BCA-13

Vardhaman Mahaveer Open University, Kota

Operating System - II

Course Development Committee

Chairman
Prof. (Dr.) Naresh Dadhich
Former Vice-Chancellor
Vardhaman Mahaveer Open University, Kota

Co-ordinator/Convener and Members

Co-ordinator/Convener

Dr. Anuradha Sharma
Assistant Professor,
Department of Botany, Vardhaman Mahaveer Open University, Kota

Members :

1. **Dr. Neeraj Bhargava**
   Department of Computer Science
   Maharishi Dyanand University, Ajmer
2. **Prof. Reena Dadhich**
   Department of Computer Science
   University of Kota, Kota
3. **Dr. Madhavi Sinha**
   Department of Computer Science
   Birla Institute of Technology & Science, Jaipur
4. **Dr. Rajeev Srivastava**
   Department of Computer Science
   LBS College, Jaipur
5. **Dr. Nishtha Kesswani**
   Department of Computer Science
   Central University of Rajasthan, Ajmer

Editing and Course Writing

Editor

Dr. Nishtha Kesswani
Department of Computer Science
Central University of Rajasthan, Ajmer

	Unit Writers	Unit No.
1.	**Dr. Nishtha Kesswani** Department of Computer Science Central University of Rajasthan, Ajmer	(1,2)
2.	**Dr. Om Prakash** Senior Software Consultant & Visiting Professor University of Rajasthan, Jaipur	(3,4,5)
3.	**Dr. Rajesh Dadhich** Department of Computer Science Govt. Polytechnic College, Kota	(6,7)
4.	**Dr. Bright Keswani** Department of Computer Science S. Gyan Vihar University, Jaipur	(8,9)
5.	**Sh. Sudesh Kumar Prajapat** Department of Computer Science Indira Gandhi National Tribal University Amarkantak (M.P.)	(10,11)
6.	**Sh. Sanjay Kumar Anand** Department of Computer Science Central University of Rajasthan, Ajmer	(12,13)
7.	**Sh. Anubha Jain** Department of Computer Science IIS University, Jaipur	(14,15)

Academic and Administrative Management

Prof. (Dr.) Vinay Kumar Pathak
Vice-Chancellor
Vardhaman Mahveer Open University, Kota

Prof. (Dr.) B.K. Sharma
Director (Academic)
Vardhaman Mahveer Open University, Kota

Prof. P.K. Sharma
Director (Regional Services)
Vardhaman Mahveer Open University, Kota

Course Material Production

Mr. Yogendra Goyal
Assistant Production Officer
Vardhaman Mahaveer Open University, Kota

Vardhaman Mahaveer Open University, Kota

CONTENTS

Operating System - II

Preface

The Course Operating System- II has been specially designed for students already having some basic knowledge of Operating systems. The text has been designed for versatile and complete insight into operating systems. This book provides some advanced topics in operating system presented in an interesting manner.

The text contains 15 Chapters intended primarily for graduate courses. The wide range of topics covered in the book makes it an excellent handbook on operating systems. It covers topics such as disk scheduling, NFS and NIS, distributed computing, distributed file system and message passing. It also covers state of the art operating systems such as Windows and Linux. Shell Scripts and Programming and Linux System administration have also been discussed in detail. The text has been supplemented with appropriate Figures for better understanding.

Extensive references and pointers to the current literature have also been provided for further reading. Each Chapter ends with self-assessment exercises that can be used for practice.

Unit - 1 : Disk Scheduling

Structure of Unit

1.0 Objective

The objective of this unit is to make you aware of some aspects of disk scheduling and scheduling algorithms. In this unit we will look inside what disk scheduling is. We will take a sneak peak at the "algorithms" in general and "selection of algorithms" in particular. We will also see how these all correlate to make the Linux system operative.

1.1 Introduction

The Disk is said to be of two basic types:

1. Fixed head disk- This has one head for each track on the disk and it requires no head movement time to service a request. This is quite expensive.
2. Movable head disk- This is much more common in use because it has a single head driven by a stepper motor that can position the head over any desired track on the disk surface.

The task of scheduling usage of sharable resources by the various processes is one of the important jobs of operating system as it is responsible for efficient use of the disk drives.

The efficiency of disk drivers means that disks must have fast access time and reasonable bandwidth. The two major components of access time and bandwidth of disks are:

- *Seek time*-the time to move the heads to the cylinder containing the desired sector.
- *Rotational latency*-the additional time to rotate the desired sector to the disk head.

This can be considered very important in case of systems with multi programming as they have a common file system. The file system is said to be common in multi programmed systems because it is shared by all the

users even though each of them may have one's own file. This common file system may be spread out over a finite number of disks or it may reside entirely on a single disk.

Thus, all processes that do disk IO are competing for access to the same physical disk or set of physical disks. Mostly as any given disk can only perform one access at a particular time, if several accesses are requested on a given disk, some order of service for the requests is established by the OS.

The only exception is that there are two or more independent head assemblies on some disks and so those can perform two or more service requests at a single time. However, even in this type of cases too, scheduling is a must if there are more requests outstanding than the available heads to serve them.

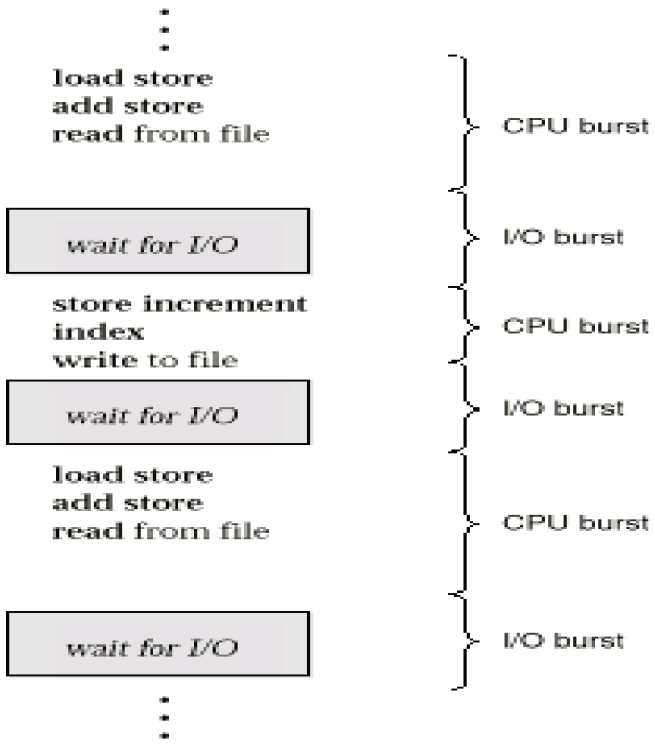


Figure 1.1 CPU and IO burst

1.2 Disk Scheduling

It is now clear that there can be number of programs in memory at the same time that results in overlapping of CPU and I/O.

There are batch programs that run without interaction from user. There may be time shared programs that run with user interaction. For both of these the common name used is Process for which burst cycle of CPU characterizes execution of their process, alternatively between CPU and I/O activity. The scheduling makes selection among the processes in memory that are ready to be executed and makes allocation of the CPU to one of them. The decision regarding scheduling takes place when a process switches from:

1. Running to waiting state

2. Running to ready state

3. Waiting to ready state

4. Terminates

The scheduling of the above processes is known as nonpreemptive. It must be noted that mostly the scheduling quantum is not used by almost all processes as shown in Figure 1.2.

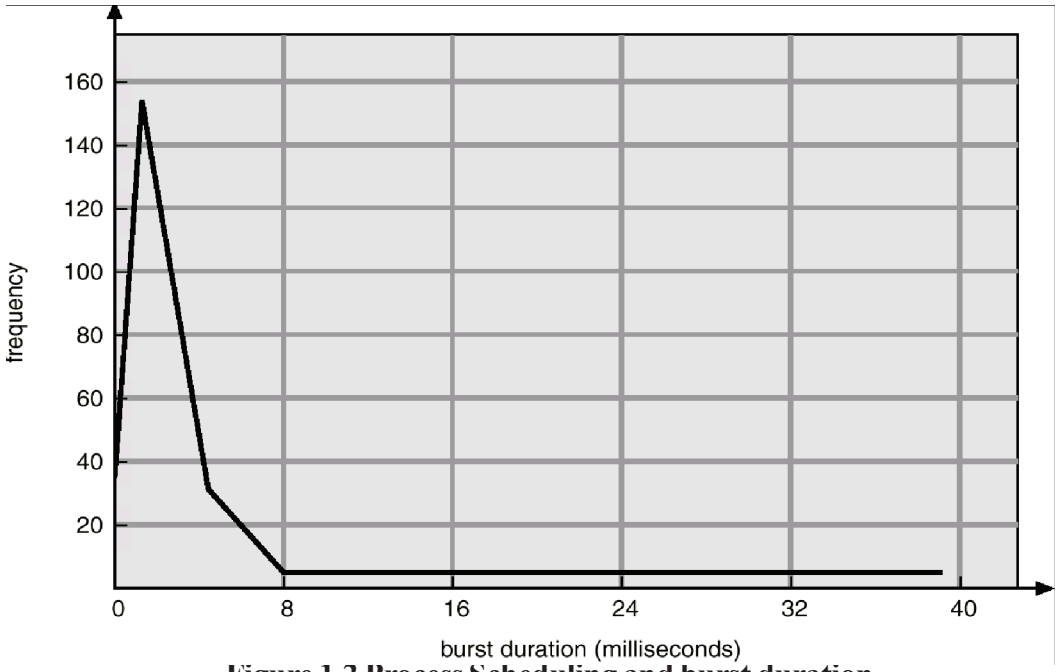


Figure 1.2 Process Scheduling and burst duration.

1.3 Scheduling Algorithms

1.3.1 First Come First Serve (FCFS)

It is similar to FIFO. It is simple, fair approach but perhaps not the best because of its poor performance as average queue time may be too long to be served. It is quite difficult to find the average queue and residence times for this. Of course, the simplest way but if disk accesses are scheduled in an order that takes into consideration some of the physical characteristics of the disk then system can be improved significantly throughout. For example, for the following processes request queue 98, 183, 37, 122, 14, 124, 65, 67, with head pointer 53, total head movement is 640 cylinders.

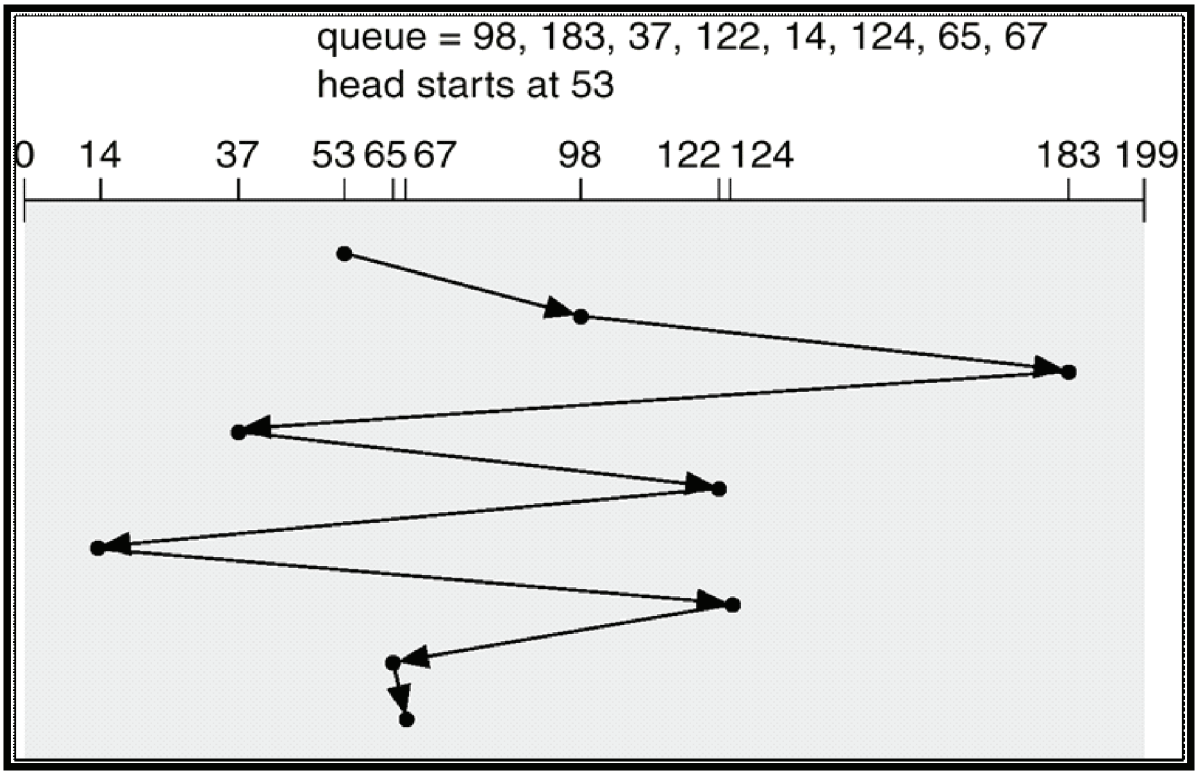


Figure 1.3 FCFS

1.3.2 SSTF: Shortest Seek Time First

It is much more efficient, but leads to starvation. It may be optimal for minimizing queue time, but may be impossible to be implemented as it tries to predict the scheduled process based on previous history. It selects the request with the minimum seek time from the current head position. It is a form of SJF scheduling; may cause starvation of some requests.

The prediction of the time used by the process on its next schedule can be given by

$$t(n+1) = w * t(n) + (1 - w) * T(n)$$

Where, t (n+1) is time of next burst.

t (n) is time of current burst.

T (n) is average of all previous bursts

W is a weighting factor emphasizing current or previous bursts.

For Example, with head pointer at 53, Total head movement:

98 + 183 + 37 + 122 + 14 + 124 + 65 + 67 = 236 tracks or < 30 tracks per access

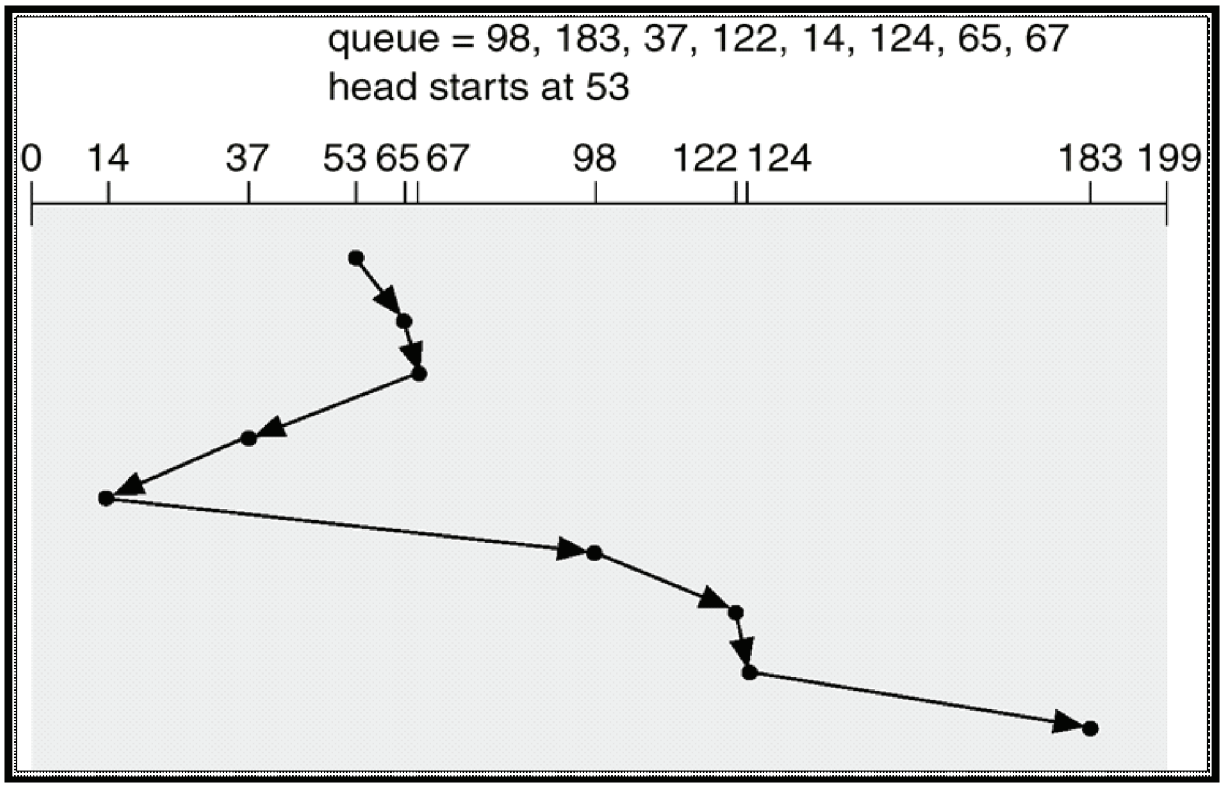


Figure 1.4 SSTF

But SSTF can be a problem on a heavily used disk. If one request is at the extreme and the other request is nearer to the centre, the extreme request can be postponed for a long time.

1.3.3 SCAN

The purpose of it is to combine efficiency with fairness. The process starts at one end of the disk with movement towards the other end, servicing requests until end, where the head movement is reversed and servicing continues. It is also known as the elevator algorithm because of its working similar to an elevator services in a building. When it goes up, it requests services in order from floors above it, but floors below it, are ignored. When it goes down, it only requests services below it. For example, with head pointer 53, Total head movement is:

98 183 + 37 + 122 + 14 + 124 + 65 + 67 = 208 tracks

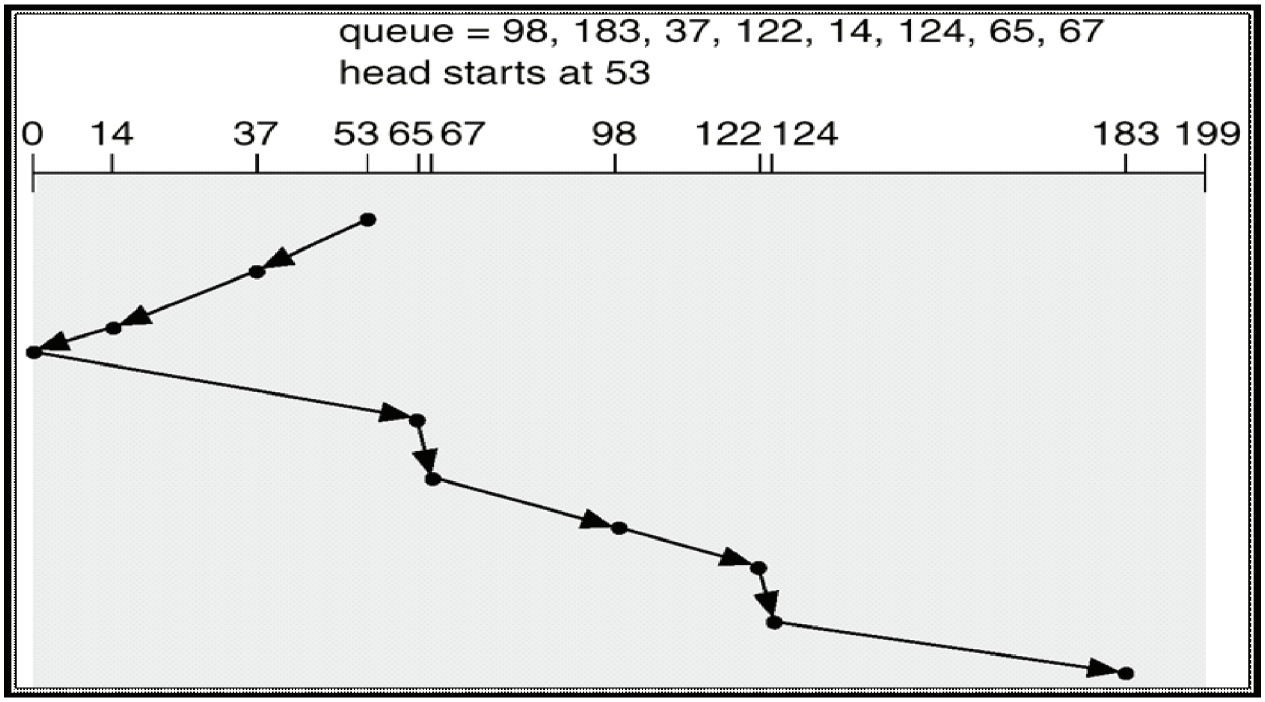


Figure 1.5 SCAN

1.3.4 C-SCAN: Circular SCAN

This algorithm is similar to SCAN. The only exception is that the disk requests services in one direction only and "jumps" to the starting of disk when the last track is reached. This results in a more uniform response time. Since a single large jump may be faster than several smaller ones, overall it may be more efficient than SCAN. By providing a more uniform wait time and treating the cylinders as a circular list, it proves better than SCAN. For example, with head pointer 53,

Total head movement:

98 183 + 37 + 122 + 14 + 124 + 65 + 67 = 322 tracks

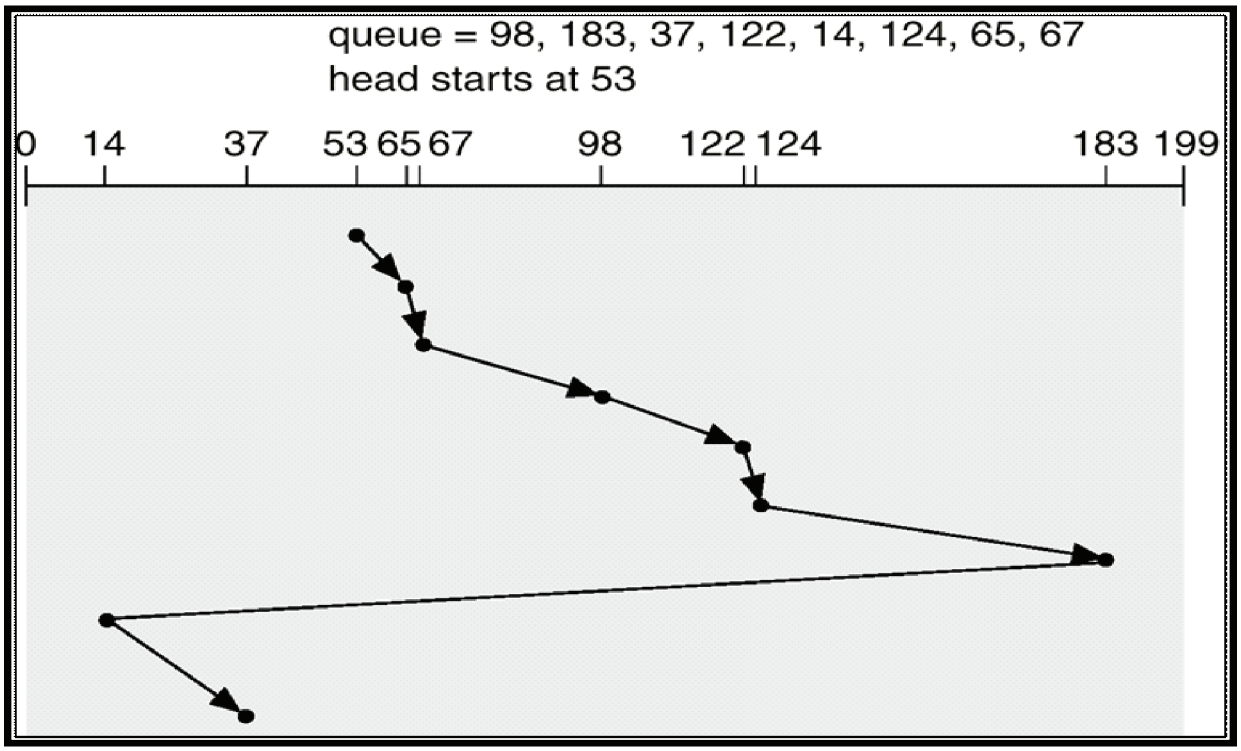


Figure 1.6 CSCAN

1.3.6 FSCAN

With the above discussed algorithms it may be possible that the arm may not move for a considerable period of time. To avoid this arm stickiness the disk request queue can be segmented, with one segment being processed at a time completely.

FSCAN is an example of such an approach. It is the policy that uses two sub queues. When a SCAN begins, all of the requests are in one of the queues, with the other empty. During the scan, requests are put into the other queue. This means that till all the old requests gets processed, service of the new requests is deferred.

1.3.7 N-step-SCAN

This policy segments the queue of disk request into sub queues of length N and the processing of these is one at a time using SCAN. Till the processing of a queue, new requests are added to some other queue. If requests available are less than N, at the end of scan, then all of them are processed with the next scan. With larger values of N, the performance approaches similar to SCAN and for N=1, it approaches FIFO.

1.3.8 Multi-Level Queue

This type of algorithm has multi queues with each queue having its own algorithm. Then priority based algorithm arbitrates between those multi level queues that can use feedback to move between queues. This method is flexible but complex. For example:

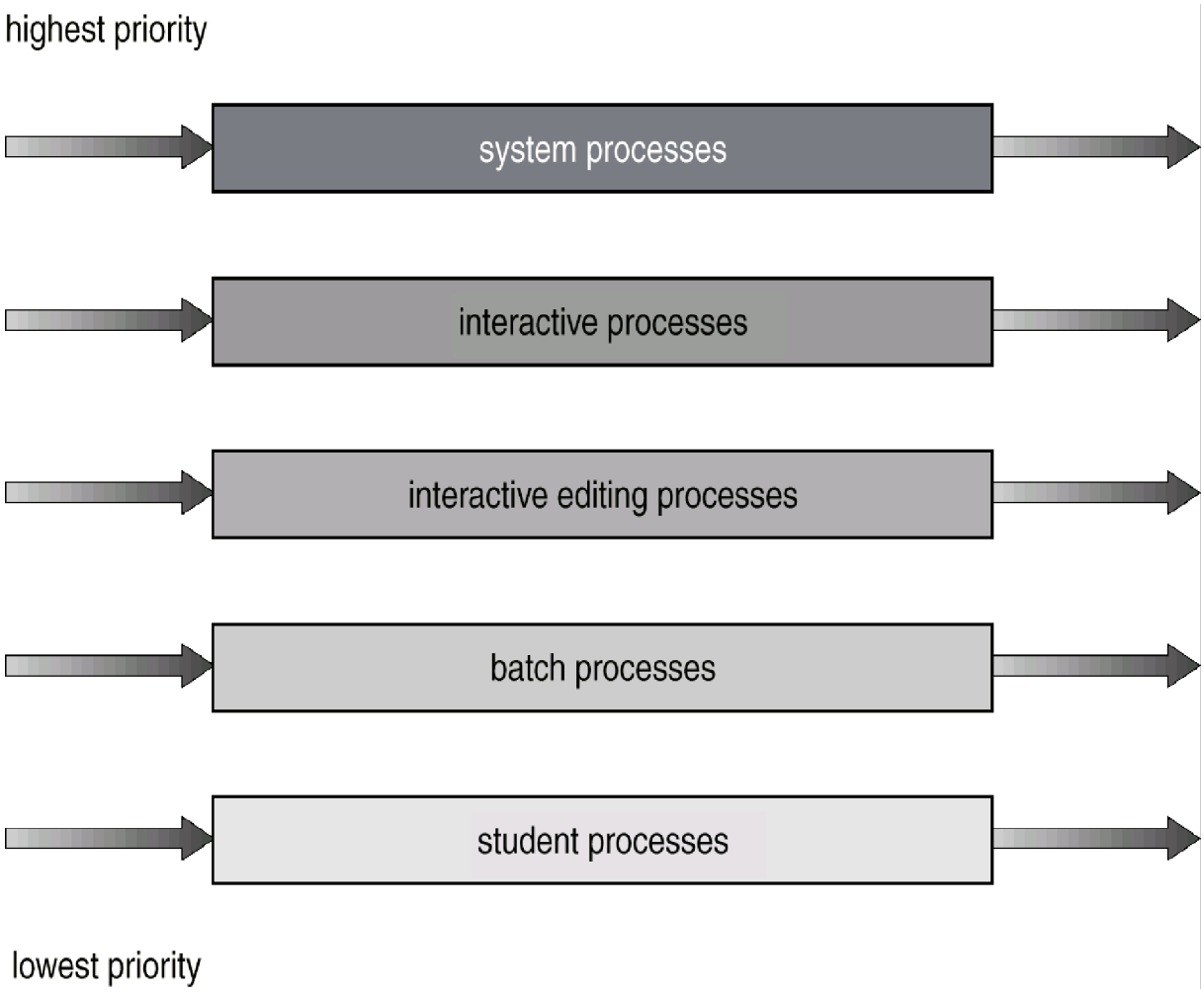


Figure 1.7 Multi-level queue

1.3.8 Multiple Processor Scheduling

We know that there are different rules for heterogeneous or homogeneous processors. For example, sharing of load in the distribution of work in such a manner that all processors have an equal amount to do work. In this each processor can schedule from a queue that is ready common or can use an arrangement by master slave.

1.4 Selection of Algorithms

To determine a particular algorithm, predetermined workload and the performance of each algorithm for that workload is to be determined. It can be said that

- SSTF is quite common and so naturally, it has a appeal
- The performance of SCAN and C-SCAN is better for system, which places a heavy load on the disk.
- Performances depend on the types & numbers of requests, which in turn are influenced by the file-allocation method.
- The algorithm must be written as a separate module of the operating system. It must be allowed to be replaced with other one, if necessary.
- For default, either SSTF or LOOK is a reasonable choice.

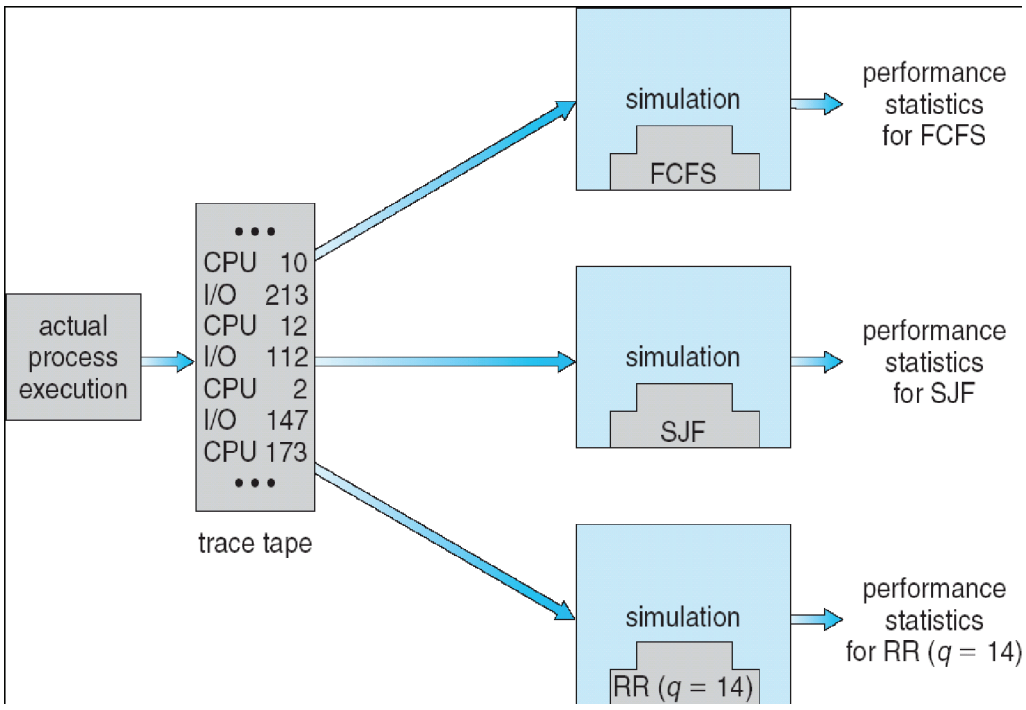


Figure 1.8 Selection of algorithm

1.5 Summary

A number of different scheduling algorithms have been discussed and which one is the best to work that depends on the application of it. The following table shows the comparison of different types of algorithms (starting at track 100):

FIFO		SSTF		SCAN		C-SCAN	
Next track Accessed	Number of tracks traversed	Next track accessed	Number of tracks traversed	Next track Accessed	Number of tracks traversed	Next track Accessed	Number of tracks traversed
55	45	90	10	150	50	150	50
58	03	58	32	160	10	160	10
39	19	55	03	184	24	184	24
18	21	39	16	90	94	18	166
90	72	38	01	58	32	38	20
160	70	18	20	55	03	39	01
150	10	150	132	39	16	55	16
38	112	160	10	38	01	58	03
184	146	184	24	18	20	90	32
Avge seek	27.5	Avge seek	27.5	Avge seek	27.8	Avge seek	35.8

The OS with general purpose may use FCFS, CSCAN, preemptive and the OS with real time can opt for priority, no preemptive as in these OS performance is never obvious and Benchmarking is everything. The three types of scheduling decisions taken by OS with respect to the execution of process are:

- **Long term:** finds when new processes are to be admitted to the system.
- **Medium term:** finds when a program is bought into main memory for execution.
- **Short term:** finds which ready process will be executed next by the processor.

The choice of algorithm depends on expected performance and on implementation complexity as shown below:

Name	Description	Remarks
Selection according to requester		
RSS	Random scheduling	For analysis and simulation
FIFO	First in First Out	Fairest of them all
PRI	Priority by process	Control outside of queue management
LIFO	Last in First Out	Maximize locality and resource utilization
Selection according to requested ITEM		
SSTF	Shortest service time first	High utilization, small queues
SCAN	Back and forth over disk	Better service distribution
CSCAN	One way with fast return	Lower service variability
N-step-SCAN	Scan of N records at a time	Service Guarantee
FSCAN	N step Scan with N=queue size at beginning of cycle	Load sensitive

1.6 Self - Assessment Exercise

1. What is disk scheduling algorithm?
2. Suppose that a disk drive has 500 cylinders, numbered 0 to 499. The drive is currently serving a request at cylinder 123, and the previous request was at cylinder 105. The queue of pending requests, in FIFO order, is

   86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130.
3. Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests, for each of the following disk-scheduling algorithms?

   FIFO

   SSTF

   SCAN (Elevator)

   C-SCAN (Modified ELevator)
4. Consider the multilevel feedback queue scheduling algorithm used in traditional Unix systems. It is designed to favour IO bound over CPU bound processes. How is this achieved? How does it make sure that low priority, CPU bound background jobs do not suffer starvation?
5. Why would a hypothetical OS always schedule a thread in the same address space over a thread in a different address space? Is this a good idea?
6. Why would a round robin scheduler NOT use a very short time slice to provide good responsive application behaviour?
7. Differentiate between pre-emptive and non-pre-emptive scheduling.
8. CPU burst time indicates the time, the process needs the CPU. The following are the set of processes with their respective CPU burst time (in milliseconds).

Processes	CPU-burst time
P1	10
P2	5
P3	5

Calculate the average waiting time if the process arrived in the following order:

(i) P1, P2 & P3 (ii) P2, P3 & P1

1.7 References

- Coffman, E. G., Klimko, L. A., and Ryan, B., "Analysis of Scanning Policies for Reducing Disk Seek Times", SIAM Journal of Computing, September 1972, Vol 1. No 3.
- Geist, Robert, and Daniel, Stephen, "A Continuum of Disk Scheduling Algorithms", *ACM Transactions on Computer Systems*, February 1987, Vol 5. No. 1.

- Gotlieb, C. C. and MacEwen, H., "Performance of Movable-Head Disk Storage Devices", *Journal of the ACM*, October 1983, Vol 20. No. 4.
- Hofri, Micha, "Disk Scheduling: FCFS vs SSTF Revisited", *Communications of the ACM*, November 1980, Vol 23, No. 11.
- Marshall Kirk McKusick, William Joy, Sam Leffler, and R. S. Fabry, "A Fast FileSystem for UNIX", *ACM Transactions on Computer Systems*, Vol. 2, No. 3, August 1984, pp. 181-197.
- Oney, Walter C., "Queuing Analysis of the Scan Policy for Moving-Head Disks", *Journal of the ACM*, July 1975, Vol 22. No. 3.
- Teorey, Toby J. and Pinkerton, Tad B., "A Comparative Analysis of Disk Scheduling Policies," *Communications of the ACM*, March 1972, Vol 15. No. 3.
- Wilhelm, Neil C., "An Anomaly in Disk Scheduling: A Comparison of FCFS and SSTF Seek Scheduling Using an Empirical Model for Disk Accesses", *Communications of the ACM*, January 1976, Volume 9, No. 1.

Unit - 2 : Linux Operating System

Structure of Unit

2.0 Objective

The objective of this unit is to make you aware of some aspects of Linux operating system. In this unit we will look inside the development of Linux operating system as open source software. We will take a sneak peak at basic commands in general and "file structure" and "shell" in particular. We will also learn about command line interface.

2.1 Introduction

A Unix-like FREE operating system, Linux has become quite popular with PC users around the world. Linux is helpful in true multitasking through its virtual memory, shared libraries, memory management, demand loading and TCP/IP networking. It is distinguished it from other operating systems in the sense that it's source code is available as free software under the GNU General Public License (GPL). This license safeguards and guarantees the freedom of any user to share, modify and again share the modified software. Some recent surveys regarding customer's preferences for OS shows the switching from the Windows NT Operating system to Linux because Linux not needs constant rebooting and it can be easily used for cost-effective computation. The Linux can be easily configured to look like Windows and equally work as good as Microsoft Office. Further to that as the Linux source code is widely available, work-around for hardware defects are reported and patched into the kernel almost overnight.

2.2 Open Source Software

A software program that tries to manage the software and hardware resource of any computer is called operating system (OS). The basic tasks of OS includes performing of control and allocation of memory, prioritizing the processing of instructions, control of input and output devices, facilitation of networking, and management of files.

Unlike earlier OS, today's OS use a mouse for input with a graphical user interface (GUI). How appropriate the OS is, it depends specifically on the CPU. Unlike Windows NT, the Linux and BSD only run on almost any CPU. Since the early 1990s there has been stiff competition between the Microsoft Windows family and the Unix-like family.

The Unix-like family has diversified operating systems. The major subcategories include System V, BSD, and Linux. These systems can run on a wide variety of machine architectures and they have been becoming quite popular in business, as well as at workstations with academic and engineering environments. Some of the Unix systems like Linux & BSD are free or open source variants and that's why they are heavily popular. Over the history, these open source systems have supplanted proprietary ones in most instances.

The Open Source software as the name suggests is open in nature where programmers can read, distribute and change code, so that day by day the code becomes mature. Programmers can adapt it, fix it, debug it, and they can do it at a speed that dwarfs the performance of software developers at conventional companies. The way this developed software becomes more flexible and better than the conventionally developed software.

The community that contributes to open source software, consists largely of programmers who have been giving their contribution for over half a decade. As with more users, the more questions are raised so the Open Source community ensures that answers keep coming, and watches the quality of the answers with a suspicious eye, which results in ever more stability and accessibility.

As open source software, Linux has accepted the challenge of the fast moving world and with the development of internet it has grown past the stage where it was almost exclusively an academic system. As a open source software Linux has been providing more than the operating system. Linux has become the leading alternative to the Microsoft's operating systems, which are installed on almost all new personal computers which use x86-compatible microprocessors.

2.3 Linux system

Unlike Windows, Linux is the best-known example of "open-source" software. The reason is that programs for it are freely available on the Internet and it can not only be obtained without payment but the users are also allowed to modify it. However it is not such free that anyone can do whatever what one wants to do with it. Almost all it is copyrighted by its authors and it's release has been under a variety of different licenses, like GNU General Public License or GPL. Under this license anyone is free to modify the Linux but the source code must be made available under the terms of the GPL after the modification of the software. However, many components of Linux are released under other similar licenses also like red hat.

2.3.1The Development Model

Linux got its name from Linus Torvalds, who thought to develop some sort of freely available academic version of UNIX, and promptly started to code. Today, there are thousands of authors, who collaborate to it through the Internet. Linux Penguin, is the official mascot of the Linux OS that was chosen by Linus Torvalds himself. The Linux Mascot is a contented, cute and cuddly creature

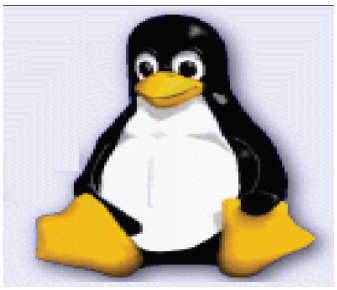


Figure 2.1 The Linux Mascot

In technical terms it is just the core of the operating system, the so called "kernel" because of its interaction directly with the hardware and supervision of the operation of other programs. But it must be mentioned that for a working Linux system many other components are also to be included without which it would not be of much use. The other components include the gcc compiler for programs based on C or C++, the bash shell, the gzip compression utility, the emacs editor and the tar and make. That means the Linux in its working condition is a system that consists of a great software and the "kernel" is only a small part of it. In its packaged form it is called a "distribution". Some of the popular distributions are Debian GNU/Linux, Red Hat, Caldera, Slackware and SuSE. Out of these Debian is entirely non-commercial and it's maintenance is done by thousands of volunteers from around the world and others are commercial. It is also interesting to note that all of them may be suitable for users as per their particular use and there is great variation in the prices of commercial distributions, from less than USD5 to more than USD100.

For all practical purposes, as its name suggests, Linux in fact is a version of Unix. Linux's modern versions are designed to be POSIX compliant. So, it can be said that for an experienced Unix user, it is not quite different from proprietary versions of Unix. However, because of its easier use and as it comes with already useful installed programs, the Linux is "out of box" than any other proprietary version of Unix.

The following figure depicts the architecture of an operating system like Unix. The Unix can be thought of composed of layers of software built around the central hardware, which provides basic services to an user. The layer of software that is nearest to the central hardware and which provides an abstraction of the hardware to the user, that insulates the user from hardware idiosyncrasies is commonly called the operating system or the "*kernel"*. The programs that are built around the "kernel" are independent of the underlying hardware. Linux *per se* refers only to the kernel of the operating system.

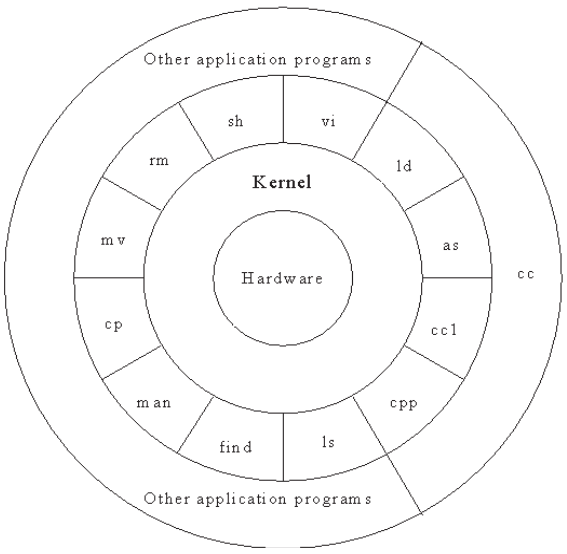


Figure 2.2 Unix like operating System

At first, Linux may seem to be strange to them who are only familiar with commercial, contemporary desktop operating systems like Windows. The reason of strangeness is the primary user interface, the command line of the bash shell, multiple copies of which run in different windows and different virtual screens under the graphical X Window System.

But no one can doubt the extreme power of the bash shell. It not only provides filename completion, command completion and numerous ways to recalling and editing previous commands but powerful programming capabilities also. However, as operating systems of Microsoft generally not work on Linux, moving from Windows to Linux normally requires installing new software.

2.3.2 Hardware Requirements and Installation

To run Linux, the computers must be with x86-compatible processors (i386 or later). Generally, available hardware is used more efficiently by Linux, but there can be problems with very recent hardware and with proprietary devices that use nonstandard protocols. Before installing Linux, it is important to know about the network card, modem, video card, sound card, and printer because sometimes it is quite difficult to know which hardware part is to be supported by Linux. The hardware parts with "Win" as name on it almost certainly will not work with Linux.

The working of Linux is often better with "generic" PCs than with big name computer manufacturers. Relatively some small companies has now started selling computers with preinstalled Linux. It is the widespread acceptability of the Linux because of which IBM, Dell, and several other large computer makers have offered support for Linux on selected servers and workstations.

If all hardware is properly identified and supported, it is quite easy to install Linux and once initial installation of a very basic system is completed the rest of the system can be installed using internet.

2.3.3 Advantages and Disadvantages of Linux

Linux is very stable and multitasks extremely well. In the absence of power failures or hardware failure or extremely ill-behaved program, a system with Linux never crashes. The system with Linux can be used for hours without interruption. Across a network, Linux works well and with Linux office machines can be accessed from home, or from 5000 kilo meters away. Although for text applications the speed of the network is not much important, for graphical applications a high-speed network is essential.

Linux is easy to share data, programs, drafts of papers, and even CPU time and its because of this feature one can put clusters of Linux machines using various types of networking hardware together with the help of the free Beowulf software.

Linux is cheaper than other commercial operating systems. Not only it is free, but it comes with so much free software also. Linux makes an ideal operating system for servers because Linux does not attempt to hide operations or limit its users to do. It is almost infinitely customizable.

The only disadvantage that appears is that Linux does not provide as much commercial software with it as compared to Microsoft's popular operating systems.

2.3.4 Command Line Interface

The Command Line Interface is a means of interaction by the user who uses commands for a computer program in the form of successive command (text). It is better than GUI as it gives more control and options to the user. Besides this using CLI is faster as only a keyboard is pretty much needed. The CLI is the primary mean to interact with most early operating systems like DOS, UNIX etc. This usually implemented

with a shell, which is a program that works to accept commands as text input and then convert them to appropriate operating system functions. Although, nowadays CLI is less widely used by users yet it is still often preferred by advanced computer users, as it often provides a more concise and powerful means to control any operating system.

Below is screenshot of BASH session:

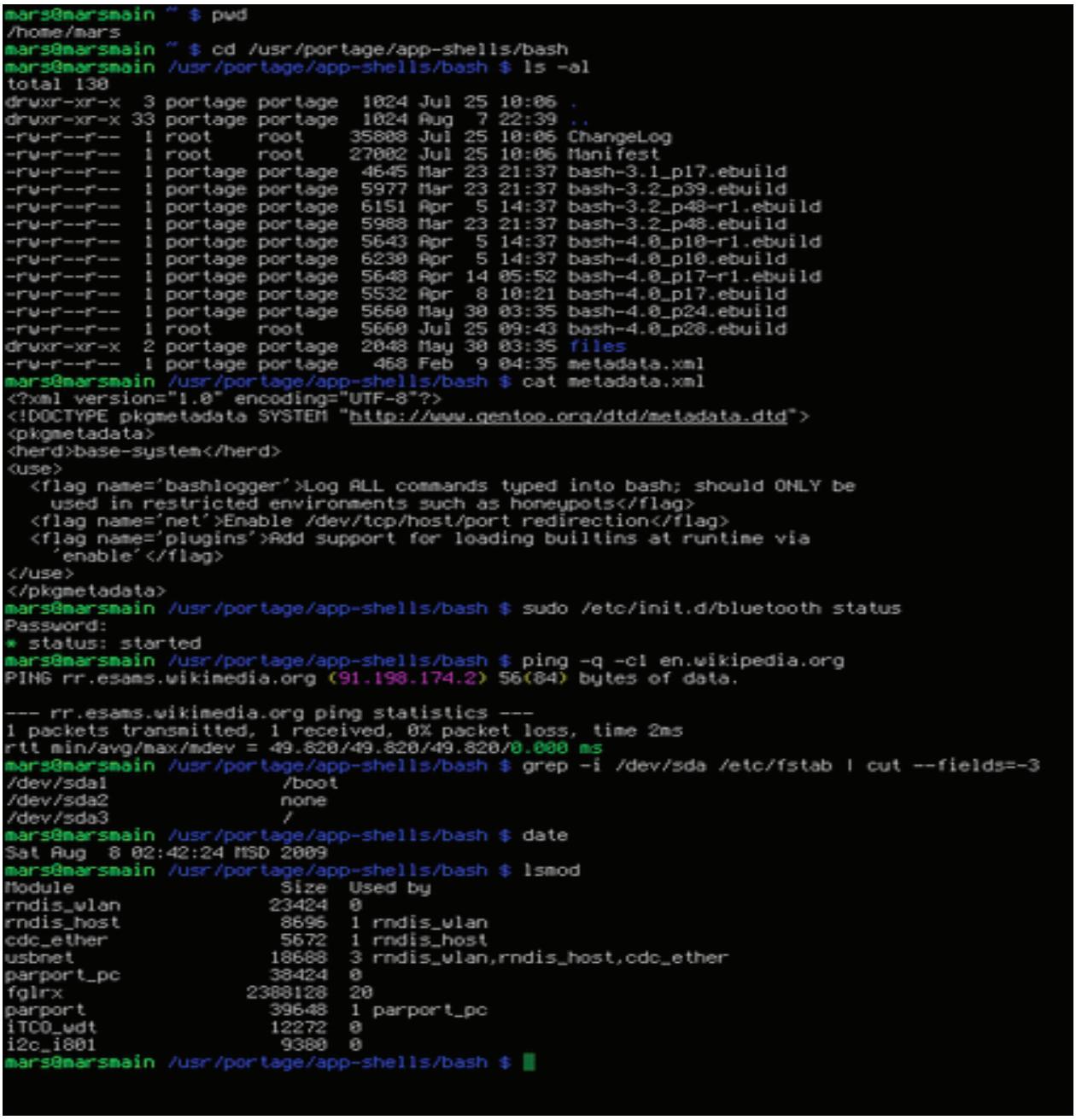


Figure 2.3 Linux shell

2.3.5 Basic Commands

The basic commands in the form of the quickies are as follows:

Quick start commands	*Command Meaning*
ls	Displays a list of files in the current working directory
cd directory	change directories

passwd	change the password for the current user
file filename	display file type of file with name filename
cat textfile	throws content of textfile on the screen
pwd	display present working directory
exit or **logout**	leave this session
man *command*	read man pages on **command**
info *command*	read Info pages on **command**
apropos *string*	search the *what is* database for strings

2.3.6 The Man Page

The manual (man) pages are an overwhelming source of documentation. They are very structured, as shown in the example given below on: **man man**.

The man page is usually read in a terminal window either in graphical mode or in text mode.

Example:

Type the following command and press ENTER

yourname@yourcomp ~> **man *man***

After pressing ENTER, the documentation for **man** will be displayed on screen as:

man(1) man(1)

NAME
man - format and display the on-line manual pages
manpath - determine user's search path for man pages

SYNOPSIS
man [-acdfFhkKtwW] [--path] [-m system] [-p string] [-C config_file]
[-M pathlist] [-P pager] [-S section_list] [section] name ...

DESCRIPTION

man formats and displays the on-line manual pages. If section is specified, man only looks in that section of the manual.name is normally the name of the manual page, which is typically the name of a command, function, or file. However, if name contains a slash (/) then man interprets it as a file specification, like man ./foo.5 or even man /cd/foo/bar.1.gz

See below for a description of where man looks for the manual page files.

OPTIONS
-C config_file
lines 1-27

To browse the next page the space bar is used. To go back to the previous page the b-key is used. The **man** will usually quit while reaching the end and b is typed to leave the man page before reaching the end.

To manipulate man pages using the available key combinations depend on the *pager* used in distribution. Mostly **less** is used to view the man pages and to scroll around.

It can be seen from the above example that usually a couple of standard sections are contained on each man page.

- The first line contains the name of the command and the id of the section in which the man page is going to be located. The man pages are ordered in chapters. There can be multiple man pages for different Commands. For example, the man page from the system admin section, the man page from the user section, and the man page from the programmer section.
- To build the index of the man page, the name of the command and a short description are given. Using the **apropos** command any given search string can be looked upon..
- A Technical notation of all the options is provided by the synopsis of command by and/or arguments this command can take. An option can be thought as a way to execute the command. The argument is what is executed it on. Some commands have no options or no arguments. Optional options and arguments are put in between "[" and "]" to indicate that they can be left out.
- A longer description of the command is given.
- Options with their descriptions are listed. Options can usually be combined.
- Environment describes the shell variables that influence the behaviour of this command.
- Sometimes sections specific to this command are provided.
- A reference to other man pages is given in the "SEE ALSO" section. Experienced users often switch to the "SEE ALSO" part using the / command followed by the search string SEE and press **Enter**.
- Usually there is also information about known bugs (anomalies) and where to report about new bugs.
- There might also be author and copyright information.

Some commands have multiple man pages. For instance, the **passwd** command has a man page in section 1 and another in section 5. By default, the man page with the lowest number is shown.

If another section than the default is to be seen, then, after the **man** command specify it:

man 5 ***passwd***

If all man pages about a command, one after the other, are to be seen the -a to man is used:

man -a ***passwd***

When the end of the first man page is reached, pressing **SPACE** again, the man page from the next section will be displayed

2.3.7 The Info Pages

In addition to the man pages, the Info pages could be read about a command, using the **info** command. Usually, these contain the most recent information and are somewhat easier to use than man pages. The info pages for some commands are referred by the man pages.

An example of info page is as shown below:

Get started by typing **info** ***info*** in a terminal window:

```
File: info.info,  Node: Top,  Next: Getting Started, Up: (dir)

Info: An Introduction
**********************
Info is a program, which you are using now, for reading documentation of computer
programs. The GNU Project distributes most of its on-line manuals in the Info format, so you
need a program called "Info reader" to read the manuals. One of such programs you are using
now.
If you are new to Info and want to learn how to use it, type the command `h' now. It brings
you to a programmed instruction sequence.
To learn advanced Info commands, type `n' twice. This brings you to `Info for Experts',
skipping over the `Getting Started' chapter.

* Menu:
* Getting Started:: Getting started using an Info reader.
* Advanced Info:: Advanced commands within Info.
* Creating an Info File:: How to make your own Info file.
--zz-Info: (info.info.gz)Top, 24 lines --Top-----------------------------

Welcome to Info version 4.2. Type C-h for help, m for menu item.
```

The arrow keys can be used to browse through the text. The movement of cursor on a line that starts with an asterisk can provide the info about the keyword when **Enter** is pressed.

The **P** and **N** keys can be used to go to the previous or next subject. The space bar is used to go to next page, no matter whether a new subject or an Info page for another command has started. The **Q** is used to quit.

2.3.8 File Structure

As in Unix, a simple description is also applicable to Linux, and that is:

“On a system with Unix, everything is a file; if it is not a file, then it is a process.”

However, there are some exceptions like:

- *Directories*: files that are lists of other files.
- *Special files*: the mechanism used for input and output.
- *Links*: a system to make a file or directory visible in multiple parts of the system’s file tree.

- *(Domain) sockets*: a special file type, similar to TCP/IP sockets, providing inter-process networking protected by the file system's access control.
- *Named pipes*: act more or less like sockets and form a way for processes to communicate with each other, without using network socket semantics.

The (ls-l) is used to display the file type. For this the first character of each input line is used as shown below:

```
1aime:~/Documents> ls –l
total 80
-rw-rw-r–1 jaime 1aime 31744 Feb 21 17:56 intro Linux.doc
-rw-rw-r–1 jaime 1aime 41472 Feb 21 17:56 Linux.doc
drwxrwxr-x 2 jaime 1aime 4096 Feb 25 11:50 course
```

The files that are more than just files are some special files like pipes and sockets, but for simplicity, we say that everything is a file. Services, programmes, images, texts etc all are files. The following table shows the types of file.

Table: File types

Symbol	Meaning
-	Regular file
d	Directory
l	Link
c	Special file
s	Socket
p	Named pipe
b	Block device

The **ls –F**, is used to indicate the type of file by suffixing file names with one of the characters "/ =*|@".

Usually, the Linux is thought of in a tree structure as shown below:

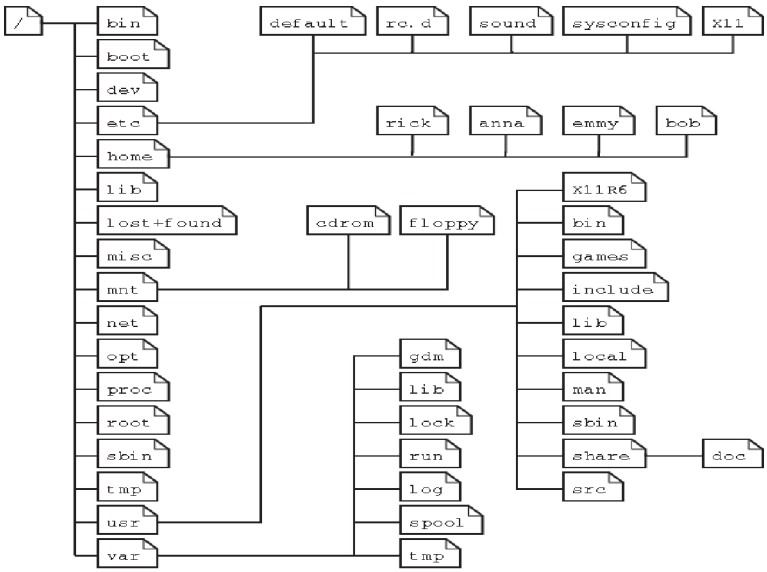


Figure 2.4 Linux File system

The shown tree layout is from RedHat and according to the OS, admin, the mission of the UNIX machine, the layout may vary.

The tree starts at the *slash* (/). This directory contains all underlying files and directories. This is called the *root directory* also. For example:

```
emmy:~> cd /
emmy:/> ls
bin/ dev/ home/ lib/ misc/ opt/ root/ tmp/ var/
boot/ etc/ initrd/ lost+found/ mnt/ proc/ sbin/ usr/
```

Table: Subdirectories of the root directory:

Directory	*Content*
/bin	Programs, common & shared by the system, the administrator and the users.
/boot	The startup files and the kernel,
/dev	Contains references to all the CPU peripheral hardware, represented as files with special properties.
/etc	Most important files are in /etc. The directory is similar to the Control Panel of Windows
/home	Home directories.
/initrd	Information for booting.
/lib	Library files for all kinds of programs needed by the system.
/lost+found	Here are files saved during failures.
/misc	For miscellaneous purposes.
/mnt	Standard mount point for external file systems.
/net	Standard mount point for entire remote file systems
/opt	Typically contains extra and third party software.
/proc	A virtual file system contains information about system resources.
/root	The administrative user's home directory.
/sbin	Programs for use by the system and the system administrator.
/tmp	Temporary space for use by the system, cleaned upon reboot,
/usr	Programs, libraries, documentation etc. for all user-related programs.
/var	Storage for all variable files and temporary files created by users.

In reality, the computer, however, doesn't get tree-structure and instead first it compares file names and inode numbers and then makes up a tree-structure. An *inode is* a kind of serial number that contains following information about the actual data for making up a file:

- Owner of the file.

- File type (regular, directory)
- Permissions on the file Section 3.4.1
- Date and time of creation, last read and change.
- Date and time this information has been changed in the inode.
- Number of links to this file.
- File size
- An address defining the actual location of the file data.

However, no information about the file name and directory is contained by the inode because these are stored is in the special directory files.

The kernel is just like the heart of the body and the communication between the peripherals and underlying hardware is managed by it. The kernel also ensures that daemons and processes starting and stopping held up exactly at the right times. Because of important tasks the kernel has,a special kernel-development mailing list containing huge information has been added and shared on this subject only.

2.3.9 The Shell

A shell is just like a language of talking to the computer. Unlike GUIs, which are almost always less capable, the shell is an advanced way of making communications with the system, as under it two-way conversation and taking initiative is allowed openly. Both communicating partners are equal, so testing of new ideas go on. The shell allows flexibility and task automation for user.

The different shell types are

- **sh** or Bourne Shell: this is the original shell related to UNIX environments. The basic shell with few features is a small program. In POSIX-compatible mode, it is emulated by **bash** shell.
- **bash** or Bourne Again Shell: the flexible, intuitive and standard GNU shell is most advisable for beginners and common users. It is also called *superset* of the Bourne shell (Bourne Again Shell) as it is compatible with the Bourne shell.
- **csh** or C Shell: the syntax of this shell resembles that of the C programming language.
- **tcsh** or Turbo C Shell: It is a superset of the C Shell, as it enhances user-friendliness and speed.
- **ksh** or the Korn shell: It is a superset of the Bourne shell but with standard configuration for UNIX users.
  The file /etc/shells give an overview of known shells on a Linux system:

  ```
  mia:~> cat /etc/shells
  /bin/bash
  /bin/sh
  /bin/tcsh
  /bin/csh
  ```

  To know which shell is under use, the following command is used: **echo $SHELL**

The following features are common in every shell but some commands are only available on systems that support job control. These commands include jobs, fg, bg etc

Table 2.1 : Common Features of Shell

Command	Meaning
>	Redirect output
>>	Append to file
<	Redirect input
<<	"Here" document (redirect input)
|	Pipe output
&	Run process in background.
;	Separate commands on same line
*	Match any character(s) in filename
?	Match single character in filename
[ ]	Match any characters enclosed
( )	Execute in subshell
` `	Substitute output of enclosed command
" "	Partial quote
' '	Full quote (no expansion)
\	Quote following character
$var	Use value for variable
$$	Process id
$0	Command name
$n	nth argument (n from 0 to 9)
$*	All arguments as a simple word
#	Begin comment
bg	Background execution
break	Break from loop statements
cd	Change directories
continue	Resume a program loop
echo	Display output
eval	Evaluate arguments
exec	Execute a new shell
fg	Foreground execution
jobs	Show active jobs
kill	Terminate running jobs
newgrp	Change to a new group
shift	Shift positional parameters
stop	Suspend a background job
suspend	Suspend a foreground job
time	Time a command
umask	Set or list file permissions
unset	Erase variable or function definitions
wait	Wait for a background job to finish

2.4 Summary

The chapter gave an overview of a Linux operating system. First, the major services provided by the operating system were described. Then, the programs that implement these services are described with a considerable lack of detail. Topics like File structure, Linux devices, and command line interface (CLI) system management utilities were included. The advanced Bourne Again Shell (BASH) shell scripting, including looping and decision making logic structures was also explained.

2.5 Self - Assessment Exercise

1. Who owns the data directory in Linux?
2. What command is used to review boot message?
3. What are seven fields in the /etc/passwd file?
4. What account is created when Linux is installed?
5. What is difference between BASH and CSH?
6. Explain the shell structure of Linux.
7. Define Open Source Software? Do you think Linux justifies the definition. Why?
8. Explain Kernel.
9. Which partitioning tool is available in Linux?
10. Differentiate between a process and a programme.
11. Define CLI. Is it better than GUI?
12. What are the types of file system in the Linux?
13. How Linux is different from Unix and Windows.

2.6 References

- Andrew S Tannenbaum and Albert S Woodhull, *OperatingSystems: Design and Implementation*, Second Edition, Prentice Hall of India,1997.
- Mark G Sobel, *Hands-on Linux*, Addison Wesley Longman Publishers,1997.
- Mark G Sobel, *A Practical Guide to Linux*, Addison Wesley Longman Publishers, 1997.
- *The Linux official web site*, http://www.linux.org/.
- *The Redhat web site*, http://www.redhat.com/.
- A feature from the Wired magazine,http://www.wired.com/wired/5.08/linux.html
- *The Cathedral and the Bazaar*, Eric. S. Raymond,http://www.tuxedo.org/~esr/writings cathedralbazaar/
- Teyssi_ere, G. (1998). \XploRe 4.0, An interactive statistical computing environment," *Journal of Applied Econometrics*, 13, 673{679.
- Welch, M. and L. Kaufman (1996). *Running Linux*, Second Edition. Sebastopol, California: O'Reilly and Associates.

Unit - 3 : Linux Files and Directories

Structure of Unit

3.0 Objective

The objective of this unit is to make you aware of files and directories in Linux operating system. In this unit we will look inside what type of directory is made up of. We will take a sneak peak at the "directories" in general and "file system" in particular. We will also see how these all correlate to make the Linux system operative.

3.1 Introduction

A *file system* is the data structure or method that is used to keep files on a hard disk; that means, the way of organizing the on the disk. There is a difference between a disk and the file system. There are a few programs whose operating is directly on the disk; if there exist a file system, it will be seriously destroyed. There are certain program whose operating is on a file system, and therefore they won't work on a disk that doesn't contain the file system. For a hard disk to be used as a file system, initialization and the writing of data on the disk is needed.

Mostly the UNIX file structure has a structure of the similar nature, but the exact details vary quite. The central concepts are:

- ***superblock:*** *The whole* information about the file structure like its size is contained by it.

- ***Inode:*** All information about a file with the exception of its name is contained by it. The storage of name is in the directory. An entry of directory has a filename and the representative number of the file is the inode. The inode has numbers of several blocks that are used for storing the data in the file.
- ***Indirect Block:*** In the inode if more space is needed to store data, the allocation of blocks is dynamically. These dynamically allocated blocks are called indirect blocks. As the name suggests, to find the data block, its number is to be found in the indirect block first.

3.2 The File/Directory Hierarchy

Linux, like Unix, has also chosen a single hierarchical directory structure. Everything starts from the root directory (/), and then expansion takes place into sub"directories.

The sorting of directories in the Linux is in descending manner; from the root directory to the sub directories according to their importance. The use of the front slashes / is to simply follow the UNIX tradition. Like Unix, Linux also chooses to be case sensitive.

The majority of Linux files is 'Second Extended File Systems', ('EXT2'). Within these file systems Linux determines which files are to be stored in which directories programs. In Linux, the documentation of programs is into:

/usr/share/doc/[program- name],
the documentation of man pages is into
/usr/share/man/man[1-9]
and info pages into
/usr/share/info.

The merging of all of these is to put the files and directories into and with the system hierarchy:

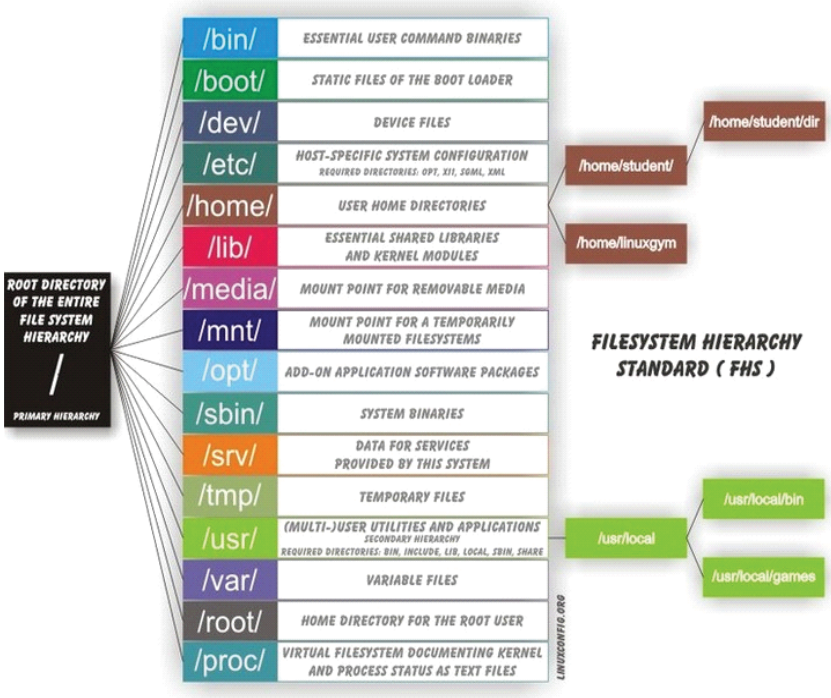


Figure 3.1 Linux File System Hierarchies

This unified file structure of Linux offers several advantages as shown in the example of the following /usr directory. This sub"directory has most executables of the system. In Linux, to mount it off another partition, an innumerable set of protocols such as NFS (Sun) can be chosen. This directory is completely transparent and local:

	shareable	unshareable
ic	/usr	/etc
	/opt	/boot
	/var/mail	/var/run
	/var/spool/news	/var/lock

cription is applicable to Linux, like Unix is:

"On a system with"Unix, everything is a file; if it is not a file, then it is a process."

However, there"are some exceptions like:

- *Directories*: files that are lists of other files.
- *Special files*: the mechanism used for input and output.
- *Links*: a system to make a file or directory visible in multiple parts of the system's file tree.
- *(D'main) sockets*: a special file type, similar to TCP/IP sockets, providing inter-process networking protected by the file system's access control'
- *Named pipes*: act more or less like sockets and form a way for processes to communicate with each other, without using network socket semantics.

The (ls-l) is used to display the file type. For this the first character of each input line is used as shown below:

```
jaime:~/Documents1aime-l
total 80
-rw-r--r-- 1 jaime jaime –44 Feb 21 17:56 intro Linux.doc
-rw-rw-r-- 1 jaime jaime 41472 Feb 21 17:56 Linux.doc
drwxrwxr-x 2 jaime jaime 4096 Feb 25 11:50 course
```

The files that are more than just files are some special files like pipes and sockets, but for simplicity, we say that everything is a file. Services, programmes, images, texts etc all are files. The following table shows the types of file.

Table: File types

Symbol	Meaning
-	Regular file
d	Directory
l	Link
c	Special file
s	Socket
p	Named pipe
b	Block device

The **ls -F**, is used to indicate the type of file by suffixing file names with one of the characters "/=*|@".

3.3 The Path

The PATH environment variable takes care of giving full path name to the command. Those directories are listed in the system through this variable where executable files can be found. This way it saves a lot of typing and memorizing locations of commands.

Naturally, it contains a lot of directories with /bin somewhere in their names. For example the **echo** command is used to display the content ("$") of the variable PATH:

rogier:> **echo $PATH**

/opt/local/bin:/usr/X11R6/bin:/usr/bin:/usr/sbin/:/bin

The path starts from the / or root directory. If it starts with a slash then it is called an absolute path, since there can be no mistake: only one file on the system can comply. In either case it is called relative path. In relative paths too the . and .. are used to indicate for the current and the parent directory.

3.4 The Root Directory

The following directories, or symbolic links to directories, are required in /, the root directory.

Type	Meaning
/bin	Essential command binaries
/boot	Static files of the boot loader
/dev	Device files
/etc	Host"specific system configuration
/lib	Essential shared libraries and kernel modules
/media	Mount point for removeable media
/mnt	Mount point for mounting a filesystem temporarily
/opt	Add"on application software packages
/sbin	Essential system binaries
/srv	Data for services provided by this system
/tmp	Temporary files
/usr	Secondary hierarchy
/var	Variable data

It is called the root directory, because it serves like the root of a tree and all directories grow though it and look like the branches of a tree, as shown below:

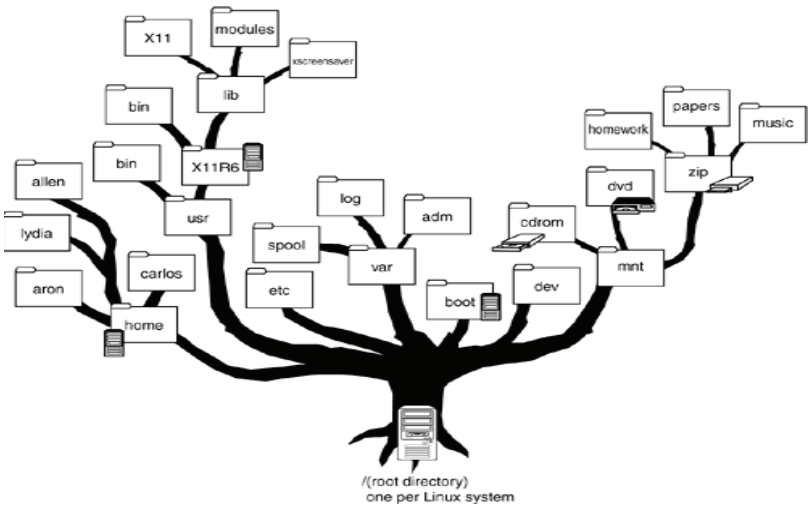
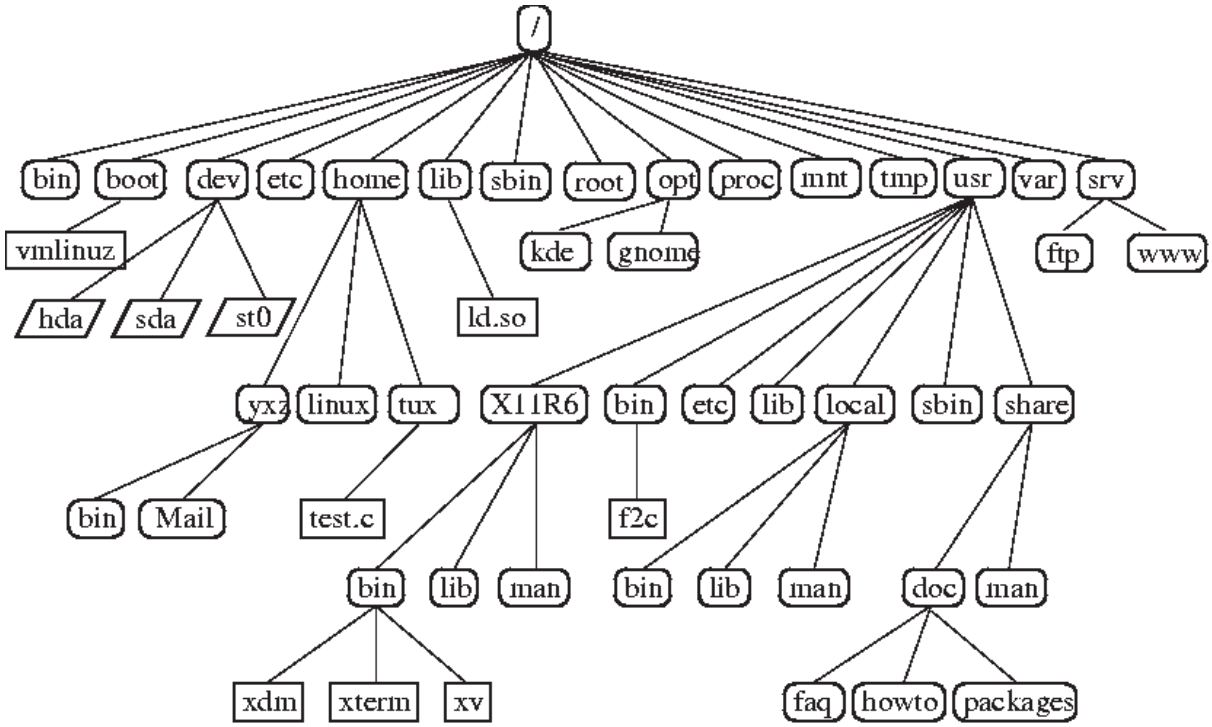


Figure 3.2 Root Directory

3.5 The File Structure

Linux file structure starts with the root directory and it contains the following sub"directories, as shown and explained below:



bin/ dev/ home/ lost+found/ proc/ sbin/ usr/ cdrom/ opt/ vmlinuz boot/ etc/ lib/ mnt/ root/ tmp/ var/ dvd/floppy/ initrd/ /tftpboot

Figure 3.3 File and Directory Structure

1.5.1 /bin

This contains many useful commands used by both the administrator and non"privileged users. The shells like bash, csh, etc are contained by it. Because of this, the binaries are considered to be essential in this directory. The essential programs contained by it must be available even if only the disk containing / is mounted. The programs which boot scripts may depend on are also contained by it.

There are no subdirectories in this directory and the location of the following commands is here:

cat	Utility to concatenate files to standard output
chgrp	Utility to change file group ownership
chmod	Utility to change file access permissions
chown	Utility to change file owner and group
cp	Utility to copy files and directories
date	Utility to print or set the system data and time
dd	Utility to convert and copy a file
df	Utility to report filesystem disk space usage
dmesg	Utility to print the kernel message buffer
echo	Utility to display a line of text

false	Utility to do nothing, unsuccessfully
hostname	Utility to show or set the system's host name
kill	Utility to send signals to processes
ln	Utility to make links between files
login	Utility to begin a session on the system
ls	Utility to list directory contents
mkdir	Utility to make directories
mknod	Utility to make block or character special files
more	Utility to page through text
mount	Utility to mount a filesystem
mv	Utility to move/rename files
ps	Utility to report process status
pwd	Utility to print name of working directory
rm	Utility to remove files or directories
rmdir	Utility to remove empty directories
sed	The 'sed' stream editor
sh	The Bourne command shell
stty	Utility to change terminal line settings
su	Utility to change user ID
sync	Utility to flush filesystem buffers
true	Utility to do nothing, successfully
umount	Utility to unmount file systems
uname	Utility to print system information

3.5.2 /boot

Everything that is required for the boot process is contained by this directory with the exception of configuration files that are not needed at boot time. This indicates that the data used before the kernel's beginning of executing user is stored by it. The data may include redundant master boot records and some other important files needed to boot. Some of the boot files are:

/boot/boot.0300 backup master boot record
/boot/boot.b the basic boot sector. A symbolic link to one of four files /boot/boot- bmp.b, /boot/boot- menu.b, /boot/boot- text.b, /boot/boot- compat.b
/boot/chain.b Used to boot non- Linux operating systems.
/boot/config- kernel- version Installed kernel configuration.
/boot/os2_d.b Used to boot to the 0S/2 operating system.
/boot/map Contains the location of the kernel.
/boot/vmlinuz Symbolic link to the kernel.
/boot/grub Contains the GRUB configuration files .
/boot/grub/device.map Maps devices in /dev.
/boot/grub/grub.conf Grub configuration file.
/boot/grub/messages Grub boot- up welcome message.
/boot/grub/splash.xpm.gz Grub boot- up background image.

3.5.3 /dev

The special or device files are located in /dev. The very interesting part of this directory is it that highlights one important aspect of the Linux "everything is a file or a directory."

This directory has hda1, hda2 etc. files that represent the various partitions on the first master drive of the system.

/dev/cdrom represent Cd rom
/dev/fd0 represent floppy drive.
/dev/dsp represent speaker device.
/dev/lp0 represent printing
/dev/js0 represent Standard game port joystick /dev/dsp represent audio device

3.5.4 /ls

The ls means list. It works in a similar way the dir command works in DOS. The typing of ls provides a listing of all the files in the current directory. There may be "hidden" files whose name start with a dot and to view them, the -a flag is used with the ls command, i.e. ls -a.

To view further information about the files, the -l flag is used with ls, i.e. ls-1. This command will show the file permissions and the file size. To have a list of all the subdirectories, the -R flag is used with the ls command, i.e. ls -R, which is a rough similar to the dir /s command in DOS.

On putting flags together, ls-1aR, one can view all the files in a directory with their permissions/size and through the subdirectories.

The command ls **–al** shows a long list of files with their properties and the destinations. The command **ls -latr** displays the same files in reversed order. Some examples are:

krissie:~/mp3> **ls**

Albums/ Radio/ Singles/ gene/ index.html

krissie:~/mp3> **ls -a**

./ .thumbs Radio gene/

../ Albums/ Singles/ index.html

krissie:~/mp3> **ls -l Radio/**

total 8

drwxr-xr-x 2 krissie krissie 4096 Oct 30 1999 Carolina/

drwxr-xr-x 2 krissie krissie 4096 Sep 24 1999 Slashdot/

krissie:~/mp3> **ls -ld Radio/**

drwxr-xr-x 4 krissie krissie 4096 Oct 30 1999 Radio/

In order not to use any option to ls most Linux versions has **ls** *aliased* to colour-ls by default. Every file type has given its own color. The standard scheme is in /etc/DIR_COLORS:

Table3.1 Color-ls default color scheme

Color	File type
blue	directories
red	compressed archives
white	text files
pink	images
cyan	links
yellow	devices
green	executables
flashing red	broken links

3.5.5 /etc

This directory is like the nervous system. All system related configuration files are contained in it or in its sub"directories. This directory must be back up regularly so that a lot of re"configuration can be saved if re"installation is needed. No binaries are located here.

For example, /etc/X11/ contains all the configuration files for the X Window System.

3.5.6 /pwd

It stands for "Print Working Directory" (present working directory). It is simply useful to show the directory in the use at the moment for scripting and referring to said current directory.

Unlike other commands, it is always almost used just by itself,

pwd

It is rarely used with options and never used with file names. This command is used when the name of the working directory is not shown by the shell. Typing of 'pwd' command results in printing of the name of working directory. This command does have only option of '-LP' to print the same. There are neither '-L' , '-P' options nor '—help' and '—version' options. These options when used with this command would give you '**invalid option**' in Ubuntu desktop. For example:

```
kucing@ubuntu-laptop:~$ pwd --help
bash: pwd: --: invalid option
pwd: usage: pwd [-LP]
```

pwd is also useful to confirm that the current directory has been actually changed.

3.5.7 /proc

This is very special directory because it is also a virtual file system and generally referred as a process information pseudo"file system. This means that no 'real' files are there but system information for runtime like system memory, hardware configuration, etc is there. This is also regarded as a control and information centre for the kernel. The files located in this can be altered and with this kernel parameters can be read or edit while the system is running.

The most distinctive feature this directory's files is that all files have a size of 0. The exception is of kcore, mtrr and self files.

The listing of directory looks like as

```
total 525256
dr- xr- xr- x 3 root root 0 Jan 19 15:00 1
dr- xr- xr- x 3 daemon root 0 Jan 19 15:00 109
dr- xr- xr- x 3 root root 0 Jan 19 15:00 170
dr- xr- xr- x 3 root root 0 Jan 19 15:00 173
dr- xr- xr- x 3 root root 0 Jan 19 15:00 178
dr- xr- xr- x 3 root root 0 Jan 19 15:00 2
dr- xr- xr- x 3 root root 0 Jan 19 15:00 3
dr- xr- xr- x 3 root root 0 Jan 19 15:00 4
dr- xr- xr- x 3 root root 0 Jan 19 15:00 421
dr- xr- xr- x 3 root root 0 Jan 19 15:00 425
dr- xr- xr- x 3 root root 0 Jan 19 15:00 433
dr- xr- xr- x 3 root root 0 Jan 19 15:00 439
```

3.5.8 /cd

The cd directory, as the name suggests, is a type of command used to change directory. The use of cd command is not quite easy as use of it clearly means the dealing with directory. The man page for cd command is as given below:

```
NAME
      cd - Change working directory
SYNOPSIS
      cd ?dirName?
_____
DESCRIPTION
      Changing the current working directory to new dir or to the home directory, if new dir is
not given. The working directory in use is a per-process resource; the cd command changes
it for all interpreters and all threads.
```

Another convenient feature of cd is the ability for any user to return directly to its home directory by merely using a tilde as shown below:

cd ~

There are only two options with cd but neither of them is used commonly. The *-P* option asks cd to use the physical directory structure and *-L* option forces it to follow symbolic links.

3.5.9 /mkdir

This directory is just like a folder to keep related files or sub-directories together at one place. The functioning of it by keeping all files in the proper directory make Linux easier and tidy to manage. The mkdir command

works similar to mkdir command of DOS. The creation of directories with the specified names supplied after it can be done by using this command. The format is **mkdir <directory1 directory2 directory3 ...>**.

Although with this a new directory is created, the mkdir command is one of the mostly used commands. Tthe manual page example of mkdir is as shown below:

```
NAME
       mkdir - make directories
SYNOPSIS
       mkdir [OPTION] DIRECTORY...
DESCRIPTION
       Creates the directories
       Mandatory arguments to long options are mandatory for short options too.      -m, --
mode=MODE
              set file mode (as in chmod), not a=rwx - umask
       -p, --parents
              no error if existing, make parent directories as needed
       -v, --verbose
              print a message for each created directory
       --help display this help and exit
       --version
              output version information and exit
```

With the mkdir command multiple directory can also be created. For example:

```
bill@slackware:~$ mkdir linux tutorial guide example
bill@slackware:~$ ls
example/ guide/
linux/ programs/
tutorial/
bill@slackware:~$
```

3.5.10 /rmdir

To delete or remove unwanted directories, to make it clean and tidy, the rmdir directory is used.

The rmdir is just opposite tool of mkdir. Unlike mkdir, the rmdir command removes directory, but only empty directory. The man page of rmdir is as follows:

```
NAME
       rmdir - remove empty directories
SYNOPSIS
       rmdir [OPTION]... DIRECTORY...
DESCRIPTION
       Remove the directories
         Fail-on-non-empty
       -p, –parents
       Remove DIRECTORY and its ancestors.
       -v, –verbose
       output a diagnostic for every directory processed
       --help display this help and exit
       --version
              output version information and exit
```

Like mkdir, many options are not available with rmdir too. The syntax is similar to the mkdir, **rmdir <new directory>**. However, rmdir is not much popular as with it only an empty directory can be deleted and even if a directory contains small file or child directory, then it is of no use.`3.5.11 /cat`

3.5.11 /cat

It is one of the most frequently used directories for displaying, combining copies and creating new files for the text files in use..

The general syntax of cat is

cat [options] [filenames] [-] [filenames]

It is most commonly used to read the contents of files. To view a text file and read it, typing of the *cat* followed by a space followed the name of the file will display the contents of a file:

cat file1

The cat is used for together stringing of copies of the contents of files. The oginal files don't get affected because concatenation occurs only to the copies. For example with the following command copies of the contents of the three files *file1*, *file2* and *file3 is concatended*:

cat file1 file2 file3

The cat is used for creation of files also, especially small files. For example, to create a new file, following command can be used:

cat > file1

3.5.12 /more

A screen full of a text ûle is displayed with the help of this command. With this a text ûle can be looked through without invoking an editor, printing the ûle, or trying to pause the terminal. After displaying the text at a time with this command the text can be searched, scrolled backwards and forwards. However, once a information passes away, it cant be seen again.

The syntax of this command is:

more[options]filename

Following are the options that can be used:

-c	Clear screen before displaying.
-e	Exit immediately after writing the last line of the last file in the argument list.
-n	Specify how many lines are printed in the screen for a given file.
+n	Starts up the file from the given number.

For Example,

more -c index.php

Clears the screen before printing the file .

3.5.13 /less

The less command is considered better than more. Like more, a screen of information can be displayed with it in a text ûle. This command allows quick view of any file or any section of that file. As the whole file is not required to be loaded in memory, it starts up faster on large files. Unlike the more command, besides scrolling forward, it can scroll back as well. It just prints the text in the given file, and not allows editing or manipulating ofthe text.

The Syntax is

less [options] filename

Following are the options that can be used with this command:

-c	Clear screen before displaying.
+n	Starts up the file from the given number.
:p	Examine the pervious file in the command line list.
:d	Remove the current file from the list of files.

For example

less +3 index.php

Start printing from 3rd line of the file.

3.5.14 /lpr

This command is used to print files. To print, the files are named on the command line to be sent to the printer. The lpr uses the standard input to read the print file if there is no list of files on the command line.

SYNTAX

lpr [-E] [-P *destination*] [-# *num-copies* [-l] [-o *option*] [-p] [-r] [-C/J/T *title*] [*file(s)*]

OPTIONS

The following options to be used **lpr**:

-E	Forces encryption when connecting to the server.
-p	Prints files to the named printer.
-#	Sets the number of copies to print from 1 to 100.
-C	Sets the job name.
-J	Sets the job name.
-T	Sets the job name.
-I	Specifies that the print file is already formatted for the destination
-o	Sets the job name.

3.5.15 /temp

The temp directory contains mostly those files, which are required temporarily. This is used to create lock files for storage of data temporarily. However as many of the files in it may be important for running programs so the files must not be deleted without knowing them as deletion of them may result in a system crash.

Usually, this is cleared out at boot or at shutdown by the local system so that people and programs may not assume that any files or directories in this directory are preserved between invocations of the program.

3.6 Summary

Proper file permissions are an extremely important part of ensuring that your website is secure. Determining the correct file permissions for any specific file requires one to know what type of information contained in the file and the purpose of that information. This chapter explained configuration files and directories on a Linux system that control user permissions, system applications, daemons, services, and other administrative tasks in a multi-user, multi-tasking environment. These tasks include managing user accounts, allocating disk quotas, managing e-mails and newsgroups, and configuring kernel parameters. This chapter also classified the config files present on a Red Hat Linux system based on their usage and the services they affect.

3.7 Self - Assessment Exercise

1. Describe three different ways of setting the permissions on a file or directory to r—r—r—. Create a file and see if this works.
2. Change to the home directory of another user directly, using cd ~username.
3. What is the difference between listing the contents of directory play with ls -l and ls -L?
4. Imagine you were working on a system and someone accidentally deleted the ls command (/bin/ls). How could you get a list of the files in the current directory?
5. Experiment with the options on the ls command. What do the d, i, R and F options do?

3.8 References

- The UNIX programming environment, Brian W. Kernighan, Rob Pike, Prentice Hall, New Jersey,1984.

- Newnes UNIX Pocket Book, Steve Heath, Butterworth”Heinemann, Great Britain, 1998.
- Suse Linux Installation and Configuration, Nazeeh Amin El”Dirghami & Youssef A. Abu Kwaik, QUE Corporation, USA, 2000.
- Inside Linux, Michael J. Tobler, New Riders Publishing, USA, 2001.
- Linux in a Nutshell 2nd Edition, Ellen Siever, O’Reilly & Associates Inc., CA, USA, 1999
- Using Caldera OpenLinux Special Edition, Allan Smart, Erik Ratcliffe, Tim Bird, David Bandel, QUE Corporation, USA, 1999.
- Linux System Security (The Administrator’s Guide to Open Source Security Tools), Scott Mann & Ellen L. Mitchell, Prentice—Hall, New—Jersey, 2000.
- XFree86 For Linux (Uncommon Solutions for the Technical Professional), Aron Hsiao, QUE Corporation, USA, 1999.
- Complete Idiot’s Guide to Linux Second Edition, Manuel Alberto Ricart, QUE Corporation, USA, 1999.
- Lions’ Commentary on UNIX 6th Edition with Source Code, John Lions, Peer”to”Peer Communications Incorporated, USA, 1996.
- The Linux System Administrators’ Guide Version 0.6.1, Lars Wirzenius, liw@iki.fi, Finland, 1998.
- SAMS Teach Yourself Shell Programming in 24 Hours, Sriranga Veerararaghavan, SAMS Publishing, USA, 1999.
- 433"252 Software Development: Principles and Tools, Zoltan Somogyi, Les Kitchen, The University of Melbourne, Department of Computer Science and Software Engineering, Australia, 2002.
- The Advanced Linux Pocketbook, Ashton Mills, ashtonmills@bigpond.com, ACP Publishing Pty Ltd, Australia, 2001.
- http://www.linuxjournal.com/article.php?sid=1104

Unit - 4 : Shell Scripts and Programming

Structure of Unit

4.0 Objective

The objective of this unit is to make you aware of some aspects of shell in Linux operating system. In this unit we will look inside what type of shell is made up of. We will take a sneak peak at the "variables" in general and "scripts" or "parameters" in particular. We will also see how these all correlate to make the Linus system operative.

4.1 Introduction

The language understood by the computer is called binary language. However to give instructions using binary language is quite difficult to read and write. Therefore operating systems have developed a special program known as Shell.

Shell accepts instructions in English and translates those into binary language understood by computers, as shown below:

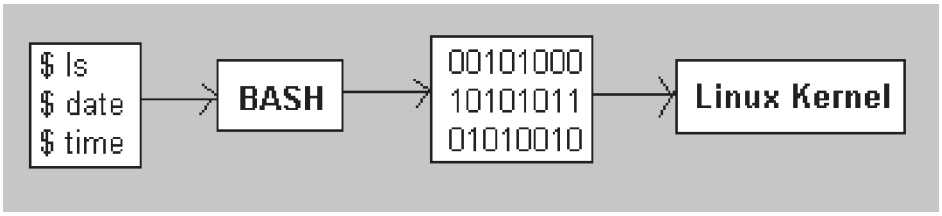


Figure 4.1 Shell

Hence shell can be considered as means of environment provided for user interaction. Shell is a commanding language interpreter, which executes command read from the standard devices or files.

Following are some most popular shells used by Linux:

Shell Name	*Developed by*	*Remark*
BASH (Bourne-Again Shell)	Brian Fox and Chet Ramey	Most common shell in Linux. Freeware shell
CSH (C SHell)	Bill Joy	The C shell's syntax and usage are very similar to the C programming
KSH (Korn SHell)	David Korn	———

The command given from user is used by any of the above and then it is told to Linux O/s what user wants. The command given by keyboard is called command line interface.

How to use Shell

Following are the commands that are needed to be typed to use shell:

The purpose	Syntax of command	Example
To know type of shell	shell	$ echo $SHELL
To see date	date	$ date
To know system user	who	$ who
Print working directory	pwd	$ pwd
List of files in cd	ls or dirs	$ ls
To create text file	cat	$ cat > myfile
To text see files	cat (file name}	$ cat myfile
To display full screen file	more(file name}	$ more myfile
To rename file	mv {file1} {file2}	$ mv sales sales99
To send mail to other	mail {user-name}	$ mail ashish
To create multiple files	ln {oldfile}(newfile}	$ ln Page1 Book1
To remove file	rm	$ rm myfile
Read your mail	mail	$ mail

4.2 Shell Variables

We know that the processing of information or data is kept in RAM. To hold this data, RAM is divided into small locations, and each of its location is given a unique number called memory address. This memory address can further be given a unique name by programmer and that is called memory variable or simply variable. It may take different values, but only one at a time.

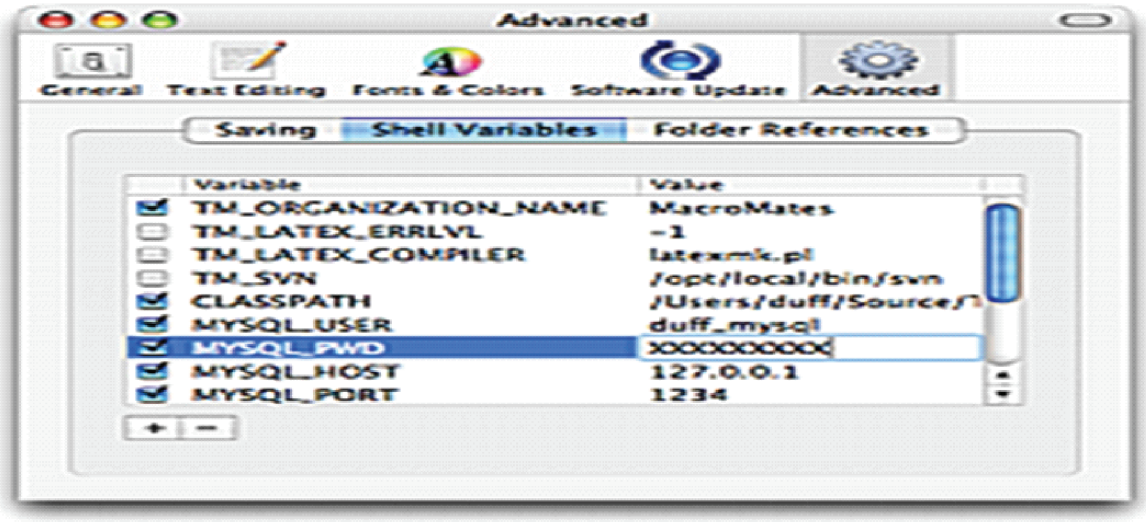


Figure 4.2 Shell Variables

In Linux, two types of variable are there:

1) **System variables** – Creation and maintenance by Linux itself defined in CAPITAL LETTERS. For example

   echo $USER

   echo $PATH

2) **User defined variables (UDV) -** Creation and maintenance by user defined in lower LETTERS.

All variables are stored and considered as strings, even if numeric values are assigned to them. As these are case sensitive, just name should be used while assigning a value to them. No spaces should be there on either side of the equals sign. The contents of a variable can be accessed within the shell by preceding its name with a $. For example

```
myname=A [ use quotes if the value contains spaces ]
myos=Linux
text = 1+2
echoYour name:$myname [ A ]
echoYour   os:$myos [ Linux ]
echo $text [ 1+2 ]
```

If a $variable expression is enclosed in double quotes, when the execution takes place, it's replaced with its value. If it is enclosed in single quotes, no substitution takes place. The special meaning of the $ symbol can be removed by prefacing it with a \.

Some of the important System variables are as given below:

System Variable	Meaning
BASH=/bin/bash	Our shell name
BASH_VERSION=1.14.7(1)	Our shell version name
COLUMNS=80	No. of columns for our screen
HOME=/home/ anurag	Our home directory
LINES=25	No. of rows for our screen
LOGNAME=	students Our logging name
OSTYPE=	Linux Our o/s type : -)
PATH=/usr/bin:/sbin:/bin:/usr/sbin	Our path settings
PS1=[\u@\h\W]\$	Our prompt settings
PWD=/home/students/Common	Our current working directory
SHELL=/bin/bash	Our shell name
USERNAME=anurag	User name currently logged in

To define User Defined Variable the following syntax is used:

Syntax: *variablename=value*

For example **$ no=10** #

To Name variables, the following are the rules to be followed:

(1) The name must begin with underscore or alphanumeric character and it must be followed by either one or more alphanumeric characters. For example:

HOME, SYSTEM_VERSION

(2) No spaces should be put on either side of the equal sign. For example

$ no=10

(3) To define NULL variable the following is used. For example

$ vech= will print no value

(4) The symbols ?,* etc, must not be used to name variable.

To print UDV the following syntax is used

Syntax: *$variablename*

For example **$ echo $vech** will print 'Bus'

4.3 Environment Variables

A Linux system is quite complex to keep a lot track of little details that come into play in interactions with other programs and no user wants to pass on these details to every program that gets run. As a mechanism, so users develop an environment. The environment can be defined as the conditions of running a program in a variable manner that is, it can be altered and played by user, as is only right in a Linux system.

The environment variables are simply a collection of values and variables that are passed to the shell or to any shell script at the starting of program. Basically, the information contained by them may be used by the program to modify its behaviour.

The set command is used to display all the shell variables, environment variables, other shell parameters and any shell function that has been defined. As the set command displays much more variables than the env command, so every shell contains environment variables.

In all systems like Unix, each has its own set of these variables. At the time of creation of process, a duplicate environment of its parent process is inherited with the exception for explicit changes that must be done between execution and running fork. All OS have environment variables; however same variable names are not used by any of them. The values of environment variables for configuration purposes can be accessed by the running programme itself.

Environment variables examples include:

- **PATH -** ls of paths. When a command is typed without the provision of full path, list of directories is checked if it contains a path that can lead to the command.
- **HOME -** indicates location of home directory of User in the file system.

- **HOME/{.AppName} –** To store application settings for programmatic purposes. APPDATA (roaming), LOCALAPPDATA or PROGRAMDATA (shared between users) is used.
- **TERM –** Indicates the use of the type of terminal. For example, dumb or vt100.
- **PS1-** specifies the display of the prompt in the Bourne Shell and its variants.
- **MAIL -** indicates where mail is to be found.
- **TEMP –** specifies location of processes/store of temporary files

All environment variables and their values are displayed by the commands set, env and printenv. The set and env are also used for setting environment variables and are their incorporation is often directly into the shell. The printenv is used to print a single variable by giving the name of the said variable as the sole argument to the command.

In Linux, the following commands are used:

- export *VARIABLE=value* # for bash, Bourne and related shells
- setenv *VARIABLE value* # for csh and related shells

- Local to process
  Environment variables are local to the process in which they were set. This implies that if two shell processes are spawned resulting in the change of value of one environment variable, that change won't be seen by the other.
- Inheritance
  A child process inherits all the environment variables and their values from the parent process by forking then replacing itself with the program to be called.
- Case-sensitive
  The environment variables are case-sensitive.
- Persistence
  The persistence of environment variables can be session-wide or system-wide.

Following is the example with the PATH environment variable:

Listing of Environment Variables with set

```
% set
PATH=/usr/local/lib/qt/bin:/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:
/usr/openwin/bin:/usr/games:.:/usr/local/ssh2/bin:/usr/local/ssh1/bin:
/usr/share/texmf/bin:/usr/local/sbin:/usr/sbin:/home/logan/bin
PIPESTATUS=([0]="0")
PPID=4978
PS1='\h:\w\$ '
PS2='> '
PS4='+ '
PWD=/home/logan
QTDIR=/usr/local/lib/qt
REMOTEHOST=ninja.tdn
SHELL=/bin/bash
% unset VARIABLE
```

An environment variable is not similar like a shell variable. However a shell variable can become an environment variable if we run the env command.

The export command makes an environment variable a shell variable. This is done by using the -n option with that. This does not destroy the variable, but it is no longer remains an environment variable.

However once a shell variable becomes environment variable, it remains an environment variable even if its value gets changed. Hence, to change its value every time, there is no meaning to use export command on a variable.

4.4 Shell Scripts

Two different ways can be used for writing shell programs. A sequence of commands can be typed and the shell can be allowed to execute them interactively. The commands can be stored in a file that can then be invoked as a program. This way of storing commands in shell is known as Shell Script. In other words it can be said that a shell script is a collection of commands all working with a little bit of programming syntax thrown in to it as shown below:

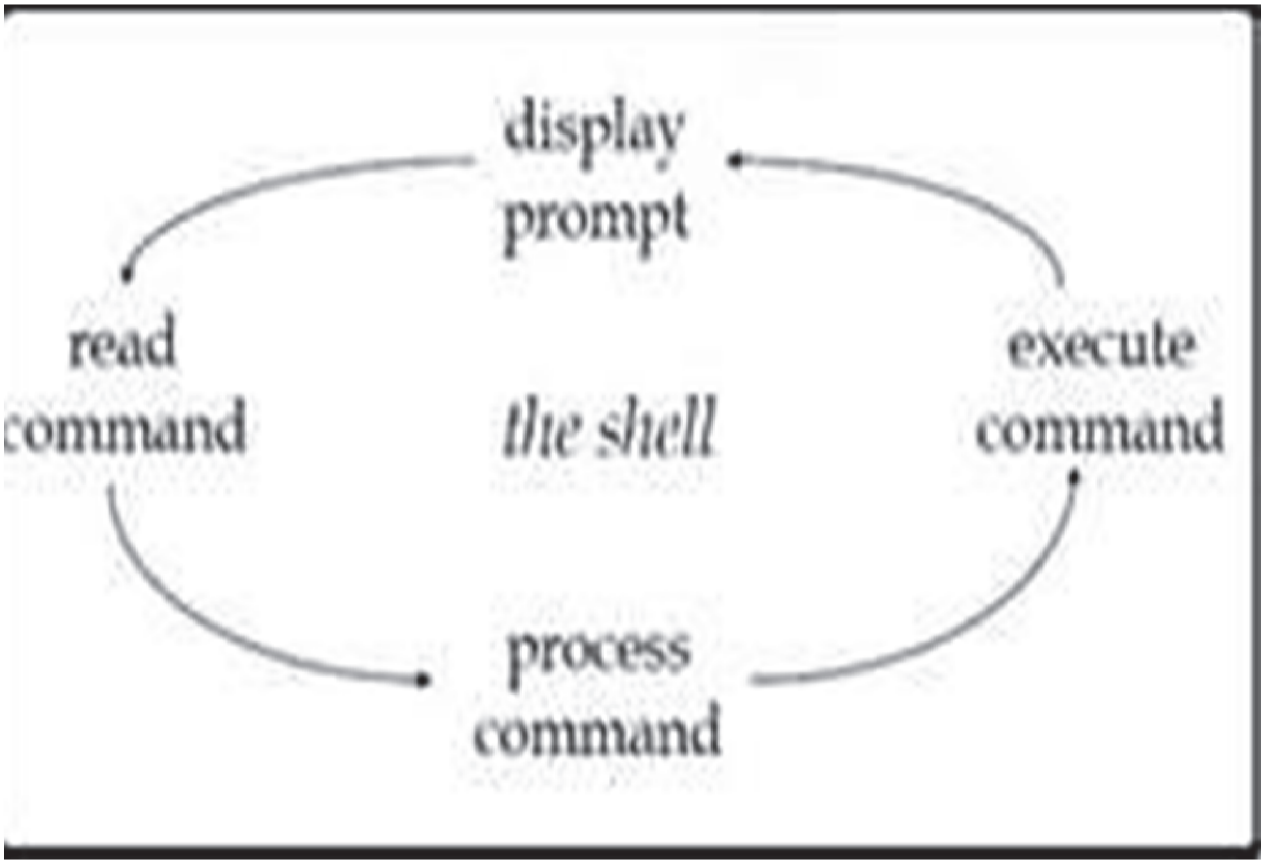


Figure 4.3 Shell Scripting

Shell script is a fundamental part of an operating system and as a ubiquitous feature of UNIX-like OS, it represents a way of writing certain types of command-line tools. The command-line tools are written in such a way that the script works on a fairly broad spectrum of computing platforms.

The performance of shell scripts can be limited because they are written in an interpreted language whose power comes from executing external program. However, they represent a powerful tool to bootstrap other technologies because they can be executed without any additional effort on every modern operating system.

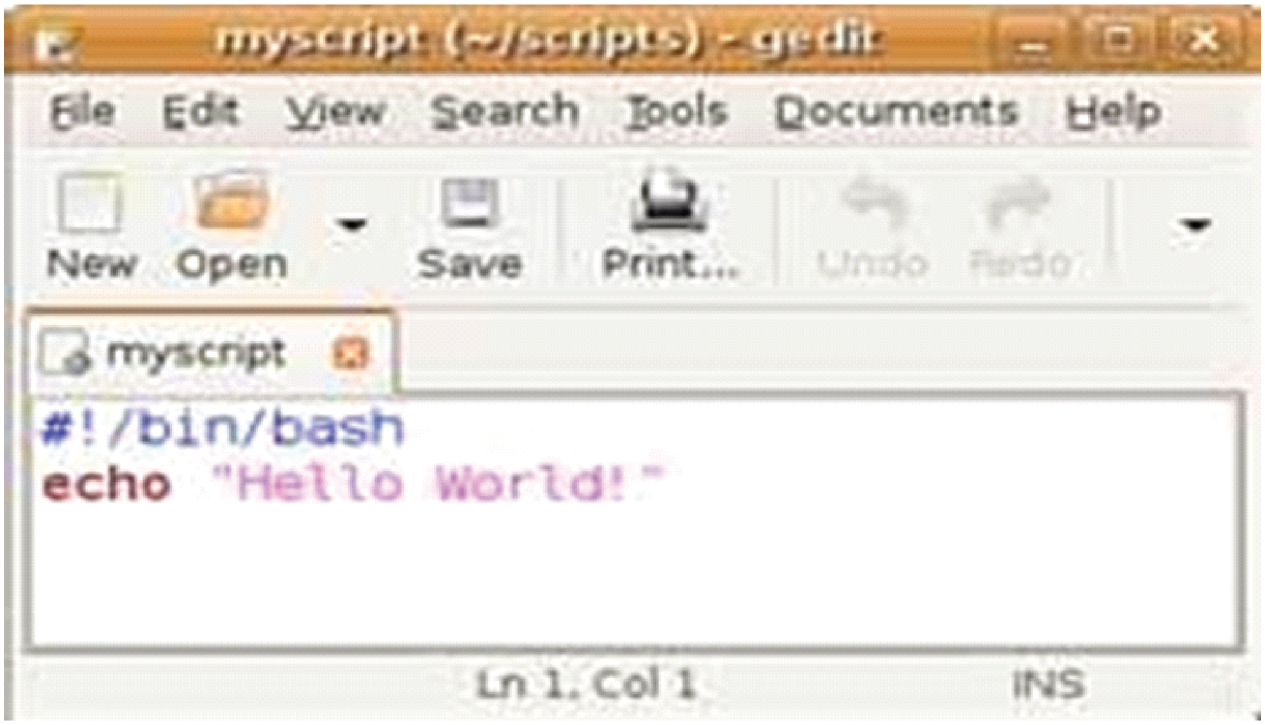


Figure 4.4 Executing Shell Scripts

Shell script

- can take inputs from user, make a file and output them on screen.
- is useful to create own commands.
- saves lots of time.
- can be used to automate some task of day today life.
- with system administration part can be also automated

For example using cat command shell script can be written as follows:

$ cat > first

#

My first shell script

#

clear

echo “Knowledge is Power”

To execute it command to be typed is

$ chmod +x first

$ ./first

Now “Knowledge is Power” gets printed on screen. To print message of variables general form of echo command that is used is as follows

echo "Message"

echo "Message variable1, variable2....variableN"

To write a shell script is like to ride a bike because learning the basics of shell scripting is similar to learning of bike. The falling off and scraping happens a lot at first. Shell script is generally considered to be a glue language, ideal for creating small pieces of code to connect tools together. Shell scripts are usually not the best choice for complex tasks. In other words it can be said that to write a script is an easy task but it is rather challenging to write a script that works consistently.

Some common shells, grouped by script syntax are:

Bourne-compatible shells

- sh
- bash
- zsh
- ksh

C-shell-compatible shells

- csh
- tcsh
- bcsh (C shell to Bourne shell translator/emulator)

Due to security reason of files, the execution permission is not given by default to the creator of Shell Script and to run shell script following two things are to be done as follows:

(1) chmod command as given below is to be used to give execution permission to script

Syntax: *chmod +x shell-script-name*

OR

Syntax: *chmod 777 shell-script-name*

(2) script can be run as

Syntax: *./your-shell-program-name*

For example **$ ./first**

Here '.'(dot) is command, and it is used in conjunction with shell script. With it current

shell is indicated that the command that is followed by the dot(.) is to be executed in the same shell.

For example

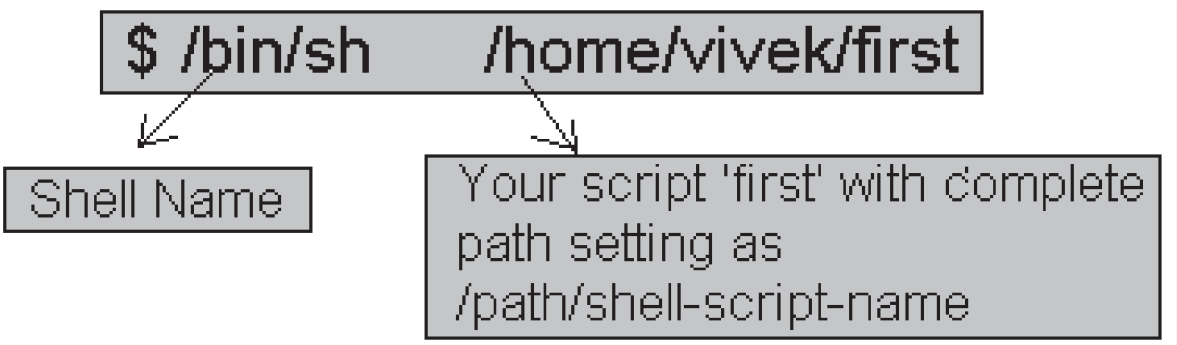
$ bash first

$ /bin/sh first

To run shell script, same directory must be used where the script is created because in different directory, because of path settings script will not run.

However, this problem too can be overcome. The complete path of script can be specified to run it from other directories. For example,

$ /bin/sh /home/vivek/first



Following are some advanced commands for execution of shell scripts:

1. **/dev/null -** This is used to send any unwanted output from program.

   Syntax is *command > /dev/null*

   For example, output of the command **$ ls > /dev/null**, is not shown on screen its send to this special file.

2. **Conditional execution (&& and ||)-** The && (read as AND) and || (read as OR) are control operators.

   The Syntax for AND is:

   command1 && command2

   It means that command2 is executed iff command1 returns an exit status of zero.

   The Syntax of OR is:

   command1 || command2

   It means that command2 is executed iff command1 returns a non-zero exit status.

3. **getopts command-**This command checks if a valid command line argument is passed to shell script or not. It is usually used in loop named while. The

   Syntax is:

   getopts {optsring} {variable1}

Errors in Shell Scripts

A shell script works usually but sometimes while writing it an error can occur when an occasionally empty environment variable does not get quoted. Following are the meaning common errors:

1 means general error

2 means return an exit status.

126 means Command cannot be executed

127 means Command not found

128+N means Command exited with Signal N

130 means Command exited with Ctrl+C

255 means exist status out of range

For example,

Change:

echo “shell script.”

to:

echoq “ shell script.”

$ ./hello.sh

Who are you?

$

The above example shows the error in shell scripts. However, it is interesting to note that the shell script simply reports the error and in spite of that shell merrily continues running. There are many different sorts of these type errors the shell reports but continue its running. The one type of error that stops the running of the shell script is called a syntax error.

It is also to be noted that in the shell what the error is the command not found and the line on which it occurs. This makes it easier to track down the error and fix it. The example of syntax error is as follows:

Change:

echo “Shell Script?”

to:

(echo “shell script?”

$ ./hello.sh

Yes. I am a shell script.

$

.

If there is a syntax error in the shell script, the shell script will abort as soon as the error is encountered by it and stop running the rest of the script, as it won’t understand what is to be done.

It can be noted in the above example that although the error is in fact at line 4, shell not decides about it until line 5 and tells us happening of anything wrong. It may prove to be very annoying because it makes debugging shell scripts painful. When the shell tells about a syntax error at line n, it should be taken that that the syntax error is somewhere between the line n and the last command the script managed to execute.

4.5 Shell Parameters

The special variables set by the shell are known as shell parameters, which cannot be modified either by the user or by a shell script. The most important parameters are the positional parameters:

• Positional parameter 0 holds the name of the shell script

• Positional parameter 1 holds the first argument passed to the script

• Positional parameter 2 holds the second argument passed to the script, etc

Except the positional parameter 0, which holds the name of the shell script, the positional parameters are set to the arguments that are given to the shell script when it gets started.

For example

./myscript.sh argon hydrogen mercury

then positional parameter 0 = ./myscript.sh

1 = argon
2 = hydrogen
3 = mercury

and all the other positional parameters are not set.

The special parameter @ is set to the value of all the positional parameters, starting from the first parameter, passed to the shell script, each value being separated from the previous one by a space. The value of this parameter is assessed using the construct ${@}. If it is accessed in double quotes ñ as in "${@}" ñ then each of the positional parameters will be treated by shell as a separate word.

The special parameter # is set to the number of positional parameters not counting positional parameter 0. Thus it is set to the number of arguments passed to the shell

script, i.e. the number of arguments on the command line when the shell script was run.

$ cd

$ examples/params.sh 0.5 62 38 hydrogen
This script is /home/y250/examples/params.sh
There are 4 command line arguments.
The first command line argument is: 0.5
The second command line argument is: 62
The third command line argument is: 38
Command line passed to this script: 0.5 62 38 hydrogen

4.6 Summary

The AIX® operating system and other UNIX-like operating systems need a way to communicate with the kernel. This is done is through the use of a shell. shell scripting is something all UNIX® users should learn how to use. Shell scripting provides with the ability to automate many tasks and can save a great deal of time. It may seem daunting at first, but with the right instruction you can become highly skilled in it. This chapter explained how to write shells scripts with programming. An overview of Shell variables and shell parameters was also given.

4.7 Self - Assessment Exercise

1. Write Script, using case statement to perform basic math operation as follows

   + addition

- subtraction

x multiplication

/division

The name of script must be 'q4' which works as follows **$ ./q4 20 / 3**, Also check for sufficient command line arguments

2. How to calculate 5.12 + 2.5 real number calculation at $ prompt in Shell ?

3. How to perform real number calculation in shell script and store result to third variable , lets say a=5.66, b=8.67, c=a+b?

4. Write script to determine whether given file exist or not, file name is supplied as command line argument, also check for sufficient number of command line argument

5. Write script to determine whether given command line argument ($1) contains "*" symbol or not, if $1 does not contains "*" symbol add it to $1, otherwise show message "Symbol is not required".

6 How do you refer to the arguments passed to a shell script?

7 What is the conditional statement in shell scripting?

8 How do you do number comparison in shell scripts?

9 How do you test for file properties in shell scripts**?**

4.8 References

- Aeleen Frisch, Essential System Administration, 3rd edition, O'Reilly and Associates, 2002,
- Cameron Newham and Bill Rosenblatt, Learning the Bash Shell, 2nd edition, O'Reilly and Associates, 1998
- Chet Ramey and Brian Fox, The GNU Bash Reference Manual, Network Theory Ltd, 2003,
- David Medinets, Unix Shell Programming Tools, McGraw-Hill, 1999
- Ellen Siever and the staff of O'Reilly and Associates, Linux in a Nutshell, 2nd edition, O'Reilly and Associates, 1999,
- Jerry Peek, Tim O'Reilly, and Mike Loukides, Unix Power Tools, 3rd edition, O'Reilly and Associates,Random House, 2002
- Kernighan, Brian W., Pike, Rob (1984), "3. Using the Shell", The UNIX Programming Environment, Prentice Hall, Inc., p. 94.
- Ken Burtch, Linux Shell Scripting with Bash, 1st edition, Sams Publishing (Pearson), 2004, 0672326426.
- Neil Matthew and Richard Stones, Beginning Linux Programming, Wrox Press, 1996
- Paul Sheer, LINUX: Rute User's Tutorial and Exposition, 1st edition, , 2002
- Stephen Kochan and Patrick Wood, Unix Shell Programming, Hayden, 1990,

Unit - 5 : System Administration in Linux

Structure of Unit

5.0 Objective

The objective of this unit is to make you aware of some aspects of system administration in Linux operating system. In this unit we will look inside what type of administration Linux operating system is made up of. We will take a sneak peak at the "root" account in general and "su" or "sudo" in particular. We will also see now time and date is set in Linux Operating system.

5.1 Introduction

The most privileged account on Linux for system administration is "root" account, which makes able to carry out all types of system administration like adding of accounts, changing of passwords, examination of log files, installation of software etc. The *root* is the account, which by default has access to all files and commands on an OS like Unix/Linux. It is also known as the *root user, root account* and the *super user*.

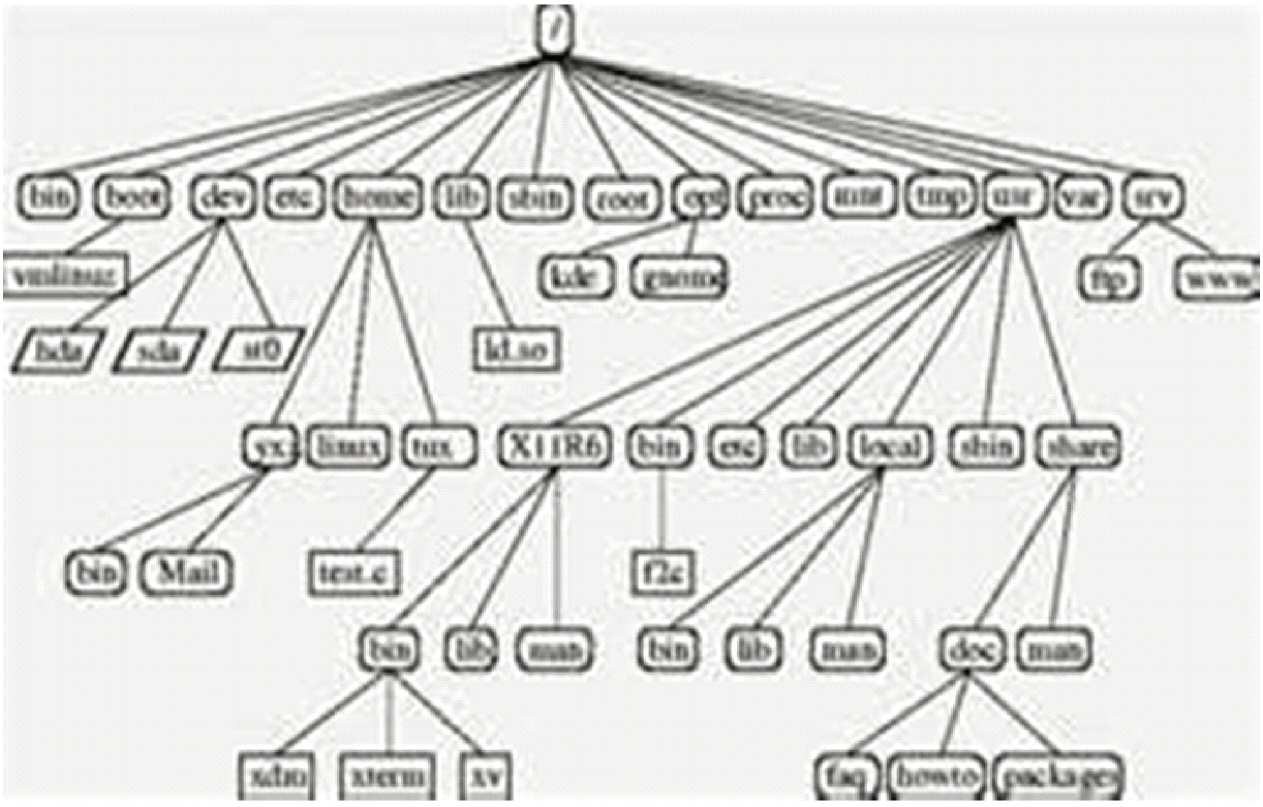


Figure 5.1 The Root Directory

5.2 The "Root" Account

The powers, which the root account has on the system, are known as root privileges. This is the most privileged one and has absolute powers like complete access to all commands and files, ability to modify the system in desired ways and to revoke and grant access permission for other users. The *root* as a term for the all-powerful administrative user is used because the root is the only account, which has permission to modify any file in the root directory. In turn, it has taken its name from the fact that the hierarchy in the OS is being designed like an inverted tree-like structure in which all directories branch off from a single directory that is analogous to the root of a tree.

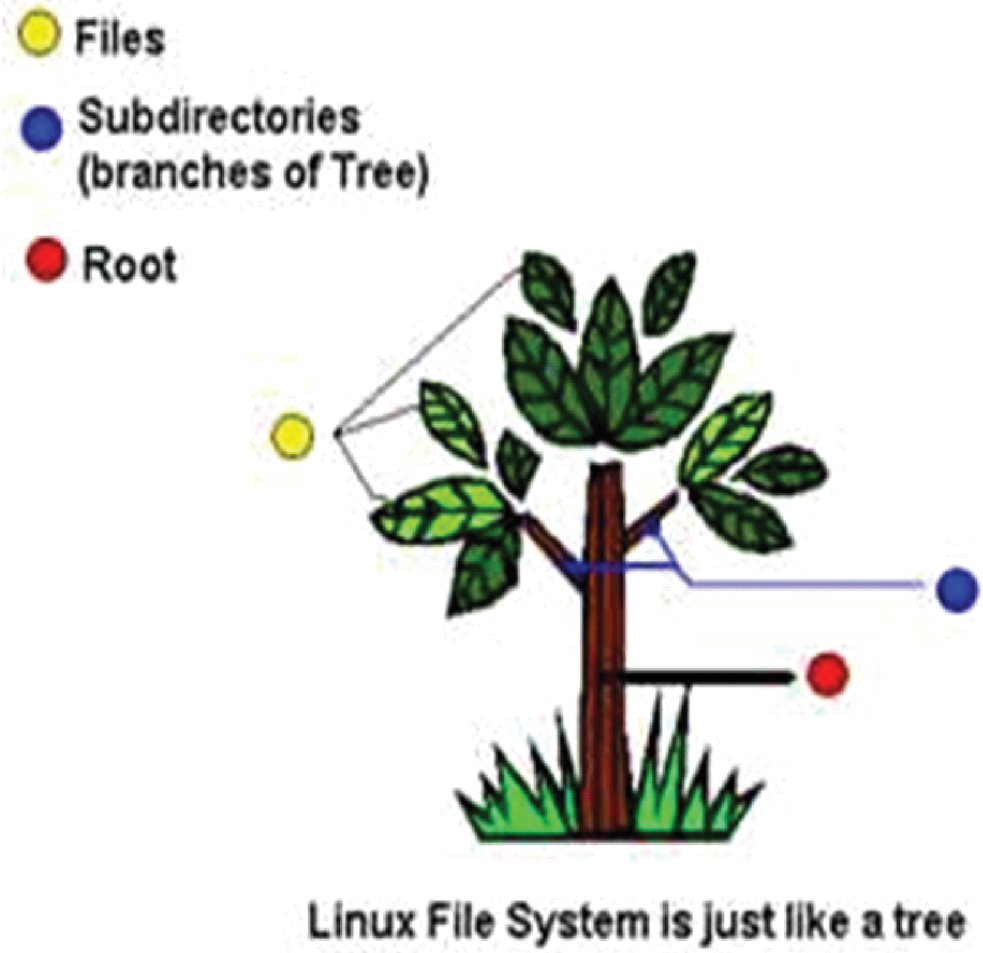


Figure 5.2 Tree Structure of Linux File System

Linux has a dedicated user account for its administration of system usually known as root and has a UID (user ID) of 0. The root account has complete access to change anything on the system. To do this su command is used which temporarily assume the privileges of the root user. E.g.

auser> su

Password: *******

root>

This command is sometimes useful to verify the configuration of another system account also. Alternatively to fully access the root privileges the sudo command can also be used that allows a system to be configured to allow certain users to run certain commands as root without having full access. This command logs a lot of information both about successful and unsuccessful attempts that provides a useful audit trail of root activities.

In the Unix like Operating Systems, each user is assigned an unique identification number, automatically and this UID is used by the system instead of the user name to identify and keep track of the users. The echo command is used to find the UID of the current user, i.e.,

echo $UID

The UID for all users including root can also be seen by looking at the command /etc/passwd that is the configuration file for user data.

However, while using the "root" one has to be most careful as it has no security restrictions imposed upon it to perform administrative duties without hassle. The OS also assumes that tasks are to be done without asking any questions. So, if one not remains very careful while using the "root" it means that it is quite easy to wipe out crucial system files.

Root login is required to perform those actions that change the settings of system for all users or to modify the accounts of users. The root account is also used for certain system operations. For example

- To add new users to the system and administer the user data.
- To install system software.
- To configure I/O devices. For example, a scanner or a TV tuner card.
- To configure system services like a web or FTP server.

The BASH shell displays '#' as the last character, to serve as a warning to give the absolute power to the user. Hence the rule of thumb is, unless, necessary absolutely, never sign in as "root", commands must be typed carefully and before pressing return, it must be double checked. Immediate Sign off from the "root" account after accomplishing the task and security of password is must.

5.3 Controlled Administrative Access

To access administrative commands, user must have administrative permissions before logging. The user that is created during installation is given access to administrative tools automatically.

To perform administrative operations, user must have access rights and to gain such access there are several ways to do so:

- login as a sudo supported user (gksu),
- unlocking an administrative tool for access (policykit),
- logging in as the root user.

Policy Kit: It is a preferred access method which is used on many administrative tools to provide access only to specified applications and only to users with administrative access for that application. The specific application is to be configured for use by policy kit is a must.

Sudo and gksu: The sudo is used for many tasks like software upgrade and installation. Any application is provided access with it with full administrative authorization but a time limit is imposed to reduce risk. The gksu command is graphical administrative tool of sudo like the Synaptic Package Manager. However, with gksu, sudo is still used to perform any command at the root level. For example moving files to an administrative directory.

Root user access, su: This access is although discouraged, but makes a provision of complete control over the entire system. This is the traditional method to access administrative tools. With the su command any user can login as the root user if the root user password is known.

5.4 Creation of User Account

The new User account can be created in the following two steps:

1. To actually create the account itself
2. To provide an alias to e"mail address

To create the account itself, initially the username to be assigned to the user is to be decided. It must be at most 8 characters long. As a next step, other information is to be entered like full name of user, user group, a user shell, password expiration value and the desired password. The user id and home directory are # (automatically assigned). The password must be between 6 to 8 characters long and *everything* should be entered in lowercase. The exception is the full name of the user that can be entered in a "pleasing format".

The Case being sensitive, identical case must be used while entering username and password.

Below is an example of adding a user name Arvind Kumar:

mail:~# **/sbin/adduser**

User to add (^C to quit): **Kumar**

That name is in use, choose another.

User to add (^C to quit): **KumarA**

Editing information for new user [kumara]

Full Name: **Arvind Kumar**

GID [100]:

Checking for an available UID after 500

First unused uid is 859

UID [859]:

Home Directory [/home/smithj]:

Shell [/bin/bash]:

Min. Password Change Days [0]:

Max. Password Change Days [30]: **90**

Password Warning Days [15]:

Days after Password Expiry for Account Locking [10]: **0**

Password [smithj]:</ **FL1539**

Retype Password:</ **Fl1539**

Sorry, they do not match.

Password:</> **FL1539**

Retype Password:</ **FL1539**

Information for new user [kumara]:

Name: Arvind Kumar

Home directory: /home/smithj

Shell: /bin/bash

Password: <hidden>

Uid: 859 Gid: 100

Min pass: 0 maX pass: 99999

Warn pass: 7 Lock account: 0

public home Directory: no

Type 'y' if this is correct, 'q' to cancel and quit the program,or the letter of the item you wish to change: **Y**

The next step is to create the alias for the e"mail account of the user. Either the user's account name can be used for e"mail address, or full name can be used to make it "easier".

To add the e"mail alias, the "/etc/aliases" file is to be edited as follows:

mail# **pico –w /etc/aliases**

To add the new alias, the following format is to be used at the bottom of the file:

Firstname.Lastname:username

For our example, the entry would be as follows:

Arvind.Kumar:kumar

After finishing the addition of alias, <**Ctrl**>"<**X**> is pressed to save the file. The "newaliases" is typed then to update the aliases database.

The user account is now created for use

Figure 5.3 System Administration

To change a password on behalf of a user, either sign on or "su" to the "root" account and ``passwduser" is to be typed then. The system will ask to enter a password but it would not echo to the screen. To change own password, ``passwd" without specifying a username is to be typed.

To disable a user account, if shadow passwords are in use, edit, as root, the "/etc/shadow" file passwords; in either case, edit the "/etc/passwd" file and then the password is replaced and stored in its encrypted form with a "*" asterisk character.

All passwords, regardless of their length are stored as encrypted strings of 13 characters. So just by replacing the password with a single "*" character, the user would not be able to sign in.

To remove a user account, the easiest way is to to use the "userdel" command, typed as "root". For example:

```
/usr/sbin/userdel          arvindkumar
```

The command will remove the username "arvindkumar" both from the file "/etc/passwd",, and the Shadow password format "/etc/shadow".

Another way is not to remove an account and rather simply *disable* it so that, the user may use his/her accounts again.

5.5 Custom Configuration and Administrative Issues

The root as administrator must know the following administration issues:

a. The "`/etc/rc.d/rc.local''` file is executed upon system start"up and contains any extra services added to server to be executed upon boot up. These are:

   - ``/etc/inetd.conf'' may look in /etc for any required specific changes
   - ``/etc/exports'' contains hosts list who are allowed to mount NFS volumes
   - ``/etc/lilo.conf'' has information of the LILO boot loader
   - ``/etc/sudoers'' contains users list who are to be given special privileges.
   - ``/etc/named.boot'' is for use of DNS

b. The root user is empowered fully, because Unix-like OS has the provision of maximum flexibility to configure their system.

c. The Unix-like OS taken it granted that the administrator knows about his/her actions exactly and only such individual(s) use the root account. That means virtually there is no safety net for the root user in the event of a careless error that could make the entire system inoperative.

d. "linuxconf" is an excellent configuration tool that makes many issues easier to do them. It runs on any means of available display environment like the console, a telnet session, or a GUI"based tool under X.

A critical means to prevent user from damaging Unix-like OS directly or to increase vulnerability of such systems to damage by others is to avoid the use of root account except when its necessary absolutely. In other words it can be said that rather than using root as logging, ordinary user accounts must be used by the administrators and then commands like kedsu, su and *sudo* can be used, which provide root privileges only as needed without requiring a new login.

For example, simply typing of su in the all-text mode and supplying the root password can make root user. The security while using su can further be increased by using option *–c that* terminates it.

The Tasks performed by root account include movement of files or directories in or out of *directories of system*, revoking or granting user privileges, copying of files, some repairs of system and installing some application programs.

5.6 Setting and Showing Time

In Linux, the symbolic link **/etc/localtime** determines the time zone, which points to a data file that describes the zone of local time. These files are located at either **/usr/lib/zoneinfo** or at **/usr/share/zoneinfo** depending on type of distribution of Linux used.

For example, on a SuSE system, the link **/etc/localtime** in New jersy would show **/usr/share/zoneinfo/US/Eastern**, the **/etc/localtime** link on Debian system would point to**/usr/lib/zoneinfo/US/Eastern**.

If with the**/usr/lib** or **/usr/share** directory, user can't find the zoneinfo directory, either the link **find / usr –print | grep zoneinfo** is to be used or your distribution's documentation is to be contacted.

To set a private time zone, user can set the TZ environment variable. If it is unset, the system time zone is assumed.

Figure 5.4

The current date and time can be shown by the date command. For example:

$ date

Sun Jul 14 21:53:41 EET DST 1996

$

That time is Sunday, 14th of July, 1996, at about ten before ten at the evening, in the time zone called ''EET DST"

The **date command** can also be used to show the universal time. For example,

$ date -u

Sun Jul 14 18:53:42 UTC 1996

$

The date command can also be used to set the kernel's clock. For example

 # date 07142157

Sun Jul 14 21:57:00 EET DST 1996

 # date

Sun Jul 14 21:57:02 EET DST 1996

#

Linux has a time zone package that knows about all existing time zones, and that can easily be updated when the rules change. The administrator of system just has to select the appropriate time zone. More to this, each and every user can set his/her own time zone irrespective of different countries over the Internet.It must also be noted that while each user can have his own time zone, only root can set the time, the clock is the same for everyone.

The track of time is kept independently by the kernel from the hardware clock. At the booting time, Linux sets the same time in its own clock as the hardware clock has but, both running of both clocks is independent. As the kernel clock always shows universal time, it does not know about time zones at all.

Below is the list of time zone variables.

Time Zone Environment Variables:		
TZ Variable	**GMT Offset**	**Description**
GMT0	0	Greewich Mean Time
UTC0	0	Universal Coordinated Time
FST2FDT	2	Fernando De Noronha Std
GST3	3	Greenland Standard Time
BST3	3	Brazil Standard Time
EST3EDT	3	Eastern Brazil Standard Time
NST3:30NDT	3.5	Newfoundland Standard Time/Newfoundland Daylight Time
AST4ADT	4	Atlantic Standard Time/ Atlantic Daylight Time
EST5EDT	5	USA Eastern Standard Time/ Eastern Daylight Time
EST6CDT	5	USA Eastern Standard Time/ Central Daylight Time
CST6CDT	6	USA Central Standard Time/ Central Daylight Time
MST7	7	USA Mountain Standard Time
MST7MDT	7	USA Mountain Standard Time/ Mountain Daylight Time
PST8PDT	8	USA Pacific Standard Time/Pacific Daylight Time, 8 hrs from GMT

AKS9AKD	9	USA Alaska Standard Time/Alaska Daylight Time
YST9YDT	9	Yukon Standard Time/Yukon Daylight Time
HST10	10	USA Hawaiian Standard Time/ Hawaiian Daylight Time
NZST-12NZDT	-12	New Zealand Standard Time/ New Zealand Daylight Time
EST-10	-10	Australian Eastern Standard Time
EST-10EDT	-10	Australian Eastern Standard Time/Australian Eastern Daylight Time
CST-9:30	-9.5	Australian Central Standard Time
CST-9:30CDT	-9.5	Australian Central Standard Time/Australian Central Daylight Time
JST-9	-9	Japan Standard Time
KST-9KDT	-9	Korean Standard Time
WST-8:00WAS-8WAD	-8	Australian Western Standard Time
CCT-8	-8	China Coast Time
HKT-8	-8	Hong Kong Time
JST-7:30	-7.5	Java Standard Time
NST-7	-7	North Sumatra Time
IST-5:30	-5.5	Indian Standard Time
IST-3:30IDT	-3.5	Iran Standard Time
MSK-3MSD	-3	Moscow Time
SAST-2SADT	-2	South Africa Standard Time/South Africa Daylight Time
EET-2EEST	-2	Eastern European Time/Eastern European Time Daylight Savings Time
MET-2METDST	-2	Middle European Time/Middle European Time Daylight Savings Time
CET-1CEST - Savings Time	1	Central European Time/Central European Time Daylight
WAT-1	-1	West Africa Time
WET0WETDST	0	Western European Time/Western European Time Daylight Savings Time

The **time** command is not used to get the system time and it's instead used to show time how long anything takes place.

The **date** command only shows or sets the clock. The **clock** command too synchronizes the software and hardware clocks. It is used when the system boots, to read the hardware clock and set the software clock. To set both clocks, first the software clock is to be set with **date command**, and then the hardware clock is to be set using the **clock -w** command.

The **-u** option to **clock** tells it that the hardware clock is in universal time.

5.7 Summary

The Chapter described the system administration aspects of using Linux. It is intended for people who know next to nothing about system administration (those saying "what is it?"), but who have already mastered at least the basics of normal usage. This chapter didn't tell one how to install Linux; System administration covers all the things that one has to do to keep a computer system in usable order. It included things like backing up files (and restoring them if necessary), installing new programs, creating accounts for users (and deleting them when no longer needed), making certain that the file system is not corrupted, and so on. If a computer were, say, a house, system administration would be called maintenance, and would include cleaning, fixing broken windows, and other such things.

5.8 Self - Assessment Exercise

1. Why Does the Computer Have the Wrong Time?
2. What is su?
3. What is sudo?
4. How can user have access to administrative operations?
5. Why the "root" is called root?
6. How the time is set and shown in Linux?
7. How an account of user is created?
8. Why the "root" account is called super user?
9. Differentiate between su, sudo and gksu?

Unit - 6 : Managing users

Structure of Unit

6.0 Objective

This unit provides a general overview of Unix user & groups accounts and user authentication. The user management commands like useradd, userdel and usermod etc are discussed. Here several ASCII configuration files that store user account information such as /etc/passwd, /etc/shadow, /etc/group and /etc/gshadow are explained. In the last section of unit controlling access to directories and file using chmod command are described. The unit focuses on the way managing users in UNIX based systems.

6.1 Introduction

User account and authentication are two of the most important areas for which system administrator is responsible. User accounts are the means by which users present themselves to the system, prove that they are who they claim to be and are granted or denied access to the information and resources on a system accordingly properly setting up and managing user accounts is one of administrator's chief tasks.

Most system provides command live utilities for manipulating user accounts and sometimes groups. The commands for managing user accounts are provided on many Unix system are: useradd, for adding new accounts; usermod, for changing settings of existing accounts and userdel for deleting user account.

/etc to the directory where all the system administration & configuration command are located user account information is stored in several ASCII configuration files.

/ etc / password

User account

/etc/shadow

Encoded passwords and password settings.

/etc/group

Group definition and memberships

/etc/gshadow

Group passwords and administration

User on Unix system are open with the files the usual set of permission give to files is rw-r—r—, which lets other users read the file but not change it any way. In order to charge default permission and set your own, chmod command is used.

6.2 Managing User Account

The system administrator has to add, remove and modify users from system and provide the default environment so that users can get their job done.

6.2.1 Adding New User Account (useradd)

- Adding a new user to the system involves the following tasks:
- Assign the user a user name, a user IDF number, and a primary group, and decide which other groups she should be a member of (if any). Enter this data into the system user account configuration files.
- Assign a password to the new account.
- Create a home directory for the user.
- Place initialization files in the user's home directory.
- Use chown and /or chgrp to give the new user ownership of his home directory and initialization files.
- Set other user account parameters appropriate for your system (possibly including password aging, account expiration date, resource limits, and system privileges).
- Grant or deny access to additional system resources as appropriate, using file protections.
- Perform any other site-specific initialization tasks.
- Test the new account

A new user account is created using the useradd command. The useradd command takes the following form:

Useradd [-c Comment] [-d dir] [-e expire] [-f inactive]

[-g group] [-u uid] [-s shell] loginanme

- c Comment

Here you specify the full name or any other textual information such as address of the user. For example:

Useradd –c 123, UIT scheme, Kota

-d dir

Here you specify the home directory of the user. Home directory is the directory where all the files and sub-directories created by the user will be saved. Usually it is / home / username

(for example, / home/raj)-e expire

This is the ending date for this login. If this field is omitted no expiration date will be used.

For example:

Useradd –e "January 1, 2015"

Or

-e 1/1/2015

-g group_Id

This refers to the primary group to which the user belongs. An existing group name or a numeric ID may be supplied with this option. – G is used for secondary group.

-u u_ID

UNIX internally uses a numeric ID to refer to each user known as the usrer ID. It is a number in the range of 0-32767 on older systems and 0-65535 on some new systems. The number 0 is reserved for the root user.

-s shell

This specifies the full pathname of the login shell, i.e. the sell which must be started when the user logs in.

Loginname

This is the login-name that you want to assign to the user. The login-name or username can be the first name of the user. The first name with the first letter of the last name, the last name, the first letter of the first name and the last name and so on. It can be anything but it must be unique in the network and consistent on all machines, a user is permitted to log into.

Here is useradd command to create user raj:

 # useradd –g CSE – G ECE, IE – S/bin/csh –d/home/raj – c "Rajesh Dadhich, Lecturer" raj

This command creates user raj, creates home directory / home / raj if it does not already exist. It also places raj in the CSE group primarily and ECE, IE groups also. H is UID will be next available number on the system. In comment (using – c option) this tell that account add for Rajesh Dadhich Lecturer.

6.2.2 Modifying User Details (usermod) :

A user's current attributes may be changed with the usermod command which accept almost all useradd option. In addition usermod supports –l option used to change the username of an existing user. For example the login shell of user raj can be set to korn shell with the following command:

 # usermod –s /bin/ksh raj

following command used to change username of an existing user.

usermod – l radhich raj.

6.2.3 Removing User Account (userdel)

Sometimes we may need to remove a user account from the system such as when an employee leaves the organization. An unneeded user account can be removed using the userdel command. For example the following commands will delete the account of user raj:

 # userdel raj

The above command will remove the entry matching raj from the /etc/passwd file and /etc/shadow.

If you want to remove the user's home directory also use the –r option as follows:

 # userdel –r raj

6.2.4 Disabling User's Account

Disabling a user's account is better than removing it because of the following reasons:

(a) The user may one day require his or her account again or may require one or more files which had been stored in the/her home directory.

(b) You may need to recover old files of which some might belong to the deleted user's ID. The files of disabled users can be recovered by enabling their accounts.

The passwd command with the-I option will mark the login as locked. Once locked, the user will not be able to log in. The command to lock a user is:

 # passwd -1 usrname

6.2.5 Changing Password (passwd)

To change the password of a user (say user raj) uses the passwd command as follows.

 # passwd raj

Note that the system will prompt you to enter the new password.

You can, as an ordinary user, also change your own password. In this case, you do not have to supply your login name. Just type the command **passwd** and the system will prompt for the new password.

6.3 Understanding Password File & Shadow Password File

The system keeps track of users via an entry in the password database. This database is maintained in /etc/passwd file, another method is the shadow format method. In this method, the account information is stored in both/etc/passwd and /etc/shadow file.

6.3.1 /etc/passwd file

This file /etc/passwd is the system's master list if information about users and every user account has an entry within it. Each entry in the password file is a single line. All the lines must have seven fields delimited by

colons (:). No comments or blank lines are allowed in this file. This file is readable and having following form.

Username: x : UID : GID: user information : home_dir: login_shell

An entry in this file will look as follows:

raj : x : 7 : 7 : Rajesh Dadhich : / home/raj : /bin/csh

The description of each of the seven filed in the above line are as follows:

Username: This is a one- to eight-character alpha-numeric filed that re represents the user's login name.

Password: The user's password is stored as encrypted string of 13 characters. If this filed is empty is means that this account has no password and hence password is not required for logging in. In systems using shadow password formate, this filed contains an "x" character (i.e. password is not stored in this file).

User ID: This specifies the user ID.

Group ID: We had learned that UNIX file permissions have three fields: owner (user ID), group (group ID) and others. The Group ID field specifies the default group for files created by this user.

Full Name: Full name of the user.

Home Directory: Usually in the form of /home/username. All user's personal files, web pages, mail forwarding etc. will be stored in the directory specified here.

Shell: This field contains the full pathname of the script or program, that is to be started by the login program, as the **shell.** If this field is empty, the Bourne shell is used by default.

6.3.2 /etc/shadow file

Most Unix operating systems support a shadow password file: An additional user-account database file designed to store the encrypted passwords. The /etc/shadow file contain the password filed in an field expanded format. It is readable only by the root (super user) It cannot be edited directly, but can be modified by the passwd command. Entries in each line are separated by colon(:)as in the /etc/ passwd file. An entry in the shadow file is as follows:

username : encloded password : changed : minlife : maxlife : warn : inactive : expires : un used

User Name: This name is used to match against the username in the passwd file.

Password: The user's password is stored as encrypted string of 13 characters. If this field is empty it means that this account has no password and hence password is not required for logging in.

Password Last-Changed Date : the number of days between January 1,1970 and the date that the password was last modified. It is stored as an integer value.

Minimum Number of Days between Password Changes: The user is not allowed to change his password until the number of days specified in this field has passed after the last password change. A value 0 means that the user may change his password at anytime.

Maximum Number of Days a Password valid: The user in not allowed to change his password until the number of days specified in this field has passed since the last change. An empty field means the password will never expire.

Number of Days to Warn User to Change Password : Number of days after which a user must be prompted to change password.

Number of Days the Login May Be Inactive: If the account is inactive for more than this number of days, the login is considered disabled.

Date when the Login is no Longer Valid: The date after which the login will get expired.

A reserved field for future use: This field is reserved for any future use.

6.4 Managing Group

UNIX groups any mechanism provided to enable arbitrary collections' of users to share files and other system resources. As such they provide one of the cornerstones of system security.

6.4.1 The Group File, /etc/group

4Unix groups are a mechanism provided to enable arbitrary collections of users to share files and other system resources. As such, they provide one of the cornerstones of system security.

4Groups may be defined in two ways:

4Implicitly, by GID; whenever a new GID appears in the fourth filed of the password file, a new group is defined.

Explicitly, by name and GID, via an entry in the file/etc/group.

Each entry in /etc/group consists of a single line with the following form:

name: *GID:additional-users

The meanings of these fields are as follows:

name

A name identifying the group. Names are often restricted to eight characters.

*

The second field is the traditional group password field, but it now holds some sort of placeholder character. Group passwords are no longer stored in the group file.

GID

This is the group's identification number. User groups generally start numbering at 100.*

Additional -user

The field holds a list of users who are members of the group, in addition to those users belonging to the group by virtue of /etc/passwd who need not be listed). Names must be separated by commas (but no space may appear within the list).

Note : Usernames and group names are independent of one another, even when the same name is both a username and a group name. Similarly, UIDs and GIDs sharing the same numerical value have no intrinsic relation to one another.

Here are some typical entries from an /etc/group file:

Comp1:* : 200 : root, njain, spahuja, sunil

Comp2:* : 300:root, dkr, mukul, kavita

The first line defines the comp1 group. It assigns the group id (GID) 200 to this group. Unix allow all users in the password file with GID 200 plus the additional users njain, spahuja, sunil and root to access this group's files. The comp2 group is also defined with GID 300 user d kr, mukul, kavita are member of comp2 group and root is member of both groups.

6.4.2 The Group Shadow File, /etc/gshadow

On same systems, an additional group configuration file is used. The file /etc/gshadow is the group shadow password file. It contains entries of the form:

group-name : encoded password : group-admins:additional-users

Where group-name is the name of the group, and encoded password is the encoded version of the group password. Group-admins is a list of users who are allowed to administer the group by changing its password and modifying memberships within the group (note that being so designated does not make them members of the specified group. additional-users in almost always a copy of the additional group members list for / etc/group; it is used by the newgrp command to determine which users can designate this group as their primary group. Both lists are comma separated and may not contain spaces.

Here are some sample entries from a group shadow file:

comp1 : xxxxx:VSS:LNJ,KKS

The group Comp1 has a group password, and users LNJ and KKS are members of it (as are any users who have it as their primary group, as defined in /etc/passwd). Its group administrator is user VSS.

On some systems, the newgrp command works slightly differently, depending on the group's entry in the group password file:

If the group has no password, newgrp fails unless the user is a member of the specified new group, either because it is her primary group or because her username is present in the additional members list in the group shadow password file, /etc/gshadow.

Because secondary group memberships for file access purposes are taken from the /etc/group file, it makes no sense for a user to appear in the group shadow file but not in the main group file. Omitting a secondary user defined in /etc/group from the shadow group list prevents him from using newgrp with that group, which might be desirable in some unusual circumstances.

If the group has a password defined, any user who knows the password can change to this group with newgrp (the command prompts for the group password).

If the group has a disabled password (indicated by an asterisk in the password field of /etc/gshadow), no user may change her primary group to that group with newrp.

6.5 Controlling Access to Directories & Files chmod

6.5.1 Types of File and Directory Access

The important issue to consider is how to protect file from unwanted access or how to allow access to those who need it. The protection on a file is referred to as its file mode on Unix systems. File modes are set with the chmod command; we'll look at chmod after discussing the file protection File modes are set with the chmod command; we'll look at chmod after discussing the file protection concepts it relies on.

Unix supports there types of file access : read, write and execute access (r, w, x). Table 6.1 shows meaning of these access types.

Table 6.1: File Access Types

Access	Meaning for a file	Meaning for a directory
r	View file contents.	Search directory contents (e.g., use Is).
w	Alter file contents.	After directory contents (e.g., delete or rename files).
x	Run executable file.	Make it your current directory (cd to it)

The file access types are fairly straightforward. It you have read access to a file, you can see what's in it. If you have write access, you can change what's in it. If you have execute access and the file is a binary executable program, you can run it. To run a script, you need both read and execute access, since the shell has to read the commands to interpret them. When you run a compiled program, the operating system loads it into memory for you and begins execution, so you don't need read access yourself.

The corresponding meanings for directories may seem strange at first, but they do make sense. If you have execute access to a directory, you can cd to it (or include it in a path that you want to cd to). You can also access files in the directory by name. However, to list all the files in the directory (i.e. to run the ls command without any arugments), you also need read access to the directory. This is consistent because a directory is just a file whose contents are the names of the files it contains, along with information pointing to their disk locations. Thus, to cd to a directory, you need only execute access since you don't need to be able to read the directory file itself. In contrast, if you want to run any command lists or use files in the directory via an explicit or implicit wildcard-e.g., ls without arguments or cat *.dat-you do need read access to the directory file itself to expand the wildcards.

Table 6.2 illustrates the workings of these various access types by listing some sample commands and the minimum access you would need to successfully execute them.

Table 6.2 File Protection Examples

Command	Minimum access needed	
	On file Itself	On directory file is in
Cd / home / raj	N/A	x
Is / home / raj/*.c	(none)	r
	r	
Is – 1 / home / raj / *.c	(none)	x
	r	
cat myfile	r	x
cat >>myflie	w	x
rm myfile	(none)	wx

Some items in this list are worth a second look. For example, when you don't have access to any of the component files, you still need only read access to a directory in order to do a simple Is; if you include-l (

or any other option that lists file sizes), you also need execute access to the directory. This is because the file sizes must be determined from the disk information, an action which implicitly changes the directory in question. In general, any operation that involves more than simply reading the list of filenames from the directory file is going to require execute access if you don't have access to the relevant files themselves.

Note especially that write access on a file is not required to delete it; write access to the directory where the file resides is sufficient (although in this case, you'll be asked whether to override the protection on the file):

$ rm example

rm : override protection 440 for example? Y

If you answer yes, the file will be deleted (the default response is no). Why does this work? Because delteting a file actually means removing its entry from the directory file (among other things), which is a form of altering the directory file, for which you need only write access to the directory. The moral is that write access to directories is very powerful and should be granted with care.

Given these considerations, we can summarize the different options for protecting directories as shown in Table 6.3

Table 6.3 Directory Protection Summary

Access granted	Resulting availability
–	Does not allow any activity of any kind within the directory or any of its subdirectories.
(no access)	
r-	Allow users to list the names of the files in the directory, but does not reveal any of their attributes (i.e., size, ownership, mode and so on).
(read access only)	
x –	
(execute access only)	Lets users work with programs in the directory specified by full pathname, but hides all other files.
r-x	
(read and execute access)	Lets users work with programs in the directory and list the contents of the directory, but does not allow them to create or delete files in the directory.
-wx	
(write and execute access)	Used for a drop-box directory. Users can change to the directory and leave files there, but can't discover the names of files placed there by others. The sticky bit is also usually set on such directories (see below).
rwx	
(full access)	Lets users work with programs in the directory, look at the contents of the directory, and create or delete files in the directory.

6.5.2 Setting File Protection

The chmod command is used to specify the access mode for files:

$ chmod access-string files

Chmod's second argument is an access string, which states the permission you want to set (or remove) for the listed files. It has three parts: the code for one or more access classes, the operator, and the code for one or more access types.

Figure 6.1 illustrates the structure of an access string. To create an access string, you choose one or more codes from the access class column, one operator from the middle column, and one or more access types form the third column. Then you concatenate them into a single string (no spaces). For example, the access string u+w says to add write access for the user owner of the file. Thus, to add write access for yourself for a file you own (ex1 for example), use:

$ chmod u+w ex1

To add write access for everybody, use the all access class:

$ chmod a+w ex1

To remove write access, use a minus sign instead of a plus sign.

$ chmod a-w ex1

This command sets the permission on the file lead to allow only read access for all users:

$ chmod a=r ex1

If execute or write access had previously been set for any access class, executing this command removes it.

ACCESS CLASS	OPERATOR	ACCESS TYPE
One or more of:		One or more of:
u	+(Add designated access)	r
g	- (Remove designated access)	w
o	= (Set exact access specified)	x
a (for all 3)		—-

Figure 6.1: Constructing an Access String for Chmod

You can specify more than one access type and more than one access class. For example, the access string g-rw says to remove read and write access from the group access. The access string go=r says to set the group and other access to read-only (no execute access, no write access), changing the current setting as needed. And the access string go+rx says to add both read and execute access for both group and other users.

You can also include more than one set of operation-access type pairs for any given access class specification. For example, the access string u+x-w adds execute access and removes write access for the user owner. You can combine multiple access strings by separating them with commas (no spaces between them). Thus, the following command adds write access for the file owner and removes write access and read access for the group and other classes for the files ex2 and ex3:

$ chmod u+w, og+r-w ex2, ex3

The chmod command supports a recursive option (-R) to change the mode of a directory and all files under it. For example, if user raj wants to protect all the files under her home directory from everyone else, she can use the command.

$ chmod – R go-rwx / home/raj

6.5.3 Octal Notation

Instead of the character arguments, one can give numeric arguments to the chmod command. The notation takes the form of an octal representation of the permissions, and it is assigned like this

Read permission	=	4
Write permission	=	2
Execute permission	=	1

When more than one permissions is associated with a particular user class, the respective number are added.

For example, if a directory has the read and execute permission for the owner, the octal representation for the owner's permission will be 4 + 1 = 5. This exercise is repeated for the other categories (group and others). The result is a three-digit octal number, with each octal digit describing the permission for each category. The sequence followed is user, group and others. That is, the first digit represents the permissions for the user, the second digit for the group and third digit for other. Permission assignment by octal notation is absolute like the = operator.

For example, instead of the command:

$ chmod a+r ex1

We can give the command

$ chmod 444

4 indicate read permission.

Similarly, instead of the command:

$ chmod ugo+rw ex1

We can give the command

$ chmod 666 ex1

6 indicates read and write permission (4+2)

To assign all permissions to the owner, read and write permission to the group; and only executable permission to others, we can use either of the following two commands:

$ chmod u+rwx, g+rw, o=x ex1

Or

$ chmod 761 ex1

777 signifies all permission and 000 indicates absence of all permissions.

6.5.4 Sticky Bit

We can add a 'sticky bit' to a directory to prevent the files within it from getting deleted. With the sticky bit attached to a file, no one except the owner of the directory and the root user, can delete files from this directory.

To add the sticky bit to a directory (say games), give the following command:

$ chmod u+t games

If you use ls -l command, you will see the permission of games directory as rwxrwxr – xt. The character't' at the end of the permission signifies that the sticky bit has been set up.

6.6 Summary

A user mean a particular individual who can log in, edit files, run program and otherwise make use of the system. Each user has username that identifies him, when a adding new user account to system, administrator assigns the user name a UID (user identification number). The administrator also assigns each new user to one or more groups : a named collection of users who generally share similar function. Each group has GID (group identification number); its system way to identifying & defining a group. Every user is a member of one or more groups.

The system administrator's (SA) one of the major responsibility is doing user management. He user useradd, usermod & userdel on command line. The SA also keeps track of users via configurating files known as password file (/etc/passwd and shadow file (/etc/shadow). He also manages groups through group file (/etc/group) and group shadow file (/etc/gshadow).

The different types of file and directory access are defined to protect files from unauthorized access. For setting file protection chmod command popularly used whenever permission related issuses are raised.

6.7 Self - Assessment Exercise

1. Explain the process of creating a new user in the UNIX system.
2. How is it that the /etc / passwd file is updated by ordinary user, using passwd command, even though the file does not have write permission?
3. Describe the contents of /etc/passwd and /etc/shadow file.
4. Can you add new user without using useradd command? Write steps
5. Explain chmod command with syntax and example. A file has got protection 744 (octal). What protections does it really have?

6.8 References

- Managing NFS & NIS O' Reilly Hal stern et.al.
- Practical UNIX & internet security O' Reilly Simso Garfinket et.al .
- Essential system Administration O' Reilly by Aeleen Frisch.
- www.Linuxhomenetworking.com

Unit -7 : NFS and NIS

Structure of Unit

7.0 Objective

This unit contains descriptions of Network File System (NFS) and Network Information Service (NIS) services; in first part we discuss basic understanding of NFS, File handle, mount & NFS protocols, configuration file information and NFS security. People consider NFS to be the heart of a distributed computing environment, because it manages the resource users are most concerned about: their files.

In next part we discuss NIS Server and Clients. The primary function of NIS is managing configuration information and making it consistent on all machines in the network. NIS provides the framework in which to use NFS. Once the framework is in place, you add users and their files into it, knowing that essential configuration information is available to every host.

7.1 Introduction

NFS and NIS are high-level networking protocols, built on several lower-level protocols. Network protocols are typically described in terms of a layered model, in which the protocols are "stacked" on top of each other. Data coming into a machine is passed from the lowest level protocol up to the highest, and data sent to other hosts moves down the protocol stack.

The standard model for networking protocols and distributed applications is the International Organization for Standardization (ISO) seven-layer model shown in Table 7-1.

Table 7-1: The ISO Seven-Layer Model

Layer	Name	Protocol/ Services
7	Application	NFS and NIS
6	Presentation	XDR
5	Session	RPC
4	Transport	TCP or UDP
3	Network	IP
2	Data Link	Ethernet
1	Physical	CAT-5

The Network File System (NFS) and the Network Information Service (NIS) provide mechanisms for solving "consistent and transparent" access problems. The NFS and NIS protocols were developed by Sun Microsystems and are now licensed to hundreds of vendors and universities, not to mention dozens of implementations from the published NFS and NFS specifications. NIS centralizes commonly replicated configuration files, such as the password file, on a single host. It eliminates duplicate copies of user and system information and allows the system administrator to make changes from one place. NFS makes remote file systems appear to be local, as if they were on disks attached to the local host. With NFS, all machines can share a single set of files, eliminating duplicate copies of files on different machines in the network. Using NFS and NIS together greatly simplifies the management of various combinations of machines, users, and file systems.

NFS provides network and file system transparency because it hides the actual, physical location of the file system. A user's files could be on a local disk, on a shared disk on a fileserver, or even on a machine located across a wide-area network. As a user, you're most content when you see the same files on all machines. Just having the files available, though, doesn't mean that you can access them if your user information isn't correct. Missing or inconsistent user and group information will break Unix file permission checking. This is where NIS complements NFS, by adding consistency to the information used to build and describe the shared file systems. A user can sit down in front of any workstation in his or her group that is running NIS and be reasonably assured that he or she can log in, find his or her home directory, and access tools such as compilers, window systems, and publishing packages. In addition to making life easier for the users, NFS

and NIS simplify the tasks of system administrators, by centralizing the management of both configuration information and disk resources.

NFS can be used to create very complex file systems, taking components from many different servers on the network. It is possible to overwhelm users by providing "everything everywhere," so simplicity should rule network design. Simplicity often satisfies the largest number of users, and it makes the system administrator's job easier.

7.2 Understanding NFS

NFS (Network File System) exists to allow remote hosts to mount partitions on a particular system and use them as though they were local file systems. This allows files to be organized in a central location, while providing the functionality of allowing authorized users continuous access to them. Some of the most notable benefits that NFS can provide are:

- Local workstations use less disk space because commonly used data can be stored on a single machine and still remain accessible to others over the network.
- There is no need for users to have separate home directories on every network machine. Home directories could be set up on the NFS server and made available throughout the network.
- Storage devices such as floppy disks, CDROM drives, and Zip drives can be used by other machines on the network. This may reduce the number of removable media drives throughout the network.

Using NFS, client can mount partitions of a server as if they were physically connected to the client. In addition to allowing remote access to file over the network, NFS allows many (relatively) low-cost computer systems to share the same high-capacity disk drive at the same time. NFS clients and servers have been written for many different operating systems.

NFS is nearly transparent. In practice, a workstation user simply logs into the work-station and begins working, accessing it as if the files were locally stored. In many environments, workstations are set up to mount the disks on the server automatically at boot time or when files on the disk are first referenced. NFS also has a network-mounting program that can be configured to mount the NFS disk automatically when an attempt is made to access files stored on remote disks.

Security problems with NFS:

- NFS is built on top of Sun's RPC (Remote Procedure Call), and in most cases uses RPC for user authentication. Unless a secure form of RPC is used, NFS can be easily spoofed.
- Even when Secure RPC is used, information sent by NFS over the network is not encrypted, and is thus subject to monitoring and eavesdropping. The data can be intercepted and replaced (thereby corrupting or trojaning files being imported via NFS).
- NFS uses the standard Unix filesystem for access control, opening the networked filesystem to many of the same problems as a local filesystem.

One of the key design features behind NFS is the concept of server statelessness. Unlike other systems, there is no "state" kept on a server to indicate that a client is performing a remote file operation. Thus, if the client crashes and is rebooted, there is no state in the server that needs to be recovered.

Alternatively, if the server crashes and is rebooted, the client can continue operating on the remote file as if nothing really happened – there is no server – side state to recreate.

NFS is based on two similar but distinct protocols: Mount & NFS, both make use of a data object known as file handle. There is also a distributed protocol for file locking.

7.2.1 File Handles

Each object on the NFS-mounted filesystem is referenced by a unique object called a file handle. A file handle is viewed by the client as being opaque – the client cannot interpret the contents. However, to the server, the contents have considerable meaning. The file handles uniquely identify every file and directory on the server computer.

The Unix NFS server stores three pieces of information inside each file handle.

Filesystem identifier

Refers to the partition containing the file (file identifiers such as inode numbers are usually unique only within a partition).

File identifier

Can be something as simple as an inode number, used to refer to a particular item on a partition.

Generation count

A number that is incremented each time a file is unlinked and recreated. The generation count ensures that when a client references a file on the server, that file is, in fact, the same file that the server thinks it is. Without a generation count, two clients accessing the same file on the same server could produce erroneous results if one client deleted the file and created a new file with the same inode number. The generation count prevents such situations from occurring when the file is recreated, the generation number is incremented, and the second client gets an error message when it attempts to access the older, now non-existent, file.

Note that the file handle does't includes a pathname; a pathname is not necessary and is, in fact, subject to change while a file is being accessed.

7.2.2 The Mount Protocol

The MOUNT protocol is used for the initial negotiation between the NFS client and the NFS server, Using MOUNT, a client can determine which filesystems are available for mounting and can obtain a token (the file handle) that is used to access the root directory of a particular filesystem. After that file handle is returned, it can thereafter be sued to retrieve file handles for other directories and files on the server.

Another benefit of the MOUNT protocol is that you can export only a portion of a local partition to a remote client, By specifying that the root is a directory on the partition, the MOUNT service will return its file handle to the client. To the client, this file handle behaves exactly like one for the root of a partition : reads, writes, and directory lookups all behave the same way.

MOUNT is an RPC service. The service is provided by the mountd or rpc.mountd daemon, which is started automatically at boot time. MOUNT is often given the RPC program number 100,005. The standard mountd normally responds to six different requests:

Request	Effect
NULL	Does nothing
MNT	Returns a file handle for a filesystem, advises the mount daemon that a client has mounted the filesystem
DUMP	Returns the list of mounted filesystems
UMNT	Removes the mount entry for this client for a particular filesystem
UMNTALL	Removes all mount entries for this client
EXPORT	Return the server's export list to the client

Although the MOUNT protocol provides useful information within an organization, the information that it provides could be used by those outside an organization to launch an attack. For this reason, you should prevent people outside your organization from accessing your computer's mount daemon. The best way to do this is by using a host-based or network-based firewall.

The MOUNT protocol is based on Sun Microsystems' RPC and External Date Representation (XDR) protocols. (For description RFC 1094)

7.2.3 The NFS Protocol

The NFS protocol takes over where the MOUNT protocol leaves off. With the NFS protocol, a client can list the contents of an exported filesystem's directories; obtain file handles for other directories and files; and even create, read, or modify files (as permitted by Unix permissions).

Here is a list of the RPC functions that perform operations on directories:

Function	Effect
CREATE	Creates (or truncates) a file in the directory
LINK	Creates a hard link
LOOKUP	Looks up a file in the directory
MKDIR	Makes a directory
READADDR	Reads the contents of a directory
REMOVE	Removes a file in the directory
RENAME	Renames a file in the directory
RMDIR	Removes a directory
SYMLINK	Creates a symbolic link

These common RPC functions can be used with files:

Function	Effect
GETATR	Gets a file's attributes (owner, length, etc.)
SETATTR	Sets some of a file's attributes
READLINK	Reads a symbolic link's path

READ	Reads from a file
WRITE	Writes to a file

In new Versions added a number of additional RPC functions. These new functions allow improved performance:

Function	**Effect**
ACCESS	Determines if a user has the permission to access a particular file or directory
FSINFO	Returns static information about a filesystem
FSSTAT	Returns dynamic information about a filesystem
MKNOD	Creates a device or special file on the remote filsystem
READDIPRLUS	Reads a directory and returns the file attributes for each entry in the directory
PATHCONF	Returns the attributes of a file specified by the pathname
COMMIT	Commits the NFS write cache to disk

All communication between the NFS client and the NFS server is based upon Sun's RPC system, which lets programs running on one computer call subroutines that are executed on another. RPC uses Sun's XDR system to allow the exchange of information between different kinds of computers. Sun built NFS upon the Internet User Datagram Protocol (UDP), believing that UDP was faster and more efficient than TCP. However, NFS required reliable transmission and, as time went on, many tuning parameters were added that made NFS resemble TCP in many respects, NFS Version 3 allows the use of TCP, which actually improves performance over low-bandwidth, high-latency links such as modem-based PPP connections because TCP's backoff and retransmission algorithms are significantly better than those is NFS.

NIS-	NFS-
Network Information System	Network Filesystem
RPC-Remote Procedure call	-
XDR-External Data Representation	
IP-Internet Protocol	
Network Transport (e.g., Ethernet)	

Figure 7.1: NFS Protocol Stack

7.2.4 Hard and Soft Mounts

If the NFS client still receives no acknowledgment, it will retransmit the request again and again, each time doubling the time that it waits between retries. If the network filesystem was mounted with the soft option, the request will eventually time our. If the network filesystem is mounted with the hard option, the client continues sending the request until the client is rebooted or gets an acknowledgement. Some BSD-derived versions of Unix also have a spongy option that is similar to hard, except that the stat, lookup, fsstat, readlink, and readdir operations behave as if they have a soft MOUNT.

NFS uses the mount command to specify whether a filesystem is mounted with the hard or soft option. To mount a filesystem soft, specify the soft option. For example:

```
/etc/mount – o soft CSE:/big/zbig
```

This command mounts the directory /big stored on the server called CSE locally in the directory /zbig. The option-o soft tells the mount program that you wish the filesystem mounted soft.

The mount a filesystem hard, do not specify the soft option:

```
/etc/mount CSE:/big / zbig
```

On some systems you need to be explicit that this is an NFS mount. You may also be able to use a URL format for the path and server. Here are examples of each:

```
mount –F nfs CSE: /big / zbig
```

```
mount nfs://CSE/bin/zbig
```

Deciding whether to mount a filesystem hard or soft can be difficult because there are advantages and disadvantages to each option. Diskless workstations often hard-mount the directories that they use to keep system programs; if a server crashes, the workstations wait until the server is rebooted, then continue file access with no problem. Filesystems containing home directories are usually hard-mounted so that all disk writes to those filesystems will be performed correctly.

On the other hand, if you mount many filesystems with the hard option, you will discover that your workstation may stop working every time any server crashes and won't work again until it reboots. If there are many libraries and archives that you keep mounted on your system, but that are not critical, you may wish to mount them soft. You may also wish to specify the intr option, which is like the hard option except that the user can interrupt it by typing the kill character (usually Ctrl-C).

As a general rule of thumb, read-only filesystem can be mounted soft without any chance of accidental loss of data. An alternative to using soft mounts is to mount everything hard (or spongy, when available) but avoid mounting your nonessential NFS partitions directly in the root directory. This practice will prevent the Unix getpwd() function from hanging when a server is down.

7.2.5 Connectionless and Stateless

NFS servers are stateless by design, Stateless means that all of the information that the client needs to mount a remote filesystem is kept on the client, instead of having additional information with the mount stored on the server. After a file handle is issued for a file, that file handle will remain good even if the server is shut down and rebooted as long as the file continues to exist and no major changes are made to the configuration of the server that would change the values (e.g., a file system rebuild or restore from tape).

Early NFS servers were also connectionless. Connectionless means that the server program does not keep track of every client that has remotely mounted the filesystem. When offering NFS over a TCP connection, however, NFS is not connection less there is one TCP connection for each mounted filesystem.

The advantage of a stateless, connectionless system is that such systems are easier to write and debug. The programmer does not need to write any code for re-establishing connections after the network server crashes and restarts because there is no connection that must be re-established. If a client crashes (or if the network becomes disconnected), valuable resources are not tied up on the server maintaining a connection and state for that client.

A second advantage of this approach is that it scales. That is, a connectionless, stateless NFS serve works equally well if 10 clients are using a filesystem or if 10,000 are using it. Although system performance suffers under extremely heavy use, every file request made by a client using NFS should eventually be satisfied, and there is no performance penalty if a client mounts a filesystem but never uses it.

7.3 Server Side NFS Security

Because NFS allows users on a network to access files stored on the server, NFS has significant security implications for the server. These implications fall into three broad categories:

Client access

NFS can (and should) be configured so that only certain clients on the network can mount filesystems stored on the server.

User authentication

NFS can (and should) be configured so that users can access and alter only files to which they have ben granted access.

Eavesdropping and data spoofing

NFS should (but does not) protect information on the network from eavesdropping and surreptitious modification.

7.3.1 Limiting Client Access: /etc/exports

The NFS server can be configured so that only certain hosts are allowed to mount filesystems on the server. This is a very important step in maintaining server security; if an unauthorized host is denied the ability to mount a filesystem, then unauthorized users on that host should not be able to access the server's files. This configuration is controlled by settings in a file. Depending on the version of Unix/ Linux/etc. that you are using, the specific file structure and usage is different.

/etc/exports

Most versions of Unix, use the/etc/exports file to designate which clients can mount the server's filesystem and what access those clients can be give. Each line in the/etc/exports file generally has the form.

```
directory-options [,more options]
```

For example, a sample/etc/exports file might look like this:

```
/-access=math,root=vmou.domain.edu

/usr -ro

/usr/spool/mail-access=math
```

The directory may be any directory or filesystem on your server. In the example, exported directories are/, /usr , and/usr/spool/mail.

The options allow you to specify a variety of security-related and performance-related options for each entry. These include:

access=machinelist

Grants access to this filesystem only to the hosts or netgroups specified in machinelist. The names of hosts and netgroups are listed and separated by colons (e.g., host1:host2:group3).

ro

Exports the directory and its contents as read-only to all clients. This option overrides whatever the file permission bits are actually set to.

rw=machinelist

Exports the filesystem read-only to all hosts except those listed, which are allowed read/write access to the filesystem.

root=machinelist

Normally, NFS changes the user ID for requests issued by the superuser on remote machines from 0 (root) to-2 (nobody). Specifying a list of hosts gives the superuser on these remote machines superuser access on the server.

anon=uid

Specifies which user ID to use on NFS requests that are not accompanied by a user ID; this might happen on a DOS client. The number specified is used for both the UID and the GID of anonymous requests. A value of-2 is the nobody user. A value of-1 usually disallows access.

You should understand that NFS maintains options on a per-filesystems basis, not on a per-directory basis. If you put two directories in the/etc/exports file that actually reside on the same filesystem, they will use the same options (usually the options used in the last export listed).

Sun's documentation of anon states that, "If a request comes from an unknown user, use the given UID as the effective user ID." This statement is very misleading; in fact, NFS by default honors "unknown' user Ids-that is, UIDs that are not in the server's /etc/passwd file-in the same way that it honors "known" UIDs because the NFS server does not ever read the contents of the/etc/passwd file. The anon option actually specifies which UID to use for NFS requests that are not accompanied by authentication credentials.

Let's look at the example/etc/exports file again:

```
/-access=math,  root=vmou.domain.edu

/usr  -ro

/usr/spool/mail-access=math
```

This example allows anybody in the group math or on the machine math to mount the root directory of the server, but only the root user on machine prose. domain.edu has superuser access to these files. The /usr filesystem is exported read-only to every machine that can get RPC packets to and from this server (usually a bad idea-this may be a wider audience than the local network). And the/usr/spool/mail directory is exported to any host in the math netgroup.

7.3.2 The Showmount Command

You can use the Unix command showmount (typically located in /usr/sbin or /usr/etc and present in most flavours of Unix) to list all of the clients that have probably mounted directories from your server. This command has the form:

```
/usr/etc/showmount  (options)  (host)
```

The options are:

-a Lists all of the hosts and which directories they have mounted

-d Lists only the directories that have been remotely mounted

-e Lists all of the filesystems that are exported;

The showmount command does not tell you which hosts are actually using your exported filesystem; it shows you only the names of the hosts that have mounted your filesystems since the last reset of the local log file. Because of the design of NFS, someone can use a filesystem without first mounting it.

7.4 Client side NFS Security

NFS can create security issues for clients as well as for NFS servers. Because the files that a client mounts appear in the client's filesystem, an attacker who is able to modify mounted files can directly compromise the client's security.

The primary system that NFS uses for authenticating servers is based on IP host addresses and hostnames. NFS packets are not encrypted or digitally signed in any way. Thus, an attacker can spoof an NFS client either by posing as an NFS server or by changing the data that is en route between a server and the client. In this way, an attacker can force a client machine to run any NFS-mounted executable. In practice, this ability can give the attacker complete control over an NFS client machine.

It's also wise to avoid mounting device files from the server. The nodev option to mount, if available, prevents character and block special devices from being interpreted as such on the client.

NFS can also cause availability and performance issues for client machines. If a client has an NFS partition on a server mounted, and the server becomes unavailable (because it crashed, or because network connectivity is lost), then the client can freeze until the NFS server becomes available. Occasionally, an NFS server will crash and restart and-despite NFS's being a connectionless and stateless protocol-the NFS client's file handles will all become stale. In this case, you may find that it is impossible to unmount the stale NFS filesystem, and your only courses of action may be to forcibly restart the client computer.

Here are some guidelines for making NFS clients more reliable and more secure:

- Try to configure your system such that it is either an NFS server or an NFS client, but not both.
- Don't allow your NFS clients to mount from NFS servers from outside your organization.
- Minimize the number of NFS servers that each client mounts. A system is usually far more reliable and more secure if it mounts two hard disks from a single NFS server, rather than mounting partitions from two NFS servers.

7.4.1 Improving NFS Security

There are many techniques that you can use to improve overall NFS security:

- Limit the use of NFS by limiting the machines to which filesystems are exported, and limit the number of filesystems that each client mounts.
- Export filesystems read-only if possible.
- Use root ownership of exported files and directories.

- Remove group write permissions from exported files and directories.
- Do not export the server's executables.
- Do not export home directories.
- Do not allow user to log into the NFS server.
- Set the portmon variable so that NFS requests that are not received from privileged ports will be ignored.
- Use showmount -e to verify that you are exporting only the filesystem you wise to export to the hosts specified, and with the correct flags.

7.5 Network Information Service(NIS)

NIS, which stands for Network Information Services, was developed by Sun Microsystems to centralize administration of UNIX (originally SunOS™) systems. It has now essentially become an industry standard; all major UNIX like systems (Solaris™, HP-UX, AIX®, Linux, NetBSD, OpenBSD, FreeBSD, etc) support NIS.NIS was formerly known as Yellow Pages, but because of trademark issues, Sun changed the name. The old term (and yp) is still often seen and used.

A major problem in running a distributed computing environment is maintaining separate copies of common configuration files such as the password, group, and hosts files. Ideally, the network should be consistent in its configuration, so that users don't have to worry about where they have accounts or if they'll be able to find a new machine on the network. The Network Information System (NIS) addresses these problems.It is a RPC-based client/server system that allows a group of machines within an NIS domain to share a common set of configuration files. This permits a system administrator to set up NIS client systems with only minimal configuration data and add, remove or modify configuration data from a single location.It is a distributed database system that replaces copies of commonly replicated configuration files with a centralized management facility. Instead of having to manage each host's files (like */etc/hosts*, */etc/passwd*, */etc/group*, */etc/others*, and so on), you maintain one database for each file on one central server. Machines that are using NIS retrieve information as needed from these databases. If you add a new system to the network, you can modify one file on a central server and propagate this change to the rest of the network, rather than changing the hosts file for each individual host on the network. For a network of two or three systems, the difference may not be crucial; but for a large network with hundreds of systems, NIS is life-saving.

7.6 NIS Server and Client

NIS is built on the client-server model. There are three types of hosts in an NIS environment: master servers, slave servers, and clients. Servers act as a central repository for host configuration information. Master servers hold the authoritative copy of this information, while slave servers mirror this information for redundancy. Clients rely on the servers to provide this information to them. Information in many files can be shared in this manner. The master.passwd, passwd, group, and hosts files are commonly shared via NIS. Whenever a process on a client needs information that would normally be found in these files locally, it makes a query to the NIS server that it is bound to instead. With the distinction between NIS servers and clients firmly established, we can see that each system fits into the NIS scheme in one of three ways:

- **A *NIS Master Server*:** This server, analogous to a Windows NT primary domain controller, maintains the files used by all of the NIS clients. The *passwd, group*, and *other* various files used by the NIS clients live on the master server. It is possible for one machine to be an NIS master server for more than one NIS domain. However, these will not the case relatively small-scale NIS environment.
- ***NIS Slave Servers*:** Similar to the Windows NT backup domain controllers, NIS slave servers maintain copies of the NIS master's data files. NIS slave servers provide the redundancy, which is needed in important environments. They also help to balance the load of the master server: NIS Clients always attach to the NIS server whose response they get first, and this includes slave-server-replies.
- ***NIS Clients*:** NIS clients, like most Windows NT workstations, authenticate against the NIS server to log on.

There are several terms and several important user processes that you will come across when attempting to implement NIS on FreeBSD, whether you are trying to create an NIS server or act as an NIS client:

Term	Description
NIS domainname	An NIS master server and all of its clients (including its slave servers) have *a NIS domainname*. Similar to an Windows NT domain name, the *NIS domainname* does not have anything to do with DNS.
rpcbind	Must be running in order to enable RPC (Remote Procedure Call, a network protocol used by NIS). If ***rpcbind*** is not running, it will be impossible to run an NIS server, or to act as an NIS client.
ypbind	"Binds" an NIS client to its NIS server. It will take the NIS domainname from the system, and using RPC, connect to the server. ***ypbind*** is the core of client-server communication in an NIS environment; if ***ypbind*** dies on a client machine, it will not be able to access the NIS server.
ypserv	Should only be running on NIS servers; this is the NIS server process itself. If **ypserv** dies, then the server will no longer be able to respond to NIS requests (hopefully, there is a slave server to take over for it). There are some implementations of NIS (but not the FreeBSD one), that do not try to reconnect to another server if the server it used before dies. Often, the only thing that helps in this case is to restart the server process (or even the whole server) or the ypbind process on the client.
rpc.yppasswdd	Another process that should only be running on NIS master servers; this is a daemon that will allow NIS clients to change their NIS passwords. If this daemon is not running, users will have to login to the NIS master server and change their passwords there.

7.7 Implementing NIS

For setting up a sample NIS environment Let us assume of a small university lab. This lab, which consists of 15 machines, currently has no centralized point of administration; each machine has its own /etc/passwd and /etc/master.passwd. These files are kept in sync with each other only through manual intervention; currently, when you add a user to the lab, you must run adduser on all 15 machines. Clearly, this has to change, so you have decided to convert the lab to use NIS, using two of the machines as servers.

Therefore, the configuration of the lab now looks something like:

Machine name	IP address	Machine role
A	20.0.0.2	NIS master
B	20.0.0.3	NIS slave
C	20.0.0.4	Faculty workstation
D	20.0.0.5	Client machine

If you are setting up a NIS scheme for the first time, it is a good idea to think through how you want to go about it. No matter what the size of your network, there are a few decisions that need to be made.

7.7.1 Choosing NIS Domain Name

This might not be the "domainname" that you are used to. It is more accurately called the "NIS domainname". When a client broadcasts its requests for info, it includes the name of the NIS domain that it is part of. This is how multiple servers on one network can tell which server should answer which request. Think of the NIS domainname as the name for a group of hosts that are related in some way.

Some organizations choose to use their Internet domainname for their NIS domainname. This is not recommended as it can cause confusion when trying to debug network problems. The NIS domainname should be unique within your network and it is helpful if it describes the group of machines it represents. For example, the CSE department at VMOU might be in the "*vmou-cse*" NIS domain. For this example, assume you have chosen the name *test-domain*.

However, some operating systems (notably SunOS) use their NIS domain name as their Internet domain name. If one or more machines on your network have this restriction, you *must* use the Internet domain name as your NIS domain name.

7.7.2 Physical Server Requirement

There are several things to keep in mind when choosing a machine to use as a NIS server. One of the unfortunate things about NIS is the level of dependency the clients have on the server. If a client cannot contact the server for its NIS domain, very often the machine becomes unusable. The lack of user and group information causes most systems to temporarily freeze up. With this in mind you should make sure to choose a machine that will not be prone to being rebooted regularly, or one that might be used for development. The NIS server should ideally be a stand alone machine whose sole purpose in life is to be an NIS server. If you have a network that is not very heavily used, it is acceptable to put the NIS server on a machine running other services, just keep in mind that if the NIS server becomes unavailable, it will affect *all* of your NIS clients adversely.

7.8 NIS Server

An NIS server is a host that contains NIS data files, called *maps*. Clients are hosts that request information from these maps. The canonical copies of all NIS information are stored on a single machine called the NIS master server. The databases used to store the information are called NIS maps. In FreeBSD, these maps are stored in /var/yp/[domainname], where [domainname] is the name of the NIS domain being served. A single NIS server can support several domains at once; therefore it is possible to have several such directories, one for each supported domain. Each domain will have its own independent set of maps.

NIS master and slave servers handle all NIS requests with the *ypserv* daemon. *ypserv* is responsible for receiving incoming requests from NIS clients, translating the requested domain and map name to a path to the corresponding database file and transmitting data from the database back to the client.

7.8.1 Setting up a NIS Master Server

Setting up a master NIS server can be relatively straight forward, depending on your needs. FreeBSD comes with support for NIS out-of-the-box. All you need is to add the following lines to /etc/rc.conf, and system will do the rest for you:

1. *nisdomainname=”test-domain”*
2. This line will set the NIS domainname to test-domain upon network setup (e.g., after reboot).
3. *nis_server_enable = “yes”*
4. This will tell system to start up the NIS server processes when the networking is next brought up.
5. *nis_yppasswdd_enable=”yes”*
6. This will enable the rpc.yppasswdd daemon which, as mentioned above, will allow users to change their NIS password from a client machine.

After setting up the above entries, run the command /etc/netstart as superuser. It will set up everything for you, using the values you defined in /etc/rc.conf. As a last step, before initializing the NIS maps, start the ypserv daemon manually:

```
# /etc/rc.d/ypserv start
```

7.8.2 Initializing the NIS Maps

The NIS maps are database files, that are kept in the /var/yp directory. They are generated from configuration files in the /etc directory of the NIS master, with one exception: the /etc/master.passwd file. This is for a good reason, you do not want to propagate passwords to your root and other administrative accounts to all the servers in the NIS domain. Therefore, before we initialize the NIS maps, you should:

```
# cp /etc/master.passwd /var/yp/master.passwd

# cd /var/yp

# vi master.passwd
```

You should remove all entries regarding system accounts (bin, tty, games, etc), as well as any accounts that you do not want to be propagated to the NIS clients (for example root and any other UID 0 (super user account).

Note: Make sure the /var/yp/master.passwd is having file permission mode 600, by using the chmod command.

When you have finished, it is time to initialize the NIS maps! FreeBSD system includes a script named *ypinit* to do this for you. Note that this script is available on most UNIX Operating Systems, but not on all. On some UNIX system it is called *ypsetup*. Because we are generating maps for an NIS master, we are going to pass the -m option to *ypinit*. To generate the NIS maps, assuming you already performed the steps above, run:

vmou# ypinit -m test-domain

```
Server Type: MASTER Domain: test-domain

Creating an YP server will require that you answer a few questions.

Questions will all be asked at the beginning of the procedure.

Do you want this procedure to quit on non-fatal errors?[y/n: n] n

Ok, please remember to go back and redo manually whatever fails.

If you don't, something might not work.

At this point, we have to construct a list of this domains YP
servers.

Please continue to add any slave servers, one per line. When you are
done with the list, type a <control D>.

master server   :  vmou

next host to add:  studycenter

next host to add:  ^D

The current list of NIS servers looks like this:

vmou

studycenter

Is this correct?  [y/n: y] y

[..output from map generation..]

NIS Map update completed.

vmou has been setup as an YP master server without any errors.
```

ypinit should have created /var/yp/Makefile from /var/yp/Makefile.dist. When created, this file assumes that you are operating in a single server NIS environment with only FreeBSD machines. Since test-domain has a slave server as well, you must edit /var/yp/Makefile:

vmou# vi /var/yp/Makefile

```
You should comment out the line that says

NOPUSH = "TRUE"
```

(If it is not commented out already.)

7.8.3 Setting Up a NIS Slave Server

Setting up an NIS slave server is even simpler than setting up the master. Log on to the slave server and edit the file /etc/rc.conf as you did before. The only difference is that we now must use the -s option when running ypinit. The -s option requires the name of the NIS master be passed to it as well, so our command line looks like:

studycenter# ypinit -s vmou test-domain

```
Server Type: SLAVE Domain: test-domain Master: vmou
Creating an YP server will require that you answer a few questions.
Questions will all be asked at the beginning of the procedure.
Do you want this procedure to quit on non-fatal errors?[y/n: n]
Ok, please remember to go back and redo manually whatever fails.
If you don't, something might not work.
There will be no further questions. The remainder of the procedure
should take a few minutes, to copy the databases from vmou.
Transferring netgroup...
ypxfr: Exiting: Map successfully transferred
Transferring netgroup.byuser...
ypxfr: Exiting: Map successfully transferred
Transferring netgroup.byhost...
ypxfr: Exiting: Map successfully transferred
Transferring master.passwd.byuid...
ypxfr: Exiting: Map successfully transferred
Transferring passwd.byuid...
ypxfr: Exiting: Map successfully transferred
Transferring passwd.byname...
ypxfr: Exiting: Map successfully transferred
Transferring group.bygid...
ypxfr: Exiting: Map successfully transferred
Transferring group.byname...
ypxfr: Exiting: Map successfully transferred
Transferring services.byname...
ypxfr: Exiting: Map successfully transferred
Transferring rpc.bynumber...
```

```
ypxfr: Exiting: Map successfully transferred
Transferring rpc.byname...
ypxfr: Exiting: Map successfully transferred
Transferring protocols.byname...
ypxfr: Exiting: Map successfully transferred
Transferring master.passwd.byname...
ypxfr: Exiting: Map successfully transferred
Transferring networks.byname...
ypxfr: Exiting: Map successfully transferred
Transferring networks.byaddr...
ypxfr: Exiting: Map successfully transferred
Transferring netid.byname...
ypxfr: Exiting: Map successfully transferred
Transferring hosts.byaddr...
ypxfr: Exiting: Map successfully transferred
Transferring protocols.bynumber...
ypxfr: Exiting: Map successfully transferred
Transferring ypservers...
ypxfr: Exiting: Map successfully transferred
Transferring hosts.byname...
ypxfr: Exiting: Map successfully transferred
studycenter has been setup as an YP slave server without any
errors.
Don't forget to update map ypservers on vmou.
```

You should now have a directory called /var/yp/test-domain. Copies of the NIS master server's maps should be in this directory. You will need to make sure that these stay updated. The following /etc/crontab entries on your slave servers should do the job:

```
20  *     *     *     *     root   /usr/libexec/ypxfr passwd.byname
21  *     *     *     *     root   /usr/libexec/ypxfr passwd.byuid
```

These two lines force the slave to sync its maps with the maps on the master server. These entries are not mandatory because the master server automatically attempts to push any map changes to its slaves. However, due to the importance of correct password information on other clients depending on the slave server, it is recommended to specifically force the password map updates frequently. This is especially important on busy networks where map updates might not always complete.

Now, run the command /etc/netstart on the slave server as well, this again starts the NIS server.

7.9 NIS Clients

An NIS client establishes what is called a binding to a particular NIS server using the ypbind daemon. ypbind checks the system's default domain (as set by the domainname command), and begins broadcasting RPC requests on the local network. These requests specify the name of the domain for which ypbind is attempting to establish a binding. If a server that has been configured to serve the requested domain receives one of the broadcasts, it will respond to ypbind, which will record the server's address. If there are several servers available (a master and several slaves, for example), ypbind will use the address of the first one to respond. From that point on, the client system will direct all of its NIS requests to that server. ypbind will occasionally "ping" the server to make sure it is still up and running. If it fails to receive a reply to one of its pings within a reasonable amount of time, ypbind will mark the domain as unbound and begin broadcasting again in the hopes of locating another server.

7.9.1 Setting Up NIS Clients

Setting up a FreeBSD machine to be a NIS client is fairly straightforward.

1. Edit the file */etc/rc.conf* and add the following lines in order to set the NIS domainname and start ypbind upon network start up:
2. *nisdomainname = " test-domain"*
3. *nis_client_enable = "yes"*
4. To import all possible password entries from the NIS server, remove all user accounts from your */etc/master.passwd* file and use *vipw* to add the following line to the end of the file:
5. `+:::::::::`

   Note: This line will afford anyone with a valid account in NIS server's password maps an account. There are many ways to configure your NIS client by changing this line.
6. To import all possible group entries from the NIS server, add this line to your /etc/group file:
   `+:*::`

To start the NIS client immediately, execute the following commands as the superuser:

```
# /etc/netstart
# /etc/rc.d/ypbind start
```

After completing these steps, you should be able to run ypcat passwd and see the NIS server's passwd map.

7.10 Summary

The Network Information Service (NIS) and Network File System (NFS) are services that allow you to build distributed computing systems that are both consistent in their appearance and transparent in the way files and data are shared.

NFS is a distributed file system An NFS server has one or more file systems that are mounted by NFS clients; to the NFS clients, the remote disks look like local disks. NFS file systems are mounted using the standard Unix *mount* command, and all Unix utilities work just as well with NFS-mounted files as they do

with files on local disks. NFS makes system administration easier because it eliminates the need to maintain multiple copies of files on several machines: all NFS clients share a single copy of the file on the NFS server. NFS also makes life easier for users: instead of logging on to many different systems and moving files from one system to another, a user can stay on one system and access all the files that he or she needs within one consistent file tree.

NIS provides a distributed database system for common configuration files. NIS servers manage copies of the database files, and NIS clients request information from the servers instead of using their own, local copies of these files. For example, the */etc/hosts* file is managed by NIS. A few NIS servers manage copies of the information in the hosts file, and all NIS clients ask these servers for host address information instead of looking in their own */etc/hosts* file. Once NIS is running, it is no longer necessary to manage every */etc/hosts* file on every machine in the network — simply updating the NIS servers ensures that all machines will be able to retrieve the new configuration file information.

7.11 Self - Assessment Exercise

1. What do you understand by NFS and NIS? Explain in brief about these services.
2. What is daemon? Enlist the NFS and NIS daemons with their brief functionality.
3. Explain the meaning of file handles, mount protocol, and hard & soft mounts in NFS.
4. Discuss how configuration file /etc/exports is useful in limiting client access. Write and explain all elements of /etc/exports file entries.
5. Discuss NIS server and client in brief. Write steps of implementing NIS server.

7.12 References

- Managing NFS and NIS O‘ Reilly series by Hal Stern et. al.
- Practical Unix and Internet Security O‘ Reilly series by Simson Garfinkel et. al.
- Essential System Administration O‘ Reilly series by Aeleen frisch
- www.Linuxhomenetworking.com

Unit - 8 : Distributed Computing

Structure of Unit

8.0 Objective

This chapter provides a general overview about

- Distributed Computing
- Distributed Process Management
- Message Passing
- Remote Procedure Calls
- Distributed Memory Management

8.1 Introduction to Distributed Computing

The high volume of networked computers, workstations, LANs has prompted users to move from a simple end user computing to a complex distributed computing environment. This transition is not just networking the computers, but also involves the issues of scalability, security etc. A Distributed Computing Environment herein referred to, as DCE is essentially an integration of all the services necessary to develop, support and

manage a distributed computing environment. Despite the advances in processor design, users still demand more performance. Eventually, single CPU technologies must give way to multiple processors parallel Computers: it is less expensive to run 10 inexpensive processors cooperatively than it is to buy a new computer 10 times as fast. This change is inevitable, and has been realized to some extent in the specialization of subsystems like bus mastering drive controllers. However, the need for additional computational power has thus far rested solely on advances in CPU technologies.

The present day computing industry depends on the efficient usage of resources. So instead of duplicating the resources at every node of computing, a remote method of accessing the resources is more efficient and saves costs. This gave rise to the field of distributed computing, where not only physical resources, but also processing power was distributed.

Distributed computing was driven by the following factors,

a) Desire to share data and resources

b) Minimize duplication of functionality

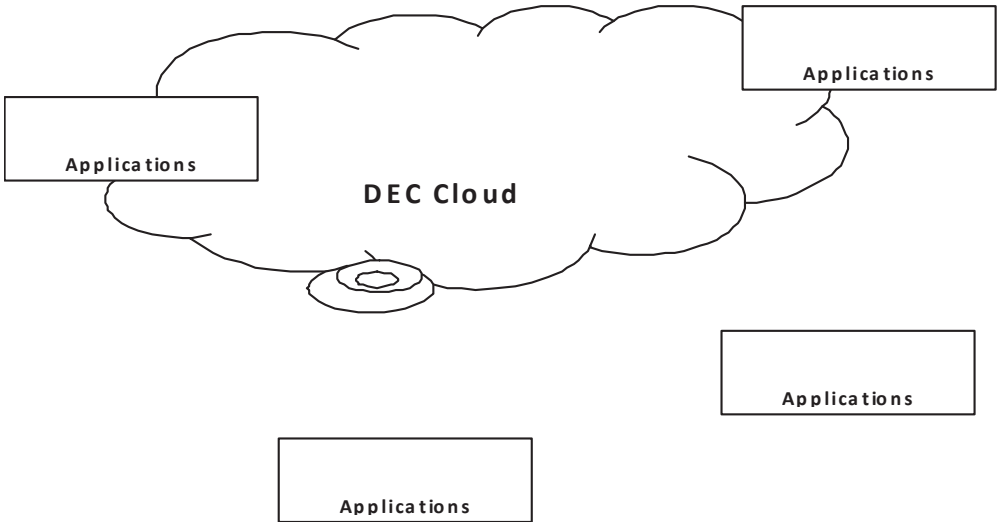
c) Increase cost efficiency

d) Increase reliability and availability of resources.

When an organization migrates from networked computing to Distributed Computing a lot of factors are to be taken into consideration. For example replication of files gives rise to consistency problems, clock synchronization becomes important, and security is a bigger consideration.

A Distributed Computing Environment addresses all these issues by providing an integrated set of cross platform, comprehensive services which aids in the development and application of distributed applications.

The following diagram gives a simple view of the DCE architecture,



The DCE cloud refers to the distributed computing environment tools that facilitate distributed computing.

A distributed system is a collection of independent computers that appear to its users as a single coherent system.

- Andrew Tannenbaum

This certainly is the ideal form of a distributed system, where the "implementation detail" of building a powerful system out of many simpler systems is entirely hidden from the user.

Unfortunately, when we look at the reality of networked computers, we find that the multiplicity of system components usually shines through the abstractions provided by the operating system and other software. In other words, when we work with a collection of independent computers, we are almost always made painfully aware of this. For example, some applications require us to identify and distinguish the individual computers by name while in others our computer hangs due to an error that occurred on a machine that we have never heard of before.

8.1.1 Examples of Distributed Systems

Probably the simplest and most well known example of a distributed system is the collection of Web servers—or more precisely, servers implementing the HTTP protocol—that jointly provide the distributed database of hypertext and multimedia documents that we know as the World-Wide Web.

The alternative to using a distributed system is to have a huge centralized system, such as a mainframe. For many applications there are a number of economic and technical reasons that make distributed systems much more attractive than their centralized counterparts.

- **Cost:** Better price/performance as long as commodity hardware is used for the component computers.
- **Performance:** By using the combined processing and storage capacity of many nodes, performance levels can be reached that are beyond the range of centralized machines.
- **Scalability:** Resources such as processing and storage capacity can be increased incrementally.
- **Reliability:** By having redundant components the impact of hardware and software faults on users can be reduced.
- **Inherent Distribution:** Some applications, such as email and the Web (where users are spread out over the whole world), are naturally distributed. This includes cases where users are geographically dispersed as well as when single resources (e.g., printers, data) need to be shared.

However, these advantages are often offset by the following problems encountered during the use and development of distributed systems:

New component: Network: Networks are needed to connect independent nodes and are subject to performance limitations. Besides these limitations, networks also constitute new potential points of failure.

Security: Because a distributed system consists of multiple components there are more elements that can be compromised and must, therefore, be secured. This makes it easier to compromise distributed systems.

Software Complexity: As will become clear throughout this course distributed software is more complex and harder to develop than conventional software; hence, it is more expensive to develop and there is a greater chance of introducing errors.

8.1.2 Hardware and Software Architectures

A key characteristic of our definition of distributed systems is that it includes both a hardware aspect (independent computers) and a software aspect (performing a task and providing a service). From a hardware point of view distributed systems are generally implemented on multicomputers.

From a software point of view they are generally implemented as distributed operating systems or middleware.

8.1.3 Multi-computers

A multicomputer consists of separate computing nodes connected to each other over a network. Multi-computers generally differ from each other in three ways:

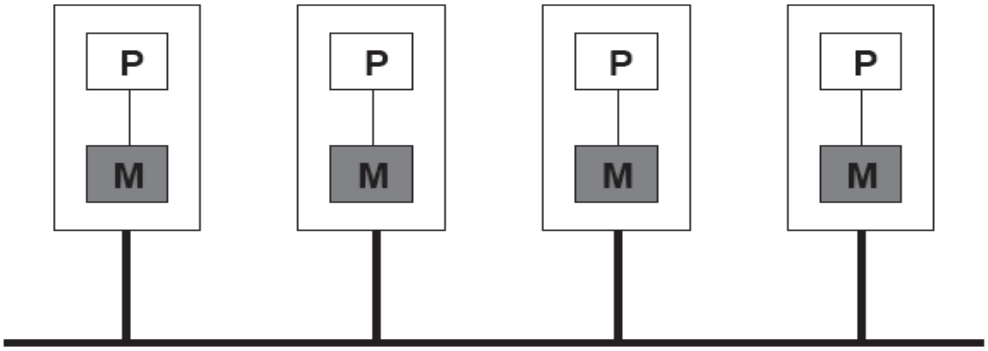


Figure 1: A multicomputer.

1. Node resources: This includes the processors, amount of memory, amount of secondary storage, etc. available on each node.

2. Network connection: The network connection between the various nodes can have a large impact on the functionality and applications that such a system can be used for. A multi computer with a very high bandwidth network is more suitable for applications that actively share data over the nodes and modify large amounts of that shared data. A lower bandwidth network, however, is sufficient for applications where there is less intense sharing of data.

3. Homogeneity: A homogeneous multicomputer is one where all the nodes are the same, that is they are based on the same physical architecture (e.g. processor, system bus, memory, etc.). A heterogeneous multicomputer is one where the nodes are not expected to be the same.

One common characteristic of all types of multicomputers is that the resources on any particular node cannot be directly accessed by any other node. All access to remote resources ultimately takes the form of requests sent over the network to the node where that resource resides.

8.1.4 Distributed Operating System

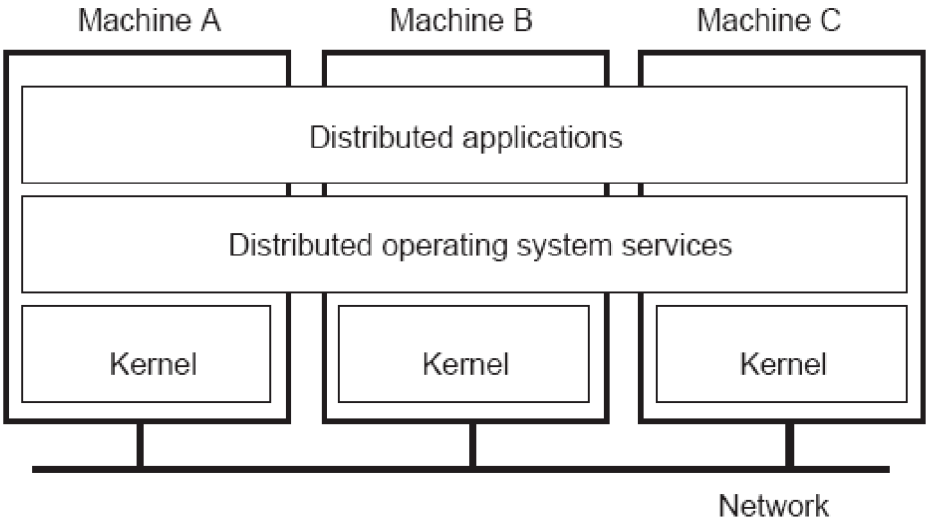


Figure 2 : A Distributed Operating System

A distributed operating system (DOS) is a an operating system that is built, from the ground up, to provide distributed services. As such, a DOS integrates key distributed services into its architecture (Figure 2). These services may include distributed shared memory, assignment of tasks to processors, masking of failures, distributed storage, interprocess communication, transparent sharing of resources, distributed resource management, etc.

A key property of a distributed operating system is that it strives for a very high level of transparency, ideally providing a single system image. That is, with an ideal DOS users would not be aware that they are, in fact, working on a distributed system. Distributed operating systems generally assume a homogeneous multicomputer. They are also generally more suited to LAN environments than to wide-area network environments.

In the earlier days of distributed systems research, distributed operating systems where the main topic of interest. Most research focused on ways of integrating distributed services into the operating system, or on ways of distributing traditional operating system services. Currently, however, the emphasis has shifted more toward middleware systems. The main reason for this is that middleware is more flexible (i.e., it does not require that users install and run a particular operating system), and is more suitable for heterogeneous and wide-area multicomputers.

8.1.5 Middleware

Whereas a DOS attempts to create a specific system for distributed applications, the goal of middleware is to create system independent interfaces for distributed applications.

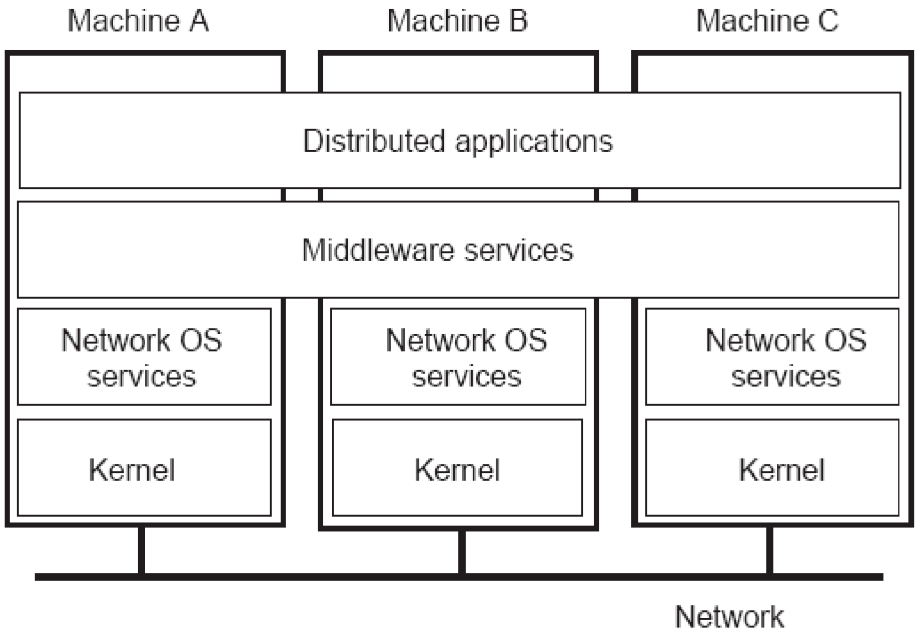


Figure 2 : A Middleware System

As shown in Figure 3 middleware consists of a layer of services added between those of a regular network OS1 and the actual applications. These services facilitate the implementation of distributed applications and attempt to hide the heterogeneity (both hardware and software) of the underlying system architectures.

The principle aim of middleware, namely raising the level of abstraction for distributed programming, is achieved in three ways:

(1) communication mechanisms that are more convenient and less error prone than basic message passing;

(2) independence from OS, network protocol, programming language, etc. and

(3) standard services (such as a naming service, transaction service, security service, etc.).

To make the integration of these various services easier, and to improve transparency and system independence, middleware is usually based on a particular paradigm, or model, for describing distribution and communication. Since a paradigm is an overall approach to how a distributed system should be developed, this often manifests itself in a particular programming model such as 'everything is a file', remote procedure call, and distributed objects. Providing such a paradigm automatically provides an abstraction for programmers to follow, and provides direction for how to design and set up the distributed applications. Paradigms will be discussed in more detail later on in the course.

Although some forms of middleware focus on adding support for distributed computing directly into a language (e.g., Erlang, Ada, Limbo, etc.), middleware is generally implemented as a set of libraries and tools that enable retrofitting of distributed computing capabilities to existing programming languages. Such systems typically use a central mechanism of the host language (such as the procedure call or method invocation) and dress remote operations up such that they use the same syntax as that mechanism resulting, for example, in remote procedure calls and remote method invocation.

Since an important goal of middleware is to hide the heterogeneity of the underlying systems (and in particular of the services offered by the underlying OS), middleware systems often try to offer a complete set of services so that clients do not have to rely on underlying OS services directly. This provides transparency for programmers writing distributed applications using the given middleware. Unfortunately this 'everything but the kitchen sink' approach often leads to highly bloated systems. As such, current systems exhibit an unhealthy tendency to include more and more functionality in basic middleware and its extensions, which leads to a jungle of bloated interfaces. This problem has been recognised and an important topic of research is investigating adaptive and reflective middleware that can be tailored to provide only what is necessary for particular applications.

With regards to the common paradigms of remote procedure call and remote method invocations, Waldo et al. [WWWK94] have eloquently argued that there is also a danger in confusing local and remote operations and that initial application design already has to take the differences between these two types of operations into account. We shall return to this point later.

8.1.6 Distributed Systems and Parallel Computing

Parallel computing systems aim for improved performance by employing multiple processors to execute a single application. They come in two flavours: shared-memory systems and distributed memory systems. The former use multiple processors that share a single bus and memory subsystem. The latter are distributed systems in the sense of the systems that we are discussing here and use independent computing nodes connected via a network (i.e., a multicomputer). Despite the promise of improved performance, parallel programming remains difficult and if care is not taken performance may end up decreasing rather than increasing.

8.1.7 Distributed Systems in Context

The study of distributed systems is closely related to two other fields: Networking and Operating Systems. The relationship to networking should be pretty obvious, distributed systems rely on networks to connect the individual computers together. There is a fine and fuzzy line between when one talks about developing networks and developing distributed systems. As we will discuss later the development (and study) of distributed systems concerns itself with the issues that arise when systems are built out of interconnected networked components, rather than the details of communication and networking protocols.

The relationship to operating systems may be less clear. To make a broad generalization operating systems are responsible for managing the resources of a computer system, and providing access to those resources in an application independent way (and dealing with the issues such as synchronization, security, etc. that arise). The study of distributed systems can be seen as trying to provide the same sort of generalized access to distributed resources (and likewise dealing with the issues that arise).

8.1.8 The DCE Cloud

It Consists of the Following Components,

a) Distributed File Service

b) Distributed Time Service

c) Security Service

d) Cell Directory Service

e) Threads Service

All these services are achieved by the use of Remote Procedure calls (RPC).

Properties of DCE

A DCE Provides a Global Computing Environment, which can interoperate with other services like DNS and X.500. This sort of global interoperability provides the much-needed interface for Write Once Run Anywhere Applications. Also the suite of components is completely integrated and interoperable, which facilitates the networking of two systems for processing even though they have different hardware and software configurations.

DCE Cells

A collection of machines, users and resources that are a part of a group and having their own directory service and security service can be called a DCE Cell. In an organization there may be a large number of cells, say one for each department.

DCE Remote Procedure Calls (RPC)

The Remote Procedure Call (RPC) in a DCE is the facility that lets users make remote procedure calls and connect to another system on the DCE. The application programmer is essentially hidden from the fact that it is a remote procedure call, by the components of RPC.

8.2 Distributed Process Management

Most processes are created and managed by a command interpreter, but any other process may also create new ones. All that is required is the capability that allows communication with a Kernel. Most users will have access to the cluster creation capability for the Kernel running on their own workstation; that is, users can create new processes on their own workstation.

The capability for creating processes on pool processors is typically kept by a "Processor Pool" service that acts as an agent for running programs on behalf of user processes. Load balancing can be achieved by the Processor Pool service when it allocates pool processors judiciously. Although clusters rarely move to a new host after being started up, migration is a central concept in the process management

mechanisms. This is became loading new clusters into memory, taking core dumps, making check-points, and doing remote debugging are all similar to migrating a cluster. In fact, if we can migrate a cluster from one machine to another, downloading, check-pointing, debugging, etc., should be simple.

Load balancing by migrating cluster is a poorly understood area and it is dubious whether it is very useful with the current sort of workstations and net-works. Migrating a five megabyte cluster, for instance, will take at least seven seconds, because that is how long it takes a fast transport protocol to copy the

memory contents over a 10 Mbit Ethernet; five megabyte programs are not at all uncommon, especially as candidates for migration: long-lived clusters are usually large too. Migration is thus rather expensive and the gain of a migrate operation must be big in order to merit one.

In spite of this, migration can be useful. When a workstation's owner logs off in the evening, the workstation can turn itself into a Pool Processor and provide process-execution service to the rest of the system. When the owner returns in the morning, however, and logs back on, the guest clusters running there could be nudged off by migrating them away to some other workstation.

8.3 Message Passing

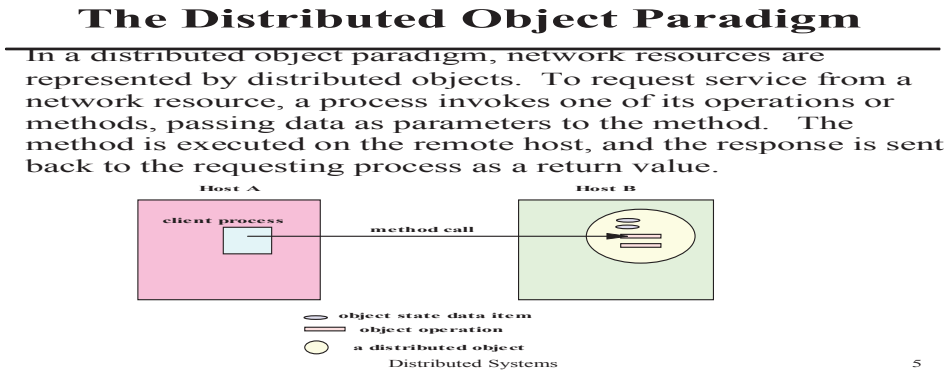
In distributed systems, there are two kinds of fundamental inter-process communication models:

a) Shared Memory and

b) Message Passing.

From a programmer's perspective, shared memory computers, while easy to program, are difficult to build and aren't scalable to beyond a few processors. Message passing computers, while easy to build and scale, are difficult to program. In some sense, shared memory model and message passing model are equivalent. One of the solutions to parallel system communication is Distributed Shared Memory (DSM), where memory is physically distributed but logically shared. DSM appears as shared memory to the applications programmer, but relies on message passing between independent CPUs to access the global virtual address space.

The **message-passing** is a common paradigm model for distributed computing, in the sense that it mimics the behavior in human communications. It is an appropriate paradigm for network services where processes interact with each other through the exchanges of messages. Message passing requires the participating processes to be tightly-coupled: throughout their interaction, the processes must be in direct communication with each other. If communication is lost between the processes (due to failures in the communication link, in the systems, or in one of the processes), the collaboration fails. The message-passing paradigm is data-oriented. Each message contains data marshaled in a mutually agreed upon format, and is interpreted as a request or response according to the protocol. The receiving of each message triggers an action in the receiving process. It is inadequate for complex applications involving a large mix of requests and responses. In such an application, the task of interpreting the messages can become overwhelming. A distributed object is one whose methods can be invoked by a **remote process**, a process running on a computer connected via a network to the computer on which the object exists.



8.4 Remote Procedure Calls

8.4.1 Definitions

RPC is a powerful technique for constructing distributed, client-server based applications. It is based on extending the notion of conventional or local procedure calling, so that the called procedure need not exist in the same address space as the calling procedure. The two processes may be on the same system, or they may be on different systems with a network connecting them. By using RPC, programmers of distributed applications avoid the details of the interface with the network. The transport independence of RPC isolates the application from the physical and logical elements of the data communications mechanism and allows the application to use a variety of transports.

8.4.2 Components of RPC

The RPC components are

a) The Interface Definition Language and its Compiler: The skeletons and stubs are created by the IDL and then compiled by the IDL compiler. The server stubs replace the remote part of the procedure call, and at the server the skeleton replaces the client.

b) Runtime RPC Library: The RPC runtime library is actively involved in the sending and receiving of remote procedure calls and finding the necessary server services and communicating between the client and the server.

c) Secure RPC Components: The Secure RPC components work along with the security APIs to provide authentication and authorization for the remote procedure calls.

d) Name Service Independent APIs: The Name Service Independent (NSI) APIs help in locating the right server to process the request. It is integrated into the directory services Component to facilitate the Association and Binding of the Client to the Server.

e) UUID Facilities: This UUID Stands for Universal Unique Identifiers. This is useful to generate UUIDs, to uniquely identify each server and client on the DCE.

8.4.3 Specifications Conformance

The DCE Architecture conforms to the Network Computing Architecture (NCA) [IRPC] specifications. Transport independence and hence the OS independence is achieved as the NCA supports both connection oriented as well as connection independent protocols.

8.4.4 Facilities Supplied by the RPC

The following are the facilities that are supplied by the DCE RPC which are shielded from the application programmer [IRPC]

a) Security services

b) Use of the Directory Service to Find the right server for remote calls

c) Managing the data formats which are different in each cell of the DCE.

d) Management of messages for example its fragmenting and reassembly.

e) The communication protocols used. RPC can communicate over TCP/IP and UDP.

8.4.5 Communication Methodology Using RPC

The Following are the commonly used communication methodologies in RPC.

Creation of the IDL File:

The interface for RPC is defined in the IDL file and not the actual procedures. The IDL file advertises the input and output of the Services offered by the remote server. The IDL file is written based on the server's procedure and then compiled using the IDL compiler. Compilation of the IDL file produce client and server stubs.

Client's View of the RPC:

The client is then provided with the Stubs generated by the compilation of the IDL file and it is incorporated into its procedure calls. A simple procedure call is now converted to a complex RPC over the network.

Server's View of RPC:

The server side has the subroutine to perform the function as given in the IDL. The server receives the parameters passed through the IDL and performs the procedure execution and sends back the results as published in the IDL to the client.

Binding:

The client finds the appropriate server to send the remote call by looking up the server's services. This is called Binding .The server when it starts must advertise the services it provides by registering with the directory services. The client then accesses the directory service to find about the server, which offers its services and then addresses that server.

8.4.6 Advantages of Using DCE RPC

The following are the advantages that are obtained by using the DCE RPC

Operating System Independence:

The RPC calls do not depend on the underlying OS's network calls mechanism

Machine Independence:

Even if the machines connecting through RPC are different, RPC can be successfully used as it provides the instructions in native format for both the client and the server.

Language Independence:

Any modern programming language can access the stubs and the skeletons that are produced by the IDL compiler.

Protocol Independence:

The server when registering with the DCE directory service explicitly states the protocol that it uses. Hence the clients can use that protocol or access a different server. The connection oriented and connection free protocols can be interchangeably used.

8.4.7 Security Inbuilt in RPC

The secure RPC is called the Authenticated RPC. There are various levels of authentication,

a) None – No Authentication

b) Connection – Authentication through encryption occurs at the first connection or handshake

c) Call Authentication – The first data packet which is sent to the server is authenticated

d) Packet Authentication – Each packet of data sent through the RPC Interface is authenticated

In addition to these levels packet integrity and privacy can be protected by the use of Cryptographic Checksums.

8.4.8 How RPC Works

An RPC is analogous to a function call. Like a function call, when an RPC is made, the calling arguments are passed to the remote procedure and the caller waits for a response to be returned from the remote procedure. Figure below shows the flow of activity that takes place during an RPC call between two networked systems. The client makes a procedure call that sends a request to the server and waits. The thread is blocked from processing until either a reply is received, or it times out. When the request arrives, the server calls a dispatch routine that performs the requested service, and sends the reply to the client. After the RPC call is completed, the client program continues. RPC specifically supports network applications.

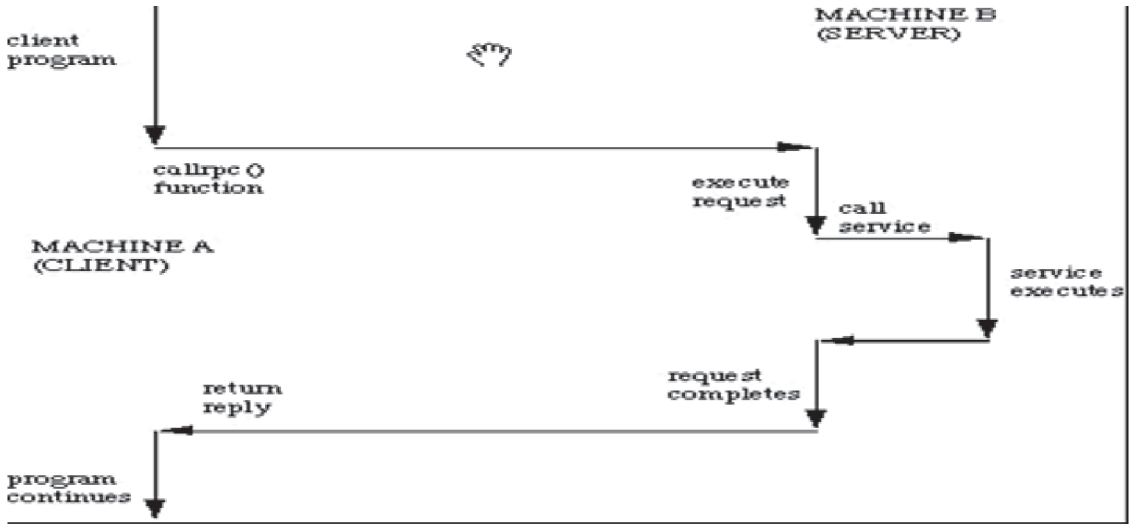


Figure 7.5 Remote Procedure Calling Mechanism

8.5 Distributed Memory Management

The memory management subsystem is one of the most important parts of the operating system. Since the early days of computing, there has been a need for more memory than exists physically in a system. Strategies have been developed to overcome this limitation and the most successful of these is virtual memory. Virtual memory makes the system appear to have more memory than it actually has by sharing it between competing processes as they need it.

Advantages:

- Shields programmer from Send/Receive primitives
- Single address space; simpli_es passing-by-reference and passing complex data structures
- Exploit locality-of-reference when a block is moved
- No memory access bottleneck, as no single bus
- Large virtual memory space
- DSM programs portable as they use common DSM programming interface

Disadvantages:

- Programmers need to understand consistency models, to write correct programs
- DSM implementations use async message-passing, and hence cannot be more efficient than message-passing implementations
- By yielding control to DSM manager software, programmers cannot use their own message-passing solutions.

Virtual memory does more than just make your computer's memory go further. The memory management subsystem provides:

- Large Address Spaces
- Protection
- Memory Mapping
- Fair Physical Memory Allocation
- Shared Virtual Memory

Threads can allocate and de-allocate blocks of memory, called **Segments**. These segments can be read and written, and can be mapped into and out of the address space of the process. A process owns at least one segment, but may have many more of them. Segments can be used for text, data, stack, or any other purpose the process desires. The operating system does not enforce any particular pattern on segment usage.

8.6 Summary

Let us sum up the different concepts we have studied till here.

- A distributed computing system is a collection of processors interconnected by a communication network in which each processor has its own local memory and other peripherals and communication between any two processors of the system takes place by message passing over the communication network.

- A distributed system includes both a hardware aspect and a software aspect. From a hardware point distributed systems are multi-computers and from a software view point they are distributed operating systems or middleware.
- A distributed operating system (DOS) is built, from the ground up, to provide distributed services.
- The message-passing is a common paradigm model for distributed computing, in the sense that it mimics the behavior in human communications.
- RPC makes the client/server model of computing more powerful and easier to program. When combined with the ONC RPCGEN protocol compiler clients transparently make remote calls through a local procedure interface.
- DSM uses simpler software interfaces, and cheaper o_-the-shelf hardware. Hence cheaper than dedicated multiprocessor systems

8.7 Self - Assessment Exercise

1. Explain the various reasons for designing applications in Distributed Processing system
2. Explain the features of Concurrency Control in Distributed Computing Environment.
3. List down various application areas where distributed computing is used

8.8 References

- John a . Sharp, " An introduction to distributed and parallel processing", Blackwell Scientific Publicaitons.
- Mukesh Singhal, Shivaratri, "Advanced Concepts in Operating System", Tata McGraw-Hill, 2001
- Nadiminti, Dias de Assunção, Buyya (September 2006). "Distributed Systems and Recent Innovations: Challenges and Benefitz". *InfoNet Magazine, Volume 16, Issue 3, Melbourne, Australia*.

Unit - 9 : Distributed Computing System – An Introduction

Structure of Unit

9.0 Objective

This unit covers a new task execution strategy called "Distributed Computing" and the following related terminologies.

- Distributed Computing System (DCS)
- Distributed Computing models
- Advantages of Distributed computing
- Security

9.1 Introduction

Distributed computing is a method of computer processing in which different parts of a program are run simultaneously on two or more computers that are communicating with each other over a network. Distributed computing is a type of **segmented** or parallel computing, but the latter term is most commonly used to refer to processing in which different parts of a program run simultaneously on two or more

processors that are part of the same computer. While both types of processing require that a program be segmented—divided into sections that can run simultaneously, distributed computing also requires that the division of the program take into account the different environments on which the different sections of the program will be running. For example, two computers are likely to have different file systems and different hardware components.

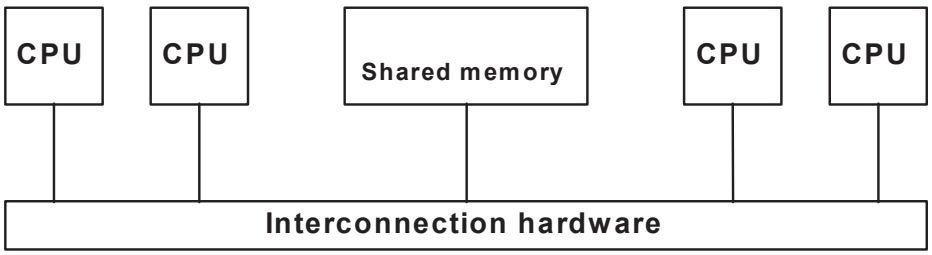
An example of distributed computing is BOINC, a framework in which large problems can be divided into many small problems which are distributed to many computers. Later, the small results are reassembled into a larger solution. Distributed computing is a natural result of using networks to enable computers to communicate efficiently. But distributed computing is distinct from computer networking or **fragmented** computing. The latter refers to two or more computers interacting with each other, but not, typically, sharing the processing of a single program. The World Wide Web is an example of a network, but not an example of distributed computing.

Advancements in microelectronic technology have resulted in the availability of fast, inexpensive processors, and advancements in communication technology have resulted in the availability of cost-effective and highly efficient computer networks. The net result of the advancements in these two technologies is that the price performance ratio has now changed to favor the use of inter-connected, multiple processors in place of a single, high-speed processor.

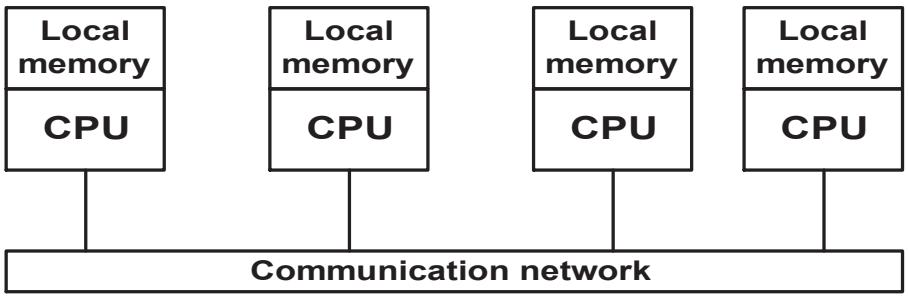
9.1.1 Basic Multiprocessor Systems

Two basic types of computer architectures consisting of interconnected, multiple processors can be distinguished as

1. ***Tightly Coupled Systems*** - systems with a single system wide primary memory (address space) that is shared by all the processors (also referred to as parallel processing systems, multiprocessors, SMMP - shared memory multiprocessors, SMS - shared memory systems, SMP – symmetric multiprocessors).



2. ***Loosely Coupled Systems*** - the systems where processors do not share memory and each processor has its own local memory (also referred to as distributed computing systems, multicomputer, DMS - distributed memory systems, MPP – massively parallel processors).



Let us see some points with respect to both tightly coupled multiprocessor systems and loosely coupled multiprocessor.

- Tightly coupled systems are referred to as parallel processing systems, and loosely coupled systems are referred to as distributed computing systems, or simply distributed systems.

- In case of tightly coupled systems, the processors of distributed computing systems can be located far from each other to cover a wider geographical area.
- In tightly coupled systems, the number of processors that can be usefully deployed is usually small and limited by the bandwidth of the shared memory.
- The Distributed computing systems are more freely expandable and can have an almost unlimited number of processors.

This has provided us with the basic idea of designing distributed operating systems. Although the field is still immature, with ongoing active research activities, commercial distributed operating systems have already started to emerge. These systems are based on already established basic concepts.

9.2 Distributed Computing System – An Outline

It is a collection of independent computers (nodes, sites) interconnected by transmission channels, that appear to the users of the system as a single computer.

Each node of distributed computing system is equipped with a processor, a local memory, and interfaces. Communication between any pair of nodes is realized only by message passing as no common memory is available. Usually, distributed systems are asynchronous, i.e., they do not use a common clock and do not impose any bounds on relative processor speeds or message transfer times.

9.3 Evolution of Distributed Computing System

Early computers were very expensive (they cost millions of dollars) and very large in size (they occupied a big room). These computers were run from a console by an operator and were not accessible to ordinary users. The job setup time was a real problem in early computers and wasted most of the valuable central processing unit (CPU) time. To increase CPU utilization several new concepts were introduced in the 1950s and 1960s like the batching together of jobs with similar needs before processing them, automatic sequencing of jobs, off-line processing by using the concepts of buffering and spooling and multiprogramming. Finally, multiprogramming improved CPU utilization by organizing jobs so that the CPU always had something to execute.

However, none of these ideas allowed multiple users to directly interact with a computer system and to share its resources simultaneously. It was not until the early 1970s that computers started to use the concept of time-sharing to overcome this hurdle. Parallel advancements in hardware technology allowed reduction in the size and increase in the processing speed of computers, causing large-sized computers to be gradually replaced by smaller and cheaper ones that had more processing capability than their predecessors.

The advent of time-sharing systems **was the first step was** distributed computing systems **because it provided us with two important concepts used in distributed computing systems-**

- The sharing of computer resources simultaneously by many users
- The accessing of computers from a place different from the main computer room.

However, in parallel, there were advancements in compute networking technology in the late 1960s and early 1970s that emerged as two key networking technologies-

- **LAN (Local Area Network)**: The LAN technology allowed several computers located within a building or a campus to be interconnected in such a way that these machines could exchange

information with each other at data rates of about 10 megabits per second (Mbps). The first high-speed LAN was the Ethernet developed at Xerox PARC in 1973

- **WAN Technology:** allowed computers located far from each other (may be in different cities or countries or continents) to be interconnected in a such a way that these machines could exchange information with each other at data rates of about 56 kilobits per second (Kbps). The first WAN was the ARPANET (Advanced Research Projects Agency Network) developed by the U.S. Department of Defense in 1969.
- **ATM Technology**: The data rates of networks continued to improve gradually in the 1980s providing data rates of up to 100 Mbps for LANs and data rates of up to 64 Kbps for WANs. Recently (early 1990s) there have been another major advancements in networking technology – the ATM (Asynchronous Transfer Mode) technology. The ATM technology is an emerging technology that is still not very well established. It will make very high speed networking possible, providing data transmission rates up to 1.2 gigabits per second (Gbps) in both LAN and WAN environments. The availability of such high-bandwidth networks will allow future distributed computing systems to support a completely new class of distributed applications, called multimedia applications, that deal with the handling of a mixture of information, including voice, video and ordinary data. The merging of computer and networking technologies gave birth to Distributed computing systems in the late 1970s.

9.4 Distributed Computing System Models

Various models are used for building distributed computing system. These models can be broadly classifies into five categories-minicomputer, workstation, workstation-server, processor-pool and hybrid**. They are briefly described below.**

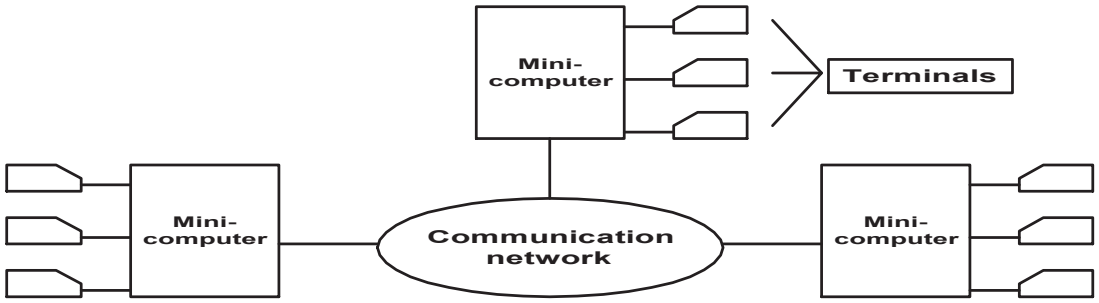
Distributed System

Item	Multiprocessor	Multicomputer	Distributed System
Node configuration	CPU	CPU, RAM, net interface	Complete computer
Node peripherals	All shared	Shared exc. maybe disk	Full set per node
Location	Same rack	Same room	Possibly worldwide
Internode communication	Shared RAM	Dedicated interconnect	Traditional network
Operating systems	One, shared	Multiple, same	Possibly all different
File systems	One, shared	One, shared	Each node has own
Administration	One organization	One organization	Many organizations

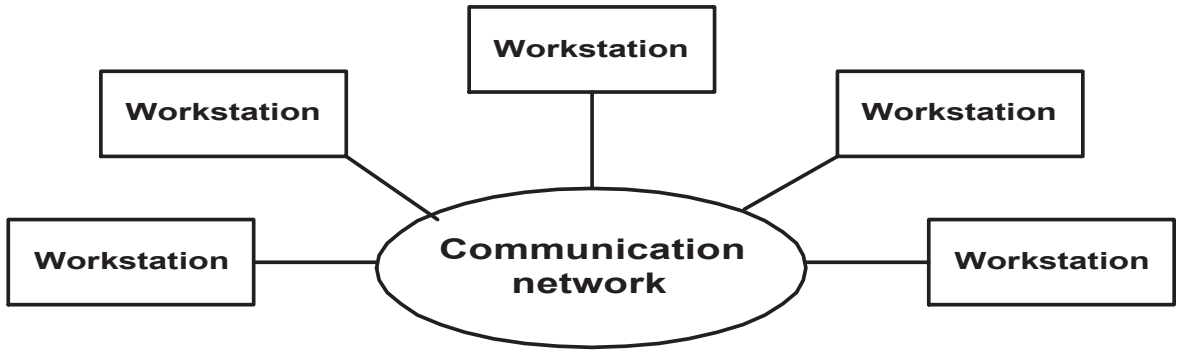
Comparison of three Kinds of Multiple CPU Systems

9.4.1 Minicomputer Model

Distributed computing system based on this model consists of a few minicomputers (or supercomputers) interconnected by a communication network. Each minicomputer is connected by several interactive terminals. Each user is logged on to one specific minicomputer, with access to remote resources available on other minicomputers. There is a simple extension of the centralized time sharing system. The example of distributed computing system based on minicomputers is the early ARPA net.



Workstation Model (NOW - Network of Workstations, P2P - Peer-to Peer)



General Characteristic:

- System consists of several workstations interconnected by a communication network.
- Every workstation may be equipped with its own disk and serving as a single-user computer.
- In such environment like company's office or a university department, at any one time (especially at night), a significant portion of the workstation are idle, resulting in the waste of large amount of CPU time.
- Main idea: interconnect all workstations by a high-speed LAN so that idle workstations may be used to process jobs of users who are logged onto other workstations and do not have sufficient processing power at their own workstations to get their jobs processed efficiently.
- User logs onto one of the workstations and submits job for execution.
- If the user's workstation does not have sufficient processing power for executing the processes of the submitted job efficiently, it transfers one or more of the processes from the user's workstation to some other workstation that is currently idle and gets the process executed there.
- The result of execution is returned to the user's workstation.
- Implementation issues:
  - How does the system find an idle workstation?
  - How is the process transferred from one workstation to get it executed on another workstation?
  - What happens to a remote process if a user logs onto a workstation that was idle until now and was being executed a process of another workstation?

- Examples of distributed computing systems based on the workstation model:
  - Sprite system; experimental system developed at Xerox PARC.

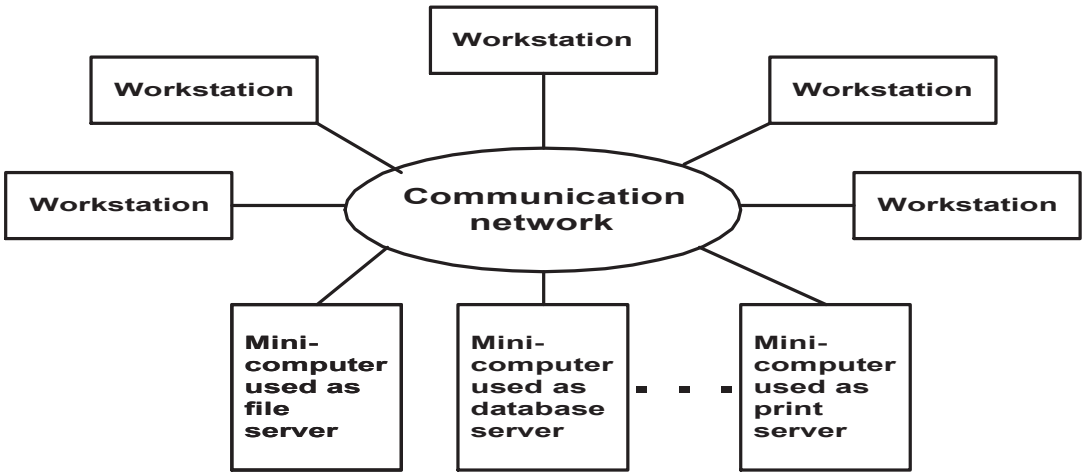
9.4.2 Workstation-Server Model

System consists of a few minicomputers and several workstations (diskless or diskful) interconnected by a communication network. In addition to the workstation, there are specialized machines running server processes (*servers*) for managing and providing access to shared resources.

Each minicomputer is used as a server machine to provide one or more types of service:

- implementing the file system;
- database service;
- print service;
- other types of service.

User logs onto a workstation called his home workstation. Normal computation activities required by the user's processes are performed at the user's home workstation. Requests for services provided by special servers are sent to a server providing that type of service that performs the user's requested activity and returns of requested processing to the user's workstation. User's processes need not be migrated to the server machines for getting the work done by those machines.



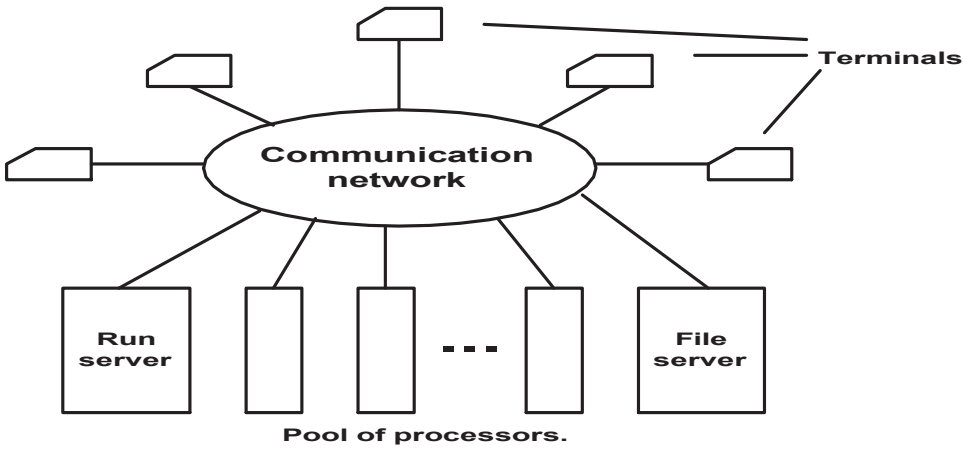
The workstation-server model has several advantages:

1. In general, it is much cheaper to use a few minicomputers equipped with large, fast disk that are accessed over the network than a large number of diskful workstations, with each workstation having a small, slow disk.
2. Diskless workstations are also preferred to diskful workstations from a system maintenance point of view. Backup and hardware maintenance are easier to perform with a few large disks than with many small disks scattered all over a building or campus. Furthermore, installing new releases of software (such as file server with new functionalities) is easier when the software is to be installed on a few file server machines than on every workstation.
3. In the workstation-server model, since the file servers manage all files, users have the flexibility to use any workstation and access the files in the same manner irrespective of which workstation the

user is currently logged on. Note that this is not the true with the workstation model, in which the workstation model, in which each workstation has its local file system, because different mechanisms are needed to access local and remote files.

4. In the workstation-server model, the request-response protocol is mainly used to access the services of the server machines. Therefore, unlike the workstation model, this model does not need a process migration facility, which is difficult to implement. The request-response protocol is known as the client-server model of communication. In this model, a client process (which in this case resides on a workstation) sends a request to server process (which in this case resides on a minicomputer) for getting some services such as reading a unit of a file. The server executes the request and sends back a reply to the client that contains the result of processing.
5. A user has guaranteed response time because workstations are not used for executing remote processes. However, the model does not utilize the processing capability of idle workstations.

9.4.3 Processor-Pool Model



The model is based on the observation that most of the time a user does not need any computer power but once in a while he may need a very large amount of computing power for a short time.

- The processors are pooled together to be shared by the users as needed.
- The pool of processors consists of a large number of microcomputers and minicomputers attached to the network.
- Each processor in the pool has its own memory to load and run a system program or an application program.
- The processors in the pool have not terminals attached directly to them, and users access the system from terminals that are attached to the network via special devices.
- A special server (*run server*) manages and allocates the processors in the pool to different users on a demand basis.
- Appropriate number of processors are temporary assigned to user‘s job by the run server.
- When the computation is completed, the processors are returned to the pool.

When a user submits a job for computation, the run server temporarily assigns an appropriate number of processors to his or her job. For example, if the user’s computation job is the compilation of a program having n segments, in which each of the segments can be compiled independently to produce separate

relocatable object files, n processors from the pool can be allocated to this job to compile all the n segments in parallel. When the computation is completed the processors are returned to the pool for use by other users.

In the processor-pool model there is no concept of a home machines. That is, a user does not log onto a particular machines but to the system as a whole. This is in contrast to other models in which each user has a home machine (e.g. a workstation or minicomputer) onto which he or she logs and runs most of his or her programs there by default. Amoeba and the Cambridge Distributed Computing Systems are examples of distributed computing systems based on the processor-pool model.

9.5 Security in Distributed Environment

Computing security is, at its core, more than a technical issue: It's a fundamental business challenge. Managers have plenty of security alternatives, but little real guidance on making intelligent decisions about them. And today's distributed, multivendor, Internet-connected environments encompass more insecure systems and networks than ever before.

Security in these systems is multilayered and can be tailored to meet company and user-specific needs Security in Distributed Computing offers the manager of distributed systems a thorough, common-sense framework for cost-effective computer security.

Security Policy determines precisely which actions the entities in a system are allowed to take and which ones are prohibited. The security policy can be enforced by following techniques:

ENCRYPTION: Transforming data into coded text that an attacker cannot understand.
AUTHENTICATION: Verifying the claimed identity of different entities.

AUTHORIZATION: Verifying whether the client is allowed to perform the requested service. .

AUDITING: Tracing each and every client's activity.

The security in distributed systems requires use of a unique, assigned login name for each user. This name must be used in conjunction with the system provided or created password unique to the user name to gain access.

The security protocol of distributed systems requires it check the login name and password against its files along with the access point to authenticate login. The system is unique in that it can accomplish this without an active server.

Each user has a personally constructed security profile. This profile only allows them access to certain areas of the files and programs located within the distributed system. This security protocol helps to keep information confidential by only allowing limited access.

9.5.1 Architecture

Various hardware and software architectures are used for distributed computing. At a lower level, it is necessary to interconnect multiple CPUs with some sort of network, regardless of whether that network is printed onto a circuit board or made up of loosely coupled devices and cables. At a higher level, it is necessary to interconnect processes running on those CPUs with some sort of communication system.

Distributed programming typically falls into one of several basic architectures or categories: Client-server, 3-tier architecture, N-tier architecture, Distributed objects, loose coupling, or tight coupling.

- **Client-server —** Smart client code contacts the server for data, then formats and displays it to the user. Input at the client is committed back to the server when it represents a permanent change.
- **3-tier architecture** — Three tier systems move the client intelligence to a middle tier so that stateless clients can be used. This simplifies application deployment. Most web applications are 3-Tier.
- **N-tier architecture —** N-Tier refers typically to web applications which further forward their requests to other enterprise services. This type of application is the one most responsible for the success of application servers.
- **Tightly coupled (clustered)** — refers typically to a set of highly integrated machines that run the same process in parallel, subdividing the task in parts that are made individually by each one, and then put back together to make the final result.
- **Peer-to-peer —**an architecture where there is no special machine or machines that provide a service or manage the network resources. Instead all responsibilities are uniformly divided among all machines, known as peers. Peers can serve both as clients and servers.
- **Space based** — refers to an infrastructure that creates the illusion (virtualization) of one single address-space. Data are transparently replicated according to application needs. Decoupling in time, space and reference is achieved.

Another basic aspect of distributed computing architecture is the method of communicating and coordinating work among concurrent processes. Through various message passing protocols, processes may communicate directly with one another, typically in a master/slave relationship. Alternatively, a "database-centric" architecture can enable distributed computing to be done without any form of direct inter-process communication, by utilizing a shared database.

9.6 Advantages of Distributed System Over Centralized System

The distributed computing systems are much more complex and difficult to build than traditional centralized systems (those consisting of a single CPU, its memory, peripherals, and one or more terminals). The increased complexity is mainly due to the fact that in addition to being capable of effectively using and managing a very large number of distributed resources, the system software of a distributed computing system should also be capable of handling the communication and security problems that are very different from those of centralized systems. For example, the performance and reliability of a distributed computing system depends to a great extent on the performance and reliability of the underlying communication network. Special software is usually needed to handle loss of messages, during transmission across the network or to prevent overloading of the network that degrades the performance and responsiveness to the users. Similarly, special software security measures are needed to protect the widely distributed shared resources and services against intentional or accidental violation of access control and privacy constraints.

Despite the increased complexity and the difficulty of building distributed computing systems, the installation and use of distributed computing systems overweigh their disadvantages. The technical needs, the economic pressures, and the major advantages that have led to the emergence and popularity of distributed computing systems are described here.

- **Inherently Distributed Applications:** Distributed computing systems come into existence in some very natural easy. For example, several applications are inherently distributed in nature and require a distributed computing system for their realization. For instance, in an employee database of a nationwide organization, the data pertaining to a particular employee are generated at the employee's

branch office, and in addition to the global need to view the entire database; there is a local need for frequent and immediate access to locally generated data at each branch office. Such applications require that some processing power be available at the many distributed locations for collecting, preprocessing, and accessing data, resulting in the need for distributed application are a computerized worldwide airline reservation system, a computerized banking system in which a customer can deposit/withdraw money from his or her account from any branch of the bank, and a factory automation system controlling robots and machines all along an assembly line.

- **Information Sharing Among Distributed Users:** Efficient person-to-person communication facility by sharing information over great distances is the one more advantage. In a distributed computing system, the users working at other nodes of the system can easily and efficiently share information generated by one of the users. This facility may be useful in many ways. For example, two or more users who are geographically far off from each other can perform a project but whose computers are the parts of the same distributed computing system.

  The use of distributed computing systems by a group of users to work cooperatively is known as computer-supported cooperative working (CSCW), or groupware.

- **Resource Sharing:** Information is not the only thing that can be shared in a distributed computing system. Sharing of software resources such as software libraries and databases as well as hardware resources such as printers, hard disks, and plotters can also be done in a very effective way among all the computers and users of a single distributed computing system

- **Better Price Performance Ration:** This is one of the most important reasons for the growing popularity of distributed computing system. With the rapidly increasing power and reduction in the price of microprocessors, combined with the increasing speed of communication networks, distributed computing systems potentially have a much better price-performance ratio than a single large centralized system. Another reason for distributed computing systems to be more cost effective than centralized systems is that they facilitate resource sharing among multiple computers.

- **Shorter Response Times and Higher Throughput:** Due to multiplicity of processors, distributed computing systems are expected to have better performance than single-processor centralized systems. The two most commonly used performance metrics are response time and throughput of user processes. That is, the multiple processors of distributed computing systems can be utilized properly for providing shorter response times and higher throughput than a single processor centralized system. Another method often used in distributed computing systems for achieving better overall performance is to distribute the load more evenly among the multiple processors by moving jobs from currently overloaded processors to lightly loaded ones.

- **Higher Reliability:** *Reliability* refers to the degree of tolerance against errors and component failures in a system. A reliable system prevents loss of information even in the event of component failures. The multiplicity of storage devices and processors in a distributed computing system allows the maintenance of multiple copies of critical information within the system. With this approach, if one of the processors fails, the computation can be successfully completed at the other processor, and if one of the storage devices fails, the computations can be successfully completed at the other processors, and if one of the storage devices fails, the information can still be used from the other storage device.

- **Availability:** An important aspect of reliability is *availability,* which refers to the fraction of time for which a system is available for use. In comparison to a centralized system, a distributed computing system also enjoys the advantage of increased availability.
- **Extensibility and Incremental Growth:** Another major advantage of distributed computing systems is that they are capable of **incremental growth**. That is, it is possible to gradually extend the power and functionality of a distributed computing system by simply adding additional resources (both hardware and software) to the system as and when the need arises. For example, additional processors can be easily added to the system to handle the increased workload of an organization that might have resulted from its expansion. **Extensibility** is also easier on a distributed computing system because addition of new resources to an existing system can be performed without significant disruption of the normal functioning of the system. Properly designed distributed computing systems that have the property of extensibility and incremental growth are called ***open distributed systems***.
- **Better Flexibility in Meeting Users Needs:** Different types of computers are usually more suitable for performing different types of computations. For example, computers with ordinary power are suitable for ordinary data processing jobs, whereas high-performance computers are more suitable for complex mathematical computations. In a centralized system, the users have to perform all types of computations on the only available computer.

9.7 Disadvantages of Distributed System Over Centralized System

Technical Issues

If not planned properly, a distributed system can decrease the overall reliability of computations if the unavailability of a node can cause disruption of the other nodes. Leslie Lamport famously quipped that: "A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable." Troubleshooting and diagnosing problems in a distributed system can also become more difficult, because the analysis may require connecting to remote nodes or inspecting communication between nodes. Many types of computation are not well suited for distributed environments, typically owing to the amount of network communication or synchronization that would be required between nodes. If bandwidth, latency, or communication requirements are too significant, then the benefits of distributed computing may be negated and the performance may be worse than a non-distributed environment.

Project-related Problems

Distributed computing projects may generate data that is proprietary to private industry, even though the process of generating that data involves the resources of volunteers. This may result in controversy as private industry profits from the data which is generated with the aid of volunteers. In addition, some distributed computing projects, such as biology projects that aim to develop thousands or millions of "candidate molecules" for solving various medical problems, may create vast amounts of raw data. This raw data may be useless by itself without refinement of the raw data or testing of candidate results in real-world experiments. Such refinement and experimentation may be so expensive and time-consuming that it may literally take decades to sift through the data. Until the data is refined, no benefits can be acquired from the computing work.

Other projects suffer from lack of planning on behalf of their well-meaning originators. These poorly planned projects may not generate results that are palpable, or may not generate data that ultimately result in finished, innovative scientific papers. Sensing that a project may not be generating useful data, the project managers

may decide to abruptly terminate the project without definitive results, resulting in wastage of the electricity and computing resources used in the project. Volunteers may feel disappointed and abused by such outcomes. There is an obvious opportunity cost of devoting time and energy to a project that ultimately is useless, when that computing power could have been devoted to a better planned distributed computing project generating useful, concrete results.

Another problem with distributed computing projects is that they may devote resources to problems that may not ultimately be soluble, or to problems that are best pursued later in the future, when desktop computing power becomes fast enough to make pursuit of such solutions practical. Some distributed computing projects may also attempt to use computers to find solutions by number-crunching mathematical or physical models. With such projects there is the risk that the model may not be designed well enough to efficiently generate concrete solutions. The effectiveness of a distributed computing project is therefore determined largely by the sophistication of the project creators.

9.7.1 Network Operating Systems

There are two Types of Distributed operating systems:

- Network Operating Systems
- Distributed Operating Systems

In the Network operating systems Users are aware of multiplicity of machines. User's access to resources of various machines is done explicitly by:

- Remote logging into the appropriate remote machine (telnet, ssh)
- Remote Desktop (Microsoft Windows)
- Transferring data from remote machines to local machines, via the File Transfer Protocol (FTP) mechanism

9.8 Issues in Designing a Distributed Operating System

In general, designing operating system is more difficult than designing a centralized operating system for several reasons. In the design of a centralized operating system, it is assumed that the operating system has access to complete and accurate information about the environment in which it is functioning. In a distributed system, the resources are physically separated, there is no common clock among the multiple processors, delivery of messages is delayed, and messages could even be lost. Due to all these reasons, a distributed operating system doest not have up-to-data, consistent knowledge about the state of the various components of the underlying distributed system. Despite these complexities and difficulties, a distributed operating system must be designed to provide all the advantage of a distributed system to its users. That is, the users should be able to view a distributed system as virtual centralized system that is flexible, efficient, reliable, secure and easy to use. To meet this requirement, the designers of a distributed operating system must deal with several design issues. Some of the key design issues are described below.

- **Transparency:** We saw that one of the main goals of a distributed operating system is to make the existence of multiple *computers invisible (transparent)* and provide a single system image to its users. That is, a distributed operating system must be designed in such a way that a collection of distinct machines connected by a communication subsystem appears to its users as a *virtual uni-processor*. The eight forms of transparency identified by the International Standards Organization's Reference Model for Open Distributed Processing [ISO 1992] are ***access transparency, location transparency, replication transparency, failure transparency, migration transparency, location transparency, concurrency transparency performance transparency, and scaling transparency***.

- **Reliability:** In general, distributed systems are expected to be more reliable than centralized systems due to the existence of multiple instances of resources. However, the existence of multiple instances of the resources alone cannot increase the systems reliability. Rather, the distributed operating system, which manages these resources, must be designed properly to increase the systems reliability by taking full advantage of this characteristic feature of a distributed system.

  For higher reliability, the fault-handling mechanisms of a distributed operating system must be designed properly to avoid faults, to tolerate faults, and to detect and recover from faults. Commonly used methods for dealing with these issues are briefly described here.

- **Flexibility:** Another important issue in the design of distributed operating systems is flexibility. The design of a distributed operating system should be flexible due to the following reasons:

  • ***Ease of modification:*** From the experience of system designers, it has been found that some parts of the design often need to be replaced/modified either because some bug is detected in the design or because the design is no longer suitable for the changed system environment or new-user requirements. Therefore, it should be easy to incorporate changes in the system in a user-transparent manner or with minimum interruption caused to the users.

  • ***Ease of enhancement:*** In every system, new functionalities have to be added from time to time to make it more powerful and easy to use. Therefore, it should be easy to add new services to the system.

  The most important design factor that influences the flexibility of a distributed operating system is the model used for designing its kernel. The *kernel* of an operating system is its central controlling part that provided basic system facilities. It operates in a separate address space that a user cannot replace or modify. The two commonly used models for kernels design in distributed operating systems are **monolithic kernel and the micro kernel.**

  • **In *monolithic kernel*** model, the kernel provides most operating system services such as process management, and inter-process communication. As a result, the kernel has a large, monolithic structure. Many distributed operating systems that are extensions or imitations of the UNIX operating system use the monolithic kernel model. This is mainly because UNIX itself has a large, monolithic kernel.

  • ***In the micro kernel*** model, the main goal is to keep the kernel as small as possible. Therefore, in this model, the kernel is a very small nucleus of software that provides only the minimal facilities necessary for implementing additional operating system services. The only services provided by the kernel in this model are inter-process communication, low–level device management and some memory management. All other operating system services, such as file-management, name management, additional process and memory management activities, and much system call handling are implemented as a user-levelserver processes.

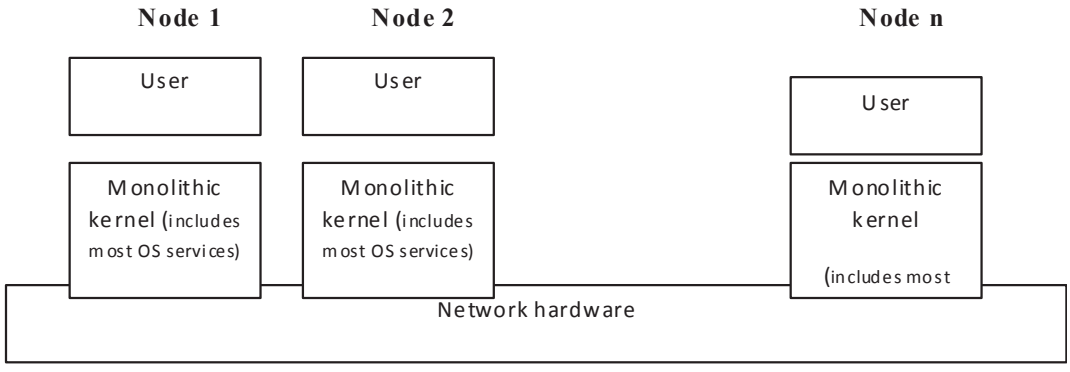


Fig 2.6(a) The Monolithic Kernel Model

- **Performance:** If a distributed system is to be used, its performance must be at least as good as a centralized system. That is, when a particular application is run on a distributed system, its overall performance should be better than or at least equal to that of running the same applications on a single-processor system. However, to achieve this goal, it is important that the various components of the operating system of a distributed system be designed properly; otherwise, the overall performance of the distributed system may turn out to be worse than a centralized system.

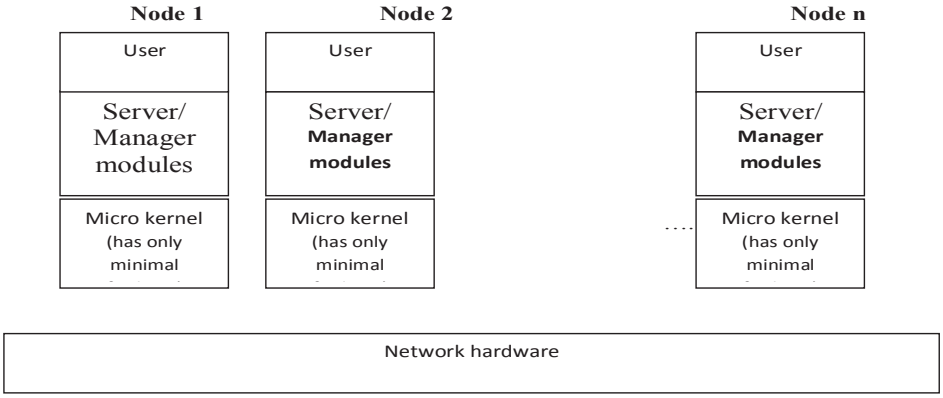


Fig 2.6(b) The Micro Kernel Model

- **Scalability:** Scalability refers to the capability of a system to adapt to increased service load. It is inevitable that a distributed system will grow with time since it is very common to add new machines or an entire sub-network to the system to take care of increased workload or organizational changes in a company. Therefore, a distributed operating system should be designed to easily cope with the growth of nodes and users in the system. That is, such growth should not cause serious disruption of service or significant loss of performance to users.

- **Heterogeneity:** A heterogeneous distributed system consists of interconnected sets of dissimilar hardware or software systems. Because of the diversity, designing heterogeneous distributed systems is far more difficult than designing homogenous distributed systems in which each system is based on the same, or closely related, hardware and software. However, as a consequence of large scale, heterogeneity is often inevitable in distributed systems. Furthermore, many users prefer often heterogeneity because heterogeneous distributed systems provide the flexibility to their users of different computer platforms for different applications.

- **Security:** In order that the users can trust the system and rely on it, the various resources of a computer system must be protected against destruction and unauthorized access. Enforcing security in a distributed system is more difficult than in a centralized system because of the lack of a single point of control and the use of insecure networks for data communication. In a centralized system, all users are authenticated by the system at login time, and the system can easily check whether a user is authorized to perform the requested operation on an accessed resource. In a distributed system, however, since the client-server model is often used for requesting and providing services, when a client sends a request message to a server, the server must have some way of knowing who is the client. This is not so simple as it might appear because any client identification field in the message cannot be trusted. This is because an intruder (a person or program trying to obtain unauthorized access to system resources) may pretend to be an authorized client or may change the message contents during transmission. Therefore, as compared to a centralized system, enforcement of security in a distributed system has the following additional requirements:

1. It should be possible for the sender of a message to know that the intended receiver received the message.

2 It should be possible for receiver of a message to know that the message was sent by the genuine sender

3. It should be possible for both the sender and receiver of a message to be guaranteed that the contents of the message were not changed while it was in transfer.

- **Emulation of Existing Operating Systems:** For commercial success, it is important that a newly designed distributed operating system be able to emulate existing poplar operating systems such as UNIX. With this property, new software can be written using the system call interface of the new operating system to take full advantage of its special features of distribution, but a vast amount to already existing old software can also be run on the same system without the need to rewrite them. Therefore, moving to the new distributed operating system will allow both types of software to be run side by side.

9.9 Summary

Let us sum up the different concepts we have studied till here.

- The existing models for distributed computing systems can be broadly classified into five categories, minicomputer, workstation-server, processor-pool and hybrid.
- Distributed computing system is much more complex and difficult to build than the traditional centralized systems. Despite the increased complexity and the difficulty of buildings, the installation and the use of distributed computing system are rapidly increasing. This is mainly because the advantages of distributed computing systems outweigh its disadvantages.
- The main advantages of distributed computing systems are (a) suitability for inherently distributed applications. (b) Sharing of information among distributed users and sharing of resources (d) better price performance ratio (e) shorter response times and higher throughout (f) higher reliability (g) extensibility and incremental growth and (h) better flexibility in meeting users needs.
- The operating systems commonly used for distributed computing systems can be broadly classified into two types: *network operating systems and distributed operating systems*. As compared to a network operating system, a distributed operating system has better transparency and fault capability and provides the image of a virtual uniprocessor to the users.
- The main issue involved in the design of a distributed operating system is transparency, reliability, flexibility, performance, scalability, heterogeneity, security and emulation of existing operating systems.

9.10 Self - Assessment Exercise

1. Differentiate between Centralised approach and Fully Distributed Approach
2. Identify the disadvantages of distributed approach in comparison to Centralised approach.
3. What are the main issues involve in the design of a distributed system.
4. Explain the different models of Distributed Computing.

5. What are the security issues related to a system with distributed approach?

9.11 References

- Attiya, Hagit and Welch, Jennifer (2004). *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. Wiley-Interscience. ISBN 0471453242
- http://en.wikipedia.org/wiki/Distributed_computing
- Nadiminti, Dias de Assunção, Buyya (September 2006). "Distributed Systems and Recent Innovations: Challenges and Benefitz". *InfoNet Magazine, Volume 16,*
- *Issue 3, Melbourne, Australia*

Unit - 10 : Distributed File System

Structure of Unit

10.0 Objective

This unit covers following aspects:

- Features of Good Distributed File System
- File Models & File Accessing Models
- File- Sharing Semantics
- System Design Issue & Design Principle

10.1 Introduction

A file system is responsible for the organization, storage, retrieval, naming, sharing, and protection of files. A Distributed File System is a network file system where a single file system can be distributed across several physical computer nodes. Separate nodes have direct access to only a part of the entire file system, in contrast to shared disk file systems where all nodes have uniform direct access to the entire storage. Example: Google file system, CODA, Hadoop.

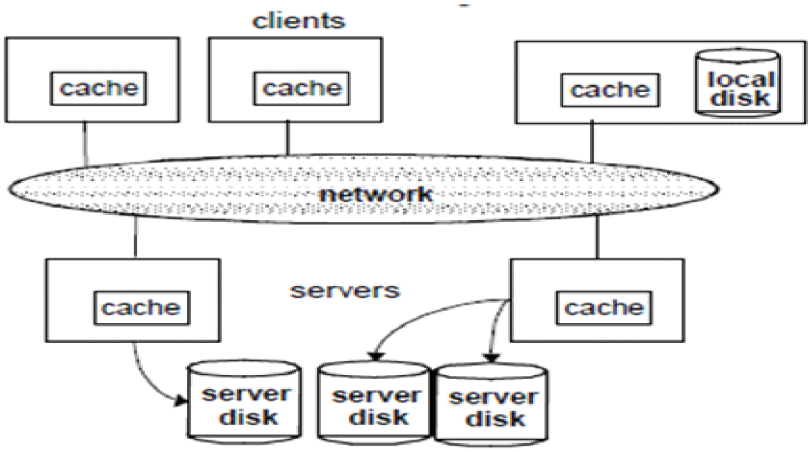


Figure 10.1: Distributed File System

10.2 Features of Good DFS

1) **Fault tolerance**: Distributed storage is composed of a large number of distributed storage components (rather than a single storage component). Increasing the number of components affects fault tolerance. Distributing the components makes the system more faults prone (because of network disruptions).

2) **Scalability:** The file system should work well in small environments (1 machine, a dozen machines) and also scale gracefully to huge ones (hundreds through tens of thousands of systems).

3) **Transparency:** The distributed systems should be perceived as a single entity by the users or the application programmers rather than as a collection of autonomous systems, which are cooperating. The users should be unaware of where the services are located and also the transferring from a local machine to a remote one should also be transparent.
   - **a.** **Access transparency** Clients are unaware that files are distributed and can access them in the same way as local files are accessed.
   - **b.** **Location transparency** A consistent name space exists encompassing local as well as remote files. The name of a file does not give it location.
   - **c.** **Concurrency transparency** All clients have the same view of the state of the file system. This means that if one process is modifying a file, any other processes on the same system or remote systems that are accessing the files will see the modifications in a coherent manner.
   - **d.** **Failure transparency** The client and client programs should operate correctly after a server failure.
   - **e.** **Heterogeneity** File service should be provided across different hardware and operating system platforms.
   - **f.** **Replication transparency** To support scalability, we may wish to replicate files across multiple servers. Clients should be unaware of this.
   - **g.** **Migration transparency** Files should be able to move around without the client's knowledge.

Other Characteristics Include

1) **Network Transparency:** Same access operation as if they are local files.

2) **Location Independence:** The file name should not be changed when the physical location of the file changes.

3) **User Mobility:** User should be able to access the file from anywhere.

4) **File Mobility:** Moves files from one place to the other in a running system.

10.3 File models & File Accessing Models

10.3.1 Distributed File System Concepts

A file service is a specification of what the file system offers to clients. A file server is the implementation of a file service and runs on one or more machines. A file itself contains a name, data, and attributes (such as owner, size, creation time, access rights). An immutable file is one that, once created, cannot be changed. Immutable files are easy to cache and to replicate across servers since their contents are guaranteed to remain unchanged. Two forms of protection are generally used in distributed file systems, and they are essentially the same techniques that are used in single-processor non-networked systems:

Capabilities

Each user is granted a ticket (capability) from some trusted source for each object to which it has access. The capability specifies what kinds of access are allowed.

Access Control Lists

Each file has a list of users associated with it and access permissions per user. Multiple users may be organized into an entity known as a group.

10.3.2 File Service Types

To provide a remote system with file service, we will have to select one of two models of operation. One of these is the upload/download model. In this model, there are two fundamental operations: read file transfers an entire file from the server to the requesting client, and write file copies the file back to the server. It is a simple model and efficient in that it provides local access to the file when it is being used. Three problems are evident. It can be wasteful if the client needs access to only a small amount of the file data. It can be problematic if the client doesn't have enough space to cache the entire file. Finally, what happens if others need to modify the same file? The second model is a remote access model. The file service provides remote operations such as open, close, read bytes, write bytes, get attributes, etc. The file system itself runs on servers. The drawback in this approach is the servers are accessed for the duration of file access rather than once to download the file and again to upload it.

Another important distinction in providing file service is that of understanding the difference between directory service and file service. A directory service, in the context of file systems, maps human-friendly textual names for files to their internal locations, which can be used by the file service. The file service itself provides the file interface (this is mentioned above). Another component of file distributed file systems is the client module. This is the client-side interface for file and directory service. It provides a local file system interface to client software (for example, the vnode file system layer of a UNIX kernel).

10.3.3 Naming Issues

In designing a distributed file service, we should consider whether all machines (and processes) should have the exact same view of the directory hierarchy. We might also wish to consider whether the name space on all machines should have a global root directory (a.k.a. super root) so that files can be accessed as, for example, //server/path. This is a model that was adopted by the Apollo Domain System, an early distributed file system, and more recently by the web community in the construction of a uniform resource locator (URL).

In considering our goals in name resolution, we must distinguish between location transparency and location independence. By location transparency we mean that the path name of a file gives no hint to where the file is located. For instance, we may refer to a file as //server1/dir/file. The server (server) can move anywhere without the client caring, so we have location transparency. However, if the file moves to server2 things will not work. If we have location independence, the files can be moved without their names changing. Hence, if machine or server names are embedded into path names we do not achieve location independence. It is desirable to have access transparency, so that applications and users can access remote files just as they access local files. To facilitate this, the remote file system name space should be syntactically consistent with the local name space. One way of accomplishing this is by redefining the way files are named and require an explicit syntax for identifying remote files. This can cause legacy applications to fail and user discontent (users will have to learn a new way of naming their files). An alternate solution is to use a file system mounting mechanism to overlay portions of another file system over a node in a local directory structure. Mounting is used in the local environment to construct a uniform name space from separate file systems (which reside on different disks or partitions) as well as incorporating special-purpose file systems into the name space (e.g. /proc on many UNIX systems allows file system access to processes). A remote file system can be mounted at a particular point in the local directory tree. Attempts to access files and directories under that node will be directed to the driver for that file system.

To summarize, our naming options are:

- machine and path naming (machine:path, ./machine/path).
- mount remote file systems onto the local directory hierarchy (merging the two name spaces).
- provide a single name space which looks the same on all machines.

The first two of these options are relatively easy to implement.

Types of Names

When we talk about file names, we refer to symbolic names (for example, server.c). These names are used by people (users or programmers) to refer to files. Another "name" is the identifier used by the system internally to refer to a file. We can think of this as a binary name (more precisely, as an address). On most UNIX file systems, this would be the device number and inode number. On MS-DOS systems, this would be the drive letter and FAT index. Directories provide a mapping from symbolic names to file addresses (binary names).

Typically, one symbolic name maps to one file address. If multiple symbolic names map onto one binary name, these are called hard links. On inode-based file systems (e.g., most UNIX systems), hard links must exist within the same device since the address (inode) is unique only on that device. On MS-DOS systems, they are not supported because file attributes are stored with the name of the file. Having two symbolic names refer to the same data will cause problems in synchronizing file attributes (how would you locate other files that point to this data?). A hack to allow multiple names to refer to the same file (whether its on the same device or a different device) is to have the symbolic name refer to a single file address but that file may have an attribute to tell the system that its contents contain a symbolic file name that should be dereferenced. Essentially, this adds a level of indirection: access a file which contains another file name, which references the file attributes and data. These files are known as symbolic links. Finally, it is possible for one symbolic name to refer to multiple file addresses. This doesn't make much sense on a local system1, but can be useful

on a networked file system to provide fault tolerance or enable the system to use the file address which is most efficient.

10.4 File- Sharing Semantics

The analysis of file sharing semantics is that of understanding how files behave. For instance, on most systems, if a read follows a write, the read of that location will return the values just written. If two writes occur in succession, the following read will return the results of the last write. File systems that behave this way are said to observe sequential semantics. Sequential semantics can be achieved in a distributed system if there is only one server and clients do not cache data. This can cause performance problems since clients will be going to the server for every file operation (such as single-byte reads). The performance problems can be alleviated with client caching. However, now if the client modifies its cache and another client reads data from the server, it will get obsolete data. Sequential semantics no longer hold. One solution is to make all the writes write-through to the server. This is inefficient and does not solve the problem of clients having invalid copies in their cache. To solve this, the server would have to notify all clients holding copies of the data.

Another solution is to relax the semantics. We will simply tell the users that things do not work the same way on the distributed file system as they did on the local file system. The new rule can be "changes to an open file are initially visible only to the process (or machine) that modified it." These are known as session semantics. Yet another solution is to make all the files immutable2. That is, a file cannot be open for modification, only for reading or creating. If we need to modify a file, we'll create a completely new file under the old name.

Immutable files are an aid to replication but they do not help with changes to the file's contents (or, more precisely, that the old file is obsolete because a new one with modified contents succeeded it). We still have to contend with the issue that there may be another process reading the old file. It's possible to detect that a file has changed and start failing requests from other processes. A final alternative is to use atomic transactions. To access a file or a group of files, a process first executes a begin transaction primitive to signal that all future operations will be executed indivisibly. When the work is completed, an end transaction primitive is executed. If two or more transactions start at the same time, the system ensures that the end result is as if they were run in some sequential order. All changes have an all or nothing property.

10.5 System Design Issue

10.5.1 Name Resolution

In looking up the pathname of a file (e.g. via the namei function in the UNIX kernel), we may choose to evaluate a pathname a component at a time. For example, for a pathname aaa/bbb/ccc, we would perform a remote lookup of aaa, then another one of bbb, and finally one of ccc). Alternatively, we may pass the rest of the pathname to the remote machine as one lookup request once we find that a component is remote. The drawback of the latter scheme is

(a) The remote server may be asked to walk up the tree by processing .. (parent node) components and reveal more of its file system than it wants and

(b) Other components cannot be mounted underneath the remote tree on the local system. Because of this, component at a time evaluation is generally favored but it has performance problems (a lot more messages). We may choose to keep a local cache of component resolutions.

10.5.2 Should Servers Maintain State?

This issue is a topic of passionate debate. A stateless system is one in which the client sends a request to a server, the server carries it out, and returns the result. Between these requests, no client-specific information is stored on the server. A stateful system is one where information about client connections is maintained on the server. In a stateless system:

- Each request must be complete – the file has to be fully identified and any offsets specified.
- Fault tolerance: if a server crashes and then recovers, no state was lost about client connections because there was no state to maintain.
- No remote open/close calls are needed (they only serve to establish state).
- No wasted server space per client.
- No limit on the number of open files on the server; they aren't "open" – the server maintains no per-client state.
- No problems if the client crashes. The server does not have any state to clean up. On a stateful system:
- requests are shorter (less info to send).
- better performance in processing the requests.
- idempotency works; cache coherence is possible.
- file locking is possible; the server can keep state that a certain client is locking a file (or portion thereof).

10.5.3 File Caching Schemes

We can employ caching to improve system performance. There are four places in a distributed system where we can hold data:

1. on the server's disk
2. in a cache in the server's memory
3. in the client's memory
4. on the client's disk

The first two places are not an issue since any interface to the server can check the centralized cache. It is in the last two places that problems arise and we have to consider the issue of cache consistency. Several approaches may be taken: write-through What if another client reads its own cached copy? All accesses would require checking with the server first (adds network congestion) or require the server to maintain state on who has what files cached. Write-through also does not alleviate congestion on writes. delayed writes Data can be buffered locally (where consistency suffers) but files can be updated periodically. A single bulk write is far more efficient than lots of little writes every time any file contents are modified. Unfortunately the semantics become ambiguous.

Write on Close

This is admitting that the file system uses session semantics.

Centralized Control

Server keeps track of who has what open in which mode. We would have to support a stateful system and deal with signalling traffic.

10.5.4 Fault Tolerance

In brief, we can say that in a computational system data are processed to produce information. Once produced, usually these information are stored in a media by the ûle system for further accesses Their necessity of being accessible implies to provide safe mechanisms for storing and accessing data/information which claims fault tolerance issues. Besides, in distributed ûle systems, not only local failures (e.g., due storage devices) should be dealt with, but also other failures inherent to the distributed environment. Following, we address some issues that are strict related with fault tolerance in distributed ûle systems.

Stateful and Stateless Services

To better understand how fault tolerance can be employed in a distributed ûle system, it is useful to understand how the services are provided. Distributed ûle systems services can be implemented by using two different service designs: stateful service or stateless service. These paradigms have contradictory concepts, however both supply the ûle operations.

Stateful Service

In this type of service, information about ûle operations are kept in the server during all the ûle session. A communication channel is established between the client and the server when a the client explicitly solicits the ûle opening. A number (identiûer) is used to deûne the communication channel then this identiûer will be used to perform ûle operations. To attend its clients, the server copies data from the storage devices to memory and let them there till the ûle closing.

Stateless Service

On the other hand, the stateless service does not establish a communication channel. Moreover, there is no necessity for explicit ûle opening and closing: before executing a ûle operation the server will automatically open and close the ûle. Each request sent to the server must deûne the desired ûle likewise, if a read or write operation is requested, it must contain the position in the ûle referring to the respective operation.

The usage of the main memory can improve performance whereas memory access is faster than disk one. The memory can be used for caching in the stateless service but its usage is not obligatory as in stateful service. As a result to this, stateful service presents an advantage when compared to the stateless one. On the contrary, the advantage of using the main memory becomes a disadvantage with respect to the fault tolerance context. If the server crashes, information stored in memory will probably be lost or at least harder to be recovered.

The consequences of servers and clients failures will depend of the used service. If a stateless server crashes, the previous ûle sessions will not be disturbed. On the other hand, if the server crashes in a stateful service, it should be able to recover the ûle session state, likewise it has inconveniences as mentioned earlier. Focusing on the client failures, in a stateful service the server should be able to realize when it happens to free the allocated memory. On the contrary, stateless service servers do not need to handle client faults. However, stateless service clients can experience a situation in which they cannot distinguish a slow server from a recovering one.

Finally, we can also compare both approaches referring to service overhead. Due to the communication channel established on the stateful service its overhead is signiûcant lower than the stateless service. The reason is that in a stateful service it is not necessary to send details about the ûle operation each time a

request is sent. Additionally, such a service can be understood as a centralized and coupled service. In contrast, a stateless service is decentralized and decoupled that delegates service tasks to clients. The great decentralized participation of clients in stateless service allows to better providing fault tolerance mechanisms once it is possible to avoid single points of failures.

10.5.5 File Replication

In distributed systems, replication techniques can be employed with different goals. Targeting consistency, for example, replication is useful for performance issues, e.g., accessing data at the same time or accessing the data copy whose network communication has a low latency. Replication can also be used for availability and fault tolerance. moreover, replication means ûle replication in distributed ûle system context. Furthermore, replication mechanisms should to keep replicas consistent even if they are used to provide fault tolerance. It implies a fault tolerance trade-off between consistency and performance. The techniques used in replication should choose one of these characteristics to prioritize.

- **Consistency vs. Performance**. When a client performs a ûle operation the ûle changes have to be updated to all its replicas. Consistency protocols manage them, providing atomicity to replica updates. In other words, these protocols ensure that all replicas of a ûle will correspond to the last change done in it. To achieve this, consistency protocols must avoid that clients open outdated replicas. This task is not trivial for distributed ûle systems that deal with mutable data (ûles). By that very fact, such protocols increase the system overhead which can degrade system performance. Consequently, the harder the consistency protocol is, the lower the performance that the system will acquire.
- **File Replication Location**. Another interesting aspect of ûle replication is the place where it will be replicated. Basically, it is possible to explore two approaches. The ûrst one relies on replicating them on the same machine - be replicas on the same storage device or on different ones. This approach can proût a RAID scheme if relied on different medias (hard disks). In opposition, the second approach concerns replicas in different machines likewise using the network infrastructure to manage them.

It is important to remember that despite the approach chosen, clients should not care about ûle replication, i.e., it should be transparent for them. However, details about ûle replication (e.g., number of replicas) can be exposed to clients if the distributed ûle system allows them to tune it by themselves. Yet, there is the possibility to provide a hybrid approach that relies on ûle replication on different machines and also taking advantage of different Medias (in some servers or in all of them). This scenario may increase the system throughput however, if well tuned, it can offer an environment to build robust fault tolerance services without degrading the system performance. Moreover, fault tolerance consistency protocols could improve their performance when relying on this hybrid approach. Fault tolerant mechanisms would proût from different machines to avoid single points of failure and the consistency protocol could be better performed relying on faster accesses to local storage devices.

10.6 Design Principle : Andrew File System (AFS)

AFS was developed in the late 80s at CMU. It uses the following design principles:

1. Callbacks: The server records that has the copy of a file.
2. Write-back cache on file close: If a file is modified, the update is propagated to server when the file is closed. The server then immediately tells all clients who own an old copy.

3. Files are cached on each client's disk. NFS caches only in clients' memory.

4. Session semantics: Updates are only visible on close.

In UNIX (single machine), updates are visible immediately to other processes that have the file open. In AFS, everyone who has the file open sees the old version; anyone who opens the file again will see the new version.

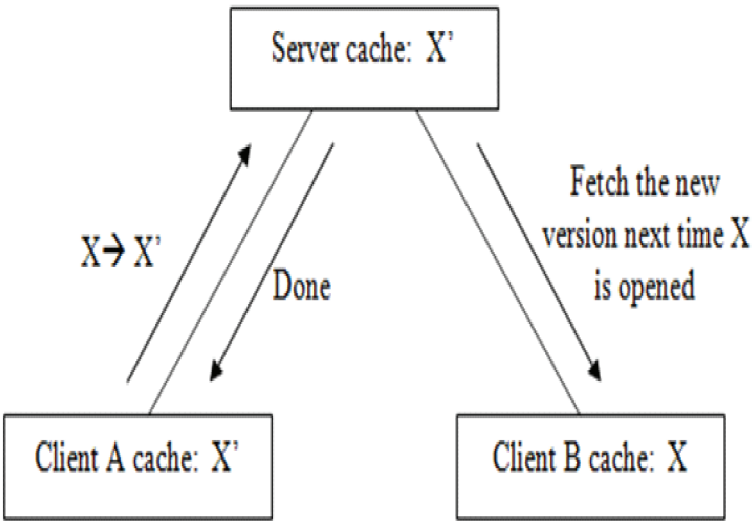


Figure 10.6: AFS Process

When a client opens a file and the file is not on the local disk, the client gets the file from the server and adds itself on the callback list. When a client closes a file, the client sends the updated copy back to the server and tells all clients to get the new version on the next open.

If the server crashes, the server loses all the callback states and needs to ask all clients to reconstruct the callback states.

10.7 Case Study

10.7.1 DCE Distributed File Service.

Distributed Computing Environment

- **DCE**

A vendor-independent distributed computing environment, DCE was defined by the Open Software Foundation (OSF), a consortium of computer manufacturers, including IBM, DEC, and Hewlett-Packard. It is not an operating system, nor is it an application. Rather, it is an integrated set of services and tools that can be installed as a coherent environment on top of existing operating systems and serve as a platform for building and running distributed applications.

A primary goal of DCE is vendor independence. It runs on many different kinds of computers, operating systems, and networks produced by different vendors. For example, some operating systems to which DCE can be easily ported include OSF/1, AIX, DOMAIN OS, ULTRIX, HP-UX, SINIX, SunOS, UNIX System V, VMS, WINDOWS, and OS/2. On the other hand, it can be used with any network hardware and transport software, including TCP/IP, X.25, as well as other similar products. As shown in Figure 10.7, DCE is middleware software layered between the DCE applications layer and the operating system and networking layer. The basic idea is to take a collection of existing machines (possibly from different vendors), interconnect them by a communication network, add the DCE software platform

on top of the native operating systems of the machines, and then be able to build and run distributed applications. Each machine has its own local operating system, which may be different from that of other machines. The DCE software layer on top of the operating system and networking layer hides the differences between machines by automatically performing data-type conversions when necessary. Therefore, the heterogeneous nature of the system is transparent to the applications programmers, making their job of writing distributed applications much simpler.

DCE Application
DCE Software
Operating System & Networking

Table 10.7(a): Position of DCE software in a DCE-based distributed system

- **DCE Creation?**

The OSF did not create DCE from scratch. Instead, it created DCE by taking advantage of work already done at universities and industries in the area of distributed computing. For this, OSF issued a request for technology (RFT), asking for tools and services needed to build a coherent distributed computing environment. To be a contender, a primary requirement was that actual working code must ultimately be provided. The submitted bids were carefully evaluated by OSF employees and a team of outside experts. Finally, those tools and services were selected that the members of the evaluation committee believed provided the best solutions. The code comprising the selected tools and services, almost entirely written in C, was then further developed by OSF to produce a single integrated package that was made available to the world as DCE. Version 1.0 of DCE was released by OSF in January 1992.

- **DCE Components**

As mentioned above, DCE is a blend of various technologies developed independently and nicely integrated by OSF. Each of these technologies forms a component of DCE. The main components of DCE are as follows:

1. **Threads Package:** It provides a simple programming model for building concurrent applications. It includes operations to create and control multiple threads of execution in a single process and to synchronize access to global data within an application.

2. **Remote Procedure Call (RPC) Facility:** It provides programmers with a number of powerful tools necessary to build client-server applications. In fact, the DCE RPC facility is the basis for all communication in DCE because the programming model underlying all of DCE is the client-server model. It is easy to use, is network- and protocol-independent, provides secure communication between a client and a server, and hides differences in data requirements by automatically converting data to the appropriate forms needed by clients and servers.

3. **Distributed Time Service (DTS):** It closely synchronizes the clocks of all the computers in the system. It also permits the use of time values from external time sources, such as those of the U.S. National Institute for Standards and Technology (NIST), to synchronize the clocks of the computers in the system with external time. This facility can also be used to synchronize the clocks of the computers of one distributed environment with the clocks of the computers of another distributed environment.

4. **Name Services:** The name services of DCE include the Cell Directory Service (CDS), the Global Directory Service (GDS), and the Global Directory Agent (GDA). These services allow resources such as servers, files, devices, and so on, to be uniquely named and accessed in a location-transparent manner.

5. **Security Service:** It provides the tools needed for authentication and authorization to protect system resources against illegitimate access.

6. **Distributed File Service (DFS):** It provides a system wide file system that has such characteristics as location transparency, high performance, and high availability. A unique feature of DCE DFS is that it can also provide file services to clients of other file systems.

The DCE components listed above are tightly integrated. It is difficult to give a pictorial representation of their interdependencies because they are recursive. For example, the name services use RPC facility for internal communication among its various servers, but the RPC facility uses the name services to locate the destination. Therefore, the interdependencies of the various DCE components can be best depicted in tabular form, as shown in Figure.

Component name	Other Component used by it
Threads	None
RPC	Threads, name, security
DTS	Threads, RPC, name, security
Name	Threads, RPC, DTS, security
Security	Threads, RPC, DTS, name
DFS	Threads, RPC, DTS, name, security

Table 10.7(b): Interdependencies of DCE components

• DCE Cells

The DCE system is highly scalable in the sense that a system running DCE can have thousands of computers and millions of users spread over a worldwide geographic area. To accommodate such large systems, DCE uses the concept of cells. This concept helps break down a large system into smaller, manageable units called cells. In a DCE system, a cell is a group of users, machines, or other resources that typically have a common purpose and share common DCE services. The minimum cell configuration requires a cell directory server, a security server, a distributed time server, and one or more client machines. Each DCE client machine has client processes for security service, cell directory service, distributed time service, RPC facility, and threads facility. A DCE client machine may also have a process for distributed file service if a cell configuration has a DCE distributed file server. Due to the use of the method of intersection for clock synchronization, it is recommended that each cell in a DCE system should have at least three distributed time servers. An important decision to be made while setting up a DCE system is to decide the cell boundaries. The following four factors should be taken into consideration for making this decision.

1. **Purpose:** The machines of users working on a common goal should be put in the same cell, as they need easy access to a common set of system resources. That is, users of machines in the same cell

have closer interaction with each other than with users of machines in different cells. For example, if a company manufactures and sells various types of products, depending on the manner in which the company functions, either a product-oriented or a function-oriented approach may be taken to decide cell boundaries [Tanenbaum 1995]. In the product-oriented approach, separate cells are formed for each product, with the users of the machines belonging to the same cell being responsible for all types of activities (design, manufacturing, marketing, and support services) related to one particular product. On the other hand, in the function-oriented approach, separate cells are formed for each type of activity, with the users belonging to the same cell being responsible for a particular activity, such as design, of all types of products.

2. **Administration:** Each system needs an administrator to register new users in the system and to decide their access rights to the system's resources. To perform his or her job properly, an administrator must know the users and the resources of the system. Therefore, to simplify administration jobs, all the machines and their users that are known to and manageable by an administrator should be put in a single cell. For example, all machines belonging to the same department of a company or a university can belong to a single cell. From an administration point of view, each cell has a different administrator.

3. **Security:** Machines of those users who have greater trust in each other should be put in the same cell. That is, users of machines of a cell trust each other more than they trust the users of machines of other cells. In such a design, cell boundaries act like firewalls in the sense that accessing a resource that belongs to another cell requires more sophisticated authentication than accessing a resource that belongs to a user's own cell.

4. **Overhead:** Several DCE operations, such as name resolution and user authentication, incur more overhead when they are performed between cells than when they are performed within the same cell. Therefore, machines of users who frequently interact with each other and the resources frequently accessed by them should be placed in the same cell The need to access a resource of another cell should arise infrequently for better overall system performance Notice from the above discussion that in determining cell boundaries the emphasis is on purpose, administration, security, and performance. Geographical considerations can, but do not have to, play a part in cell design. For better performance, it is desirable to have as few cells as possible to minimize the number of operations that need to cross cell boundaries. However, subject to security and administration constraints, it is desirable to have smaller cells with fewer machines and users. Therefore, it is important to properly balance the requirements imposed by the four factors mentioned above while deciding cell boundaries in a DCE system.

10.7.2 Sun Network File System (NFS)

Sun's NFS is one of the most popular and widespread distributed file systems in use today.

The design goals of NFS were:

- Any machine can be a client and/or a server.
- NFS must support diskless workstations (that are booted from the network). Diskless workstations were Sun's major product line.
- Heterogeneous systems should be supported: clients and servers may have different hardware and/or operating systems. Interfaces for NFS were published to encourage the widespread adoption of NFS.

- High performance: try to make remote access as comparable to local access through caching and read-ahead.

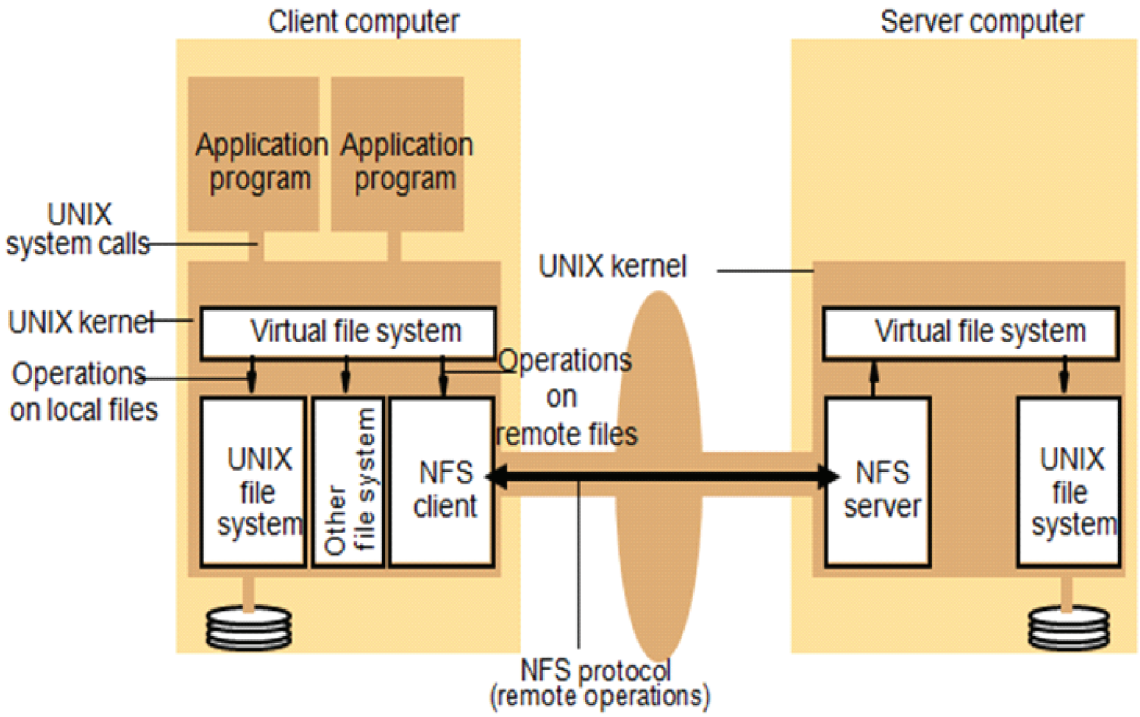


Figure 10.7(c): NFS Architecture

From a transparency point of view NFS offers:

Access Transparency

Remote (NFS) files are accessed through normal system calls; the protocol is implemented under the VFS (vnode) layer in UNIX.

Location Transparency

The client adds remote file systems to its local name space via mount. File systems must be exported at the server. The user is unaware of which directories are local and which are remote. The location of the mount point in the local system is up to the client's administrator.

Failure Transparency

NFS is stateless; UDP is used as a transport. If a server fails, the client retries.

Performance Transparency

Caching at the client will be used to improve performance

No migration Transparency

The client mounts machines from a server. If the resource moves to another server, the client must know about the move.

No support for Unix Semantics

NFS is stateless, so stateful operations such as file locking are a problem. All UNIX file system controls may not be available.

Devices

Since NFS had to support diskless workstations, where every file is remote, remote device

files had to refer to the client's local devices. Otherwise there would be no way to access local devices in a diskless environment.

- **NFS Protocols**

The NFS client and server communicate over remote procedure calls (Sun's RPC) using two protocols: the mounting protocol and the directory and file access protocol. The mounting protocol is used to request a access to an exported directory (and the files and directories within that file system under that directory). The directory and file access protocol is used for accessing the files and directories (e.g. read/write bytes, create files, etc.). The use of RPC's external data representation (XDR) allows NFS to communicate with heterogeneous machines. The initial design of NFS ran only with remote procedure calls over UDP. This was done for two reasons. The first reason is that UDP is somewhat faster than TCP but does not provide error correction (the UDP header provides a checksum of the data and headers). The second reason is that UDP does not require a connection to be present. This means that the server does not need to keep per-client connection state and there is no need to re-establish a connection if a server was rebooted.

The lack of UDP error correction is remedied in the fact that remote procedure calls have built-in retry logic. The client can specify the maximum number of retries (default is 5) and a timeout period. If a valid response is not received within the timeout period the request is re-sent. To avoid server overload, the timeout period is then doubled. The retry continues until the limit has been reached. This same logic keeps NFS clients fault-tolerant in the presence of server failures: a client will keep retrying until the server responds.

- **Mounting Protocol**

The client sends the pathname to the server and requests permission to access the contents of that directory. If the name is valid and exported (listed in /etc/dfs/sharetab on System V release 4 versions of UNIX, and /etc/exports on many other versions) the server returns a file handle to the client. This file handle contains all the information needed to identify the file on the server: {file system type, disk ID, inode number, and security info}. Mounting an NFS file system is accomplished by parsing the path name, contacting the remote machine for a file handle, and creating an in-core vnode at the mount point. A vnode points to an inode for a local UNIX file or, in the case of NFS, an rnode. The rnode contains specific information about the state of the file from the point of view of the client.

- **Directory and File Access Protocol**

Clients send RPC messages to the server to manipulate files and directories. A file is accessed by performing a lookup remote procedure call. This returns a file handle and attributes. It is not like an open in that no information is stored in any system tables on the server. After that, the handle may be passed as a parameter for other functions. For example, a read(handle, offset, count) function will read count bytes from location offset in the file referred to by handle. The entire directory and file access protocol is encapsulated in sixteen functions.

These are:

null	no-operation but ensure that connectivity exists
lookup	lookup the file name in a directory
create	create a file or a symbolic link

remove	remove a file from a directory
rename	rename a file or directory
read	read bytes from a file
write	write bytes to a file
link	create a link to a file
symlink	create a symbolic link to a file
readlink	read the data in a symbolic link (do not follow the link)
mkdir	create a directory
rmdir	remove a directory
readdur	read from a directory
getattr	get attributes about a file or directory (type, access and modify times, and access permissions)
setattr	set file attributes
statfs	get information about the remote file system

- **Accessing Files**

Files are accessed through conventional system calls (thus providing access transparency). If you recall conventional UNIX systems; a hierarchical pathname is dereference to the file location with a kernel function called namei. This function maintains a reference to a current directory looks at one component and finds it in the directory, changes the reference to that directory, and continues until the entire path is resolved. At each point in traversing this pathname, it checks to see whether the component is a mount point, meaning that name resolution should continue on another file system. In the case of NFS, it continues with remote procedure calls to the server hosting that file system.

Upon realizing that the rest of the pathname is remote, name will continue to parse one component of the pathname at a time to ensure that references to and to symbolic links become local if necessary. Each component is retrieved via a remote procedure call which performs an NFS lookup. This procedure returns a file handle. An in-core rnode is created and the VFS layer in the file system creates a vnode to point to it.

The application can now issue read and write system calls. The file descriptor in the user's process will reference the in-core vnode at the VFS layer, which in turn will reference the in core rnode at the NFS level which contains NFS-specific information, such as the file handle. At the NFS level, NFS read, write, etc. operations may now be performed, passing the file handle and local state (such as file offset) as parameters. No information is maintained on the server between requests; it is a stateless system. The RPC requests have the user ID and group ID number sent with them. This is a security hole that may be stopped by turning on RPC encryption.

- **Problems**

The biggest problem with NFS is file consistency. The caching and validation policies do not guarantee session semantics. NFS assumes that clocks between machines are synchronized and performs no clock

synchronization between client and server. One place where this hurts is in distributed software development environments. A program such as make, which compares times of files (such as object and source) to determine whether to regenerate them, can either fail or give confusing results.

Because of its stateless design, open with append mode cannot be guaranteed to work. You can open a file, get the attributes (size), and then write at that offset, but you'll have no assurance that somebody else did not write to that location after you received the attributes. In that case your write will overwrite the other once since it will go to the old end-of-file byte offset. Also because of its stateless nature, file locking cannot work. File locking implies that the server keeps track of which processes have locks on the file. Sun's solution to this was to provide a separate process (a lock manager) that does keep state.

One common programming practice under UNIX file systems for manipulating temporary data in files is to open a temporary file and then remove it from the directory. The name is gone, but the data persists because you still have the file open. Under NFS, the server maintains no state about remotely opened files and removing a file will cause the file to disappear. Since legacy applications depended on this, Sun's solution was to create a special hack for UNIX: if the same process that has a file opens attempts to delete it, it is instead moved to a temporary name and deleted on close. It's not a perfect solution, but it works well. Permission bits might change on the server and disallow future access to a file. Since NFS is stateless, it has to check access permissions each time it receives an NFS request. With local file systems, once access is granted initially, a process can continue accessing the file even if permissions change. By default, no data is encrypted and Unix-style authentication is used (used ID, group ID). NFS supports two additional forms of authentication: Diffie-Hellman and Kerberos. However, data is never encrypted and user-level software should be used to encrypt files if this is necessary.

10.7.3 OSF

Distributed file systems are an important component of an overall plan for distributed computing. Two such plans are currently being promulgated, one by OSF and the other by UI. A high-level view of these plans is shown in Figure.

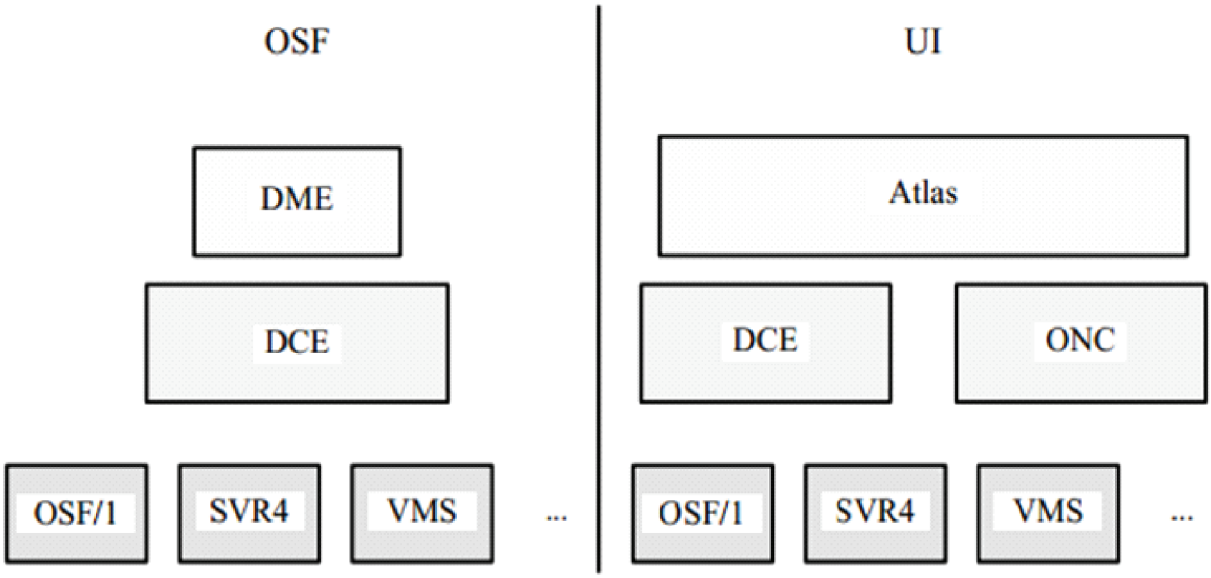


Figure 10.7(d): Competing Distributed Architecture Plans

UNIX International (UI), with the aid of (a portion of) the computer industry, has devised an overall framework for an industry-standard distributing computing architecture. The Open Software Foundation (OSF), in the meantime, has been developing an actual distributed computing architecture, known as DCE (for distributed computing environment). Fortunately, as it turns out, OSF's DCE fits within the UI Atlas view of the world. However, Sun's ONC (Open Network Computing) also fits within this scheme. The intent of both groups is that, whatever distributed architecture is adopted, it will support (or be supportable by) most existing operating systems, not just UNIX. DCE is currently well into development. Two initial, "functionality" versions of it have been released, and a production-quality release is expected within a year. As a separate project, OSF is working on DME (distributed management environment), whose concern is the management of services within a distributed environment.

10.8 Summary

In this unit, we discussed the various distributed file system. A file system is responsible for the organization, storage, retrieval, naming, sharing, and protection of files. A Distributed File System is a network file system where a single file system can be distributed across several physical computer nodes. Fault tolerance, scalability and transparency etc are the features of good distributed file system. Each file has a list of users associated with it and access permissions per user. Multiple users may be organized into an entity known as a group. These are managed by the Access Control List. In distributed ûle systems, not only local failures (e.g., due storage devices) should be dealt with, but also other failures inherent to the distributed environment. Stateless and stateful services are generally considered in it. At last, we compared various file system like AFS, NFS etc. in case study.

10.9 Self - Assessment Exercise

1. Briefly explain the features of Good DFS.
2. What do you mean by File System? Explain different File System issues.
3. Explain DCE.
4. Explain the NFS and its protocols.
5. Explain OSF with suitable diagram.

10.10 References

- Distributed Operating Systems by Andrew S. Tenenbaumb
- Distributed File Systems: Concepts and example by ELIEZER LEVY and ABRAHAM SILBERSCHATZ

Unit - 11: Message Passing

Structure of Unit

11.0 Objectives

This chapter provides-

- Features and desirable issues in IPC
- Buffering and different Strategies
- Encoding and decoding of message data
- Process addressing and failure handling

11.1 Introduction

Message passing is a method by which an object sends data to another object or requests other object to invoke method. This is also known as interfacing. It acts like a messenger from one object to other object to convey specific instructions. In this model, processes or objects can send and receive messages (comprising zero or more bytes, complex data structures, or even segments of code) to other processes. By waiting for messages, processes can also synchronize. In other words we can say that Message Passing is a system or communication where messages are sent from a sender to one or more recipients. Forms of

messages include (remote) method invocation, signals, and data packets. When designing a message passing system several choices are made:

- Whether messages are transferred reliably
- Whether messages are guaranteed to be delivered in order
- Whether messages are passed one-to-one (unicast), one-to-many (multicast or broadcast), many-to-one (client–server), or many-to-many (AllToAll).
- Whether communication is synchronous or asynchronous.

Interprocess communication (IPC) basically requires information sharing among two or more processes. Two basic methods for information sharing are as follows:

- Original sharing, or **shared-data approach**;
- Copy sharing, or **message-passing approach**.

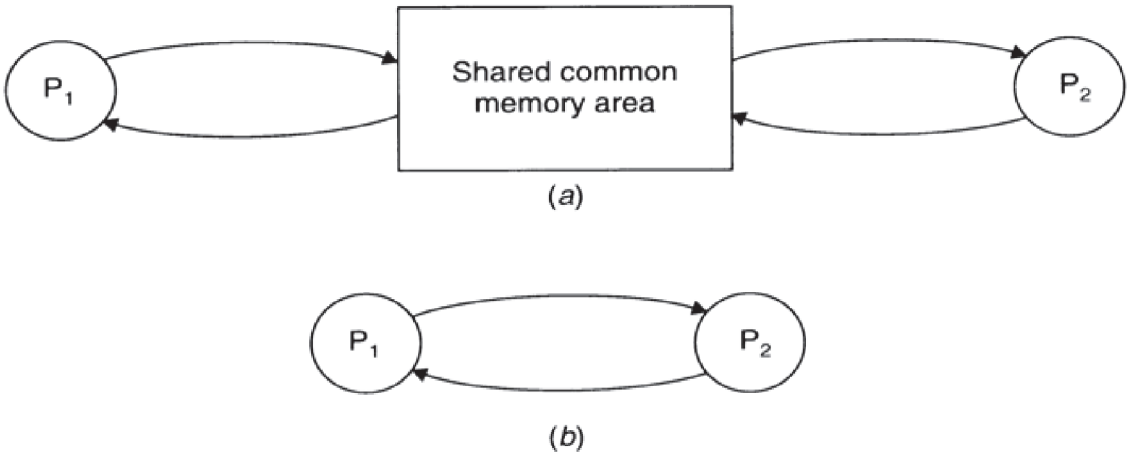


Figure 11.2 Message Passing

Two basic intercrosses communication paradigms: the shared data approach and message passing approach.

In the shared-data approach, the information to be shared is placed in a common memory area that is accessible to all processes involved in an IPC.

In the message-passing approach, the information to be shared is physically copied from the sender process's space to the address space of all the receiver processes, and this is done by transmitting the data to be copied in the form of messages (message is a block of information).

A message-passing system is a subsystem of distributed operating system that provides a set of message-based IPC protocols, and does so by shielding the details of complex network protocols and multiple heterogeneous platforms from programmers. It enables processes to communicate by exchanging messages and allows programs to be written by using simple communication primitives, such as send and receive.

11.2 Features of a Good Message Passing System

11.2.1 Simplicity

A message passing system should be simple and easy to use. It should be possible to communicate with old and new applications, with different modules without the need to worry about the system and network aspects.

11.2.2 Uniform Semantics

In a distributed system, a message-passing system may be used for the following two types of interprocess communication:

- **Local Communication**, in which the communicating processes are on the same node;
- **Remote Communication**, in which the communicating processes are on different nodes.

Semantics of remote communication should be as close as possible to those of local communications. This is an important requirement for ensuring that the message passing is easy to use.

11.2.3 Efficiency

An IPC protocol of a message-passing system can be made efficient by reducing the number of message exchanges, as far as practicable, during the communication process. Some optimizations normally adopted for efficiency include the following:

- avoiding the costs of establishing and terminating connections between the same pair of processes for each and every message exchange between them;
- minimizing the costs of maintaining the connections;
- Piggybacking of acknowledgement of previous messages with the next message during a connection between a sender and a receiver that involves several message exchanges.

11.2.4 Correctness

Correctness is a feature related to IPC protocols for group communication. Issues related to correctness are as follows:

- atomicity;
- ordered delivery;
- Survivability.

Atomicity ensures that every message sent to a group of receivers will be delivered to either all of them or none of them. Ordered delivery ensures that messages arrive to all receivers in an order acceptable to the application. Survivability guarantees that messages will be correctly delivered despite partial failures of processes, machines, or communication links.

11.2.5 Other features

- Reliability
- Flexibility
- Security
- Portability

11.3 Desirable Issues in IPC by Message Passing & Synchronization

11.3.1 Introduction

A message is a block of information formatted by a sending process in such a manner that it is meaningful to the receiving process. It consists of a fixed-length header and a variable-size collection of typed data objects. The header usually consists of the following elements:

- **Address:** It contains characters that uniquely identify the sending and receiving processes in the network.

- **Sequence Number:** This is the message identifier (ID), which is very useful for identifying lost messages and duplicates messages, in case of system failures.
- **Structural Information:** This element also has two parts. The type part specifies whether the data to be passed on to the receiver is included within the message or the message only contains a pointer to the data, which is stored somewhere outside the contiguous portion of the message. The second part of this element specifies the length of the variable-size message data.

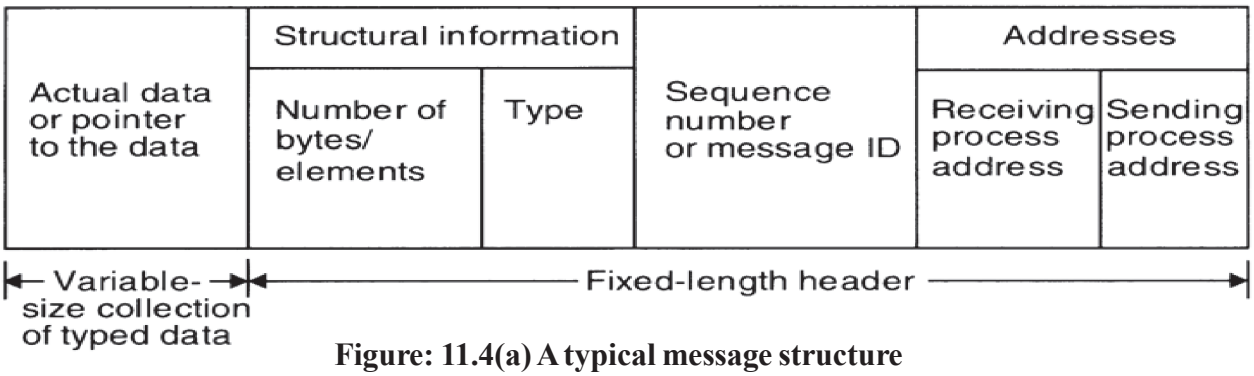


Figure: 11.4(a) A typical message structure

11.4.2 Synchronization

A central issue in the communication structure is the synchronization imposed on the communicating processes by the communication primitives. The semantics used for synchronization may by broadly classify as blocking and nonblocking types. A primitive is said to have nonblocking semantics if its invocation does not block the execution of its invoker (the control returns almost immediately to the invoker); otherwise a primitive is said to be of the blocking type.

In case of a blocking send primitive, after execution of the send statement, the sending process is blocked until it receives an acknowledgement from the receiver that the message has been received. On the other hand, for nonblocking send primitive, after execution of the send statement, the sending process is allowed to proceed with its execution as soon as the message has been copied to a buffer.

In the case of blocking receive primitive, after execution of the receive statement, the receiving process is blocked until it receives a message. On the other hand, for a nonblocking receive primitive, the receiving process proceeds with its execution after execution of the receive statement, which returns control almost immediately just after telling the kernel where the message buffer is.

An important issue in a nonblocking receive primitive is how the receiving process knows that the message has arrived in the message buffer. One of the following two methods is commonly used for this purpose:

- **Polling**: In this method, a test primitive is provided to allow the receiver to check the buffer status. The receiver uses this primitive to periodically poll the kernel to check if the message is already available in the buffer.
- **Interrupt**: In this method, when the message has been filled in the buffer and is ready for use by the receiver, a software interrupt is used to notify the receiving process.

A variant of the nonblocking receive primitive is the conditional receive primitive, which also returns control to the invoking process almost immediately, either with a message or with an indicator that no message is available.

When both the send and receive primitives of a communication between two processes use blocking semantics, the communication is said to be synchronous, otherwise it is asynchronous. The main drawback of synchronous communication is that it limits concurrency and is subject to communication deadlocks.

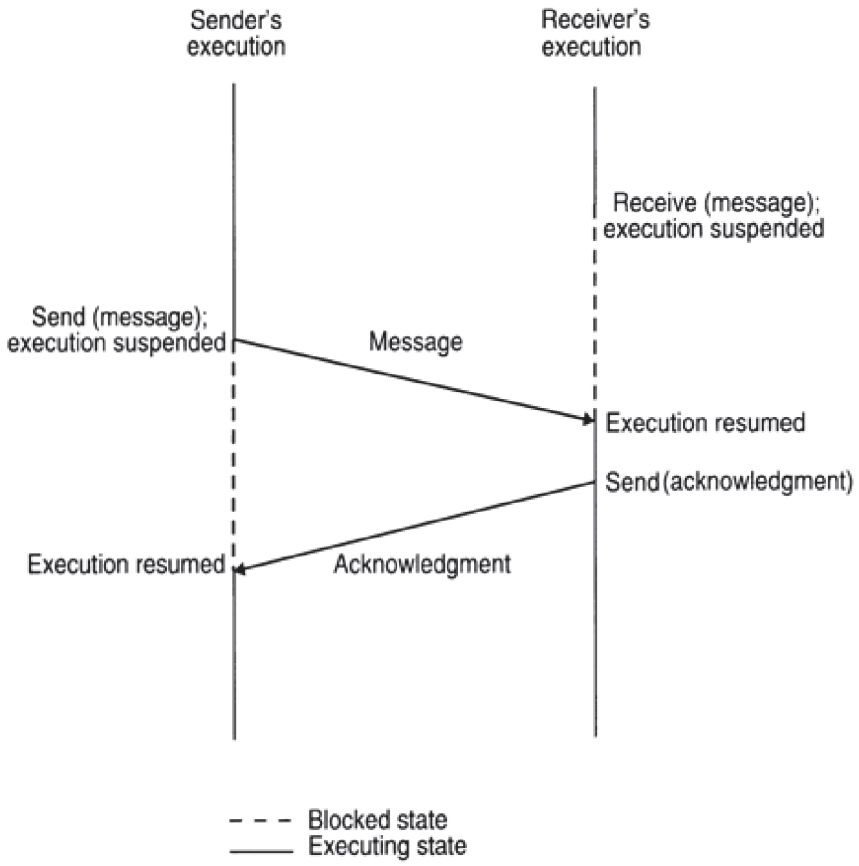


Figure 11.4(b) Synchronous mode of communication

Synchronous mode of communication with both **sends** and **receives** primitives having blocking-type semantics

11.4 Buffering

In the standard message passing model, messages can be copied many times: from the user buffer to the kernel buffer (the output buffer of a channel), from the kernel buffer of the sending computer (process) to the kernel buffer in the receiving computer (the input buffer of a channel), and finally from the kernel buffer of the receiving computer (process) to a user buffer.

11.4.1 Null Buffer (No Buffering)

In this case, there is no place to temporarily store the message. Hence one of the following implementation strategies may be used:

- The message remains in the sender process's address space and the execution of the send is delayed until the receiver executes the corresponding receive.
- The message is simply discarded and the time-out mechanism is used to resend the message after a timeout period. The sender may have to try several times before succeeding.

11.4.2 Strategies

The three types of buffering strategies used in interprocess communication

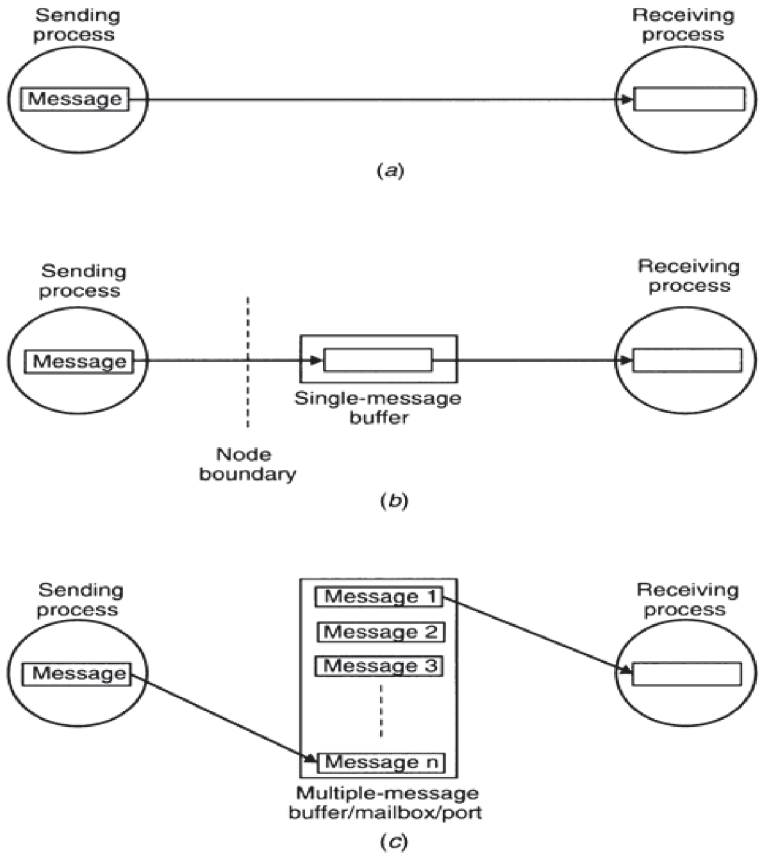


Figure 11.5 Buffering Strategy

11.4.2.1 Single-Message Buffer

In single-message buffer strategy, a buffer having a capacity to store a single message is used on the receiver's node. This strategy is usually used for synchronous communication, an application module may have at most one message outstanding at a time.

11.4.2.2 Unbounded-Capacity Buffer

In the asynchronous mode of communication, since a sender does not wait for the receiver to be ready, there may be several pending messages that have not yet been accepted by the receiver. Therefore, an unbounded-capacity message-buffer that can store all unreceived messages is needed to support asynchronous communication with the assurance that all the messages sent to the receiver will be delivered.

11.4.2.3 Finite-Bound Buffer

Unbounded capacity of a buffer is practically impossible. Therefore, in practice, systems using asynchronous mode of communication use finite-bound buffers, also known as multiple-message buffers. In this case message is first copied from the sending process's memory into the receiving process's mailbox and then copied from the mailbox to the receiver's memory when the receiver calls for the message.

When the buffer has finite bounds, a strategy is also needed for handling the problem of a possible buffer overflow. The buffer overflow problem can be dealt with in one of the following two ways:

- **Unsuccessful Communication.** In this method, message transfers simply fail, whenever there is no more buffer space and an error is returned.
- **Flow-controlled Communication**. The second method is to use flow control, which means that the sender is blocked until the receiver accepts some messages, thus creating space in the buffer for new messages. This method introduces synchronization between the sender and the receiver and may result in unexpected deadlocks. Moreover, due to the synchronization imposed, the asynchronous send does not operate in the truly asynchronous mode for all send commands.

11.5 Multi Datagram Messages

Almost all networks have an upper bound of data that can be transmitted at a time. This size is known as maximum transfer unit (MTU). A message whose size is greater than MTU has to be fragmented into multiples of the MTU, and then each fragment has to be sent separately. Each packet is known as a datagram. Messages larger than the MTU are sent in miltipackets, and are known as multidatagram messages.

11.5.1 Keeping Track of Lost and Out-of-Sequence Packet in Multidatagram Messages

A message transmission can be considered to be complete only when all the packets of the message have been received by the process to which it is sent. For successful completion of a multidatagram message transfer, reliable delivery of every packet is important. A simple way to ensure this is to acknowledge each packet separately (called **stop-and-wait protocol**). To improve communication performance, a better approach is to use a single acknowledgment packet for all the packets of a multidatagram message (called **blast protocol**). However, when this approach is used, a node crash or a communication link failure may lead to the following problems:

- One or more packets of the multidatagram message are lost in communication.
- The packets are received out of sequence by the receiver.

An efficient mechanism to cope with these problems is to use a bitmap to identify the packets of a message.

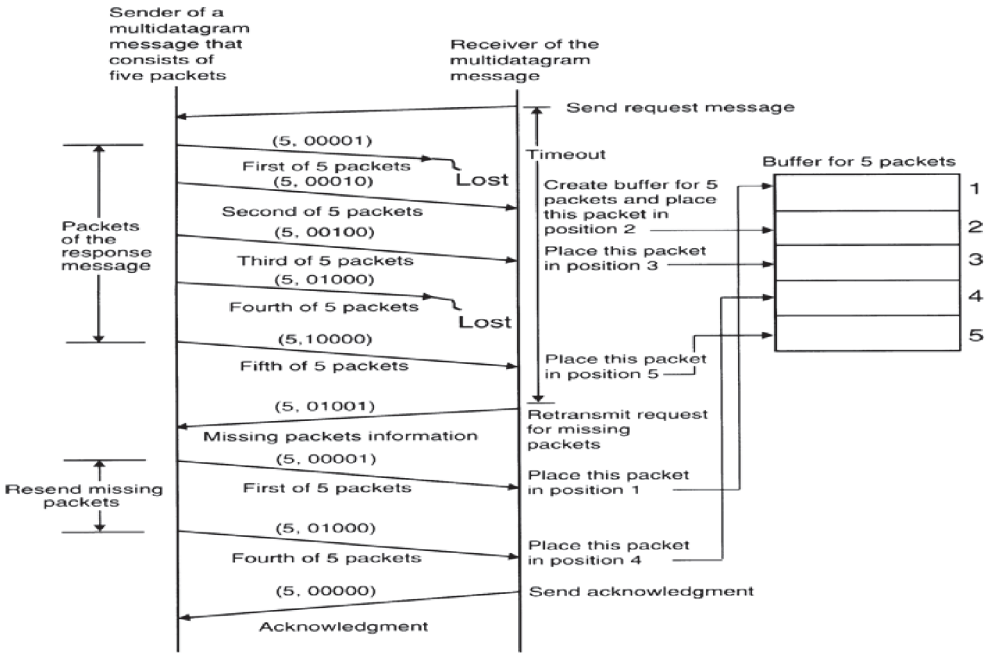


Figure 11.6 Packets in Multidatagram Messages

An example of the use of a bitmap to keep track of lost and out of sequence packets in a multidatagram message transmission

11.6 Encoding and Decoding of Message Data

A message data should be meaningful to the receiving process. This implies that, ideally, the structure of program objects should be preserved while they are being transmitted from the address space of the sending process to the address space of the receiving process. However, even in homogenous systems, it is very difficult to achieve this goal mainly because of two reasons:

- An absolute pointer value loses its meaning when transferred from one process address space to another.
- Different program objects occupy varying amount of storage space. To be meaningful, a message must normally contain several types of program objects, such as long integers, short integers, variable-length character strings, and so on.

In transferring program objects in their original form, they are first converted to a stream form that is suitable for transmission and placed into a message buffer. The process of reconstruction of program object from message data on the receiver side is known as decoding of message data. One of the following two representations may by used for the encoding and decoding of a message data:

- In **tagged representation** the type of each program object along with its value is encoded in the message.
- In **untagged representation** the message data only contains program object. No information is included in the message data to specify the type of each program object.

11.7 Process Addressing

Another important issue in message-based communication is addressing (or naming) of the parties involved in an interaction. For greater flexibility a message-passing system usually supports two types of process addressing:

- **Explicit Addressing**: The process with which communication is desired is explicitly named as a parameter in the communication primitive used.
- **Implicit Addressing**: The process willing to communicate does not explicitly name a process for communication (the sender names a server instead of a process). This type of process addressing is also known as functional addressing.

11.7.1 Methods to Identify a Process (naming)

(a) **send** (process_id, message)
Send a message to the process identified by "process_id".

(b) **receive** (process_id, message)
Receive a message from the process identified by "process_id".

(c) **send_any** (service_id, message)
Send a message to any process that provides the service of type "service_id".

(d) **receive_any** (process_id, message)
Receive a message from any process and return the process identifier ("process_id") of the process from which the message was received.

A simple method to identify a process is by a combination of machine_id and local_id. The local_id part is a process identifier, or a port identifier of a receiving process, or something else that can by used to uniquely identify a process on a machine. The machine_id part of the address is used by the sending machine's kernel to send the message to the receiving process's machine, and the local_id part of the address is then used by the kernel of the receiving process's machine to forward the message to the process for which it is intended.

A drawback of this method is that it does not allow a process to migrate from one machine to another if such a need arises.

To overcome the limitation of the above method, process can be identified by a combination of the following three fields: machine_id, local_id and machine_id.

- The first field identifies the node on which the process was created.
- The second field is the local identifier generated by the node on which the process was created.
- The third field identifies the last known location (node) of the process.

Another method to achieve the goal of location transparency in process addressing is to use a two-level naming scheme for processes. In this method each process has two identifiers: a high-level name that is machine independent (an ASCII string) and the low-level name that is machine dependent (such as pair (machine_id, local_id). A name server is used to maintain a mapping table that maps high-level names of processes to their low-level names.

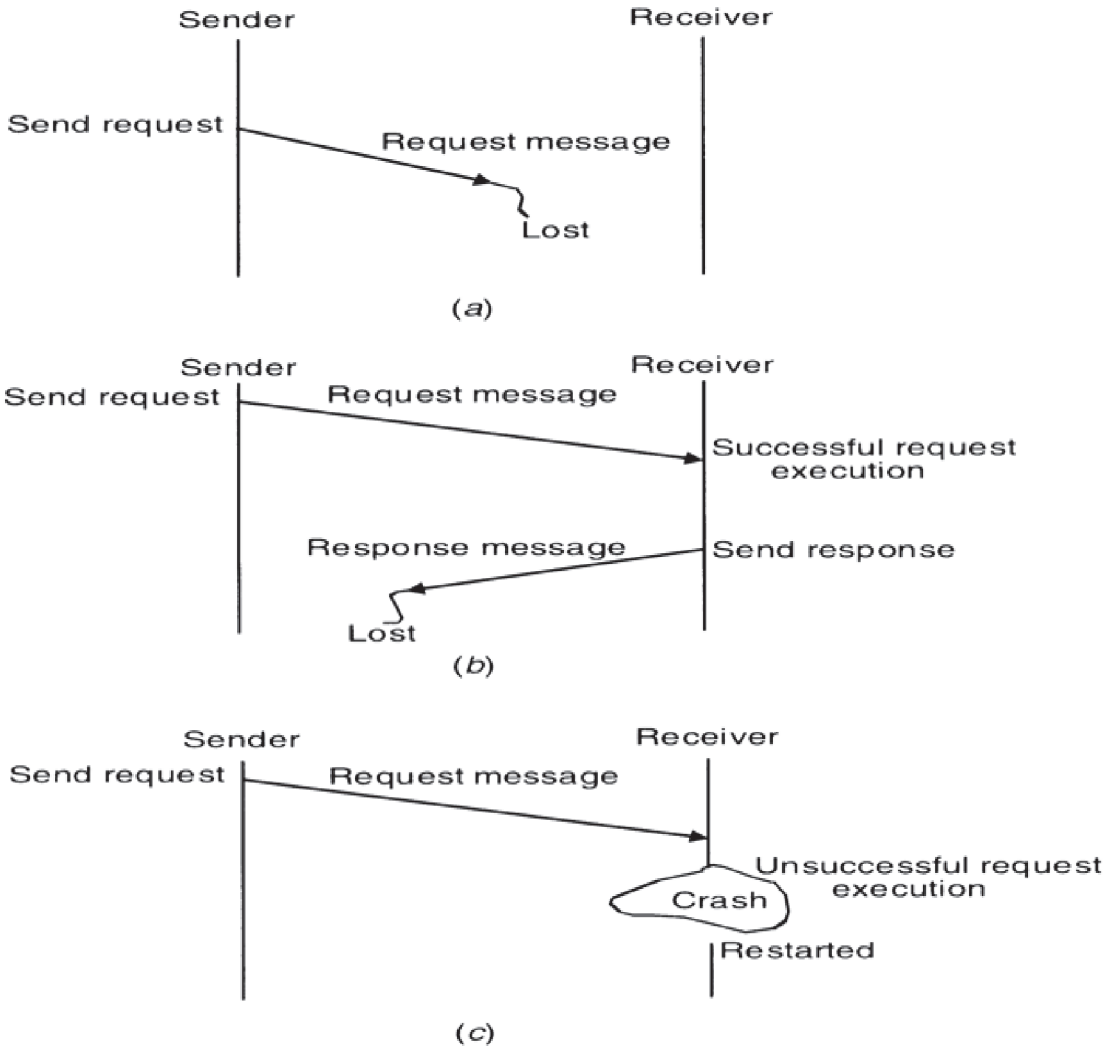


Figure 11.8 Process Addressing

11.8 Failure Handling

During interprocess communication partial failures such as a node crash or communication link failure may lead to the following problems:

- **Loss of Request Message:** This may happen either due to the failure of communication link between the sender and receiver or because the receiver's node is down at the time the request message reaches there.
- **Loss of Response Message:** This may happen either due to the failure of communication link between the sender and receiver or because the sender's node is down at the time the response message reaches there.
- **Unsuccessful Execution of the Request**: This may happen due to the receiver's node crashing while the request is being processed.

11.8.1 Possible Problems in IPC due to Different Types of System Failures

Four-message reliable IPC protocol for client-server communication between two processes works as follows:

- The client sends a request message to the server.
- When the request message is received at the server's machine, the kernel of that machine returns an acknowledgment message to the kernel of the client machine. If the acknowledgment is not received within the timeout period, the kernel of the client machine retransmits the request message.
- When the server finishes processing the client's request it returns a reply message (containing the result of processing) to the client.
- When the reply is received at client machine, the kernel of that machine returns an acknowledgment message to the kernel of the server machine. If the acknowledgment message is not received within the timeout period, the kernel of the server machine retransmits the reply message.

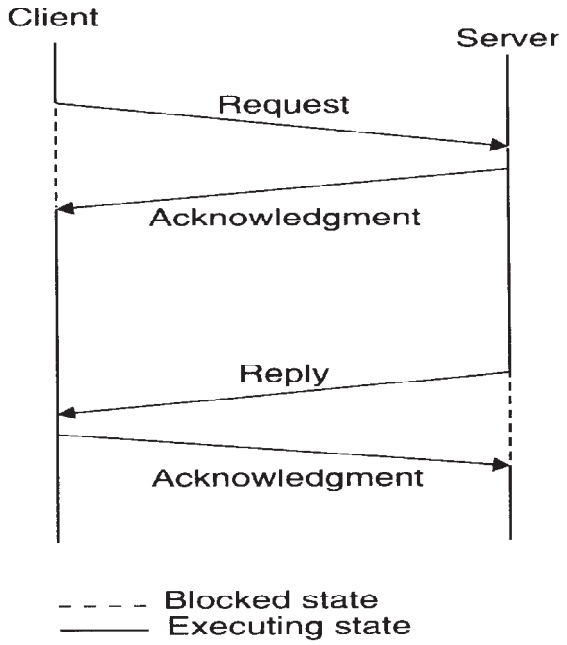


Figure 11.9(a): Failure Handling (The four message reliable IPC)

In client-server communication, the result of the processed request is sufficient acknowledgment that the request message was received by the server. Based on this idea, a **three-message reliable** IPC protocol for client-server communication between two processes works as follows:

- The client sends a request message to the server.
- When the server finishes processing the client's request, it returns a reply message (containing the result of processing) to the client. The client remains blocked until the reply is received. If the reply is not received within the timeout period, the kernel of the client machine retransmits the request message.
- When the reply message is received at the client's machine, the kernel of that machine returns an acknowledgment message to the kernel of the sever machine. If the acknowledgment message is not received within the timeout period, the kernel of the server machine retransmits the reply message.

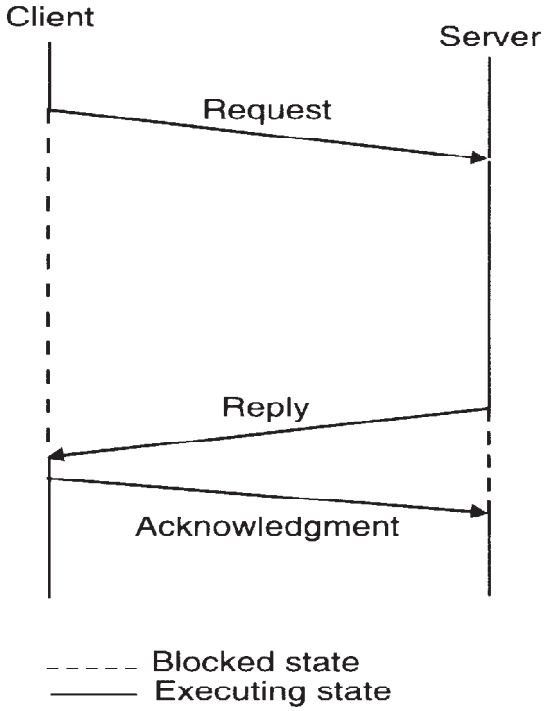


Figure 11.9(b): Failure Handling (The three message reliable IPC)

A problem occurs if a request processing takes a long time. If the request message is lost, it will be retransmitted only after the timeout period, which has been set to a large value to avoid unnecessary retransmission of the request message. On the other hand, if the timeout value is not set properly taking into consideration the long time needed for request processing, unnecessary retransmissions of the request message will take place.

The following protocol may be used to handle this problem:

- The client sends a request message to the server.
- When the request message is received at the server's machine, the kernel of that machine starts a timer. If the server finishes processing the client's requests and returns the reply message to the client before the timer expires, the reply serves as the acknowledgment of the request message. Otherwise, a separate acknowledgment is sent by the kernel of the server machine to acknowledge the request message. If an acknowledgement is not received within the timeout period, the kernel of the client machine retransmits the request message.

- When the reply message is received, at the client's machine, the kernel of that machine returns an acknowledgment message to the kernel of the server machine. If the acknowledgment message is not received within the timeout period, the kernel of the server retransmits the reply message.

A message-passing system may be designed to use the following two-message IPC **protocol** for client-server communication between two processes:

- The client sends a request message to the server and remains blocked until a reply is received from the server.
- When the server finishes processing the client's request, it returns a reply message (containing the result of processing) to the client. If the reply is not received within the timeout period, the kernel of the client machine retransmits the request message.

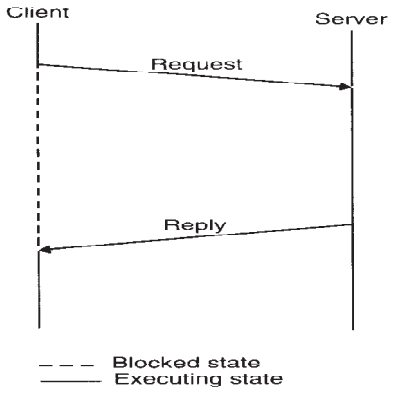


Figure 11.9(c): Failure Handling

Three message IPC protocol used in many systems for client-server communication

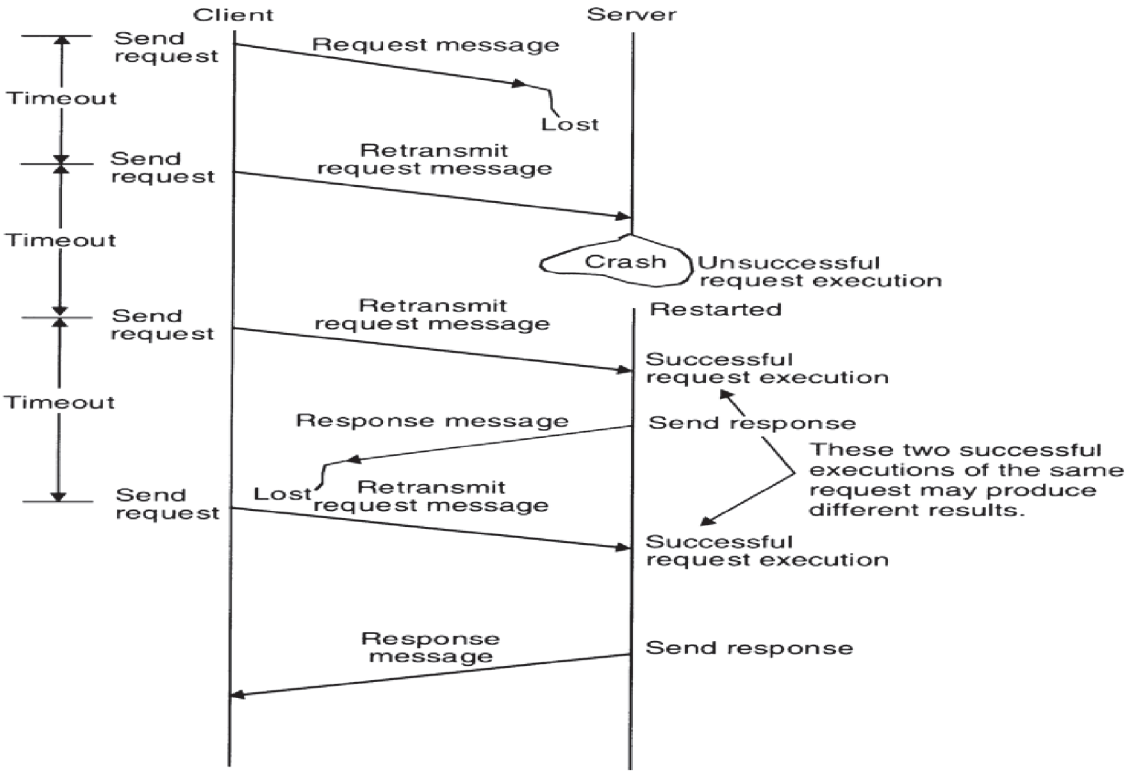


Figure 11.9(d): Failure Handling

An example of fault tolerant communication between a client and a server.

The most elementary form of message-based interaction is one-to-one communication (also known as point-to-point, or unicast communication) in which a single-sender process sends a message to a single-receiver process. For performance and ease of programming, several highly parallel distributed applications require that a message-passing system should also provide group communication facility. Depending on single or multiple senders and receivers, the following three types of group communication are possible:

- **One to many** (single sender and multiple receivers).
- **Many to one** (multiple senders and single receivers).
- **Many to many** (multiple senders and multiple receivers).

One-to-Many Communication

In this scheme, there are multiple receivers for a message sent by a single sender. One-to-many scheme is also known as multicast communication. A special case of multicast communication is broadcast communication, in which the message is sent to all processors connected to a network.

One-to-many may refer to:

- Multivalued function, a one-to-many function in mathematics
- Fat link, a one-to-many link in hypertext
- Point-to-multipoint communication, communication which has a one-to-many relation

11.9 Summary

In this unit we have discussed about the message passing. A message is a block of information formatted by a sending process in such a manner that it is meaningful to the receiving process. Message passing is a method by which an object sends data to another object or requests other object to invoke method. This is also known as interfacing. Interprocess communication (IPC) basically requires information sharing among two or more processes. Two basic methods for information sharing are as follows: Original sharing, or shared-data approach; and Copy sharing, or message-passing approach. Simplicity, uniform Semantics, efficiency and correctness are the features of good message passing. At last we discussed the buffering and its strategies and the multi diagram message.

11.10 Self - Assessment Exercise

1. Explain the good features of Message Passing system.
2. What are the desirable issues in IPC by Message Passing & Synchronization?
3. What is the role of Buffering?
4. Explain Multi Datagram Messages.
5. What are the Methods to Identify a Process (naming) Process Addressing
6. Write down the possible problems in IPC due to different types of system failures.

11.11 References

- Distributed Operating Systems by Andrew S. Tenenbaumb.
- Distributed File Systems: Concepts and example by ELIEZER LEVY and ABRAHAM SILBERSCHATZ

Unit – 12 : Remote Procedure Calls

Structure of Unit

12.0 Objective

After reading this material you will know-

- Understand the RPC and its goal.
- How RPC Mechanism is implemented and how stub is generated.
- Understand different issues which is consider in RPC
- Understand the Secure RPC and Lightweight RPC

12.1 Introduction

RPC is a powerful technique for constructing distributed, client-server based applications. It is based on extending the notion of conventional or local procedure calling, so that the called procedure need not exist in the same address space as the calling procedure. The two processes may be on the same system, or they may be on different systems with a network connecting them. By using RPC, programmers generally avoid the details of the interface of distributed applications or network. The transport independence of RPC isolates the application from the physical and logical elements of the data communications mechanism and allows the application to use a variety of transports.

RPC builds the client/server model and making more powerful and easier program. Basically, Remote Procedure Call (RPC) is a protocol in which one program can use to request a service from a program located in another computer in a network without having to understand network details. A procedure call is also sometimes known as a function call or a subroutine call. The requesting program is a client and the service-providing program is the server. Like a regular or local procedure call, an RPC is a synchronous operation requiring the requesting program to be suspended until the results of the

remote procedure are returned. However, the use of lightweight processes or threads that share the same address space allows multiple RPCs to be performed concurrently.

Generally, the program statement that use RPC is compiled into an executable program, a stub is included in the compiled code that acts as the representative of the remote procedure code. When the program is run and the procedure call is issued, the stub receives the request and forwards it to a client runtime program in the local computer. The client runtime program has the knowledge of how to address the remote computer and server application and sends the message across the network that requests the remote procedure. Similarly, the server includes a runtime program and stub that interface with the remote procedure itself. Results are returned the same way.

12.1.1 Goals

Remote Procedure Call (RPC) provides a different prototype/example for accessing network services. Instead of accessing remote services by sending and receiving messages, a client invokes services by making a local procedure call. The local procedure hides the details of the network communication.

When making a remote procedure call:

1. The calling environment is suspended, procedure parameters are transferred across the network to the environment where the procedure is to execute, and the procedure is executed there.
2. When the procedure finishes and produces its results, its results are transferred back to the calling environment, where execution resumes as if returning from a regular procedure call.

The main goal of RPC is to hide the existence of the network from a program. As a result, RPC doesn't quite fit into the OSI model:

1. The message-passing nature of network communication is hidden from the user. The user doesn't first open a connection, read and write data, and then close the connection. Indeed, a client often does not even know they are using the network!
2. RPC often omits many of the protocol layers to improve performance. Even a small performance improvement is important because a program may invoke RPCs often. For example, on (diskless) Sun workstations, every file access is made via an RPC.

RPC is especially well suited for client-server (e.g., query-response) interaction in which the flow of control alternates between the caller and callee. Conceptually, the client and server do not both execute at the same time. Instead, the thread of execution jumps from the caller to the callee and then back again.

12.2 RPC Model

The RPC model describes how cooperating processes on different network nodes can communicate and coordinate activities. The paradigm of RPC is based on the concept of a procedure call in a programming language. The semantics of RPC are almost identical to the semantics of the traditional procedure call. The major difference is that while a normal procedure call takes place between procedures of a single process in the same memory space on a single system, RPC takes place between a client process on one system and a server process on another system where both the client system and the server system are connected to a network.

There are several RPC models and implementations. A popular model and implementation is the Open Software Foundation's Distributed Computing Environment (DCE). The Institute of Electrical

and Electronics Engineers defines RPC in its ISO Remote Procedure Call Specification, ISO/IEC CD 11578 N6561, ISO/IEC, November 1991.

RPC spans the Transport layer and the Application layer in the Open Systems Interconnection (OSI) model of network communication. This makes it easier to develop an application that includes multiple programs distributed in a network.

Alternative methods for client/server communication include message queuing and IBM's Advanced Program-to-Program Communication (APPC).

The basic operation of RPC is illustrates in Figure 12.2. A normal client application issues a normal procedure call to a client stub. The client stub receives arguments from the calling procedure and returns arguments to the calling procedure. An argument may instantiate

-an input parameter,

-an output parameter, or

-an input/output parameter

Here, the term input argument/parameter refers to a parameter which may be either an input parameter or an input/output parameter, and the term output argument refers to either an output parameter or an input/output parameter.

The client stub converts the input arguments from the local data representation to a common data representation, creates a message containing the input arguments in their common data representation, and calls the client runtime, usually an object library of routines that supports the functioning of the client stub. The client runtime transmits the message with the input arguments to the server runtime which is usually an object library that supports the functioning of the server stub. The server runtime issues a call to the server stub which takes the input arguments from the message, converts them from the common data representation to the local data representation of the server, and calls the server application which does the processing.

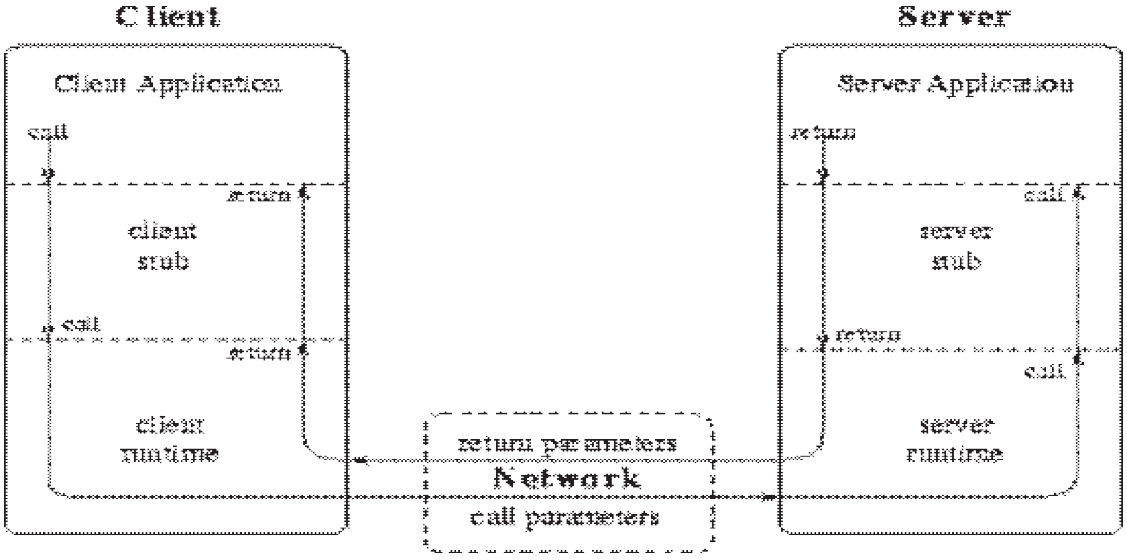


Figure 12.2: RPC Model

When the server application has completed, it returns to the server stub the results of the processing in the output arguments. The server stub converts the output arguments from the data representation of the server to the common data representation for transmission on the network and encapsulates the output arguments into a message which is passed to the server runtime. The server runtime transmits the

message to the client runtime which passes the message to the client stub. Finally, the client stub extracts the arguments from the message and returns them to the calling procedure in the required local data representation.

In addition to normal procedure call, RPC is a synchronous operation, i.e., the client process is blocked until processing by the server is complete. This is not acceptable for many applications. As a consequence, the RPC model is enhanced to include the concept of a lightweight process. A lightweight process (also known as a thread) is an independent execution path within a normal process. A normal process can consist of several lightweight processes, each behaving like a normal process from the point of view of CPU use. However, all lightweight processes of the same process share the same address space. Thus, context switches between lightweight processes may be done more economically than context switches between normal processes.

In order to achieve asynchronous operation, a client application initiates an RPC call in a lightweight process and then proceeds with other processing. The application can recognize the completion of the RPC by some technique such as a status check or a software interrupt.

12.3 Implementing RPC Mechanism

Basically Remote Procedure is different from the local procedure which refers to the transparency. The implementation of an RPC mechanism is to achieve the goal of semantic transparency. It is based on the concept of stubs which provide a perfectly normal procedure call abstraction by concealing from programs the interface to the underlaw RPC system. Generally, an RPC involves a client process and a server process. Therefore we conceal the interface of the underlying RPC system from both the client and server processes, a separate stub procedure is associated with each of the two processes. To hide the existence and functional details of the underlying network, an RPC communication package is used on both the client and server sides. The implementation of an RPC mechanism usually involves the following five elements of program:

1. The client

2. The client stub

3. The RPC Runtime

4. The server stub

5. The server

The client, the client stub and one instance of RPC Runtime execute on the client machine, while the server, the server stub and another instance of RPC Runtime execute on the server machine. The job of each of these elements is described below.

Client

The client is a user process that initiates a remote procedure call. To make a remote procedure call, the client makes a perfectly normal local call that invokes a corresponding procedure in the client stub.

Client Stub

The client stub is responsible for carrying out the following two tasks:

1. On receipt of a call request from the client, it packs a specification of the target procedure and the arguments into a message and then asks the local RPC Runtime to send it to the server stub.

2. On receipt of the result of procedure execution, it unpacks the result and passes it to the client.

RPC Runtime

The RPC Runtime handles transmission of messages across the network between client and server machines. It is responsible for retransmissions, acknowledgments, packet routing and encryption. The RPC Runtime on the client machine receives the call request message from the client stub and sends it to the server machine. It also receives the message containing the result of procedure execution from the server machine and passes it to the client stub.

On the other hand, the RPC Runtime on the server machine receives the message containing the result of procedure execution from the server stub and sends it to the client machine. It also receives the call request message from the client machine and passes it to the server stub.

Server Stub

The job of the server stub is very similar to that of the client stub. It performs the following two tasks:

1. On receipt of the call request message from the local RPC Runtime, the server stub unpacks it and makes a perfectly normal call to invoke the appropriate procedure in the server.

2. On receipt of the result of procedure execution from the server, the server stub packs the result into a message and then asks the local RPC Runtime to send it to the client stub.

Server

On receiving a call request from the server stub, the server executes the appropriate procedure and results the result of procedure execution to the server stub.

12.4 Stub Generation

The idea behind RPC is to make a remote procedure call which looks as much as possible like a local one. In other words, we can say that RPC should be transparent—the calling procedure should not be aware that the called procedure is executing on a different machine or vice versa. Suppose that a program needs to read some data from a file. The programmer puts a call to read in the code to get the data.

In a traditional (single-processor) system, the read routine is extracted from the library by the linker and inserted into the object program. It is a short procedure, which is generally implemented by calling an equivalent read system call. In other words, the read procedure is a kind of interface between the user code and the local operating system.

Even though read does a system call, it is called in the usual way, by pushing the parameters onto the stack, Thus the programmer does not know that read is actually doing something fishy.

RPC achieves its transparency in an analogous way. When read is actually a remote procedure (e.g., one that will run on the file server's machine), a different version of read, called a client stub, is put into the library. Also like the original one, it too, does a call to the local operating system. Only unlike the original one, it does not ask the operating system to give it data. Instead, it packs the parameters into a message and requests that message to be sent to the server. Following the call to send, the client stub calls receive, blocking it until the reply comes back.

When the message arrives at the server, the server's operating system passes it up to a server stub. A server stub is the server-side equivalent of a client stub:

It is a piece of code that transforms requests coming in over the network into local procedure calls. Typically the server stub will have called receive and be blocked waiting for incoming messages. The server stub unpacks the parameters from the Stubs can be generated in one of the following two ways:

Manually

In this method, the RPC implementer provides a set of translation functions from which a user can construct his or her own stubs. This method is simple to implement and can handle very complex parameter types.

Automatically

This is the more commonly used method for stub generation. Its user Interface Definition Language (IDL) is used to define the interface between a client and a server. An interface definition is mainly a list of procedure names that is supported by the interface, together with the types of their arguments and results. This is sufficient information for the client and server to independently perform compile-time type checking and to generate appropriate calling sequences.

An interface definition also contains other information that helps RPC reduce data storage and the amount of data transferred over the network. An interface definition has information to indicate whether each argument is input, output or both- only input arguments need be copied from client to server and only output arguments need be copied from server to client. Similarly an interface definition also has information about type definition, enumerated types and defined constants that side uses to manipulate data from RPC calls making it unnecessary for both the client and the server to store this information separately.

A server program that implements procedure in an interface is said to export the interface and a client program that calls procedures from an interface is said to import the interface. When writing a distributed application, a programmes first writer an interface definition using the IDL. We can then write the client program that imports the interface and the server program that exports the interface.

The interface definition is processed using an IDL compiler to generate components that can be combined with client and server programs, without making any changes to the existing compilers. In particular, from an interface for each procedure in the interface, the appropriate marshaling and unmarshaling operations in each stub procedure and a header file that supports the data types in the interface definition. The header file is included in the source files of the client and server programs, the client stub procedures are compiled and linked with the client program and the server stub procedures are compiled and linked with the server program. An IDL compiler can be designed to process interface definitions for use with different languages, enabling clients and servers written in different languages, to communicate by using remote procedure calls.

To summarize, a remote procedure call occurs in the following steps:

1. The client procedure calls the client stub in the normal way.

2. The client stub builds a message and calls the local operating system.

3. The client's OS sends the message to the remote OS.

4. The remote OS gives the message to the server stub.

5. The server stub unpacks the parameters and calls the server.

6. The server does the work and returns the result to the stub.

7. The server stub packs it in a message and calls its local OS.

8. The server's OS sends the message to the client's OS.

9. The client's OS gives the message to the client stub.

10. The stub unpacks the result and returns to the client.

12.5 RPC Messages

An RPC is initiated by the client, which sends a request message to a known remote server to execute a specified procedure with supplied parameters. The remote server sends a response to the client, and the application continues its process. There are many variations and subtleties in various implementations, resulting in a variety of different (incompatible) RPC protocols. While the server is processing the call, the client is blocked (it waits until the server has finished processing before resuming execution), unless the client sends an asynchronous request to the server, such as an XHTTP call.

An important difference between remote procedure calls and local calls is that remote calls can fail because of unpredictable network problems. Also, callers generally must deal with such failures without knowing whether the remote procedure was actually invoked. Idempotent procedures (those that have no additional effects if called more than once) are easily handled, but enough difficulties remain that code to call remote procedures is often confined to carefully written low-level subsystems.

12.6 RPC Issues

There are various issues that must be addressed:

12.6.1 Calling Semantics and Naming

An important issue in distributed systems is the parameter-passing semantics of remote procedure calls or remote invocations i.e. how should parameters be passed across machine boundaries? In non-distributed systems, the programmer typically has a choice of call-by-reference or call-by-value. However, in most distributed systems, call-by-reference is not possible because addresses cannot be passed across the network; therefore, call-by-value (or call-by-value-result) must be used.

Still, with call-by-value, the system must be able to transmit the addressed entity, which may be a data type instance, across the network. The process of placing parameters into a message packet, and later taking them off and reconstructing the data, is called marshalling in an RPC system. Marshalling can be done in several ways, depending on the support provided to the programmer by the RPC system.

In some RPC systems, a programmer must explicitly code calls to marshalling procedures, a tedious and error-prone process. In most systems a stub compiler program reads a high-level specification of a server's interface and automatically produces calls to marshalling procedures; these calls are part of the stub called locally when an RPC occurs. In the Argus system, each abstract data type can be asked to encode itself into a form suitable for transmission to other systems.

This gives the programmer the flexibility to define a standardized on-the-wire format for a data type. The major problem is not that call-by-value and call-by-reference are different, but that call-by-value replicates a data type instance during the duration of the RPC call. Assuming that concurrency exists in a distributed system, each instance represents a state that is potentially shared among several processes. Concurrent access to that state must be controlled through a synchronization mechanism, such as monitors. In fact, a principal objective of abstract data types is the synchronization of shared access, as well as implementation independence.

In a call-by-value RPC, however, the server is not accessing shared state, but is accessing a copy of the state at the time the argument is accessed; it may be out of date. Similarly, with call-by value- result, the effect will be the same as call-by-reference only if no other accesses

to that state occurred on the source node in the duration of the call. The result is that all shared instances must be controlled by servers, since only the server can guarantee sequential access if needed. Having a data type with concurrency control is not sufficient if an instance of that data type can be passed as a call-by-value result parameter. This is an issue that exists in centralized systems independent of RPC; however, the problem is greatly magnified by the existence of true concurrency in distributed systems.

An alternative approach to the problem of calling semantics and naming is the distributed virtual memory implemented in the Ivy system. In Ivy, a single address space exists across nodes; the memory manager traps program accesses to shared store and exchanges coherency messages with memory managers on other nodes. From the programmer's point of view, distributed processes communicate through shared memory as they would on a centralized system. Distributed processes can therefore exchange and share addresses. If one built an RPC system on top of Ivy, call-by-reference could be trivially used for parameter passing.

In distributed object-oriented systems, it is also possible to have a single object name space that spans machine boundaries. Each object has a network-wide unique identifier (its ID or address) that can be used to reference the object in a location independent manner. This obviously implies a level of indirection; all object references must occur through a descriptor mechanism so that remote references can be detected and trapped. Given such a mechanism, call-by-reference can be provided easily. In fact, conceptually the only parameter-passing mechanism in an object oriented system is call-by-object-reference. Thus, to give someone access to an object, all one does is to pass its address, independent of the location of the object or the receiver of its address.

The problem with call-by-reference in a distributed object-oriented system is performance. When one object performs a remote invocation of another, passing it references to parameter objects, the receiver will probably cause remote invocations when it attempts to invoke those parameters. In many cases, these remote invocations are unavoidable. However, there are cases in which a call-by-value mechanism would make more sense with respect to performance. For example, if the parameter object is immutable, then sending a copy of it would be semantically equivalent to call-by-reference Obviously this can be done for immutable objects such as integers, as well as for structures of immutable.

It may also be possible to decrease the cost of a remote call-by-reference parameter by moving the parameter object to the calling site for the duration of the call. The Emerald system has this capability, and moving objects has been shown to have performance improvement potential, depending on the nature of the parameter object being moved (e.g. its size and the number of active invocations). Once again, moving an object is different than call-by-value because the object represents shared state.

Summary of these are given below:

Marshalling:

- Parameters must be marshalled into a standard representation.

- Parameters consist of simple types (e.g., integers) and compound types (e.g., C structures or Pascal records). Moreover, because each type has its own representation, the types of the various parameters must be known to the modules that actually do the conversion. For example, 4 bytes of characters would be uninterpreted, while a 4-byte integer may need to the order of its bytes reversed.

Semantics:

- Call-by-reference is not possible that means the client and server don't share an address space. That is, addresses referenced by the server correspond to data residing in the client's address space.
- One approach is to simulate call-by-reference using copy-restore. In copy-restore, call-by-reference parameters are handled by sending a copy of the referenced data structure to the server, and on return replacing the client's copy with that modified by the server.
- However, copy-restore doesn't work in all cases. For instance, if the same argument is passed twice, two copies will be made, and references through one parameter only changes one of the copies.

12.6.2 Exception Handling

It refers to -how are errors handled? A major difference between centralized and distributed systems is in their failure modes. Failures are a difficult matter in distributed systems, and particularly in systems such as Emerald in which distribution is transparent. In both RPC and distributed object systems, much effort goes into making remote and local calls identical to the extent possible in each particular system. In general, hiding the details of distribution from the programmer simplifies the programming of distributed applications. Failures constitute a notable exception to this rule.

Although it is possible to make local and remote invocations largely identical, it is not possible to make remote invocations obey all of the properties of local invocations. For example, whereas local procedures on a single node can raise error conditions, they do not time-out, they do not become temporarily unreachable and they do not crash independently of the calling procedure. Thus, a local procedure call is typically not equipped to handle such failures.

12.6.3 Transport/Communication Protocol

It refers to -what transport protocol should be used? Generally, reliable stream protocols are designed for a different purpose: high throughput. The cost of setting up and terminating a connection is insignificant in comparison to the amount of data exchanged. Most of the elapsed time is spent sending data.

With RPC, low latency is more important than high throughput. If applications are going to use RPC much like they use regular procedures (e.g., over and over again), performance is crucial.

RPC can be characterized as a specific instance of transaction-oriented communication, where:

- A transaction consists of a single request and a single response.
- A transaction is initiated when a client sends a request and terminated by the server's response.

A transaction-oriented transport protocol should efficiently handle the following cases:

1. Transactions in which both the request and response messages fit in a single packet. The response can serve as an acknowledgment, and the client handles the case of lost packets by retransmitting the original request.
2. Large multi-packet request and response messages, where the data does not necessarily fit in a single packet. For instance, some systems use RPC to fetch pages of a file from a file server. A single-packet request would specify the file name, the starting position of the data desired, and the number of bytes to be read. The response may consist of several pages (e.g. 8K bytes) of data.

A communications link depends on a set of communications protocols. A communications protocol is a clearly defined set of operational rules and procedures for communications.

Communications protocols include a transport protocol (from the Transport Layer of the OSI network architecture) such as the Transmission Control Protocol (TCP) or the User Datagram Protocol(UDP); and the corresponding network protocol (from the OSI Network Layer), such as the Internet Protocol (IP).

For an RPC client and server to communicate, their RPC runtimes must use at least one identical communications protocol, including a common RPC protocol, transport protocol, and network protocol. An RPC protocol is a communications protocol that supports the semantics of the DCE RPC API and runs over specific combinations of transport and network protocols. DCE RPC provides two RPC protocols: the connectionless RPC protocol and the connection-oriented RPC protocol.

Connectionless (Datagram) RPC Protocol

This protocol runs over a connectionless transport protocol such as UDP. The connectionless protocol supports broadcast calls.

Connection-oriented RPC Protocol

This protocol runs over a connection-oriented transport protocol such as TCP.

Each binding uses a single RPC protocol and a single pair of transport and network protocols. Only certain combinations of communications protocols are functionally valid (are actually useful for interoperation); for instance, the RPC connectionless protocol cannot run over connection-oriented transport protocols such as TCP. DCE RPC supports the following combinations of communications protocols:

- RPC connection-oriented protocol over the Internet Protocol Suite, Transmission Control Protocol (TCP/IP)
- RPC connectionless over the Internet Protocol Suite, User Datagram Protocol (UDP/IP)

The following table lists the network protocols supported by RPC and the type of RPC connection for which the protocol is used.

RPC-Supported Network Protocols

Protocol	RPC Type
Transmission Control Protocol (TCP)	Connection–oriented
Sequenced Packet Exchange (SPX)	Connection–oriented
Named Pipe	Connection–oriented
HTTP	Connection–oriented
User Datagram Protocol (UDP)	Connectionless
Cluster Datagram Protocol (CDP)	Connectionless

Table12.6.3: RPC-Supported Network Protocols

12.6.4 Binding

It refers to -how does the client know who to call, and where the service resides?

The most flexible solution is to use dynamic binding and find the server at run time when the RPC is first made. The first time the client stub is invoked, it contacts a name server to determine the transport address at which the server resides.

Binding is the process of establishing communication between two parties. Through binding, the parties determine the addresses of their partners and the protocols to be used for the communication. On a single node, modules communicate using procedure call and return, and address each other using virtual addresses; the binding is performed by the compiler or linker. In a network, there are various choices for how and when binding is made, and these choices affect both the performance and flexibility of the application. In the most static case, binding is performed at compile time. That is, the two communicating programs know the explicit network locations of their partners. In the most general case, binding can be performed at every interaction. Options exist in the middle that permit trade-offs between flexibility and performance; RPC systems have typically chosen early binding, which permits call time performance optimizations, whereas invocation-oriented systems have chosen late binding, which permits flexibility.

12.7 Security

The Lightweight Remote Procedure Call mechanism tries to address the inefficiencies in normal RPC usage in the common case where small footprint calls are made on the same machine. This is done by avoiding overhead associated to the execution path required for the complex but infrequent cases.

Small kernel OS aims at placing components in separate domains. An RPC like mechanism is used for communication. But this 'one-size fits all' approach doesn't account for the real usage pattern: although the bulk of communication is between domains on the same machine, and small-simple parameters are involved, the overhead for the heavyweight machinery (marshalling, access validation, scheduling, dispatch, etc) is still paid.

They provide a communication mechanism designed for concurrency that maintains domain protection at a lower cost by dispatching a shared argument stack directly to the server domain. Linkage records are used to facilitate the transfer of control. Security is enforced by Binding Objects, which are capabilities obtained at bind-time.

The main technique is to optimize the common execution path. The Lightweight RPC caller still calls into a stub, but instead of a complicated dispatch mechanism, the stub simply uses the A-stack (shared with the callee) to pass arguments, no unnecessary data copying is done. A-stacks are pre allocated. Execution-

stacks on the server are not associated with A-stacks at bind time, rather dynamically, so that large E-stack space explosion is avoided. E-stacks are 'activated' directly by the kernel, no dispatching overhead. A-stacks can be shared between procedures in the same interface with similar size requirements. Stubs are generated in assembly.

There is an upfront cost that is paid at bind time for setting up the A-stacks. Multiple A-stacks allow for concurrency, but the concurrency limit is bound at bind time. In the uncommon case, RPC like overhead (marshalling, Modula-stubs, etc) is still involved, so LRPC maintains two coexisting mechanisms.

The shared stack optimization only holds for their language environment, it would not work for the more common case of C-like languages.

The mechanism fits well into a system that already uses RPC, where communication is done only through arguments/return values. It is unlikely that one could take it a step further, where local LRPC caller/callee would also share global data. In a way, they are not comparing apples to apples: the assembly generated stubs for LRPC increase performance of LRPC, while Modula stubs for RPC decrease the RPC performance; this blurs comparison of the rest of the mechanisms (which would be more accurate if assembly-generated stubs were also used for old RPC).

12.8 Lightweight RPC

Lightweight Remote Procedure Call (LRPC) is a communication facility designed and optimized for communication between protection domains on the same machine. In contemporary small-kernel operating systems, existing RPC systems incur an unnecessarily high cost when used for the type of communication that predominates between protection domains on the same machine. This cost leads system designers to coalesce weakly related subsystems into the same protection domain, trading safety for performance.

LRPC achieves a level of performance for cross-domain communication that is significantly better than conventional RPC systems, while still retaining their qualities of safety and transparency. Four techniques contribute to the performance of LRPC:

- Simple control transfer. The client's thread executes the requested procedure in the server's domain.
- Simple data transfer. The parameter-passing mechanism is similar to that used by procedure call. A shared argument stack, accessible to both client and server, can often eliminate redundant data copying.
- Simple stubs. LRPC uses a simple model of control and data transfer, facilitating the generation of highly optimized stubs.
- Design for concurrency. LRPC avoids shared data structure bottlenecks and benefits from the speedup potential of a multiprocessor.

12.9 Summary

Remote Procedure Call (RPC) is a powerful technique for constructing distributed, client/server based applications. Client/server is a computational architecture that involves client processes requesting service from server processes. The client/server software architecture is a versatile, message-based and modular infrastructure that is intended to improve, usability, flexibility, interoperability, and salability as compared to centralized, mainframe, time sharing computing.

RPC is based on extending the notion of conventional or local procedure calling, so that the called procedure need not exist in the same address space as the calling procedure. The two processes may be on the same system, or they may be on different systems with a network connecting them.

By using RPC, programs on networked platforms can communicate with remote (and local) resources. RPC allows network applications to use specialized kinds of procedure calls designed to hide the details of underlying networking mechanisms. The complexity involved in the development of distributed processing is reduced by keeping the semantics of a remote call the same whether or not the client and server are collocated on the same system.

12.10 Self - Assessment Exercise

1. What are the differences between a local call and a remote call?
2. What are the goals of RPC?
3. Explain the RPC Model with a suitable diagram.
4. Explain different elements used in RPC Process.
5. Write down the steps for generating the RPC.
6. Explain "call by reference" vs. "call by value".
7. Explain Naming.
8. What are stub and skeleton and why are they needed in remote procedure calls?
9. How does so called marshalling solve the problem of different byte ordering of sender and receiver?
10. Differentiate the Connectionless and Connection-Oriented RPC protocol.
11. Explain the Lightweight RPC.

12.11 References

- Operating System Concepts by Silberschatz Galvin Gagne
- System programming and operating system by D. M. Dhamdhere
- Operating Systems, Madnick, Madnick by Madnick Se
- Operating Systems by I.A. Dhotre

Unit - 13 : Real Time System

Structure of Unit

13.0 Objective

The main objective of this chapter is to know about the real time system. To know the aim and major classification of it's along with its examples.

This unit also includes the desirable characteristics of it and features of real-time kernel. Also we will know the basic architecture of it for implementing the RTOS.

13.1 Introduction

Real-time systems play a considerable role in our society, and they cover a spectrum from the very simple to the very complex. Examples of current real-time systems include the control of domestic appliances like washing machines and televisions, the control of automobile engines, telecommunication switching systems, military command and control systems, industrial process control, flight control systems, and space shuttle and aircraft avionics.

All of these involve gathering data from the environment, processing of gathered data, and providing timely response. A concept of 'time' is the distinguishing issue between real-time and non-real-time systems. When a usual design goal for non-real-time systems is to maximize system's throughput, the goal for real-time system design is to guarantee, that all tasks are processed within a given time. The taxonomy of time introduces special aspects for real-time system research.

In other words, we can say that word 'time' means that the correctness of the system not only depend on the logical result of the computation but also on the time at which the results are produced. The word 'real' indicates that the reaction of the system to external events must occur during their evolution. As a consequence, the system time must be measured using the same time scale used for measuring the time in the controlled environment

Timeliness is the single most important aspect of a real -time system. These systems respond to a series of external inputs, which arrive in an unpredictable fashion. The real-time systems process these inputs, take appropriate decisions and also generate output necessary to control the peripherals connected to them. As defined by Donald Gillies "A real-time system is one in which the correctness of the computations not only depends upon the logical correctness of the computation but also upon the time in which the result is produced. If the timing constraints are not met, system failure is said to have occurred."

It is essential that the timing constraints of the system are guaranteed to be met. Guaranteeing timing behavior requires that the system be predictable.

Any system in which the time at which output is produced is significant. This is usually because the input corresponds to some movement in the physical world, and the output has to relate to that same movement. The lag from input time to output time must be sufficiently small for acceptable timeliness. Timeliness is a function of the total system: for example- missile guidance requires output within a few milliseconds of input whereas scheduling of steamships requires responses measured in days. Real-time systems are usually considered to be those in which the response time is of order milliseconds. In this context, real time system provides an interactive system where interactive systems are those with response times of order seconds.

13.1.1 Definitions

Failure is inability to perform according to specification. In the case of real-time systems the 'failed' specification may be lack of correctness or the failure to produce the response by the required time.

A real-time system is one whose correctness is based on both the correctness of the outputs and their timeliness. The 'novelty' here is that the system is time critical.

Definition 1:

A real-time system is one that must process information and produce a response within a specified time, else risk severe consequences, including failure. That is, in a system with a real-time constraint it is no good to have the correct action or the correct answer after a certain deadline: it is either by the deadline or it is useless!

Definition 2 (modified from The Oxford Dictionary of Computing):

Any system in which the time at which output is produced is significant. This is usually because the input corresponds to some event in the physical world, and the output has to relate to that same event. The lag from input time to output time must be sufficiently small for acceptable timeliness

Definition 3:

Any information processing activity or system which responds to externally generated input stimuli within a finite and specified period.

Examples:

Aircraft control, ticket reservation system at airport, over-temperature monitor in nuclear power station, mobile phone, oven temperature controller, Doppler blood-flow monitor, ECG/arrhythmia monitor.

13.1.2 Aim/Objectives

The objective of real time systems is not to minimize the average response time of a given set of tasks but it is to meet the individual timing requirements of each task. However short the average response time is,

without a scientific methodology, we would never be able to guarantee the individual timing requirements of each task in all possible circumstances. When several computation activities have different constraints, average performance has little significance for the correct behaviour of the system

13.1.3 Major Classification of Real Time System

The design of a real -time system must specify the timing requirements of the system and ensure that the system performance is both correct and timely. The major classification of Real-time systems may be subdivided into *hard* and *soft*, depending on the severity of failure to meet a deadline for output.

The three types of time constraints:

- **Hard:** A late response is incorrect and implies a system failure. An example of such a system is of medical equipment monitoring vital functions of a human body where a late response would be considered as a failure.

  Basically, it is absolutely imperative that operations are completed within their deadline in this case if an operation is completed after the time in which it is supposed to then it is considered useless e.g. systems in space shuttle. In which case if the system did not react within the strict deadline it could result in serious issues such as the loss of life or damage to the equipment such as the shuttle crashing. As can be seen in this situation a late response would be pointless.

More specially, we can say that-

- Must meet its deadline.
- Conflicts with time-sharing systems, not supported by general-purpose operating systems.
- Often used as a control device in a dedicated application such as industrial control and robotics
- Secondary storage limited or absent, data stored in short term memory, or read-only memory (ROM).

Other examples of hard real time activities include:

- Sensory data acquisition
- Detection of critical conditions
- Actuator serving
- Low level control of critical system components etc.

- **Soft:** Timeliness requirements are defined by using an average response time. If a single computation is late, it is not usually significant, although repeated late computation can result in system failures. An example of such a system includes airlines reservation systems.

  Although it is emphasised that deadlines are important they are not as strict as in hard real-time systems. Deadlines in this case can be missed occasionally but the important thing in this type of system is to try and reduce the amount of deadlines that are missed. Examples of soft real-time systems include operating systems.

  More specifically, we can say that-

  - A critical real-time task gets priority over the other tasks (Deadline desirable but not mandatory).
  - Limited utility in industrial control of robotics

- Useful in applications (multimedia, virtual reality) requiring advanced operating-system features.

Other examples of soft activities include:

- The command interpreter of the user interface
- Handling input data from a keyboard
- Displaying messages on the screen.
- Representation of system state variables.

- **Firm:** This is a combination of both hard and soft timeliness requirements. The computation has a shorter soft requirement and a longer hard requirement. For example, a patient ventilator must mechanically. Ventilate the patient, a certain amount in a given time period. A few seconds' delay in the initiation of breath is allowed, but not more than that.

The processes in the Real time may also be classified as follows:

- **Hard Real Time Tasks:** A real time task is said to be hard if missing its deadline may cause catastrophic consequences on the environment under control.
- **Soft Real Time Tasks:** A real time task is said to be soft if meeting its deadline is desirable for performance reasons, but missing its deadline does not cause serious damage to the environment and does not jeopardize correct system behavior.

All practical systems can be said to be real-time systems because they must produce an output or respond to the user's commands within a reasonable amount of time (insurance company responding to letters, word processor displaying what was typed on the screen, mobile phones responding with delays that allow 'comfortable' conversation). These systems where 'uncomfortably' long response times are a nuisance (such as windows2000) but the system still functions even if deadlines are sometimes not met are called soft real-time systems. Systems where failure to meet response time constraints leads to catastrophic system failure (aircraft crashing, car skidding, patient dying before corrective action is performed) are called hard real-time systems.

13.1.4 Example of Real Time System - MARS

MARS – Maintainable Real time System is a fault tolerant distributed real time system developed at the University of Vienna to support complex control applications, such as air traffic control systems, railway switching systems, where hard deadlines are imposed by the control environment. The MARS architecture consists of a set of computing nodes (clusters) connected through high speed communication channels. Each cluster is composed of a number of acquisition and processing units (components) interconnected by a synchronous real time bus, the MARS bus. Each component is a self-contained computer on which a set of real time application tasks and an identical copy of the MARS operating system are executed. The MARS configuration is shown as follows:

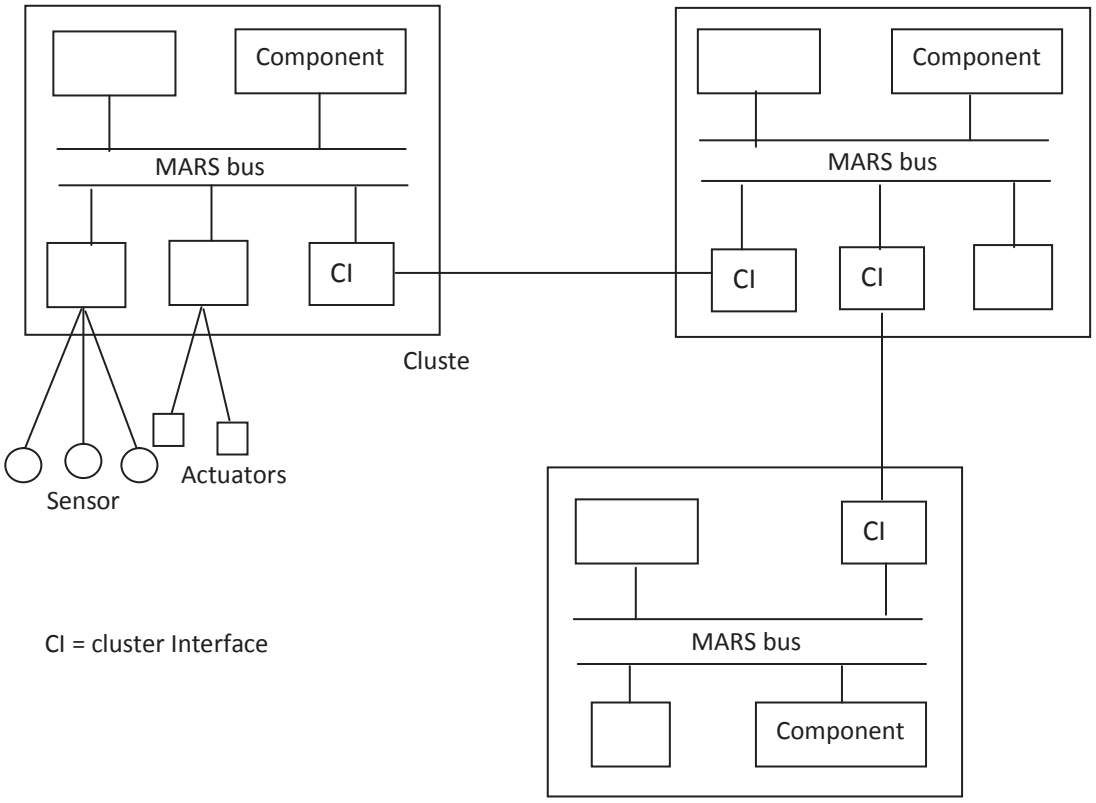


Figure 13.1.4 MARS Structure

The main feature that distinguishes MARS from other distributed real-time operating systems is its deterministic behavior even in peak-load conditions. Fault-tolerance is achieved at the cluster level through active redundant components, which are grouped in a set of Fault Tolerant Units (FTUs). High error detection coverage is achieved by use of software mechanisms at the kernel level and hardware mechanisms at the processor level.

Within an FTU, a single redundant component fails silently; that is, it either operates correctly or does not produce any results. This facilitates system maintainability and extensibility since redundant components may be removed from a running cluster, repaired and integrated later without affecting the operation of the cluster.

All MARS components have access to a common global time base, the system time, with known synchronization accuracy. It is used to test the validity of real time information, detect timing errors, control the access to the real time bus, and discard the redundant information.

The software residing on a MARS component can be split into the following three classes:

- **Operating System Kernel:** its primary goals are resource management and hardware transparency.
- **Hard Real Time Tasks (HRT):** HRT tasks are periodic activities that receive, process, and send messages. Each instance of a task is characterized by a hard deadline. The set of HRT tasks consists of application tasks and system tasks, which perform specific functions of the kernel, such as time synchronization and protocol conversions.
- **Soft Real Time Tasks (SRT):** SRT tasks are activities that are not subject to strict deadlines. Usually, they are periodic tasks scheduled in background during the idle time of the processor.

13.2 Desirable Characteristics

In any real time application, the various control activities can be seen as members of team acting together to accomplish one common goal, which can be the control of a nuclear power plant or an aircraft. This means that the tasks are not all independent and it is strictly necessary to support independent address spaces and a high degree of predictability. Some very important properties that real time systems should have to support critical applications are:

- **Timeliness:** Results have to be correct not only in their value but also in the time domain. As a consequence, the operating system must provide specific kernel mechanisms for time management and handling tasks with explicit time constraints and different criticalness.
- **Design for Peak Load:** Real time systems must not collapse when they are subject to peak load conditions, so they must be designed to handle all anticipated scenarios.
- **Predictability:** To guarantee a minimum level of performance, the system must be able to predict the consequence of any scheduling decision. If some task cannot be guaranteed within its time constraints, the system must notify this fact in advance, so that alternative actions can be planned in order to cope with the situation.
- **Fault Tolerance:** single hardware and software failures should not cause the system to crash. Thus, critical components of real time systems have to be designed to be fault tolerant.
- **Maintainability:** the architecture of a real time system should be designed according to a modular structure to ensure that possible system modifications are easy to perform.

13.2.1 Real Time System Tasks

Periodic and Deadline-oriented Tasks

The period is the amount of time between each iteration of a regularly repeated task. Such repeated tasks are called periodic tasks. This is the timing constraints within which the system has to perform the assigned task successfully. That is, real-time system is one that has to meet the timing constraints to avoid any possible failure. Thus every periodic task has to be finished within the defined period. It does not mean that a real-time system has to act fast.

Every task has to be deadline-oriented. That is, every task has to be accomplished within the set deadline. The deadline is a constraint on the latest time at which the operation has to come the end.

Cruise Control

Consider a cruise control mechanism on an automobile. The basic operation of cruise control is to keep the speed of the vehicle constant. Suppose the driver has selected 60 mph as the desired speed. If the vehicle is going slower or faster than the selected speed, then the embedded computer sends a signal to the engine controller to set the speed right. The frequency in which the computer checks whether the current speed of the vehicle is as per the set speed is called control rate and it is fixed by the control system designer. The checking frequency, on one side, should meet specifications but on the other side, it should not be obstructive to system functioning.

Aperiodic Tasks

All real-time tasks need not to be periodic. Aperiodic tasks respond to randomly arriving events. Consider anti-lock braking. If the driver presses the brake pedal, the car must respond very quickly. The response

time is the time between the moment the brake pedal is pressed, and the moment the anti-lock braking software actuates the brakes. If the response time was one second, an accident might occur. So, the fastest possible response is desired. But, like the cruise control algorithm, fastest is not necessarily best, because it is also desirable to keep the cost of parts down by using small microcontrollers. The point here is the application has to specify a worst-case response time and both the hardware and software has to be designed to meet the specifications.

13.3 Features of Real-Time Kernel

A kernel is a central component of an operating system. It acts as an interface between the user applications and the hardware. The sole aim of the kernel is to manage the communication between the software (user level applications) and the hardware (CPU, disk memory etc).

Typically, a kernel includes an interrupt handler that handles all requests or completed I/O operations that compete for the kernel's services, a scheduler that determines which programs share the kernel's processing time in what order, and a supervisor that actually gives use of the computer to each process when it is scheduled.

Following are some basic features of Real-Time Kernel

1. Task Management.
2. Timer Management.
3. Message Management.
4. Memory Management.

Each of these is discussed below.

Task Management

Task management includes the following.

1. Task Creation.
2. Task Scheduling.
3. Task Priorities.
4. Task Deletion (optional)

Task Creation

Ideally task creation should be dynamic that means task creation should be allowed during run-time. Systems that allow only static task creation (done at compile time by filling in a table for example) are less flexible, but by no means unusable. Most real-time systems do not require dynamic task creation, and furthermore, dynamic task creation can be effectively simulated by creating the maximum number of tasks at compile time, and enabling each as system requirements demand it.

Task Scheduling

A flexible system will allow task scheduling via events, time-slicing, or cooperation or any mix of the preceding.

Task Priorities

Priority based pre-emptive scheduling is an important feature. Priorities must be real-time responsive, and preferably dynamic.

Task Deletion

The ability to delete a task is a useful feature in dynamic environments.

Timer Management

Timer management should provide for pause, one-shot timers, periodic timers, and differential timer management. In addition, it is important that a timer be dispatched as a message to the task that started the timer. In this way much flexibility is added. For example, message aliasing can be used to allow task A to start a timer for task B; message cancellation can be used to cancel a timer, and so forth. Basically, any of the features of the message management system can be used to facilitate timer management.

Message Management

Messages are used for inter-task communication and the kernel message management system should have at least the features outlined below.

Message Queuing

Sometimes a burst of requests can be sent to a task and in this case the messages must by save in a queue so that the task can process each message in turn.

Variable Size Messages

Since messages can have varying amounts data, the kernel should allow for variable size messages. This can be accomplished by reserving one field of the message for a data pointer. For smaller data items a fixed block of bytes is typically reserved in the message for convenience.

Wait on a Message

Tasks that expect messages should be able to suspend, if they so desire, until a message arrives; this is referred to as the ability to "wait" for a message.

Selective Wait on a Message

A task should be able to wait for a particular message, or any one of a list of messages, or for any message at all regardless of its message id number.

Message Aliasing

Sometimes it is convenient for task A to send a message to task B, but force task B to respond to task C. This can be done if task A can change to sender field in the message so that when task B responds to the sender, he really responds to task C.

Timed Wait on a Message

Sometimes you only want to wait so long for a message and then time out.

Peeking

Sometimes it is convenient for a task to peek into its message queue and scan the messages, but remove none, leaving the queue intact.

Message Priorities

Priority refers to the order that means which message will be handled by the server next. Sometimes it is important to have the ability to assign priorities to messages. Consider task A which has 5 messages in its queue. If task B determines that an emergency stop is required, and sends task A an EMERGENCY_STOP message, then without message priorities, this important message would be number 6 in the queue. However, by sending the message with a higher priority, it can be forced to the front of the message queue.

Message Identity

It is important for the receiver of a message to know who sent the message so that he can respond if necessary.

Message Throttling

This feature allows the programmer to specify, on a queue by queue basis, the maximum number of messages allowed.

Memory Management

The kernel should provide for high speed deterministic memory management. Fixed block memory management is preferred because of its high speed and the elimination of fragmentation.

In addition to features it has also some responsibilities-

- Thread Scheduling
- Interrupt and exception handling
- Low level processes synchronization
- Recover after a power failure

13.4 Implementation of Real-Time OS

The implementing the Real Time OS depends on various aspects:

13.4.1 Basic Structure

- The most important component of RTOS is its kernel (Monolithic & Microkernel).
- BSP or Board Support Package makes an RTOS target-specific (It's a processor specific code onto (processor) which we like to have our RTOS running).

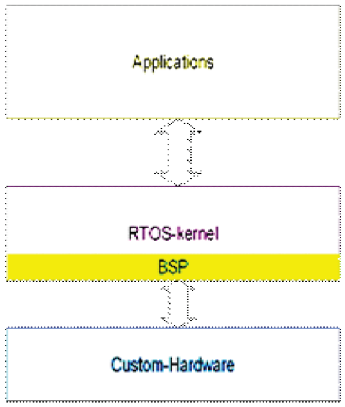


Figure 13.4.1: RTOS Kernel

13.4.2 RTOS KERNEL: Tasks

- A task is basic unit of execution in RTOS.
- RTOS scheduler needs to be deterministic ~ O (1) or O (n).

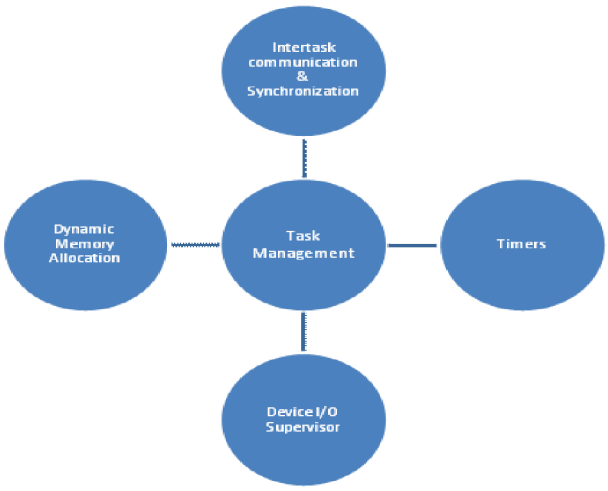


Figure13.4.2: RTOS Kernel Tasks

13.4.3 Scheduling Policy

Scheduling refers to the order which specify the next process. Scheduling policies that are available in a RTOS are:

- Clock driven
- Priority driven (RMS & EDF)

13.4.4 Task Control Block

The task control block generally refers to the data structure where various tasks are grouped together. This data structure is called the Task Control Block. The TCB generally contains the-

- Task ID
- Keep track the status of each task
- Signal and events performed by particular task.
- Memory and some system variable etc.

These are illustrated below:

TID
SAVED_TASK_STATE
Resource 1(Signals)
Resource 2 (events)
Resource 3 (Shared Memory)
...
System Variables

Figure 13.4.4: Task Control Block

13.5 Summary

Real-Time systems span a large part of computer industry. So far most of the real-time systems research has been mostly confined to single node systems and mainly for processor scheduling. This needs to be extended for multiple resources and distributed nodes. Real-time systems are expanding to several other domains such as automotive industry and embedded real-time systems. Especially the marriage of the Internet with multimedia applications has opened several new volume applications.

13.6 Self - Assessment Exercise

1. Define Real Time System. Explain the different class of Real Time System
2. What are the desirable characteristics of RTOS? Explain.
3. How Aperiodic tasks is different from periodic tasks?
4. Explain different features of Real Time Kernel.
5. Explain in brief the implementation components of RTOS.

13.7 References

- Operating System Concepts by Silberschatz Galvin Gagne
- System programming and operating system by D. M. Dhamdhere

Unit - 14 : Multimedia Systems

Structure of Unit

14.0 Objectives

In this lesson we will learn the preliminary concepts of Multimedia. We will discuss the various benefits and applications of multimedia. After going through this chapter the reader will be able to :

- Define multimedia
- List the elements of multimedia
- Enumerate the different applications of multimedia
- Describe the different stages of multimedia software development

14.1 Introduction

Multimedia has become an inevitable part of any presentation. It has found a variety of applications right from entertainment to education. The evolution of internet has also increased the demand for multimedia content. **Multimedia** is the media that uses multiple forms of information content and information processing (e.g. text, audio, graphics, animation, video, interactivity) to inform or entertain the user. *Multimedia* also refers to the use of electronic media to store and experience multimedia content. Multimedia is similar to traditional mixed media in fine art, but with a broader scope. The term "rich media" is synonymous for interactive multimedia.

The term ***multimedia*** describes a wide range of applications that are in popular use today. These include audio and video files such as MP3 audio files, DVD movies, and short video clips of movie previews or news stories downloaded over the Internet. Multimedia applications also include live webcasts (broadcast over the World Wide Web) of speeches or sporting events and even live webcams that allow a viewer in Manhattan to observe customers at a cafe in Paris. Multimedia applications need not be either audio or video; rather, a multimedia application often includes a combination of both. For example, a movie may consist of separate audio and video tracks. Nor must multimedia applications be delivered only to desktop personal computers. Increasingly, they are being directed toward smaller devices, including personal digital assistants (PDAs) and cellular telephones. For example, a stock trader may have stock quotes delivered in real time to her PDA.

Multimedia may be broadly divided into two categories

14.1.1 Linear Multimedia

Linear active content progresses without any navigational control for the viewer such as a cinema presentation. It is a non-interactive form of multimedia in which a project starts at a beginning and runs through to the end. Conventional "television" is essentially "linear multimedia" (although switching channels could be said to involve interactivity). However, there is no ability for the end user to control when elements are to be delivered. Some multimedia projects are like television in that they present material in a linear fashion from beginning to end.

Linear multimedia can be distinguished from non-linear multimedia because it has literally no interactivity of any kind. It lacks any extra features that a user can take advantage of, such as the ability to choose different options, click on icons, control the flow of the media, or change the pace at which the media is displayed.

The main reason to use linear multimedia over the more interactive and fun non-linear types of multimedia is to aid in teaching or training. Linear multimedia works exceedingly well for providing information to large groups of people such as at training sessions, seminars, workplace meetings, or study groups.

14.1.2 Non Linear Multimedia

In Non Linear Multimedia the end user is given navigational control to wander through multimedia content at will. The user can control what is seen and when it will be seen. Non-linear content offers user interactivity to control progress as used with a computer game or used in self-paced computer based training.

14.2 Elements of Multimedia System

Multimedia means that computer information can be represented through audio, graphics, image, video and animation in addition to traditional media(text and graphics). Hypermedia can be considered as one type of particular multimedia application.

- Audio
- Video
- Graphics
- Images

14.3 Categories of Multimedia

Multimedia may be broadly divided into **linear** and **non-linear** categories. Linear active content progresses without any navigation control for the viewer such as a cinema presentation. Non-linear content offers user interactivity to control progress as used with a computer game or used in self-paced computer based training. Non-linear content is also known as hypermedia content.

Multimedia presentations can be live or recorded. A recorded presentation may allow interactivity via a navigation system. A live multimedia presentation may allow interactivity via interaction with the presenter or performer.

14.4 Features of Multimedia

Multimedia presentations may be viewed in person on stage, projected, transmitted, or played locally with a media player. A broadcast may be a live or recorded multimedia presentation. Broadcasts and recordings can be either analog or digital electronic media technology. Digital online multimedia may be downloaded or streamed. Streaming multimedia may be live or on-demand.

Multimedia games and simulations may be used in a physical environment with special effects, with multiple users in an online network, or locally with an offline computer, game system, or simulator.

14.5 Applications of Multimedia

Multimedia finds its application in various areas including, but not limited to, advertisements, art, education, entertainment, engineering, medicine, mathematics, business, scientific research and spatial, temporal applications.

14.5.1 A Few Application Areas of Multimedia

Creative Industries

Creative industries use multimedia for a variety of purposes ranging from fine arts, to entertainment, to commercial art, to journalism, to media and software services provided for any of the industries listed below. An individual multimedia designer may cover the spectrum throughout their career. Request for their skills range from technical, to analytical and to creative.

A presentation using Powerpoint.

Corporate presentations may combine all forms of media content.

Commercial

Much of the electronic old and new media utilized by commercial artists is multimedia. Exciting presentations are used to grab and keep attention in advertising. Industrial, business to business, and interoffice communications are often developed by creative services firms for advanced multimedia presentations beyond simple slide shows to sell ideas or liven-up training..

Entertainment and Fine Arts

In addition, multimedia is heavily used in the entertainment industry, especially to develop special effects in movies and animations. Multimedia games are a popular pastime and are software programs available either as CD-ROMs or online. Some video games also use multimedia features.

Education

In Education, multimedia is used to produce computer-based training courses (popularly called CBTs) and reference books like encyclopedia and almanacs. A CBT lets the user go through a series of presentations, text about a particular topic, and associated illustrations in various information formats. Edutainment is an informal term used to describe combining education with entertainment, especially multimedia entertainment.

Engineering

Software engineers may use multimedia in Computer Simulations for anything from entertainment to training such as military or industrial training. Multimedia for software interfaces are often done as collaboration between creative professionals and software engineers.

Industry

In the Industrial sector, multimedia is used as a way to help present information to shareholders, superiors and coworkers. Multimedia is also helpful for providing employee training, advertising and selling products all over the world via virtually unlimited web-based technologies.

Mathematical and Scientific Research

In Mathematical and Scientific Research, multimedia is mainly used for modeling and simulation. For example, a scientist can look at a molecular model of a particular substance and manipulate it to arrive at a new substance. Representative research can be found in journals such as the Journal of Multimedia.

Medicine

In Medicine, doctors can get trained by looking at a virtual surgery or they can simulate how the human body is affected by diseases spread by viruses and bacteria and then develop techniques to prevent it.

14.5.2 Multimedia in Public Places

In hotels, railway stations, shopping malls, museums, and grocery stores, multimedia will become available at stand-alone terminals or kiosks to provide information and help. Such installation reduce demand on traditional information booths and personnel, add value, and they can work around the clock, even in the middle of the night, when live help is off duty.

A menu screen from a supermarket kiosk that provide services ranging from meal planning to coupons. Hotel kiosk list nearby restaurant, maps of the city, airline schedules, and provide guest services such as automated checkout. Printers are often attached so users can walk away with a printed copy of the information. Museum kiosk are not only used to guide patrons through the exhibits, but when installed at each exhibit, provide great added depth, allowing visitors to browser though richly detailed information specific to that display.

14.6 Stages of Multimedia Application Development

A Multimedia application is developed in stages as all other software are being developed. In multimedia application development a few stages have to complete before other stages being, and some stages may be skipped or combined with other stages.

Following are the four basic stages of multimedia project development:

14.6.1 Planning and Costing:

This stage of multimedia application is the first stage which begins with an **idea** or need. This idea can be further refined by outlining its messages and objectives. Before starting to develop the multimedia project, it is necessary to plan what writing skills, graphic art, music, video and other multimedia expertise will be required. It is also necessary to estimate the time needed to prepare all elements of multimedia and prepare a **budget** accordingly. After preparing a budget, a **prototype** or proof of concept can be developed.

14.6.2 Designing and Producing:

The next stage is to execute each of the planned tasks and create a finished product.

14.6.3 Testing:

Testing a project ensure the product to be free from bugs. Apart from bug elimination another aspect of testing is to ensure that the multimedia application meets the objectives of the project. It is also necessary to test whether the multimedia project works properly on the intended deliver platforms and they meet the needs of the clients.

14.6.4 Delivering:

The final stage of the multimedia application development is to pack the project and deliver the completed project to the end user. This stage has several steps such as implementation, maintenance, shipping and marketing the product.

14.7 Compression

Because of the size and rate requirements of multimedia systems, multimedia files are often compressed from their original form to a much smaller form. Once a file has been compressed, it takes up less space for storage and can be delivered to a client more quickly. Compression is particularly important when the content is being streamed across a network connection. In discussing file compression, we often refer to the compression ratio, which is the ratio of the original file size to the size of the compressed file. For example, an 800-KB file that is compressed to 100 KB has a compression ratio of 8:1.

14.7.1 Image Compression

The best image quality at a given bit-rate (or compression rate) is the main goal of image compression, however, there are other important properties of image compression schemes:

Scalability generally refers to a quality reduction achieved by manipulation of the bitstream or file (without decompression and re-compression). Other names for scalability are *progressive coding* or *embedded bitstreams*.m Despite its contrary nature, scalability also may be found in lossless codecs, usually in form of coarse-to-fine pixel scans. Scalability is especially useful for previewing images while downloading them (e.g., in a web browser) or for providing variable quality access to e.g., databases. There are several types of scalability:

- **Quality Progressive** or layer progressive: The bitstream successively refines the reconstructed image.
- **Resolution Progressive**: First encode a lower image resolution; then encode the difference to higher resolutions
- **Component Progressive**: First encode grey; then color.

Region of Interest Coding. Certain parts of the image are encoded with higher quality than others. This may be combined with scalability (encode these parts first, others later).

Meta Information. Compressed data may contain information about the image which may be used to categorize, search, or browse images. Such information may include color and texture statistics, small preview images, and author or copyright information.

Processing Power. Compression algorithms require different amounts of processing power to encode and decode. Some high compression algorithms require high processing power.

14.7.2 Video Compression

To digitize and store a 10-second clip of full-motion video in your computer requires transfer of an enormous amount of data in a very short amount of time. Reproducing just one frame of digital video component video at 24 bits requires almost 1MB of computer data; 30 seconds of video will fill a gigabyte hard disk. Full-size, full-motion video requires that the computer deliver data at about 30MB per second. This overwhelming

technological bottleneck is overcome using digital video compression schemes or *codecs* (coders/decoders). A codec is the algorithm used to compress a video for delivery and then decode it in real-time for fast playback.

Real-time video compression algorithms such as MPEG, P*64, DVI/Indeo, JPEG, Cinepak, Sorenson, ClearVideo, RealVideo, and VDOwave are available to compress digital video information. Compression schemes use Discrete Cosine Transform (DCT), an encoding algorithm that quantifies the human eye's ability to detect color and image distortion. All of these codecs employ lossy compression algorithms.

In addition to compressing video data, *streaming* technologies are being implemented to provide reasonable quality low-bandwidth video on the Web. Microsoft, RealNetworks, VXtreme, VDOnet, Xing, Precept, Cubic, Motorola, Viva, Vosaic, and Oracle are actively pursuing the commercialization of streaming technology on the Web. QuickTime, Apple's software-based architecture for seamlessly integrating sound, animation, text, and video (data that changes over time), is often thought of as a compression standard, but it is really much more than that.

14.7.2.1 MPEG

The MPEG standard has been developed by the Moving Picture Experts Group, a working group convened by the International Standards Organization (ISO) and the International Electro-technical Commission (IEC) to create standards for digital representation of moving pictures and associated audio and other data. MPEG1 and MPEG2 are the current standards.

14.7.2.2 DVI/Indeo

DVI is a property, programmable compression/decompression technology based on the Intel i750 chip set. This hardware consists of two VLSI (Very Large Scale Integrated) chips to separate the image processing and display functions.

Two levels of compression and decompression are provided by DVI: Production Level Video (PLV) and Real Time Video (RTV). PLV and RTV both use variable compression rates. DVI's algorithms can compress video images at ratios between 80:1 and 160:1. DVI will play back video in full-frame size and in full color at 30 frames per second.

14.7.2.3 Optimizing Video Files for CD-ROM

CD-ROMs provide an excellent distribution medium for computer-based video: they are inexpensive to mass produce, and they can store great quantities of information. CDROM players offer slow data transfer rates, but adequate video transfer can be achieved by taking care to properly prepare your digital video files.

- Limit the amount of synchronization required between the video and audio. With Microsoft's AVI files, the audio and video data are already interleaved, so this is not a necessity, but with QuickTime files, you should "flatten" your movie. *Flattening* means you interleave the audio and video segments together.
- Use regularly spaced key frames, 10 to 15 frames apart, and temporal compression can correct for seek time delays. *Seek time* is how long it takes the CD-ROM player to locate specific data on the CD-ROM disc. Even fast 56x drives must spin up, causing some delay (and occasionally substantial noise).
- The size of the video window and the frame rate you specify dramatically affect performance. In QuickTime, 20 frames per second played in a 160X120-pixel window is equivalent to playing 10 frames per second in a 320X240 window.

14.8 Requirements of Multimedia Kernels

We refer to a multimedia environment, consisting of workstations and servers connected by a network dealing with video and audio streams. These streams require Real Time (RT) handling. Fortunately the requirements are not so demanding as those in Hard Real Time (HRT) systems. Therefore resources are easier to obtain and less restrictive in use. This implies a more relaxed behavior of workstation kernels with respect to timeliness and opens the possibility of having a better average use of resources.

Such a kernel may (1) respond in an opportunistic way by serving the most demanding process immediately and not caring about the others or it may (2) make flexible Quality of Service (QoS) contracts with applications. In case of emergency it should be possible to break a contract. Both types of kernels are considered and (dis)advantages are compared.

We will consider two types of RT kernels:

- Opportunistic kernel
- Contract kernel

Sometimes these kernels are also referred to as dynamic or static respectively. In a dynamic kernel no information about the arrival of tasks is known a priory. Therefore the scheduling decisions have to be made at the moment a task arrives. In a static kernel however, arrival times and resource usage are known a priori. So the scheduling decisions can be done off-line. We will give an overview of both kernel types and will compare the advantages and disadvantages.

14.8.1 The Opportunistic Kernel

In the opportunistic kernel we think of a workstation with a RT kernel that will adapt itself to the immediate needs of the system as well as possible. The kernel has no knowledge about the behavior of its environment in the future. It simply serves the most important process until this process has completed or until a more important process is requesting service. Therefore the kernel is called opportunistic.

In the opportunistic kernel jobs will arrive without a priori knowledge. These are scheduled according to a priority. This priority is typically derived from parameters as used in classical scheduling policies such as Static Priority Scheduling, Shortest Process Time, Shortest Slack Time (elapsed time minus its estimated completion time) or Earliest Deadline First. Among others Liu and Layland proved that a given set of processes, scheduled by any satisfying scheduling algorithm, could also be scheduled by a deadline driven algorithm. Without any precautions these techniques may lead to the phenomenon of priority inversion. This could happen when a high priority task is blocked for a resource that is held by a low priority task. The latter may not proceed due to its low priority, thus blocking the high priority task.

Depending on the synchronization policy such as the Fixed Priority Protocol, the Basic Inheritance Protocol, the Priority Ceiling Protocol or the RT Transaction Protocol (RTTP) a dispatcher assigns processes to the processor(s).

- The Fixed Priority Protocol generally suffers from priority inversion. The Basic Inheritance Protocol, the Priority Ceiling Protocol and the RT Transaction Protocol provide methods to avoid priority inversion.
- The Basic Inheritance Protocol does this by inheritance of priority. Low priority processes, owning shared resources that are also requested by high priority processes, inherit the high priority from the waiting processes. A primary disadvantage of this scheme is the impossibility of avoiding transitive waiting.
- The Priority Ceiling Protocol avoids priority inheritance and also transitive waiting. The basic idea is to make way for high priority jobs, even if it is not certain that they will become active. The rule is that a medium priority job may not pre-empt a low priority job if the low priority job holds resources that could be claimed by a high priority job. The priority ceiling associated with a resource is the highest priority of a job that ever can claim this resource

14.8.2 The Contract Kernel

In the contract kernel a distributed application tries to establish a contract between several parties, such as a producer (workstation), a network and a consumer (workstation). Of course this can be extended to a client/server environment. The network is typically an Asynchronous Transfer Mode (ATM) network. Such a network can give some statistical guarantee of bandwidth and it can support RT behavior under the assumption that both end systems keep to the contract. The service required by the network is specified by QoS requirements such as bandwidth, delay, jitter and error rate. However, it might happen that some thrashing still occurs occasionally. In this type of network HRT guarantees cannot be given. For multimedia application this is not necessary, as we have indicated already in the introduction.

For the periodic processes we can estimate the future needs of processor time and resource usage. A rational technique would be to compute a schedule at the time a contract is made. However, sporadic unexpected events might make it difficult to keep to this schedule. This requires some extra adaptability of our kernel. We might have to push the processor beyond the average agreed values in the contract. This certainly does not mean that the schedulability criteria of Liu and Layland are not valid any more. On the contrary, their Rate Monotonic algorithm is still very useful and it can give an easy first hand impression for admission control of a task.

14.9 CPU Scheduling

In computer science, scheduling is the method by which threads, processes or data flows are given access to system resources (e.g. processor time, communications bandwidth). This is usually done to load balance a system effectively or achieve a target quality of service. The need for a scheduling algorithm arises from the requirement for most modern systems to perform multitasking (execute more than one process at a time) and multiplexing (transmit multiple flows simultaneously).

The scheduler is concerned mainly with:

- Throughput - The total number of processes that complete their execution per time unit.
- Latency, specifically:
  - Turnaround time - total time between submission of a process and its completion.
  - Response time - amount of time it takes from when a request was submitted until the first response is produced.
- Fairness / Waiting Time - Equal CPU time to each process (or more generally appropriate times according to each process' priority). It is the time for which the process remains in the ready queue.

In practice, these goals often conflict (e.g. throughput versus latency), thus a scheduler will implement a suitable compromise. Preference is given to any one of the above mentioned concerns depending upon the user's needs and objectives.

In real-time environments, such as embedded systems for automatic control in industry (for example robotics), the scheduler also must ensure that processes can meet deadlines; this is crucial for keeping the system stable. Scheduled tasks are sent to mobile devices and managed through an administrative back end.

14.9.1 Scheduling Disciplines

Scheduling disciplines are algorithms used for distributing resources among parties which simultaneously and asynchronously request them. Scheduling disciplines are used in routers (to handle packet traffic) as well as in operating systems (to share CPU time among both threads and processes), disk drives (I/O scheduling), printers (print spooler), most embedded systems, etc.

The main purposes of scheduling algorithms are to minimize resource starvation and to ensure fairness amongst the parties utilizing the resources. Scheduling deals with the problem of deciding which of the outstanding requests is to be allocated resources. There are many different scheduling algorithms. In this section, we introduce several of them.

In packet-switched computer networks and other statistical multiplexing, the notion of a scheduling algorithm is used as an alternative to first-come first-served queuing of data packets.

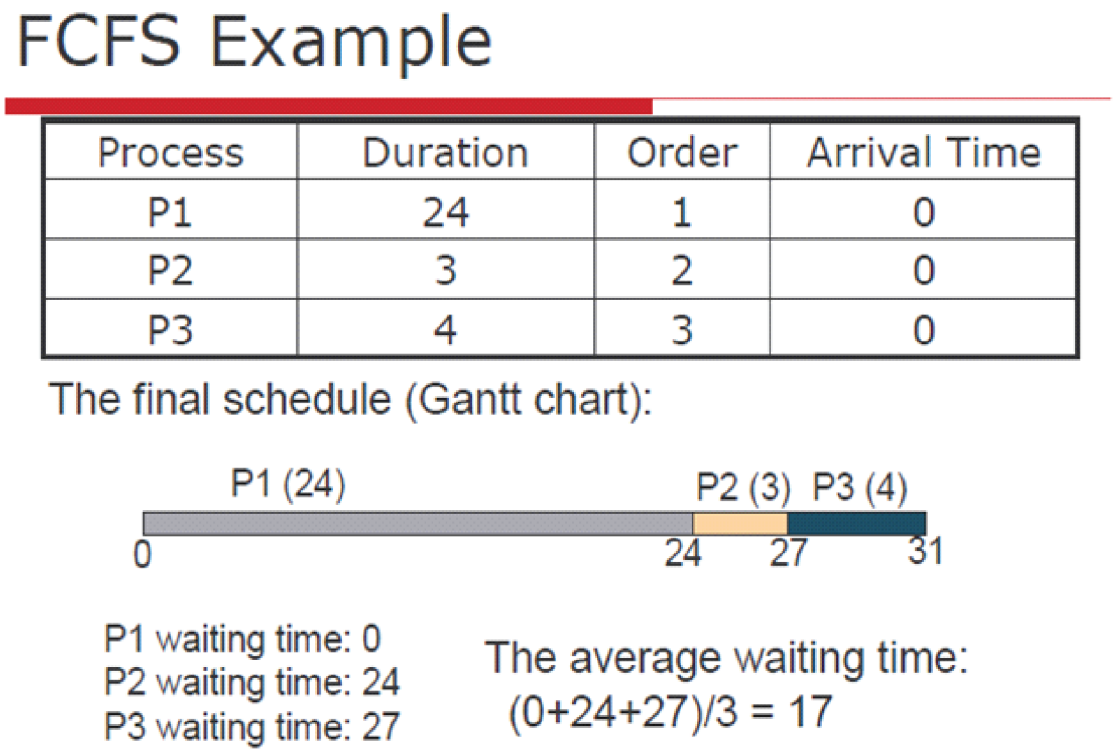
14.9.1.1 FIFO

FIFO is an acronym for **First In, First Out**, which is an abstraction related to ways of organizing and manipulation of data relative to time and prioritization. This expression describes the principle of a queue processing technique or servicing conflicting demands by ordering process by first-come, first-served (**FCFS**) behavior: where the persons leave the queue in the order they arrive, or waiting one's turn at a traffic control signal.

FCFS is also the jargon term for the FIFO operating system scheduling algorithm, which gives every process CPU time in the order they come. In the broader sense, the abstraction LIFO, or Last-In-First-Out is the opposite of the abstraction FIFO organization. The difference perhaps is clearest with considering the less commonly used synonym of LIFO, FILO (meaning First-In-Last-Out). In essence, both are specific cases of a more generalized list (which could be accessed anywhere). The difference is not in the list (data), but in the rules for accessing the content. One sub-type adds to one end, and takes off from the other, its opposite takes and puts things only on one end

Algorithm:

- Since context switches only occur upon process termination, and no reorganization of the process queue is required, scheduling overhead is minimal.
- Throughput can be low, since long processes can hog the CPU
- Turnaround time, waiting time and response time can be high for the same reasons above
- No prioritization occurs, thus this system has trouble meeting process deadlines.
- The lack of prioritization means that as long as every process eventually completes, there is no starvation. In an environment where some processes might not complete, there can be starvation.
- It is based on Queuing



14.9.1.2 Shortest Remaining Time

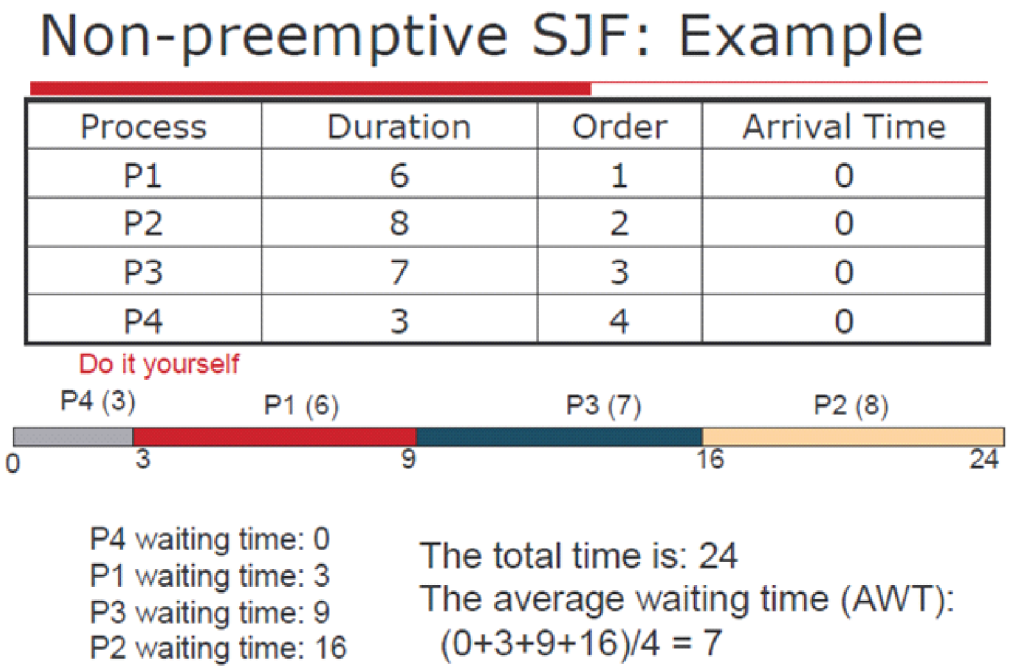
Shortest remaining time, also known as **shortest remaining time first (SRTF)**, is a scheduling method that is a preemptive version of shortest job next scheduling. In this scheduling algorithm, the process with the smallest amount of time remaining until completion is selected to execute. Since the currently executing process is the one with the shortest amount of time remaining by definition, and since that time should only reduce as execution progresses, processes will always run until they complete or a new process is added that requires a smaller amount of time.

Shortest remaining time is advantageous because short processes are handled very quickly. The system also requires very little overhead since it only makes a decision when a process completes or a new process is added, and when a new process is added the algorithm only needs to compare the currently executing process with the new process, ignoring all other processes currently waiting to execute

Algorithm:

With this strategy the scheduler arranges processes with the least estimated processing time remaining to be next in the queue. This requires advanced knowledge or estimations about the time required for a process to complete.

- If a shorter process arrives during another process' execution, the currently running process may be interrupted (known as preemption), dividing that process into two separate computing blocks. This creates excess overhead through additional context switching. The scheduler must also place each incoming process into a specific place in the queue, creating additional overhead.
- This algorithm is designed for maximum throughput in most scenarios.
- Waiting time and response time increase as the process' computational requirements increase. Since turnaround time is based on waiting time plus processing time, longer processes are significantly affected by this. Overall waiting time is smaller than FIFO, however since no process has to wait for the termination of the longest process.
- No particular attention is given to deadlines, the programmer can only attempt to make processes with deadlines as short as possible.
- Starvation is possible, especially in a busy system with many small processes being run.



14.9.1.3 Fixed Priority Pre-emptive Scheduling

Fixed-priority pre-emptive scheduling is a scheduling system commonly used in real-time systems. With fixed priority pre-emptive scheduling, the scheduler ensures that at any given time, the processor executes the highest priority task of all those tasks that are currently ready to execute.

The pre-emptive scheduler has a clock interrupt task that can provide the scheduler with options to switch after the task has had a given period—the time slice. This scheduler system has the advantage of making sure no task hogs the processor for any time longer than the time slice. However this scheduling scheme also has the downfall of process or thread lockout, as priority is given to higher priority tasks the lower priority tasks could wait an indefinite amount of time. One common method of arbitrating this situation is the use of aging. Aging will slowly increment a process/thread(s) priority which is in the wait queue to ensure some degree of fairness. Most Real-time operating systems (RTOSs) have pre-emptive schedulers. Also turning off time slicing effectively gives you the non-pre-emptive RTOS.

Algorithm:

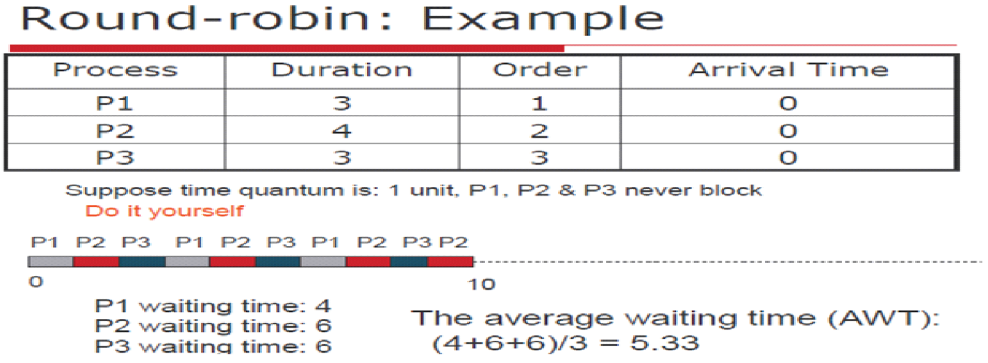
The OS assigns a fixed priority rank to every process, and the scheduler arranges the processes in the ready queue in order of their priority. Lower priority processes get interrupted by incoming higher priority processes.

- Overhead is not minimal, nor is it significant.
- FPPS has no particular advantage in terms of throughput over FIFO scheduling.
- Waiting time and response time depend on the priority of the process. Higher priority processes have smaller waiting and response times.
- Deadlines can be met by giving processes with deadlines a higher priority.
- Starvation of lower priority processes is possible with large amounts of high priority processes queuing for CPU time.

14.9.1.4 Round-robin Scheduling

In order to schedule processes fairly, a round-robin scheduler generally employs time-sharing, giving each job a time slot or *quantum* (its allowance of CPU time), and interrupting the job if it is not completed by then. The job is resumed next time a time slot is assigned to that process. In the absence of time-sharing, or if the quanta were large relative to the sizes of the jobs, a process that produced large jobs would be favoured over other processes.

Example: If the time slot is 100 milliseconds, and *job1* takes a total time of 250 ms to complete, the round-robin scheduler will suspend the job after 100 ms and give other jobs their time on the CPU. Once the other jobs have had their equal share (100 ms each), *job1* will get another allocation of CPU time and the cycle will repeat. This process continues until the job finishes and needs no more time on the CPU.



Algorithm:

The scheduler assigns a fixed time unit per process, and cycles through them.

- RR scheduling involves extensive overhead, especially with a small time unit.
- Balanced throughput between FCFS and SJF, shorter jobs are completed faster than in FCFS and longer processes are completed faster than in SJF.
- Poor average response time, waiting time is dependent on number of processes, and not average process length.
- Because of high waiting times, deadlines are rarely met in a pure RR system.
- Starvation can never occur, since no priority is given. Order of time unit allocation is based upon process arrival time, similar to FCFS.

14.10 Disk Scheduling

As we know that on a single computer we can perform many operations at a time so that management is also necessary on all the running processes those are running on the system at a time. With the help or advent of the multi-programming we can execute many programs at a time. So far controlling and providing the memory to all the processes operating system uses the concept of disk scheduling.

In this the time of cpu is divided into the various processes and also determines that all the processes will work properly. So that disk scheduling will specifies that at which time which process will be executed by the cpu. So that the scheduling means to execute all the processes those are given to a cpu at a time. The scheduling is used for divide the total time of the cpu between the number or processes so that the processes can execute concurrently at a single time. For sharing the time or for dividing the total time of the cpu, the cpu uses the following the scheduling techniques.

14.10.1 FCFS or First Come First Serve

In this jobs or processes are executed in the manner in which they are entered into the computer. In this operating system creates a queue which contains the sequence order in which they are to be executed and the sequence in which the cpu will execute the process.. In this all the jobs are performed according to their sequence order as they have entered. In this the job which had requested first will firstly performed by the cpu. And the jobs those are entered later will be executed in to their entering order.

14.10.2 SSTF or Shortest Seek Time First

In this technique the operating system will search for the shortest time means this will search which job will takes a less time of cpu for running. And after examining all the jobs, all the jobs are arranged in the sequence wise or they are organized into the priority order. The priority of the process will be the total time which a process will use for execution. The shortest seek time will include all the time means time to enter and time to completion of the process.

14.10.3 C-scan Scheduling

In the c-scan all the processes are arranged by using some circular list. Circular list is that in which there is no start and end point of the list means the end of the list is the starting point of the list. In the c-scan scheduling the cpu will search for the process from start to end and if an end has found then this again start from the starting process. Because many times when a cpu is executing the processes then may a user wants

to enter some data means a user wants to enter some data so that at that situation the cpu will again execute that process after the input operation. So that c-scan scheduling is used for processing same processes again and again.

14.10.4 Look Scheduling

In the look scheduling the cpu scans the list from starting to end of the disk in which the various processes are running and in the look scheduling the cpu will scan the entire disk from one end to the second end.

14.10.5 Round Robin

In the round robin scheduling the time of cpu is divided into the equal numbers which is also called as quantum time. Each process which is request for execution will consumes the equal number of times of the cpu and after the quantum time of first process, the cpu will automatically goes to the next process. But the main problem is that after the completion of the process the time will also be consumed by the process. Means if a process either or not have some operations to perform the time of cpu will also be consume by the cpu , so this is the wastage of the time of the cpu.

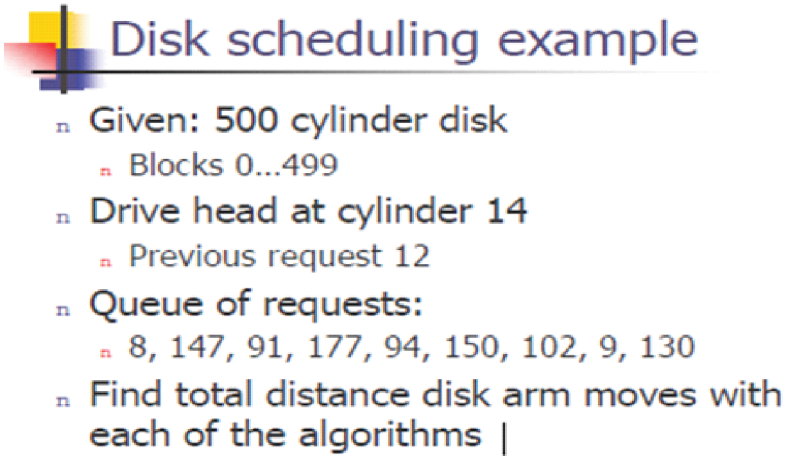
14.10.6 Priority Scheduling

In this each process have some priorities assign to them, means each and every process will be examined by using the total time which will be consumed by the process. The total time of the process, and number of times a process needs some input and output and number of resources will be examines to set the priorities of the processes. So that all the processes are arranged into the form of these criteria's and after that they will be processed by the cpu.

14.10.7 Multilevel Queue

The multilevel queue is used when there are multiple queues for the various different processes as we know that there are many different types of works those are to be performed on the computers at a time. So that for organizing the various or different types of queues the cpu maintains the queues by using this technique. In this all the queues are collected and organized in the form of some functions which they are requesting to perform. So that the various types of queues are maintained this contains all the processes which have same type.

Example:



14.11 Network Management

Network management refers to the activities, methods, procedures, and tools that pertain to the operation, administration, maintenance, and provisioning of networked systems.

- Operation deals with keeping the network (and the services that the network provides) up and running smoothly. It includes monitoring the network to spot problems as soon as possible, ideally before users are affected.
- Administration deals with keeping track of resources in the network and how they are assigned. It includes all the "housekeeping" that is necessary to keep the network under control.
- Maintenance is concerned with performing repairs and upgrades—for example, when equipment must be replaced, when a router needs a patch for an operating system image, when a new switch is added to a network. Maintenance also involves corrective and preventive measures to make the managed network run "better", such as adjusting device configuration parameters.

- Provisioning is concerned with configuring resources in the network to support a given service. For example, this might include setting up the network so that a new customer can receive voice service.

A common way of characterizing network management functions is FCAPS—Fault, Configuration, Accounting, Performance and Security.

Functions that are performed as part of network management accordingly include controlling, planning, allocating, deploying, coordinating, and monitoring the resources of a network, network planning, frequency allocation, predetermined traffic routing to support load balancing, cryptographic key distribution authorization, configuration management, fault management, security management, performance management, bandwidth management, Route analytics and accounting management.

Data for network management is collected through several mechanisms, including agents installed on infrastructure, synthetic monitoring that simulates transactions, logs of activity, sniffers and real user monitoring. In the past network management mainly consisted of monitoring whether devices were up or down; today performance management has become a crucial part of the IT team's role which brings about a host of challenges—especially for global organizations.

A small number of accessories methods exist to support network and network device management. Access methods include the SNMP, command-line interface (CLIs), custom XML, CMIP, Windows Management Instrumentation (WMI), Transaction Language 1, CORBA, NETCONF, and the Java Management Extensions (JMX). Internet service providers (ISP) use a technology known as deep packet inspection in order to regulate network congestion and lessen Internet bottlenecks.

Schemas include the WBEM, the Common Information Model, and MTOSI amongst others.

Medical Service Providers provide a niche marketing utility for managed service providers; as HIPAA legislation consistently increases demands for knowledgeable providers. Medical Service Providers are liable for the protection of their clients confidential information, including in an electronic realm. This liability creates a significant need for managed service providers who can provide secure infrastructure for transportation of medical data.

14.12 Summary

- Multimedia is a woven combination of text, audio, video, images and animation.
- Multimedia systems finds a wide variety of applications in different areas such as education, entertainment etc.
- The categories of multimedia are linear and non-linear.
- The stages for multimedia application development are Planning and costing, designing and producing, testing and delivery.
- Compression is to reduce size of image or video for better multimedia experience.
- Multimedia kernel performs real Time (RT) handling of streaming.
- Scheduling is the method by which threads, processes or data flows are given access to system resources
- Scheduling disciplines are algorithms used for distributing resources among parties which simultaneously and asynchronously request them

- The disk scheduling is used for divide the total time of the cpu between the number or processes so that the processes can execute concurrently at a single time
- Network management refers to the activities, methods, procedures, and tools that pertain to the operation, administration, maintenance, and provisioning of networked systems

14.13 Self - Assessment Exercise

1. List five applications of multimedia?
2. Five batch jobs A through E arrive at the system at almost the same time and form a queue of A,B, C, D, E. They have estimated running time of 10, 6, 2, 4 and 8 minutes respectively. Suppose that RR is used to schedule them. What is the turnaround time for C process? What is the turnaround time for D process?
3. What is Multimedia Kernel? List its requirements.
4. What is compression?
5. Explain what do you mean by Image compression and video compression?
6. What do you mean by Scheduling?
7. What is CP scheduling? State and explain the algorithms.
8. What is disk scheduling? State and explain the algorithms
9. What do you mean by Network management? Explain.

14.14 References

- Multimedia Making it work" By Tay Vaughan
- "Multimedia in Practice – Technology and applications" By Jeffcoat
- Mullender S.J., Leslie I.M., McAuly D. "Operating Support for Distributed Multimedia", Proceedings of Summer Usenix Conference, Boston MA, June 1994.
- Hopper A.: "Pandora, an experimental system for multimedia applications", ACM Operating Systems Review, 24 (2), April 1990, pp. 19-34.
- Massalin H.: "Synthesis: An efficient implementation of fundamental operating system services", Ph.D. Thesis, Columbia University, 1992.
- Hyden E.A.: "Operating System Support for Quality of Service", Ph.D. Thesis, University of Cambridge, Febr. 1994.
- Liu C.L., Layland J.W.: "Scheduling algorithms for multi-processing in a hard-real-time environment", J.Ass. Comput. Mach., Vol. 20, Jan. 1973, pp. 46-61
- A. Clemm: Network Management Fundamentals. CiscoPress, 2006
- Fingerpointing, Frustrated Network Engineers, and the Application Performance Blame Game

Unit - 15 : Windows Operating System - A Case Study

Structure of Unit

15.0 Objective

In this lesson we will look inside what windows operating system is made up of. We will take a sneak peak at the windows architecture in general. We will also see what do we mean by Windows file system. We will then read about FAT and NTFS systems and there comparative analysis.

15.1 Introduction

Windows 7 is built on the Windows Vista core, but Windows 7 has limited the files that load at startup to help with the core performance of the operating system. They have also removed many of the fluff items that Windows Vista used, thus allowing for better performance. When Microsoft first released Windows 7 as a beta, there was a 64-bit version but no 32-bit version. This did not go over well with the Internet bloggers. I even saw a petition online to have a 32-bit version released. The funny thing is that I also saw a petition asking Microsoft not to release a 32-bit version. The logic behind this was it would force users and manufacturers to upgrade everything to 64-bit. Well, Microsoft has released Windows 7 as both a 32-bit and a 64-bit version.

Microsoft could not just release a 64-bit version of Windows 7. This would alienate many

users with 32-bit computer systems, and it would cost Microsoft a large share of the client-side software market. Users already have to deal with the PC vs. Mac commercials! So Windows 7 users have a choice of either 32-bit or 64-bit.

Before going into the architecture in detail, let's see what we mean by 32-bit and 64-bit

15.1.1 32-Bit vs. 64-Bit

When you hear the terms 32-bit and 64-bit, this is referring to the CPU or processor. The number represents how the data is processed. It is processed either as 2 ^ 32 or as 2 ^ 64. The larger the number, the larger the amount of data that can be processed at any one time. Think of a large highway that has 32 lanes. Vehicles can travel on those 32 lanes only. When traffic gets backed up, they can only use these lanes, and this can cause traffic delays. But now think of a 64-lane highway and how many more vehicles can travel on that highway. This is an easy way of thinking of how 32-bit and 64-bit processors operate. The problem here is that if you have a 32-lane highway, you can't just set up 64 vehicles on this highway and let them go. You need to have the infrastructure to allow for 64 vehicles by having 64 lanes. This is the same with computers. Your computer has to be configured to allow you to run a 64-bit processor. So what does all of this mean to the common user or administrator? Well, it's all about RAM. A 32-bit operating system can handle up to 4 GB of RAM and a 64-bit processor can.

15.1.2 Windows 7 Architecture

If you want to truly know how Windows 7 works and what makes it tick, you need to dig under the hood. Windows 7 doesn't boot from an initialization file. Instead, the operating system uses the Windows boot manager to initialize and start the operating system.

The boot environment dramatically changes the way the operating system starts. The boot environment was created by Microsoft to resolve several prickly problems related to boot integrity, operating system integrity, and firmware abstraction. The boot environment is loaded prior to the operating system, making it a pre operating system environment. As such, the boot environment can be used to validate the integrity of the startup process and the operating system itself before actually starting the operating system.

The boot environment is an extensible abstraction layer that allows the operating system to work with multiple types of firmware interfaces without requiring the operating system to be specifically written to work with these firmware interfaces. Rather than updating the operating system each time a new firmware interface is developed, firmware interface developers can use the standard programming interfaces of the boot environment to allow the operating system to communicate as necessary through the firmware interfaces.

Firmware interface abstraction is the first secret ingredient that makes it possible for Windows 7 to work with BIOS-based and EFI-based computers in exactly the same way, and this is one of the primary reasons Windows 7 achieves hardware independence..

The next secret ingredient for Windows 7 hardware independence is Windows Imaging Format (WIM). Microsoft distributes Windows 7 on media using WIM disk images. WIM uses compression and single-instance storage to dramatically reduce the size of image files. Using compression reduces the size of the image in much the same way that Zip compression reduces the size of files. Using single-instance storage reduces the size of the image because only one physical copy of a file is stored for each instance of that file in the disk image.

Because WIM is hardware independent, Microsoft can use a single binary for each supported architecture:

- One binary for 32-bit architectures

- One binary for 64-bit architectures
- One binary for Itanium architectures

The final secret ingredient for Windows 7 hardware independence is modularization. Windows 7 uses modular component design so that each component of the operating system is defined as a separate independent unit or module. Because modules can contain other modules, various major features of the operating system can be grouped together and described independently of other major features. Because modules are independent from each other, modules can be swapped in or out to customize the operating system environment.

Windows 7 includes extensive support architecture. At the heart of this architecture is built-in diagnostics and troubleshooting. Microsoft designed built-in diagnostics and troubleshooting to be self-correcting and self-diagnosing, and failing that, to provide guidance while you are diagnosing problems.

Windows 7 includes network awareness and network discovery features. Network awareness tracks changes in network configuration and connectivity. Network discovery controls a computer's ability to detect other computers and devices on a network.

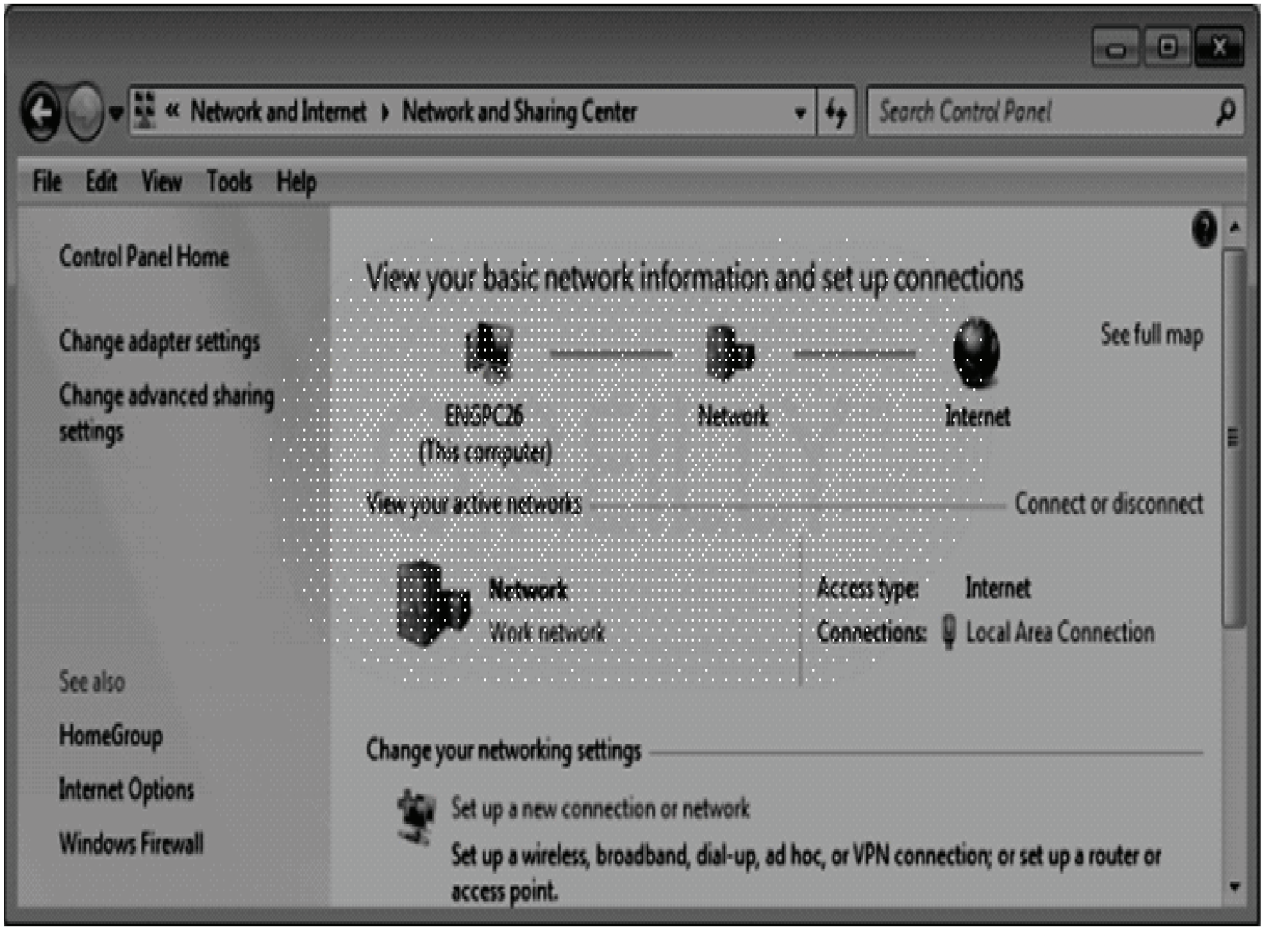
Network awareness allows Windows 7 to detect the current network configuration and connectivity status, which is important because many networking and security settings depend on the type of network to which a computer running Windows 7 is connected. Windows 7 has separate network configurations for domain networks, private networks, and public networks and is able to detect:

- When you change a network connection
- Whether the computer has a connection to the Internet
- Whether the computer can connect to the corporate network over the Internet

Unlike all earlier versions of Windows, Windows Firewall in Windows 7 supports connectivity to multiple networks simultaneously and multiple active firewall profiles. Because of this, the active firewall profile for a connection depends on the type of connection.

If you disconnect a computer from one network switch or hub and plug it into a new network switch or hub, you might inadvertently cause the computer to think it is on a different network, and depending on Group Policy configuration, this could cause the computer to enter a lockdown state in which additional network security settings are applied. As shown in figure, you can view the network connection status in the Network and Sharing Center. In Control Panel, under Network and Internet, click Network and Sharing Center to access this management console.

Windows 7 tracks the identification status of all networks to which the computer has been connected. When Windows 7 is in the process of identifying a network, the Network and Sharing Center shows the Identifying Networks state. This is a temporary state for a network that is being identified. After Windows 7 identifies a network, the network becomes an Identified Network and is listed by its network or domain name in the Network and Sharing Center.



If Windows 7 is unable to identify the network, the network is listed with the Unidentified Network status in the Network and Sharing Center. In Group Policy, you can set default location types and user permissions for each network state, as well as for all networks, by using the policies for Computer Configuration under Windows Settings\Security Settings\Network List Manager Policies.

When you are working with the Network And Sharing Center, you can attempt to diagnose a warning status by using Windows Network Diagnostics—another key component of the diagnostics and troubleshooting framework. To start diagnostics, click the warning icon in the network map or click Troubleshoot Problems, and then click Internet Connections. Windows Network Diagnostics then attempts to identify the network problem and provide a possible solution.

The Windows diagnostics and troubleshooting infrastructure offers improved diagnostics guidance, additional error reporting details, expanded event logging, and extensive recovery policies. Although Windows XP and earlier versions of Windows include some help and diagnostics features, those features are, for the most part, not self-correcting or self-diagnosing. Windows 7, on the other hand, can detect many types of hardware, memory, and performance issues and resolve them automatically or help users through the process of resolving them.

As shown below, Windows diagnostics and troubleshooting features are divided into 15 broad diagnostics areas. In Group Policy, you can configure how these features work by using the Administrative Templates policies for Computer Configuration under System\Troubleshooting and Diagnostics.

15.1.3 Key Diagnostics Areas in Windows 7

DIAGNOSTIC AREA	DESCRIPTION	REQUIREMENTS
Application compatibility	Supports the Program Compatibility Assistant (PCA) for diagnosing drivers blocked due to compatibility issues. PCA can detect failures caused by applications trying to load legacy Windows DLLs or trying to create COM objects that have been removed by Microsoft. PCA can detect several types of application installation failures. These installation failures can be related to applications that do not have privileges to run as an administrator but must be installed with elevated privileges as well as applications that fail to launch child processes that require elevation. In this case, PCA provides you with the option to restart the installer or the update process as an administrator.	Diagnostic Policy Service, Program Compatibility Assistant Service
Boot performance	Supports automatic detection and troubleshooting of issues that affect boot performance. Root causes of boot performance issues are logged to the event logs. Can also assist you in resolving related issues.	Diagnostic Policy Service
Corrupted file recovery	Supports automatic detection, troubleshooting, and recovery of corrupted files. If Windows detects that an important operating system file is corrupted, Windows attempts notification and recovery, which requires a restart in most cases for full resolution.	Diagnostic Policy Service
External support	Supports the Microsoft Support Diagnostic Tool (MSDT) for collecting and sending diagnostic data to a support professional to resolve a problem. Msdt.exe is stored in the %SystemRoot%\System32 folder and through policy settings can be configured for local and remote troubleshooting or remote troubleshooting only.	Diagnostic Policy Service
Fault-tolerant heap	Supports automatic detection and correction of common memory management issues related to the heap used by the operating system.	Diagnostic Policy Service
Memory leak	Supports automatic detection and troubleshooting of memory leak issues. A memory leak occurs if an application or system component doesn't completely free areas of physical memory after it is done with them.	Diagnostic Policy Service

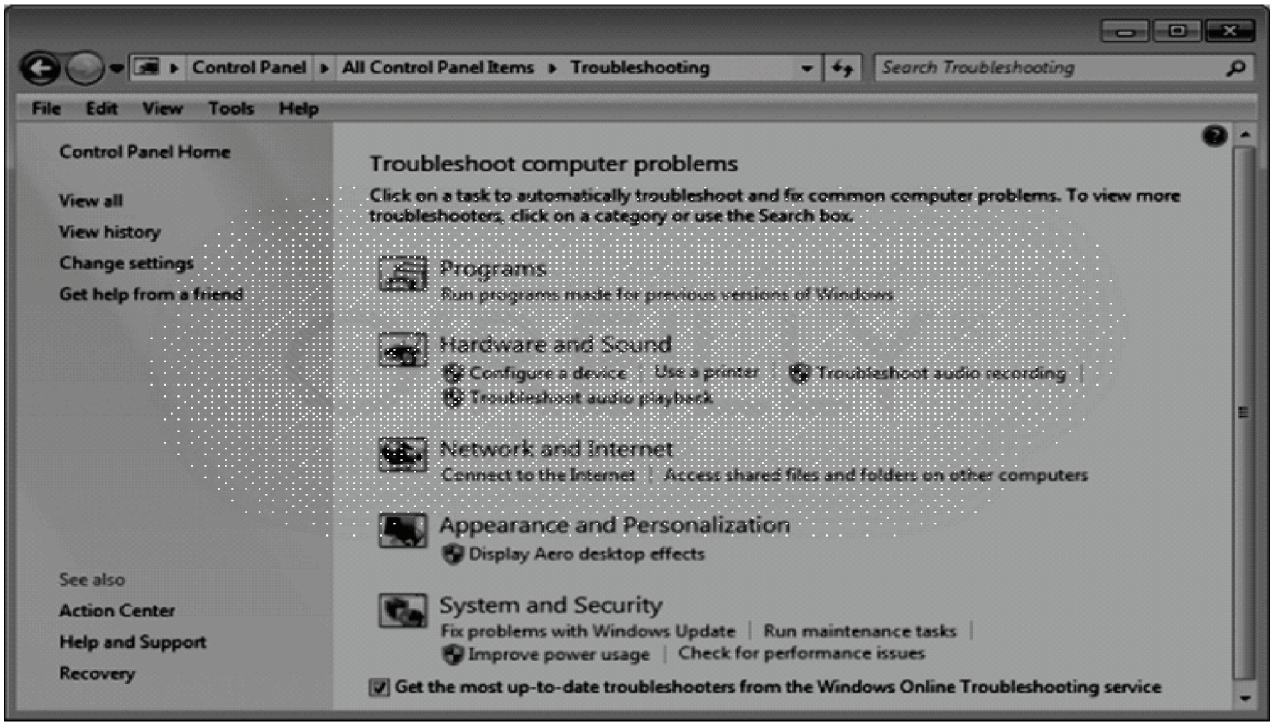
DIAGNOSTIC AREA	DESCRIPTION
MSI corrupted file recovery	Supports automatic detection, troubleshooting, and recovery of corrupted MSI applications. If Windows detects that application files are corrupted, Windows attempts notification and recovery.
Performance PerfTrack	Supports automated tracking and reporting of responsiveness events to Microsoft's Software Quality Management (SQM) team.
Resource exhaustion	Supports automatic detection and troubleshooting to resolve issues related to running out of virtual memory. Can also alert you if the computer is running low on virtual memory and identify the processes consuming the largest amount of memory, allowing you to close any or all of these high-resource-consuming applications directly from the Close Programs To Prevent Information Loss dialog box. An alert is also logged in the event log.
Scheduled maintenance	Supports diagnostics that run periodically via the Task Scheduler to detect and resolve system problems.
Scripted diagnostics	Supports Action Center and controls whether users can access troubleshooting content and troubleshooting tools.
Shutdown performance	Supports automatic detection and troubleshooting of issues that affect shutdown performance. Root causes of shutdown performance issues are logged to the event logs. Can also assist you in resolving related issues.
Standby/resume performance	Supports automatic detection and troubleshooting of issues that affect standby/resume performance on desktop computers. Root causes of standby/resume performance issues are logged to the event logs. Can also assist you in resolving related issues.
System responsiveness	Supports automatic detection and troubleshooting of issues that affect the overall responsiveness of the operating system. Root causes of responsiveness issues are logged to the event logs. Can also assist you in resolving related issues.

Other diagnostics features of Windows 7 include:

- Restart Manager
- Action Center and troubleshooters
- Startup Repair tool
- Performance Diagnostics console
- Windows Memory Diagnostics

In Windows XP and earlier versions of Windows, an application crash or hang is marked as Not Responding, and it is up to the user to exit and then restart the application. Windows 7 attempts to automatically resolve the issues related to unresponsive applications by using Restart Manager. Restart Manager can shut down and restart unresponsive applications automatically. In many cases, this means that you may not have to intervene to try to resolve issues with frozen applications.

A failed installation and nonresponsive conditions of applications and drivers are also tracked through Action Center. Should such an event occur, the Action Center notification icon will show a red circle with an X through it. If you click the notification icon, Windows 7 displays a summary report of current issues. As discussed previously, you can click the link provided to open a possible solution or to get more information. If these processes fail, access the Action Center main window and then scroll down to display the Troubleshooting and Recovery links.



Clicking Troubleshooting opens the Troubleshooting window. As shown in figure above, several troubleshooters are provided. These troubleshooters can help users quickly resolve common problems without requiring administrator support. The troubleshooters include:

- Programs for compatibility issues with applications designed for earlier versions of Windows.
- Hardware And Sound for issues with hardware devices, audio recording, and audio playback.
- Network And Internet for issues with connecting to networks and accessing shared folders on other computers.

- Appearance And Personalization for issues with the display appearance and personalization settings. To quickly resolve display issues with Aero, click Display Aero Desktop Effects.
- System And Security for issues with Windows Update, power usage, and performance. Click Run Maintenance Tasks to clean up unused files and shortcuts and perform other routine maintenance tasks.

To resolve startup problems, Windows 7 uses the Startup Repair tool (StR), which is installed automatically and started when a system fails to boot. After it is started, StR attempts to determine the cause of the startup failure by analyzing startup logs and error reports. Then StR attempts to fix the problem automatically. If StR is unable to resolve the problem, it restores the system to the last known working state and then provides diagnostic information and support options for further troubleshooting.

Startup Repair performs many tests during diagnostics and troubleshooting. These tests can take anywhere from 5 to 30 minutes or more depending on the configured hardware, and they include these specific tests:

- Check for updates determine whether newly applied updates are affecting startup.
- System disk test determines whether there is a problem with the system disk that is preventing startup. If so, StR can attempt to repair any missing or corrupt files.
- Disk failure diagnosis Determines whether any of the configured disks have failed.
- Disk metadata test determines whether any of the available disks have a problem with their metadata that is preventing startup. The metadata associated with a disk depends on how a disk is partitioned and the file system format of disk partitions.
- Target OS test Determines whether the operating system you are attempting to start has a specific issue that is preventing startup.
- Volume content check examines the content of disk volumes to ensure that volumes are accessible.
- Boot manager diagnosis Determines whether there is a problem with the boot manager or boot manager entries that are preventing startup.
- System boot log diagnosis Examines system boot log entries from previous startups to see if there are specific errors that might be related to the startup issue.
- Event log diagnosis Examines event log entries to see if there are specific errors that might be related to the startup issue.
- Internal state check Checks the current internal state of the preboot environment.
- Boot status test Checks the current boot status in the preboot environment.
- Setup state check Determines whether the computer is in a setup state.
- Registry hives test Checks the computer's registry hives.
- Windows boot log diagnosis Examines the Windows boot log entries to see if there are specific errors that might be related to the startup issue.
- Bug check analysis performs a basic bug check analysis of the operating system.
- Access control test determines whether access controls in the preboot environment are preventing startup of the operating system.

- File system test (chkdsk) performs a basic file system test using Chkdsk.
- Software install log diagnosis Examines software installation log entries to see if there are specific errors that might be related to the startup issue.
- Fallback diagnosis determines whether any flags have been set that indicate the computer should fall back to a previous state to correct the startup issue. If so, StR will attempt to restore the previous state.

Error detection for devices and failure detection for disk drives also is automated. If a device is having problems, hardware diagnostics can detect error conditions and either repair the problem automatically or guide the user through a recovery process. With disk drives, hardware diagnostics can use fault reports provided by disk drives to detect potential failure and alert you before this happens. Hardware diagnostics can also help guide you through the backup process after alerting you that a disk might be failing.

Windows 7 can automatically detect performance issues, which include slow application startup, slow boot, slow standby/resume, and slow shutdown. If a computer is experiencing degraded performance, Windows diagnostics can detect the problem and provide possible solutions. For advanced performance issues, you can track related performance and reliability data in the Performance Monitor console, which can be opened from the Administrative Tools menu.

Windows 7 can also detect issues related to memory leaks and failing memory. If you suspect that a computer has a memory problem that is not being automatically detected, you can run Windows Memory Diagnostics manually by completing the following steps:

1. Click Start, type mdsched.exe in the Search box, and then press Enter.
2. Choose whether to restart the computer and run the tool immediately or schedule the tool to run at the next restart.
3. Windows Memory Diagnostics runs automatically after the computer restarts and performs a standard memory test. If you want to perform fewer or more tests, press F1, use the Up and Down Arrow keys to set the Test Mix as Basic, Standard, or Extended, and then press F10 to apply the desired settings and resume testing.
4. When testing is completed, the computer restarts. You'll see the test results when you log on.

If a computer crashes because of failing memory and Memory Diagnostics detects this, you are prompted to schedule a memory test the next time the computer is started.

15.2 Windows File System

A file system (or filesystem) is a means to organize data expected to be retained after a program terminates by providing procedures to store, retrieve and update data as well as manage the available space on the device(s) which contain it. A file system organizes data in an efficient manner and is tuned to the specific characteristics of the device. A tight coupling usually exists between the operating system and the file system. Some file systems provide mechanisms to control access to the data and metadata. Ensuring reliability is a major responsibility of a file system. Some file systems allow multiple programs to update the same file at nearly the same time.

File systems are used on data storage devices, such as hard disk drives, floppy disks, optical discs, or flash memory storage devices, to maintain the physical locations of the computer files. They may provide access to data on a file server by acting as clients for a network protocol (e.g. NFS, SMB, or 9P clients), or they

may be virtual and exist only as an access method for virtual data (e.g. procfs). This is distinguished from a directory service and registry.

15.2.1 Microsoft Windows

Windows makes use of the FAT, NTFS, exFAT and ReFS file systems (the latter is only supported and usable in Windows Server 8; Windows cannot boot from it).

Windows uses a drive letter abstraction at the user level to distinguish one disk or partition from another. For example, the path C:\WINDOWS represents a directory WINDOWS on the partition represented by the letter C. Drive C: is most commonly used for the primary hard disk partition, on which Windows is usually installed and from which it boots. This "tradition" has become so firmly ingrained that bugs came about in older applications which made assumptions that the drive that the operating system was installed on was C. The use of drive letters, and the tradition of using "C" as the drive letter for the primary hard disk partition, can be traced to MS-DOS, where the letters A and B were reserved for up to two floppy disk drives. This in turn derived from CP/M in the 1970s, and ultimately from IBM's CP/CMS of 1967.

```
C:\Temp> dir
 Volume in drive C is C
 Volume Serial Number is 74F5-B93C

 Directory of C:\Temp

2009-08-25  11:59    <DIR>          .
2009-08-25  11:59    <DIR>          ..
2007-03-01  11:37         2,321,600 AdobeUpdater12345.exe
2009-04-03  10:01            27,988 dd_depcheckdotnetfx30.txt
2009-04-03  10:01               764 dd_dotnetfx3error.txt
2009-04-03  10:01            32,572 dd_dotnetfx3install.txt
2009-06-09  13:46            35,145 GenProfile.log
2009-08-05  12:11               155 KB969856.log
2009-04-20  08:37               402 MSI29e0b.LOG
2009-04-09  16:34            38,895 offcln11.log
2009-04-03  16:02    <DIR>          OfficePatches
2009-07-14  14:30    <DIR>          OHotfix
2009-08-25  10:52            16,384 Perflib_Perfdata_c30.dat
2009-04-03  10:01             1,744 uxeventlog.txt
2009-08-25  11:42        50,245,632 WFV2F.tmp
2009-04-20  10:07             1,397 {AC76BA86-7AD7-1033-7B44-A81200000003}.ini
2009-04-20  10:13               617 {AC76BA86-7AD7-1033-7B44-A81300000003}.ini
              13 File(s)     52,723,295 bytes
               4 Dir(s)  83,570,208,768 bytes free
```

15.3 FAT

File Allocation Table (FAT) is the name of a computer file system architecture and a family of industry standard files systems utilizing it.

The FAT file system is a legacy file system which is simple and robust. It offers good performance even in light-weight implementations, but cannot deliver the same performance, reliability and scalability as some modern file systems. It is however supported for compatibility reasons by virtually all existing operating systems for personal computers, and thus is a well-suited format for data exchange between computers and devices of almost any type and age from the early 1980s up to the present.

Originally designed in the late 1970s for use on floppy disks, it was soon adapted and used almost universally on hard disks throughout the DOS and Windows 9x eras for two decades. With the introduction of more powerful computers and operating systems, and the development of more complex file systems for them, it is no longer the default file system for usage on hard drives by most modern desktop operating systems.

Today, FAT file systems are still commonly found on floppy disks, solid-state memory cards, flash memory cards, and on many portable and embedded devices. It is also utilized in the boot stage of EFI-compliant computers.

The name of the file system originates from the file system's prominent usage of an index table, the FAT, statically allocated at the time of formatting. The table contains entries for each cluster, a contiguous area of disk storage. Each entry contains either the number of the next cluster in the file, or else a marker indicating end of file, unused disk space, or special reserved areas of the disk. The root directory of the disk contains the number of the first cluster of each file in that directory; the operating system can then traverse the FAT table, looking up the cluster number of each successive part of the disk file as a cluster chain until the end of the file is reached. In much the same way, sub-directories are implemented as special files containing the directory entries of their respective files.

As disk drives have evolved, the maximum number of clusters has significantly increased, and so the number of bits used to identify each cluster has grown. The successive major versions of the FAT format are named after the number of table element bits: 12 (FAT12), 16 (FAT16), and 32 (FAT32). Each of these variants is still in use. The FAT standard has also been expanded in other ways while generally preserving backward compatibility with existing software.

A FAT file system is composed of four different sections:

- The Reserved sectors, located at the very beginning.

The first reserved sector (logical sector 0) is the Boot Sector (aka Volume Boot Record (VBR)). It includes an area called the BIOS Parameter Block (with some basic file system information, in particular its type, and pointers to the location of the other sections) and usually contains the operating system's boot loader code.

Important information from the Boot Sector is accessible through an operating system structure called the Drive Parameter Block (DPB) in DOS and OS/2. The total count of reserved sectors is indicated by a field inside the Boot Sector, and is usually 32 on FAT32 file systems.

For FAT32 file systems, the reserved sectors include a File System Information Sector at logical sector 1 and a Backup Boot Sector at logical sector 6. While many other vendors have continued to utilize a single-sector setup (logical sector 0 only) for the bootstrap loader, Microsoft's boot sector code has grown to spawn over logical sectors 0 and 2 since the introduction of FAT32, with logical sector 0 depending on sub-routines in logical sector 2. The Backup Boot Sector area consists of three logical sectors 6, 7, and 8 as well. In some cases, Microsoft also uses sector 12 of the reserved sectors area for an extended boot loader.

- **The FAT Region.**

This typically contains two copies (may vary) of the File Allocation Table for the sake of redundancy checking, although rarely used, even by disk repair utilities.

These are maps of the Data Region, indicating which clusters are used by files and directories. In FAT12 and FAT16 they immediately follow the reserved sectors. Typically the extra copies are kept in tight synchronization on writes, and on reads they are only used when errors occur in the first FAT. In FAT32, it is possible to switch from the default behavior and select a single FAT out of the available ones to be used for diagnosis purposes.

The first two clusters (cluster 0 and 1) in the map contain special values.

- **The Root Directory Region.**

This is a Directory Table that stores information about the files and directories located in the root directory. It is only used with FAT12 and FAT16, and imposes on the root directory a fixed maximum size which is pre-allocated at creation of this volume. FAT32 stores the root directory in the Data Region, along with files and other directories, allowing it to grow without such a constraint. Thus, for FAT32, the Data Region starts here.

- **The Data Region.**

This is where the actual file and directory data is stored and takes up most of the partition. Traditionally, the unused parts of the data region are initialized with a filler value of 0xF6 as per the INT 1Eh's Disk Parameter Table (DPT) during format on IBM compatible machines, but also used on the Atari Portfolio. 8-inch CP/M floppies typically came pre-formatted with a value of 0xE5; by way of Digital Research this value was also used on Atari ST formatted floppies. Amstrad used 0xF4 instead. Some modern formatters wipe hard disks with a value of 0x00, whereas a value of 0xFF, the default value of a non-programmed flash block, is used on flash disks to reduce wear. The latter value is typically also used on ROM disks. (Some advanced formating tools allow to configure the format filler byte.)

The size of files and subdirectories can be increased arbitrarily (as long as there are free clusters) by simply adding more links to the file's chain in the FAT. Note however, that files are allocated in units of clusters, so if a 1 KB file resides in a 32 KB cluster, 31 KB are wasted.

FAT32 typically commences the Root Directory Table in cluster number 2: the first cluster of the Data Region.

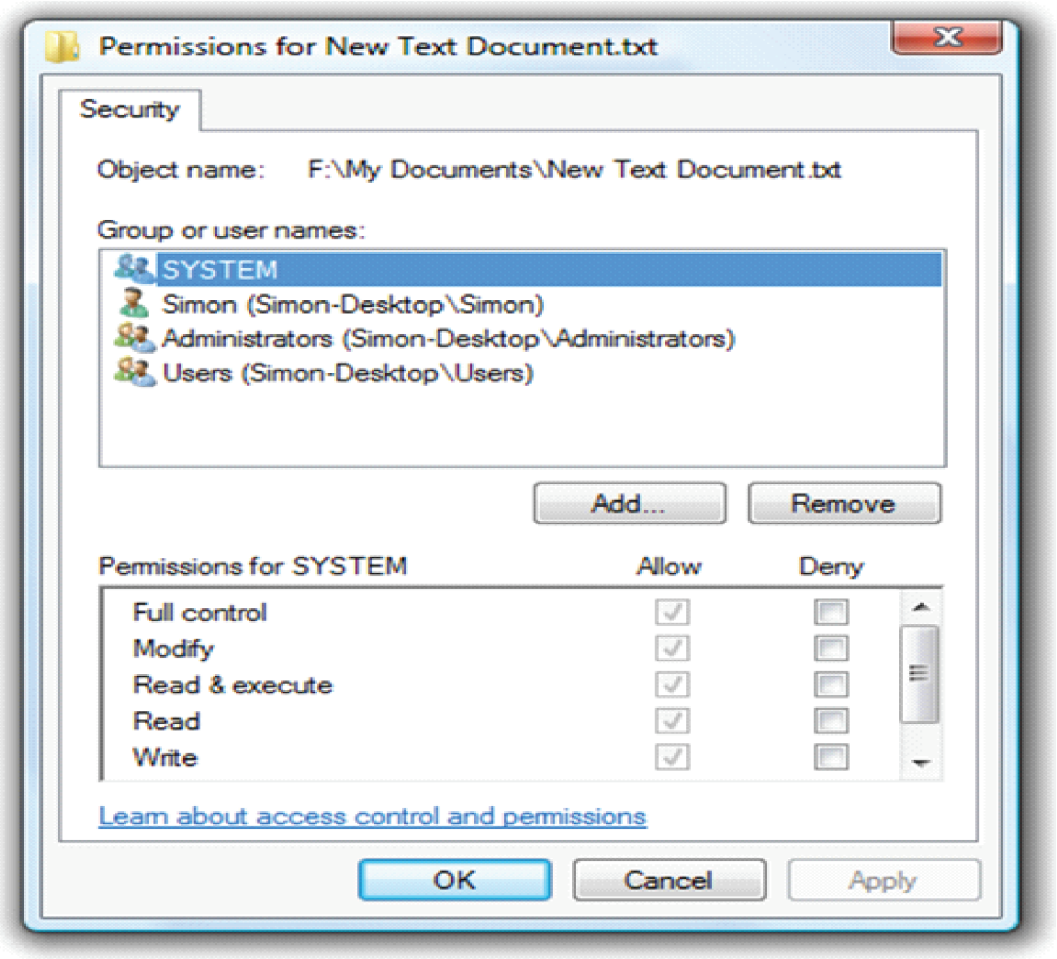
FAT uses little-endian format for all entries in the header (except for, where explicitly mentioned, for some entries on Atari ST boot sectors) and the FAT(s). It is possible to allocate more FAT sectors than necessary for the number of clusters. The end of the last FAT sector can be unused if there are no corresponding clusters. The total number of sectors (as noted in the boot record) can be larger than the number of sectors used by data (clusters × sectors per cluster), FATs (number of FATs × sectors per FAT), and hidden sectors including the boot sector: this would result in unused sectors at the end of the volume. If a partition contains more sectors than the total number of sectors occupied by the file system it would also result in unused sectors at the end of the volume.

15.4 NTFS

NTFS (New Technology File System) is a proprietary file system developed by Microsoft Corporation for its Windows line of operating systems, beginning with Windows NT 3.1 and Windows 2000, including Windows XP, Windows Server 2003, and all their successors to date.

NTFS supersedes the FAT file system as the preferred file system for Microsoft's Windows operating systems. NTFS has several improvements over FAT and HPFS (High Performance File System), such as improved support for metadata, and the use of advanced data structures to improve performance, reliability, and disk space utilization, plus additional extensions, such as security access control lists (ACL) and file system journaling.

15.4.1 Internals



NTFS filesystem permissions on a Windows Vista system.

In NTFS, all file data—file name, creation date, access permissions (by the use of access control lists), and contents—are stored as metadata in the Master File Table. This abstract approach allowed easy addition of file system features during Windows NT's development—an interesting example is the addition of fields for indexing used by the Active Directory software.

NTFS allows any sequence of 16-bit values for name encoding (file names, stream names, index names, etc.). This means UTF-16 codepoints are supported, but the file system does not check whether a sequence is valid UTF-16 (it allows any sequence of short values, not restricted to those in the Unicode standard).

Internally, NTFS uses B+ trees to index file system data. Although complex to implement, this allows faster file look up times in most cases. A file system journal is used to guarantee the integrity of the file system metadata but not individual files' content. Systems using NTFS are known to have improved reliability compared to FAT file systems.

The Master File Table (MFT) contains metadata about every file, directory, and metafile on an NTFS volume. It includes filenames, locations, size, and permissions. Its structure supports algorithms which minimize disk fragmentation. A directory entry consists of a filename and a "file ID", which is the record number representing the file in the Master File Table. The file ID also contains a reuse count to detect stale references. While this strongly resembles the W_FID of Files-11, other NTFS structures radically differ.

15.4.2 Limitations

The following are a few limitations of NTFS:

- **Compression:**

  The compression algorithms in NTFS are designed to support cluster sizes of up to 4 kB. When the cluster size is greater than 4 kB on an NTFS volume, NTFS compression is not available.

- **Maximum cluster size:**

  The maximum cluster size is 64 kB.

- **File names:**

  File names are limited to 255 UTF-16 code points. Certain names are reserved in the volume root directory and cannot be used for files. These are $MFT, $MFTMirr, $LogFile, $Volume, $AttrDef, . (dot), $Bitmap, $Boot, $BadClus, $Secure, $Upcase, and $Extend. (dot) and $Extend are both directories; the others are files. The NT kernel limits full paths to 32,767 UTF-16 code points.

- **Maximum volume size:**

  In theory, the maximum NTFS volume size is 264"1 clusters. However, the maximum NTFS volume size as implemented in Windows XP Professional is 232"1 clusters partly due to partition table limitations. For example, using 64 kB clusters, the maximum Windows XP NTFS volume size is 256 TBs minus 64 KBs. Using the default cluster size of 4 kB, the maximum NTFS volume size is 16 TB minus 4 kB. (Both of these are vastly higher than the 128 GB limit lifted in Windows XP SP1.) Because partition tables on master boot record (MBR) disks only support partition sizes up to 2 TB, dynamic or GPT volumes must be used to create NTFS volumes over 2 TB. Booting from a GPT volume to a Windows environment requires a system with UEFI and 64-bit support.

- **Maximum file size:**

  As designed, the maximum NTFS file size is 16 EB (16 × 10246 bytes) minus 1 kB or 18,446,744,073,709,550,592 bytes. As implemented, the maximum NTFS file size is 16 TB minus 64 kB or 17,592,185,978,880 bytes.

- **Alternate data streams:**

  Windows system calls may handle alternate data streams. Depending on the operating system, utility and remote file system, a file transfer might silently strip data streams. A safe way of copying or moving files is to use the BackupRead and BackupWrite system calls, which allow programs to enumerate streams, to verify whether each stream should be written to the destination volume and to knowingly skip unwanted streams.

15.4.3 Comparison between NTFS and FAT

NTFS	FAT 16/32
• Default File system In Windows XP, 2k and NT • Support For Drives over 40gb, Files over GB • Allows extended file names, foreign characters • Has a severely crippled maintenance system in chkdsk • Chkdsk is notoriously slow • Increased security with file encryption • Smaller file clusters, 4kb • Compression to reduce disk space • User permissions for files and folders • File copies are "undone" if interrupted, cluster chains is cleaned • Small files are kept in Master File Table at the beginning of the drive • Not compatible with different operating systems on the same computer	• Fat 16 not compatible with XP, FAT is more compatible with other operating Systems(Windows 95, etc) • FAT 16 has 8.3 character limitation • Has better, more and interactive recovery utilities (scandisk) • Scandisk is very quick • Just a space for the OS to read files • Faster on drives less than 10gb • FAT 16 cluster size is 32kb • Cluster chains containing data from interrupted copies are marked as damaged • Master File Table are separate from files

15.4.4 Some Random Facts

- Fat 16 was developed in 1981 for dos
- Fat 16 was designed to handle floppies
- Fat 32 is an extension of Fat 16
- Fat 32 introduced in service pack 2 of Windows 95
- Operating systems may recognise Fat16, but not Fat 32 (Win NT)
- You can go from FAT to NTFS but not the other way around
- FAT = File Allocation Table
- NTFS = New Technology File System

15.4.5 NTFS or FAT?

If you really only want to choose one way or another, here are two very important considerations:

- For files above 4gb, and hard disks above 32gb, go for NTFS
- For smaller drives, files and better recovery tools go for FAT
- But why not go for both, which is the best option in my opinion.
- Set aside some FAT so that you can run recovery tools, especially scandisk, so that you have something usable when things go awry, instead of the awful Windows System Tools.

- Then set the rest to NTFS so that you have better security on personal files, support for large files and drive.

15.5 More Detailed Comparison

Criteria	NTFS	FAT32	FAT16
Operating System	Windows NT Windows 2000 Windows XP	Windows 98 Windows ME Windows 2000 Windows XP	DOS All versions of Microsoft Windows
Limitations			
Max Volume Size	2TB	2TB	2GB
Max Files on Volume	Nearly Unlimited	Nearly Unlimited	~65000
Max File Size	Limit Only by Volume Size	4GB	2GB
Max Clusters Number	Nearly Unlimited	268435456	65535
Max File Name Length	Up to 255	Up to 255	Standard - 8.3 Extended - up to 255
File System Features			
Unicode File Names	Unicode Character Set	System Character Set	System Character Set
System Records Mirror	MFT Mirror File	Second Copy of FAT	Second Copy of FAT
Boot Sector Location	First and Last Sectors	First Sector	First Sector
File Attributes	Standard and Custom	Standard Set	Standard Set
Alternate Streams	Yes	No	No
Compression	Yes	No	No
Encryption	No	No	No
Object Permissions	Yes	No	No
Disk Quotas	No	No	No
Sparse Files	No	No	No
Reparse Points	No	No	No
Volume Mount Points	No	No	No
Overall Performance			
Built-In Security	Yes	No	No
Recoverability	Yes	No	No
Performance	Low on small volumes High on Large	High on small volumes Low on large	Highest on small volumes Low on large
Disk Space Economy	Max	Average	Minimal on large volumes
Fault Tolerance	Max	Minimal	Average

15.6 Summary

- Windows 7 is built on the Windows Vista core, but Windows 7 has limited the files that load at startup to help with the core performance of the operating system
- When you hear the terms 32-bit and 64-bit, this is referring to the CPU or processor. The number represents how the data is processed
- Windows 7 doesn't boot from an initialization file. Instead, the operating system uses the Windows boot manager to initialize and start the operating system

- Windows 7 includes extensive support architecture.
- Windows 7 includes network awareness and network discovery features
- A file system (or filesystem) is a means to organize data expected to be retained after a program terminates by providing procedures to store, retrieve and update data as well as manage the available space on the device(s) which contain it.
- File Allocation Table (FAT) is the name of a computer file system architecture and a family of industry standard files systems utilizing it.
- NTFS supersedes the FAT file system as the preferred file system for Microsoft's Windows operating system.

15.7 Self - Assessment Exercise

1. What do you mean by Windows Architecture?
2. What are differences between 32-bit and 64-bit architectures?
3. Explain Key Diagnostics Areas in Windows 7.
4. What is File operating system? Explain.
5. What is a Fat system? Where it is used?
6. What is a NTFS system? Where it is used?
7. Explain limitations of NTFS system.
8. Explain differences between FAT and NTFS file systems.

15.8 References

- O'reilly answers by adfm
- "Windows XP: Format backup drives using NTFS". Microsoft. September 7, 2006. http://www.microsoft.com/windowsxp/using/setup/tips/advanced/ntfs.mspx.
- Mark Russinovich. "Inside Win2K NTFS, Part 1". Microsoft Developer Network. http://msdn2.microsoft.com/en-us/library/ms995846.aspx. Retrieved 2008-04-18.
- Microsoft TechNet (2003-03-28). "How NTFS Works". Windows Server 2003 Technical Reference. http://technet.microsoft.com/en-us/library/cc781134(v=ws.10).aspx. Retrieved 2011-09-12.
- Richard Russon and Yuval Fledel. "NTFS Documentation". http://dubeyko.com/development/FileSystems/NTFS/ntfsdoc.pdf. Retrieved 2011-06-26.
- Rick Vanover. "Windows Server 8 data deduplication". http://www.techrepublic.com/blog/datacenter/windows-server-8-data-deduplication-what-you-need-to-know/4887. Retrieved 2011-12-02.
- Custer, Helen (1994). Inside the Windows NT File System. Microsoft Press. ISBN 978-1-55615-660-1.