how many blocks already allocated: 3
Enter blocks already allocated: 1 3 5

Enter index starting block and length: 2 2

2 ———> 1

3 Block is already allocated

4 ———> 1

Do you want to enter more file(Yes - 1/No - 0)0

11. Simulate Bankers Algorithm for Dead Lock Avoidance.

Ans :

```

#include<stdio.h>
struct file
{
    int all[10];
    int max[10];
    int need[10];
    int flag;
};

void main()
{
    struct file f[10];
    int fl;
    int i, j, k, p, b, n, r, g, cnt=0, id, newr;
    int avail[10],seq[10];
    clrscr();
    printf("Enter number of processes — ");
    scanf("%d",&n);
    printf("Enter number of resources — ");
    scanf("%d",&r);
    for(i=0;i<n;i++)
    {
        printf("Enter details for P%d",i);
        printf("\nEnter allocation\t — \t");
        for(j=0;j<r;j++)
            scanf("%d",&f[i].all[j]);
        printf("Enter Max\t\t — \t");
        for(j=0;j<r;j++)
            scanf("%d",&f[i].max[j]);
        f[i].flag=0;
    }

    printf("\nEnter Available Resources\t — \t");
    for(i=0;i<r;i++)
        scanf("%d",&avail[i]);

```

```

printf("\nEnter New Request Details — ");
printf("\nEnter pid \t — \t");
scanf("%d",&id);
printf("Enter Request for Resources \t — \t");
for(i=0;i<r;i++)
{
    scanf("%d",&newr);
    f[id].all[i] += newr;
    vail[i]=avail[i] - newr;
}

for(i=0;i<n;i++)
{
    for(j=0;j<r;j++)
    {f[i].need[j]=f[i].max[j]-f[i].all[j]; if(f[i].need[j]<0)
    f[i].need[j]=0;
    }

    cnt=0;
    fl=0;
    while(cnt!=n)
    {
        g=0;
        for(j=0;j<n;j++)
        {
            if(f[j].flag==0)
            {
                b=0;
                for(p=0;p<r;p++)
                {
                    if(avail[p]>=f[j].need[p])
                    b=b+1;
                    else
                    b=b-1;
                }

                if(b==r)
                {
                    printf("\nP%d is visited",j); seq[fl++] = j;
                    f[j].flag=1; for(k=0;k<r;k++)
                    avail[k]=avail[k]+f[j].all[k];
                    cnt=cnt+1;
                    printf("("); for(k=0;k<r;k++)
                    printf("%3d",avail[k]);
                    printf(")");
                    g=1;
                }
            }
        }
    }
}

```

```

if(g==0)
{
    printf("\n REQUEST NOT GRANTED —
    DEADLOCK OCCURRED");
    printf("\n SYSTEM IS IN UNSAFE STATE");
    goto y;
}

printf("\nSYSTEM IS IN SAFE STATE");
printf("\nThe Safe Sequence is — (");
for(i=0;i<fl;i++)
    printf("P%d ",seq[i]);
printf(")");

y: printf("\nProcess\t\tAllocation\t\tMax\t\t\tNeed\n");
for(i=0;i<n;i++)
{
    printf("P%d\t",i);
    for(j=0;j<r;j++)
        printf("%6d",f[i].all[j]);
    for(j=0;j<r;j++)
        printf("%6d",f[i].max[j]);
    for(j=0;j<r;j++)
        printf("%6d",f[i].need[j]);
    printf("\n");
}

getch();
}

```

Output :

```

Enter the no of processes:4
Enter the no of resource classes:3
Enter the total existed resource in each class:3 2 2
Enter the allocated resources:1 0 0 5 1 1 2 1 1 0 0 2
Enter the process making the new request:2
Enter the requested resource:1 1 2
Enter the process which are n blocked or running:
process 2:
1 2
process 4:
1 0
process 5:
1 0
Deadlock will occur

```

12. Simulate Bankers Algorithm for Dead Lock Prevention.*Ans :*

```

#include < stdio.h>
#include < conio.h>
void main()
{
    int allocated[15][15], max[15][15], need[15][15], avail[15], tres[15], work[15], flag[15];
    int pno, rno, i, j, prc, count, t, total;
    count = 0;
    clrscr();
    printf("\n Enter number of process:");
    scanf("%d", &pno);
    printf("\n Enter number of resources:");
    scanf("%d", &rno);
    for(i = 1; i <= pno; i++)
    {
        flag[i] = 0;
    }
    printf("\n Enter total numbers of each resources:");
    for(i = 1; i <= rno; i++)
    {
        scanf("%d", &tres[i]);
    }
    printf("\n Enter Max resources for each process:");
    for(i = 1; i <= pno; i++)
    {
        printf("\n for process %d:", i);
        for(j = 1; j <= rno; j++)
        {
            scanf("%d", &max[i][j]);
        }
    }
    printf("\n Enter allocated resources for each process:");
    for(i = 1; i <= pno; i++)
    {
        printf("\n for process %d:", i);
        for(j = 1; j <= rno; j++)
        {
            scanf("%d", &allocated[i][j]);
        }
    }
    printf("\n available resources:\n");
    for(j = 1; j <= rno; j++)
    {
        avail[j] = 0;
        total = 0;
        for(i = 1; i <= pno; i++)
        {

```

```

            total += allocated[i][j];
        }
        avail[j] = tres[j] - total;
        work[j] = avail[j];
        printf("    %d \t", work[j]);
    }
    do
    {
        for(i = 1; i <= pno; i++)
        {
            for(j = 1; j <= rno; j++)
            {
                need[i][j] = max[i][j] - allocated[i][j];
            }
        }
        printf("\n Allocated matrix Max need");
        for(i = 1; i <= pno; i++)
        {
            printf("\n");
            for(j = 1; j <= rno; j++)
            {
                printf("%4d", allocated[i][j]);
            }
            printf(" | ");
            for(j = 1; j <= rno; j++)
            {
                printf("%4d", max[i][j]);
            }
            printf(" | ");
            for(j = 1; j <= rno; j++)
            {
                printf("%4d", need[i][j]);
            }
        }
        prc = 0;
        for(i = 1; i <= pno; i++)
        {
            if(flag[i] == 0)
            {
                prc = i;
                for(j = 1; j <= rno; j++)
                {

```

```

    if(work[j] < need[i][j])
    {
        prc=0;
        break;
    }
}
if(prc!=0)
break;
}
if(prc!=0)
{
    printf("\n Process %d completed",i);
    count++;
    printf("\n Available matrix:");
    for(j=1;j<= mo;j++)
    {
        work[j] += allocated[prc][j];
        allocated[prc][j] = 0;
        max[prc][j] = 0;
        flag[prc] = 1;
        printf("   %d",work[j]);
    }
}
}while(count!=pno&&prc!=0);
if(count==pno)
    printf("\nThe system is in a safe state!!");
else
    printf("\nThe system is in an unsafe state!!");
getch();
}

```

Output

```

Enter number of process:5
Enter number of resources:3
Enter total numbers of each resources:10 5 7
Enter Max resources for each process:
for process 1:7 5 3
for process 2:3 2 2
for process 3:9 0 2
for process 4:2 2 2
for process 5:4 3 3

```

Enter allocated resources for each process:

for process 1:0 1 0

for process 2:3 0 2

for process 3:3 0 2

for process 4:2 1 1

for process 5:0 0 2

available resources:

2 3 0

Allocated matrix			Max			need		
0	1	0	7	5	3	7	4	3
3	0	2	3	2	2	0	2	0
3	0	2	9	0	2	6	0	0
2	1	1	2	2	2	0	1	1
0	0	2	4	3	3	4	3	1

Process 2 completed

Available matrix: 5 3 2

Allocated matrix			Max			need		
0	1	0	7	5	3	7	4	3
0	0	0	0	0	0	0	0	0
3	0	2	9	0	2	6	0	0
2	1	1	2	2	2	0	1	1
0	0	2	4	3	3	4	3	1

Process 4 completed

Available matrix: 7 4 3

Allocated matrix			Max			need		
0	1	0	7	5	3	7	4	3
0	0	0	0	0	0	0	0	0
3	0	2	9	0	2	6	0	0
0	0	0	0	0	0	0	0	0
0	0	2	4	3	3	4	3	1

Process 1 completed

Available matrix: 7 5 3

Allocated matrix			Max			need		
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
3	0	2	9	0	2	6	0	0
0	0	0	0	0	0	0	0	0
0	0	2	4	3	3	4	3	1

Process 3 completed

Available matrix: 10 5 5

Allocated matrix			Max	need
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	2	4	3

Process 5 completed

Available matrix: 10 5 7

The system is in a safe state!!

13. Write a C program to simulate disk scheduling algorithms.*Ans :*

```
(a) FCFS
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int RQ[100], i, n, TotalHeadMoment = 0, initial;
    printf("Enter the number of Requests\n");
    scanf("%d", &n);
    printf("Enter the Requests sequence\n");
    for(i=0; i<n; i++)
        scanf("%d", &RQ[i]);
    printf("Enter initial head position\n");
    scanf("%d", &initial);
    // logic for FCFS disk scheduling
    for(i=0; i<n; i++)
    {
        TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial);
        initial = RQ[i];
    }
    printf("Total head moment is %d", TotalHeadMoment);
    return 0;
}
```

Output

```
Enter the number of Request
8
Enter the Requests Sequence
9518034119111236264
```

Enter initial head position

50

Total head movement is 644

(b) SCAN

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int RQ[100], i, j, n, TotalHead Moment = 0, initial,
    size, move;
    printf("Enter the number of Requests\n");
    scanf("%d", &n);
    printf("Enter the Requests sequence\n");
    for(i=0; i<n; i++)
        scanf("%d", &RQ[i]);
    printf("Enter initial head position\n");
    scanf("%d", &initial);
    printf("Enter total disk size\n");
    scanf("%d", &size);
    printf("Enter the head movement direction for high 1
    and for low 0\n");
    scanf("%d", &move);
    // logic for Scan disk scheduling
    /*logic for sort the request array */
    for(i=0; i<n; i++)
    {
        for(j=0; j<n-i-1; j++)
        {
            if(RQ[j] > RQ[j+1])
            {
                int temp;
                temp = RQ[j];
                RQ[j] = RQ[j+1];
                RQ[j+1] = temp;
            }
        }
    }
    int index;
    for(i=0; i<n; i++)
    {
        if(initial < RQ[i])
        {
            index = i;
```

```

break;
}
}
// if movement is towards high value
if(move == 1)
{
for(i = index; i < n; i++)
{
    TotalHeadMoment = TotalHeadMoment + abs(RQ[i] -
initial);
    initial = RQ[i];
}
// last movement for max size
TotalHeadMoment = TotalHeadMoment + abs(size -
RQ[i-1]-1);
    initial = size-1;
for(i = index-1; i >= 0; i--)
{
    TotalHeadMoment = TotalHeadMoment + abs(RQ[i] -
initial);
    initial = RQ[i];
}
}
// if movement is towards low value
else
{
for(i = index-1; i >= 0; i--)
{
    TotalHeadMoment = TotalHeadMoment + abs(RQ[i] -
initial);
    initial = RQ[i];
}
// last movement for min size
TotalHeadMoment = TotalHeadMoment + abs(RQ[i+1]-
0);
    initial = 0;
for(i = index; i < n; i++)
{
    TotalHeadMoment = TotalHeadMoment + abs(RQ[i] -
initial);
    initial = RQ[i];
}
}

```

```

printf("Total head movement is %d", Total Head
Moment);
return 0;
}

```

Output

Enter the number of Request

8

Enter the Requests Sequence

9518034119111236264

Enter initial head position

50

Enter total disk size

200

Enter the head movement direction for high 1 and for low 0

1

Total head movement is 337

(c) C-SCAN

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
int main()
```

```
{
```

```
    int RQ[100], i, j, n, Total Head Moment = 0, initial,
size, move;
```

```
    printf("Enter the number of Requests\n");
```

```
    scanf("%d", &n);
```

```
    printf("Enter the Requests sequence\n");
```

```
for(i = 0; i < n; i++)
```

```
    scanf("%d", &RQ[i]);
```

```
    printf("Enter initial head position\n");
```

```
    scanf("%d", &initial);
```

```
    printf("Enter total disk size\n");
```

```
    scanf("%d", &size);
```

```
    printf("Enter the head movement direction for high 1
and for low 0\n");
```

```
    scanf("%d", &move);
```

```
// logic for C-Scan disk scheduling
```

```
/*logic for sort the request array */
```

```
for(i = 0; i < n; i++)
```

```
{
```

```
for(j = 0; j < n-i-1; j++)
```

```
{
```

```
if(RQ[j] > RQ[j+1])
```

```

{
int temp;
    temp=RQ[j];
    RQ[j]=RQ[j+1];
    RQ[j+1]=temp;
}

}
}
int index;
for(i=0;i<n;i++)
{
if(initial<RQ[i])
{
    index=i;
break;
}
}
// if movement is towards high value
if(move==1)
{
for(i=index;i<n;i++)
{
    TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-
initial);
    initial=RQ[i];
}
// last movement for max size
    TotalHeadMoment=TotalHeadMoment+abs(size-
RQ[i-1]-1);
/*movement max to min disk */
    TotalHeadMoment=TotalHeadMoment+abs(size-
1-0);
    initial=0;
for( i=0;i<index;i++)
{
    TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-
initial);
    initial=RQ[i];
}
}
// if movement is towards low value
else

```

```

{
for(i=index-1;i>=0;i--)
{
    TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-
initial);
    initial=RQ[i];
}
// last movement for min size
    TotalHeadMoment=Total Head Moment+abs
(RQ[i+1]-0);
/*movement min to max disk */
    Total Head Moment = Total Head Moment +
abs(size-1-0);
    initial =size-1;
for(i=n-1;i>=index;i--)
{
    TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-
initial);
    initial=RQ[i];
}
}
printf("Total head movement is %d", Total Head
Moment);
return0;
}

```

Output

```

Enter the number of Request
8
Enter the Requests Sequence
9518034119111236264
Enter initial head position
50
Enter total disk size
200
Enter the head movement direction for high 1
and for low 0
1
Total head movement is 382

```

FACULTY OF INFORMATICS
M.C.A. I Year II Semester Examination
Model Paper - I
OPERATING SYSTEMS

Time : 3 Hours]

Max. Marks : 70

(5 × 14 = 70 Marks)

Note : Answer all the question according to the internal choice

ANSWERS

1. Explain various basic unix commands with an examples. (Unit-I, Q.No.2)
OR
2. Explain the structure of the Operating system. (Unit-I, Q.No.15)
3. What is the use of page replacement algorithm? Explain about various page replacement algorithms. (Unit-II, Q.No.11)
OR
4. How memory is allocated to the process? Explain how to resolve dynamic storage allocation problem? (Unit-II, Q.No.5)
5. What is file? Explain about the structure of the file and its attributes. (Unit-III, Q.No.1)
OR
6. Write about Directory structure of file system. (Unit-III, Q.No.8)
7. What are the Goals and Principles of Protection? (Unit-IV, Q.No.1)
OR
8. What is Encryption? Write about Symmetric and Asymmetric Encryption. (Unit-IV, Q.No.12)
9. Discuss about the kernel modules of Linux Systems (Unit-V, Q.No.3)
OR
10. Write about programmer interface of Windows 7. (Unit-V, Q.No.16)

FACULTY OF INFORMATICS
M.C.A. I Year II Semester Examination
Model Paper - II
OPERATING SYSTEMS

Time : 3 Hours]

Max. Marks : 70

(5 × 14 = 70 Marks)

Note : Answer all the question according to the internal choice**ANSWERS**

1. Explain about File Permission of unix. (Unit-I, Q.No.4)
OR
2. Discuss briefly about various multi threading issues. (Unit-I, Q.No.25)
3. Write a briefly note on fragmentation. (Unit-II, Q.No.6)
OR
4. Define thrashing. Explain the techniques of thrashing. (Unit-II, Q.No.14)
5. What are the various File Access Mechanisms? (Unit-III, Q.No.3)
OR
6. Explain about directory implementation methods. (Unit-III, Q.No.10)
7. Write about Access Control Mechanism. (Unit-IV, Q.No.5)
OR
8. Write about various user authentication processes. (Unit-IV, Q.No.15)
9. Write about the design principles of windows. (Unit-V, Q.No.12)
OR
10. What are the various components of Linux System ? (Unit-V, Q.No.1)

FACULTY OF INFORMATICS

M.C.A. I Year II Semester Examination

Model Paper - III

OPERATING SYSTEMS

Time : 3 Hours]

Max. Marks : 70

(5 × 14 = 70 Marks)

Note : Answer all the question according to the internal choice**ANSWERS**

1. Explain briefly about awk. (Unit-I, Q.No.9)
OR
2. What is multi processor scheduling explain? (Unit-I, Q.No.30)
3. What is Swapping? Explain. (Unit-II, Q.No.3)
OR
4. Consider a main memory with five page frames and the following sequence of page references: 3, 8, 2, 3, 9, 1, 6, 3, 8, 9, 3, 6, 2, 1, 3. which one of the following is true with respect to page replacement policies First-In-First-out (FIFO) and Least Recently Used (LRU)? (Unit-II, Q.No.12)
5. What are Mass (Secondary) Storage Devices? Explain. (Unit-III, Q.No.11)
OR
6. Write about various communication I/O devices. (Unit-III, Q.No.17)
7. Write about the implementation of cryptography in secure socket layers (SSL). (Unit-IV, Q.No.14)
OR
8. What is Language Based Protection? Explain. (Unit-IV, Q.No.8)
9. Write about the process management of Linux System. (Unit-V, Q.No.4)
OR
10. Describe briefly about Windows Architecture. (Unit-V, Q.No.11)

FACULTY OF INFORMATICS
M.C.A II-Semester(CBCS) Examination
November - 2021
OPERATING SYSTEMS

Time : 2 Hours]

[Max. Marks : 70

PART - A - (4 × 17^{1/2} = 70 Marks)

Note : Answer any four Questions.

1. (a) What are the Operating System services and AWK Explain in detail.
(b) Solve the dining philosopher's problem using monitors instead of semaphores.
2. (a) What is process synchronization and monitors.
(b) A system has four processes and five locatable resources. The current application and maximum needs are as follows :

	Allocated	Maximum	Available
Process A	10211	112113	00X11
Process B	20110	22210	
Process C	11010	21310	
Process D	11110	11221	

- What is the smallest value of x for which this is a safe state?
3. (a) How the fragmentation is avoided give an examples
(b) If FIFO page replacement is used with four page frames and eight pages how many page fault will occur with the reference string 0 1 7 2 3 2 7 1 0 3 if the four frames are initially empty? Now repeat this problem for optimal.
 4. (a) Elaborate segmentation with paging.
(b) Explain demand paging with a suitable example.
 5. (a) State the allocation methods and which method is best.
(b) Tell about the mass storage structures.
 6. (a) Describe I/O systems and file concepts.
(b) Illustrate directory implementation of file systems.
 7. (a) What are the goals and domain of protection ?
(b) Mention the access control and access rights.
 8. (a) How to implement the security defenses brief.
(b) List the computer security classification and discuss them.
 9. (a) Show the design principles of Linux systems.
(b) Discuss the windows 7 design principles.
 10. (a) Differentiate the terminal services and fast user switching file systems.
(b) Why are the kernel modules what they do? Define input and output.

FACULTY OF INFORMATICS
M.C.A II - Semester (CBCS) Examination,
April - 2022
OPERATING SYSTEMS

Time : 3 Hours]**[Max. Marks : 70**

Note : Answer any five questions from the following. All questions carry equal marks.

1. (a) What are system components and regular expressions. Explain in detail.
 (b) Define grep? Why A.W.K. give an example?
2. (a) Write about file permissions? Show how counting semaphores (i.e semaphores that can hold an arbitrary value) can be implemented using only binary or ordinary machine instructions.
 (b) Define CPU scheduling criteria. Consider the following set of prdcesses, with the length of the CPU-burst time given in milliseconds.

Process	Burst time
P ₁	10
P ₂	1
P ₃	2
P ₄	1
P ₅	5

The processes are assumed to have arrived in the order P₁, P₂, P₃, P₄ and P₅ all at time 0. Draw Gantt Chart illustrating the Execution of these processes using FCFS and SJF, under preemptive scheduling and calculate TAT of each process and waiting time of each process and response time of each process.

3. (a) List the memory management strategies with example architecture?
 (b) If L R U page replacement is used with four page frames and eight pages how many page fault will occur with the reference string 0 1 7 2 3 2 7 1 0 3 if the four frams are initially empty? Now repeat this problem for optimal.
4. (a) Define thrashing? Discuss paging.
 (b) What is page fault and how it is handled give an explanation.
5. (a) How many access methods are there in file and which one is the best access method give an example.
 (b) Illustrate file system implementation.
6. (a) Differ file system structure and mass storage structure.
 (b) Why allocation methods? Which method is the best method explain?

PREVIOUS QUESTION PAPERS

- 7 (a) Mentions the principles and domains of system protection.
 (b) Elaborate cryptography.
- 8 (a) What are the features of language based protection.
 (b) Tell about firewalling.
- 9. (a) Brief about design principles and kernel module.
 (b) Illustrate the case study of Linux systems.
- 10. (a) What is the design principals of windows 7 brief
 (b) State the networking and programmer interface.

FACULTY OF INFORMATICS
MCA II - Semester (CBCS) Examinations,
April / May - 2023
OPERATING SYSTEMS

Time : 3 Hours]**[Max. Marks : 70**

Note: I. Answer one question from each unit. All questions carry equal marks.

II. Missing data, if any, may be suitably assumed.

UNIT - I

1. (a) Write the syntax and examples to all the commands related to directories.
(b) Illustrate process states with its figure.

(OR)

2. (a) Describe and explain priority scheduling with an illustration.
(b) Explain the functionality of chmod command.

UNIT - II

3. (a) Illustrate segmentation mechanism with its figure.
(b) Explain the concept of thrashing.

(OR)

4. (a) Illustrate the FIFO page replacement algorithm.
(b) Explain the steps of demand paging with its diagram.

UNIT - III

5. (a) Write about the implementation of tree structured directory system.
(b) Illustrate the SCAN disk scheduling algorithm with a request queue (0-199):
98,183,37,122,14,124,64,67. The head pointer is at 53.

(OR)

6. (a) Explain with illustration the concept of linked list allocation method.
(b) Describe the characteristics of I/O devices.

UNIT - IV

7. (a) Implement an access matrix for system protection.
(b) Classify the security methods for the computer.

(OR)

8. (a) Write notes on cryptography and its applications.
(b) Explain the concept of firewalls.

UNIT - V

9. (a) Explain the kernel module of linux system.
(b) Discuss about the programmer interface.

(OR)

10. (a) Explain the file system of windows 7.
(b) Write notes on memory management in linux.