NETWORK APPLICATION SERVER

PROJECT WORK DONE AT
E-BRAHMA TECHNOLOGIES PVT. LTD.,
PEELAMEDU,
COIMBATORE - 641 004.

PROJECT REPORT P- 814

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE AWARD OF THE DEGREE OF
M.Sc [APPLIED SCIENCE] SOFTWARE ENGINEERING
OF BHARATHIAR UNIVERSITY, COIMBATORE.

SUBMITTED BY

SARAVANAN.P.R
REG NO. **9937S0091**

UNDER THE GUIDENECE OF

External Guide	Internal guide
Miss.Vidya E-BRAHMA TECHNOLOGIES PVT. LTD., Coimbatore - 4	Mr.S.Ganesh Babu Dept. Of CSE. Coimbatore - 6

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
KUMARAGURU COLLEGE OF TECHNOLOGY
COIMBATORE - 641 006
MAY 2002 - AUG 2002

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY

(Affiliated to Bharathiar University)

COIMBATORE - 641 002

SEPTEMBER - 2002

CERTIFICATE

This is to certify that the project entitled

NETWORK APPLICATION SERVER

DONE BY

SARAVANAN.P.R

REG NO. **9937S0091**

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF **M.Sc [Applied science] SOFTWARE ENGINEERING** OF BHARATHIYAR UNIVERSITY

[illegible] 27/9/02

Professor and HOD

[illegible] 26/09/2002

Internal Guide

Submitted to University Examination held on 26/9

[illegible] 26/9

Internal Examiner

[illegible]

External Examiner

e-Brahma - A Center for IT Excellence Private Limited
PSG STEP, Software Park II
Peelamedu
Coimbatore-641 004, India
Tel:+91 (422) 593020, 592403
Telefax:+91 (422) 592403
e-mail : nsridar@eth.net

3 September, 2002

TO WHOMSOEVER IT MAY CONCERN

This is to certify that the Mr.P.R.Saravanan pursuing fourth year M.Sc Software Engineering at Kumaraguru College of Technology, Coimbatore, has sucessfully completed his project titled "**Network Application Server**" in the area of Networking. The duration of their project was from May 2002 till August 2002.

During this period, we found him to be sincere and hard working.

With regards

[illegible]

Sridar Natarajan
(Chief Executive Officer)

ACKNOWLEDGEMENT

My debts for Assistance in preparing this project Report are more numerous that can be identified here.

I express my sincere thanks to **Prof.Dr.S.Thangasamy B.E. (Hons), Ph.D.,** Head of the Department, Computer Science and Engineering for his valuable suggestions and advice.

I extend my sincere thanks to our course coordinator **Mrs.S.Devaki M.S** for providing constant support and taking great interest in the welfare of the students.

I am immensely thankful to my guide **Mr.S.Ganesh Babu M.C.A.** for the valuable guidance and support throughout my project.

My sincere thanks to, **Mr.V.Paramamsivam**, Chairman and Director, e-Brahma Technologies, Coimbatore.

I express my gratitude to **Miss.Vidya**, my internal guide for her kind help and valuable guidance in completing the project.

My sincere thanks to my parents who contributed immeasurably for shaping me into the computer profession and I also thank my friends for guiding me from behind.

CONTENTS

	TITLE	Page NO.
	SYNOPSIS	i
1	INTRODUCTION	01
	1.1 Purpose	
	1.2 Scope	
	1.3 Definitions & Acronyms	
	1.4 Outline of Report	
2	SYSTEM STUDY	06
	2.1 Introduction	
	2.2 Existing System	
	2.3 Proposed System	
	2.4 System Model	
	2.5 Communication Protocols	
	2.6 Sockets	
	2.7 Process Model	
	2.8 Server Model	
	2.9 Inter Process Communication	
3	REQUIREMENT ANALYSIS	17
	3.1 Functional Requirements	
	3.2 Hardware and Software Requirements	
	3.3 Resource Requirements	
	3.4 Testing Requirements	

4	SYSTEM ANALYSIS	25
	4.1 Project Phases	
	4.2 Constraints, Dependencies and Assumptions	
	4.3 Software Process Model	
5	SYSTEM DESIGN	28
	5.1 Design Constraints	
	5.2 Functional Model	
	5.3 Modular Design	
	5.4 Logical Data Design	
	5.5 Process Design	
6	IMPLEMENTATION ISSUES	66
	6.1 Data Structures	
	6.2 Implementation Description	
7	TESTING TECHNIQUES	69
	7.1 White-Box Testing	
	7.2 Black-Box Testing	
8	TESTING STRATEGIES	72
	8.1 Unit Testing	
	8.2 System Testing	
	8.2.1 Security testing	
	8.2.2 Stress testing	
9	FUTURE ENHANCEMENTS	76
	CONCLUSION	77
	BIBLIOGRAPHY	78

SYNOPSIS

The project entitled as **"NETWORK APPLICATION SERVER"** is being Carried out in **e-Brahma Technologies (P) Ltd.**, Coimbatore.

This project consist of two applications **Enhanced Mail Server** and **Distributed Application Management** .The main objective of the first application is to get the services provided by any contemporary mail service provider on the Internet. The main Objective of the second application is to get the services provided by an application Present in remote systems.

Mail server Application aims at implementation of the Simple Mail Transfer Protocol and Post Office Protocol 3 with meticulous attention to the standards followed which makes the application server compatible with all coeval mail servers. Distributed Application Management aims at fetching the required application from any system in the Network.

The two applications are primarily a system with client-server architecture, which enables communication between the server and a number of clients on the same network at the same time.

The project is implemented using C language in Linux platform, which incorporates several networking features.

CHAPTER 1

1. INTRODUCTION

The first application Enhances Mail Server is basically like as all other Internet Mail Servers deals with transfer of mail between various users with some newly defined features.

Windows provides transparency to the user by making the application present in a remote system to be present in the local system. The second Application "Distributed Application Management " is basically such type of utility program for Linux environment that provides the facility to the user to invoke any application program present in the network.

1.1 Purpose:

Application 1:

The first application in this project concerned with the transfer of mail from one user to another and to read the mails received and to know the status of the mails send by the user and to schedule every users address book.

Application 2:

The second application in this project concerned with the usage of applications present in remote systems, which is connected by network.

1.2 Scope

The both applications are developed such that it fits to the Linux network environment. Both applications demand the use of communication protocols such as TCP/IP for connection -oriented data transfer and demand the use of sockets.

Application 1:

Context:

It deals with transfer of mail between the various users in the network. This involves the usage of the various RFC standards and Protocols for transferring of mail and for reading them. It demands the use of threads for connecting the server with the user and run multiple users at he same time.

Information Objectives:

The primary objective of this application is to provide the client with the mail transfer and mail reading facility.

Function and Performance:

SMTP Client -Server used to send mail and POP Client-Server used to receive the mail.

Application 2:

Context:

The second application in this project deals with using application cross systems. It demands process for connecting the server with the users and to run multi-users at the same time.

Information Objectives:

The requested application is the customer-visible data object and it is invoked automatically. The user needs to input the name of the application as the input data object.

Function and Performance:

The software searches for the specified application over the network, transfers it to the host system and invokes it using the overlay concept.

1.3 Definitions, Acronyms and Abbreviations Protocol:

A set of rules that governs the operation of functional units such as system, terminals, etc. to achieve communication.

Connection-oriented data transfer:

A protocol for exchanging data in which a logical connection is established between the end points (e.g., virtual circuit).

Socket:

A socket is a data structure maintained by a BSD-Unix system to handle network connections. A socket is created using the system call ``socket''. It returns an integer that is like a file descriptor: it is an index into a table and ``read'' and ``write'' to the network using this socket file descriptor.

System Call:

The direct entry points provided by the kernel through which an active process can obtain services from the kernel.

Inter Process Communication (IPC):

The facilities provided by the operating system for two processes to communicate with each other (pipes, FIFO, message queues, semaphores, shared memory).

IP Address:

Internet Protocol Address (a 32-bit address).

TCP:

Transmission Control Protocol.

1.4 OUTLINE OF REPORT

The chapter 2 provides detailed study on the project, its process model and networking concepts. The requirements and analysis is discussed in detail in chapters 3 and 4. Chapter 5 gives the logical and process design along with its constraints. The implementation issues are dealt in chapter 6. Chapter 7 and 8 provides information about testing methods and strategies. Chapter 9 deals with future enhancements of the project.

CHAPTER 2

SYSTEM STUDY

2.1 Introduction

Application 1:

In order to develop the first application, the knowledge about the existing SMTP and POP3 systems is needed and also it is important to know the hashing technique, threads, knowledge of structures and files in Linux are needed.

Application 2:

For developing the second application knowledge about data structures like circular queue and trees, process are needed.

Both applications require the knowledge of sockets in Linux platform and communication protocol used in networking. This chapter provides a detailed account on the above topics

2.2 Existing System

Application 1:

The existing system provides the user options to read and write mails from his user login and it checks for the provides the users with options to save their friends and relatives details in their address book. The system also sends a fail report if a mail does not get delivered. The system also maintains the details of the messages of every individual user separately.

Application 2:

The existing system includes a remote application user for windows environment. The applications are transported through the network drive linked to the neighborhood computers present in the network. The client can use the application only if it is present in the server.

2.3 Proposed System

Application 1:

The proposed system is for Linux environment where the users are provided with all the basic options of the existing system along with certain new options and services. The clients are provided an option to schedule their friends and relatives birthday and anniversary day and inform the user about it a day before. Every user in the system is provided an option that helps one to know whether the receiver has read the mail sent by him. It also concerned the indication of whether any mails received from important persons.

Application 2:

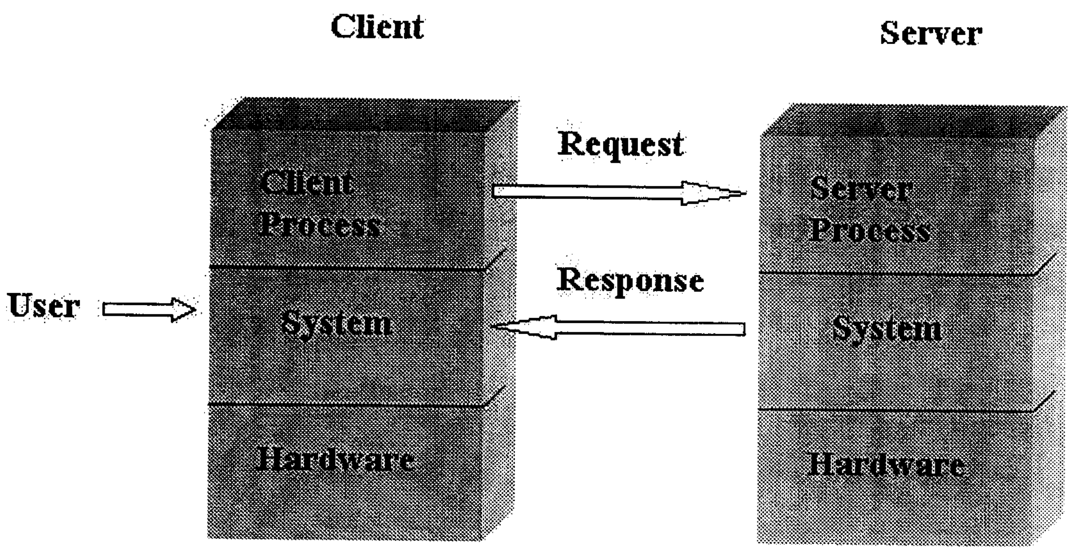
The proposed system is for Linux environment where a client can make request to the concurrent server, which transport the application to the client. The client can make use of the existing application in any system connected to the network.

2.4 System Model

The system model indicated the way in which the different entities within the system are setup. Networking system model has to be used for supporting network access. The different system models that are used by many applications are

- ❖ Peer - to - peer
- ❖ Filter
- ❖ Client - server

Client-server model architecture is described in the following figure:



2.5 Communication Protocols

Open System Interconnection (OSI) and TCP/IP protocol suite are the two important communication protocols. TCP/IP is now widely employed than OSI because of many reasons. The main reason is incomplete implementation of the OSI protocol. The detailed account on TCP/IP is given below.

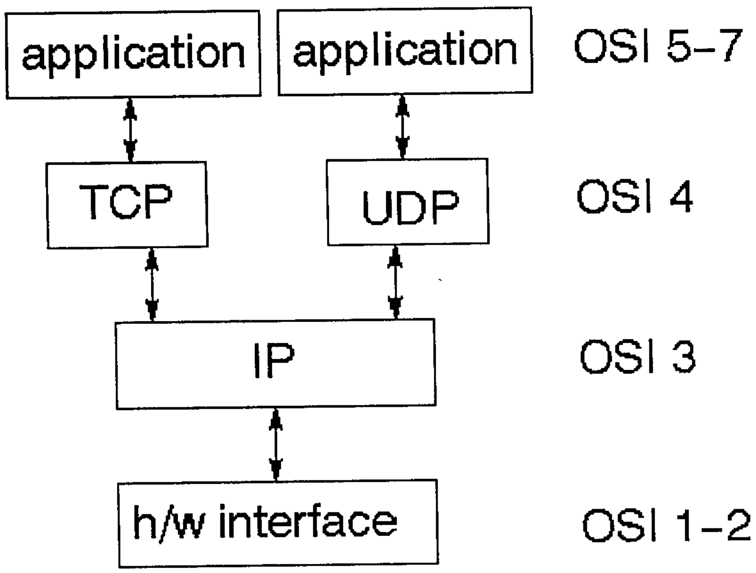
TCP/IP

The OSI model was devised using a committee process wherein the standard was set up and then implemented. Some parts of the OSI standard are obscure, some parts cannot easily be implemented, and some parts have not been implemented.

The TCP/IP protocol was devised through a long-running DARPA project. This worked by implementation followed by RFCs (Request For Comment). TCP/IP is the principal Unix networking protocol.

TCP/IP Stack

The TCP/IP stack is shorter than the OSI one:



P-814

Fig. 2.2 TCP / IP Stack

TCP is a connection-oriented protocol and UDP (User Datagram Protocol) is a connectionless protocol.

IP Data grams

The IP layer provides a connectionless and unreliable delivery system. It considers each datagram independently of the others. The higher layers must supply any association between datagrams.

The IP layer supplies a checksum that includes its own header. The header includes the source and destination addresses.

The IP layer handles routing through an Internet. It is also responsible for breaking up large datagrams into smaller ones for transmission and reassembling them at the other end.

UDP

UDP is also connectionless and unreliable. What it adds to IP is a checksum for the contents of the datagram and port numbers. These are used to give a client/server model.

TCP

TCP supplies logic to give a reliable connection-oriented protocol above IP. It provides a virtual circuit that two processes can use to communicate.

Internet addresses

In order to use a service you must be able to find it. The Internet uses an address scheme for machines so that they can be located. The address is a 32-bit integer, which gives the IP address. This encodes a network ID and more addressing. The network ID falls into various classes according to the size of the network address.

Network address

Class A use 8 bits for the network address with 24 bits left over for other addressing. Class B uses 16-bit network addressing. Class C use 24-bit network addressing and class D uses all 32.

Subnet address

Internally, the Linux network is divided into sub networks. Building 11 is currently on one sub network and uses 10-bit addressing, allowing 1024 different hosts.

Host address

8 bits are finally used for host addresses within out subnet. This places a limit of 256 machines that can be on the subnet.

Total address

The 32-bit address is usually written as 4 integers separated by dots

Symbolic names

Each host has a name. This can be found from the Unix user level command "root@localhost$hostname".

Port addresses

A service exists on a host, and is identified by its port. This is a 16-bit number. To send a message to a server you send it to the port for that service of the host that it is running on. This is not location transparency. Certain of these ports are ``well known''. They are listed in the file /etc/services in Linux system. For example, ftp: 21/tcp, time: 37/tcp, time: 37/udp, finger 79/tcp. TCP/IP reserves ports in the region 1-255. The system may reserve more. User processes may have their own ports above 1023.

2.6 Sockets

Berkeley sockets are the BSD Unix system calls for this. They are part of the BSD Unix kernel. They have also been adopted by the PC world. They form the lowest practical level of doing client/server on both Windows and Unix.

A socket is a data structure maintained by a BSD-Unix system to handle network connections. A socket is created using the call ``socket''. It returns an integer that is like a file descriptor: it is an index into a table and ``reads'' and ``writes'' to the network use this ``socket file descriptor''. Two processes wishing to communicate over a network create a socket each. These are similar to two ends of a pipe - but the actual pipe does not yet exist.

2.7 Process Model

The process model is concerned with establishing connection between systems. The connection-oriented protocol in the transport layer i.e. TCP is used for communication between systems. The fundamental mechanism for the communication between a client and server is shown in the following figure.

Connection oriented (TCP)

One process (server) makes its socket known to the system using ``bind''. This will allow other sockets to find it. It then ``listens'' on this socket to ``accept'' any incoming messages. The other process (client) establishes a network connection to it, and then the two exchange messages. As many messages as needed may be sent along this channel, in either direction.

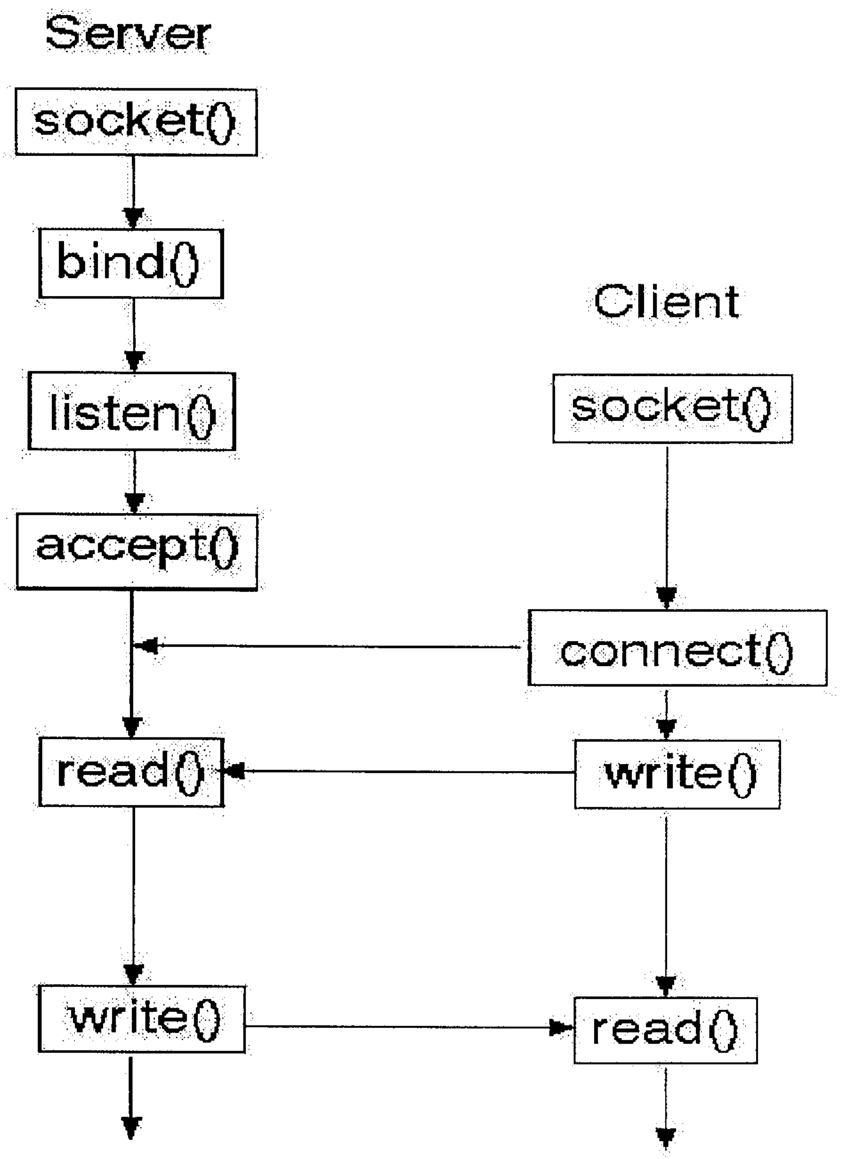


Fig 2.3 Socket Flow

2.8 Server Model

There are two types of Server. They are Concurrent server and Iterative server.

Concurrent Server:

A concurrent server invokes another process to handle each client request so that the original server process can go back to sleep, waiting for the next client request. Naturally, this type of servers requires an operating system that allows multiple processes to run at the same time. Server handles most client requests that deal with a file of information (e.g., print a file, read or write a file) in a concurrent fashion by the server. As the amount of the processing required handling each request depends on the size of the file.

Iterative Server:

When the server can handle a client's request in a known, short amount of time, the server process handles the request by itself. We call this iterative server. The server typically handles a time-of-day service in an iterative fashion.

2.9 Interprocess Communication

Since network programming involves the interaction of two or more processes, we must look carefully at the different methods that are available for different processes to communicate with each other. In traditional single process programming different modules within the single process can communicate with each other using global variables, functions calls, and the arguments and results passed back and forth between functions and their callers. But when, dealing with separate processes, each executing within its own address space, there are more details to consider. When time-shared, multiprogramming operating systems were developed over 20 years ago, one design goal was to assure certain that separate processes wouldn't interfere with each other. For two processes to communicate with each other, they must both agree to it, and the operating system must provide some facilities for the Interprocess Communications (IPC).

While some form of IPC is required for network programming, the use of IPC is by no means restricted to processes executing on different systems, communicating through a network of some form. Indeed, there are many instances where IPC can and should be used between processes on a single computer system.

CHAPTER 3

3.Functional Specification

Functional Requirements of Mail Server

Functional Requirement 1:

Introduction: Facility to login in into his mail id and create a new login.
Input: The mail id and the password of the login.
Process: The server checks for the validation of the user id and his password using hashing principle and grants him permission to access his mail.
Output: The server authorizes the user depending on the validity of his password and his ID.

Functional Requirement 2:

Introduction: Facility for send mail to another user.
Input: The various commands involved and the text to be transferred and the receiver id.
Process: The server checks for the existence of the receiver id and the syntax of the commands involved. The server sends a failures report if the receiver id is not present.
Output: The server sends the message to the receiver if the commands are valid.

Functional Requirement 3:

Introduction: Facility to list and read the messages user has obtained.

Input: The message number and the number of lines to be listed.
Process: The server checks whether the message is present in his mailbox.
Output: The server sends the message to the user and he is allowed to read the message.

Functional Requirement 4:

Introduction: Facility for the user to schedules his friend's birthday and anniversary.
Input: The details in the address book.
Process: The server goes through the user address book of every user and checks if any entry in the address book matches with the next date of the system for tier corresponding birthday and anniversary day. The server also checks for the status of the message he has send to the receiver and reports him about its status.
Output: The client receives separate mails informing him of his friends' birthday and anniversary day.

Functional Requirement 5:

Introduction: Facility for the user to know the status of the mails he has sent.
Input: The user id and password.
Process: The server checks for the user id and password and opens the corresponding file, which contains the status of the mail send by the user.
Output: The server provides of the message he has send to the various users and reports him about its status.

Functional Requirement 6:

Introduction: The server announces when a message comes from an important person whenever user gets login.
Input: The details name and login ID of the important person is given in the address book.
Process: The server goes through the user address book of logged user and checks if any entry in the address book matches with the received mail.
Output: A message announces the arrival of important message.

Functional requirements of Distributed Application management

Functional Requirement 1:

Introduction: Facility to make a request by the client to the server.
Input: The application name that has to be imported.
Process: The application name is get from the user and the data block that has to be sent to the server is built and sent.
Output: The block of data consisting the application name and client details is sent to the concurrent server.

Functional requirement 2:

Introduction: Facility to handle the request of the client by the server.
Input: The block of data sent by the client.

Process: The server receives the request from the client and spawns another process to handle the request. The application is searched in the server.
Output: if the application is present in the server then it is exported to the requested client else the server send signals to the other clients present in thr network to transfer the application if they have to the requested client.

Functional requirement 3:

Introduction: Facility for the client to handle the signal sent by the server.
Input: The application name and IO address.
Process: The client searches for the client for the specified application in it and it is found then exports that to the specified client.
Output: If the client has the application then it is transferred to the specified client IP address and a report message for successful is sent to the server else a report message for failure is sent.

3.2 Hardware and Software Requirements.

Establishing a network requires rs232 cables, network hub, LAN cards, and two or more PCs. A UPS is needed for regulated power supply.

Linux operating system has to be installed in the PCs used. The editors vi or vim, which are provided as part of the OS, is needed for project developing i.e. coding. The compiler g++ is needed for developing the project and also for debugging the project, which comes along with Linux, is needed for creating object module of the software.

The compiler **cc,** along with Linux, is needed for creating object module of the software.

LINUX:

Linux is an Operating System for several types of computer platforms but primarily for Intel- based PCs. The system has been designed and built by hundreds of programming scattered around the world. The goal has been to create a UNIX clone free of any commercially copyright Software, which the entire world can use.

Linux was developed in the early 1990's by Linux torvald along with other programmers around the world.

Linux is basically a UNIX clone, which means that with LINUX we get many of the advantages of UNIX. LINUX multitasking is fully preemptive; meaning that we can run multiple programs at the same time and each program seems to process continuously.

LINUX allows starting a file transfer, printing a document, copy a file, using a CD-ROM, and playing a game all at the same time.

WHY USE LINUX

LINUX is selected because it is only operating system today that's freely available to provide multitasking and multiprogramming and multiprocessing capabilities for multiple users on IBM PC- compatible Hardware platforms. You aren't locked into upgrading every few years and paying outrageous sums to update all your applications. Many applications for LINUX are freely available on the Internet. Thus we have access to the sources code to modifying and expand the Operating System to our needs something you can't with commercial Operating Systems such as Windows NT and Windows 95.

FEATURES OF LINUX

Here are some reasons why LINUX could be the best Operating System.

LINUX Distributions has thousands of dollars worth of Software for no cost.

LINUX is a complete Operating System that is

- ❖ Stable -its crash of an application is much less likely to bring down the Operating System under LINUX.
- ❖ Reliable -LINUX servers are offered to run up for hundreds of day compared with the regular reboots required with a Windows system.
- ❖ Extremely Powerful

Comes with a complete development environment, including C, C++, FORTRAN compilers, toolkits such as Qt and scripting languages such as PERL, AWK. C Compiler for Windows alone would set your backs hundred of dollars.

Excellent networking facilities: allow you to share modems etc., The ideal environment to run servers such as a web server (e.g. Apache) or Ftp server.

Wide variety of commercial Software is available if your needs aren't satisfied by the free Software.

An Operating System that is easily upgradeable.

About C:

The language selected for this project is **"C"** efficient for network-oriented project.

This language is efficient, powerful and compact.

C has emerged as the language of choice for most applicable due to speed, portability and compactness code. **C** is highly portable; programs running on computer can be used with another computer with different Operating System with slight or no modification.

Program written in **C** are efficient and fast, that is due to its variety of data types and powerful operators. Several standard functions are available in C, which can be used for developing programs.

Another important feature of C is its ability to extend itself.

A C program is basically a collection of functions that are supported by the C library.

We can continuously add our own functions to C library. With the availability of large number of functions the programming task becomes simple.

3.3 Resource Requirements

The resources such as Linux manual pages for system calls and shell commands are needed for reference during implementation. As to satisfy the client's request, the server must be loaded with ample amount of applications.

3.4 Testing Requirements

Testing Requirements for the two modules includes the following things:

- Three to four systems in a network
- The server programs are run under different root as user
- The client programs are run on other systems.
- The client can request an application, which is present in the server.
- The client is made to request an application, which is present in other client.
- The client is made to request an application, which is not present in any systems in the network.
- The output is verified for the above three cases.

CHAPTER 4

SYSTEM ANALYSIS

In the analysis phase, the project is divided into many stages or phases and each phase is carried out in a sequential fashion.

4.1 Project phases

PROJECT PHASES
SRS Documentation
Project Coding
Testing
Implementation
Detailed Design Documentation

4.2 Constraints, Dependencies and Assumptions

Application 1:

The constraints and dependencies of the first application are as follows:

- ❖ Username and password must be in small letters while the commands are in capital letters.
- ❖ Each user must contain an individual file system for the details of the message send, and for those, which he has received, and an address book.
- ❖ The server program has to run all the time.

The assumptions are as follows:

- ❖ The user is believed that he knows the syntax and commands for the mail transfer.

Application 2:

- ❖ Sharing the application is possible only with the client connected to the network.
- ❖ The requested application must have the permission to be executed.
- ❖ The dependent files transferred to the client before execution of the application.
- ❖ The server program must be run under root as the user.

The assumptions are as follows:

The requested application is an executable file for Linux platform. Some of the executable files on Linux platform are .tar, .out, etc.

4.2 Software Process Model

Linear Sequential Model

The software process model followed by us to develop the project is linear sequential model. Sometimes called the classic life cycle model or waterfalls model, the linear sequential model suggests a systematic, sequential approach to software development that begins at the system level and progresses through analysis, design, coding, testing and support. The following figure illustrates the linear sequential model for software engineering.

The classic life cycle paradigm has a definite and simple structure. It provides a template into which methods for analysis, design, coding, testing, and support can be placed.

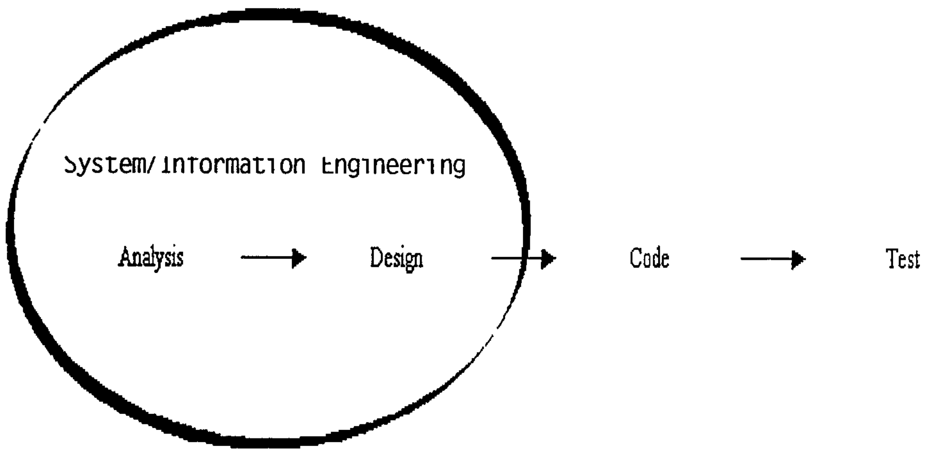


Fig 4.1 Linear Sequential Model

CHAPTER 5

SYSTEM DESIGN

5.1 Design Constraints

Portability:

The project has to be designed in such a way that it can be ported to other Linux systems.

Reliability:

The project has to be designed in such a way that it works for all sorts of input from the system environment. For example, the server performs the mailing operation, which is requested by the client. If the request is not present in the server file system then failure message code has to be sent to the client. This ensures reliability in mailing operation.

Performance:

The system performance is measured by its execution time and space occupied. Hence in order to ensure performance, the system has to be designed efficiently.

Scalability:

Any number of clients can be added to or removed from the network without exceeding the network load. The scalability can be ensured when the system supports any number of clients without any significant decrease in performance.

5.2 Functional Model

Data Flow Diagram

The Data Flow Diagram (DFD) that shows the information flow continuity of the system along with its entities are shown in the following figures.

Application 1:

The main objective of the system is to provide the client (user) with the mail transfer and mail reading facility. The design phase starts with identifying the entities present in the system - server, client that places request, user who sends mail through the client to the SMTP server and the server acts as the messenger between sender and receiver. Using the POP server the receiver can read his mails. The Data Flow Diagram (DFD) that shows the information flow continuity of the system along with its entities is shown in the following figure.

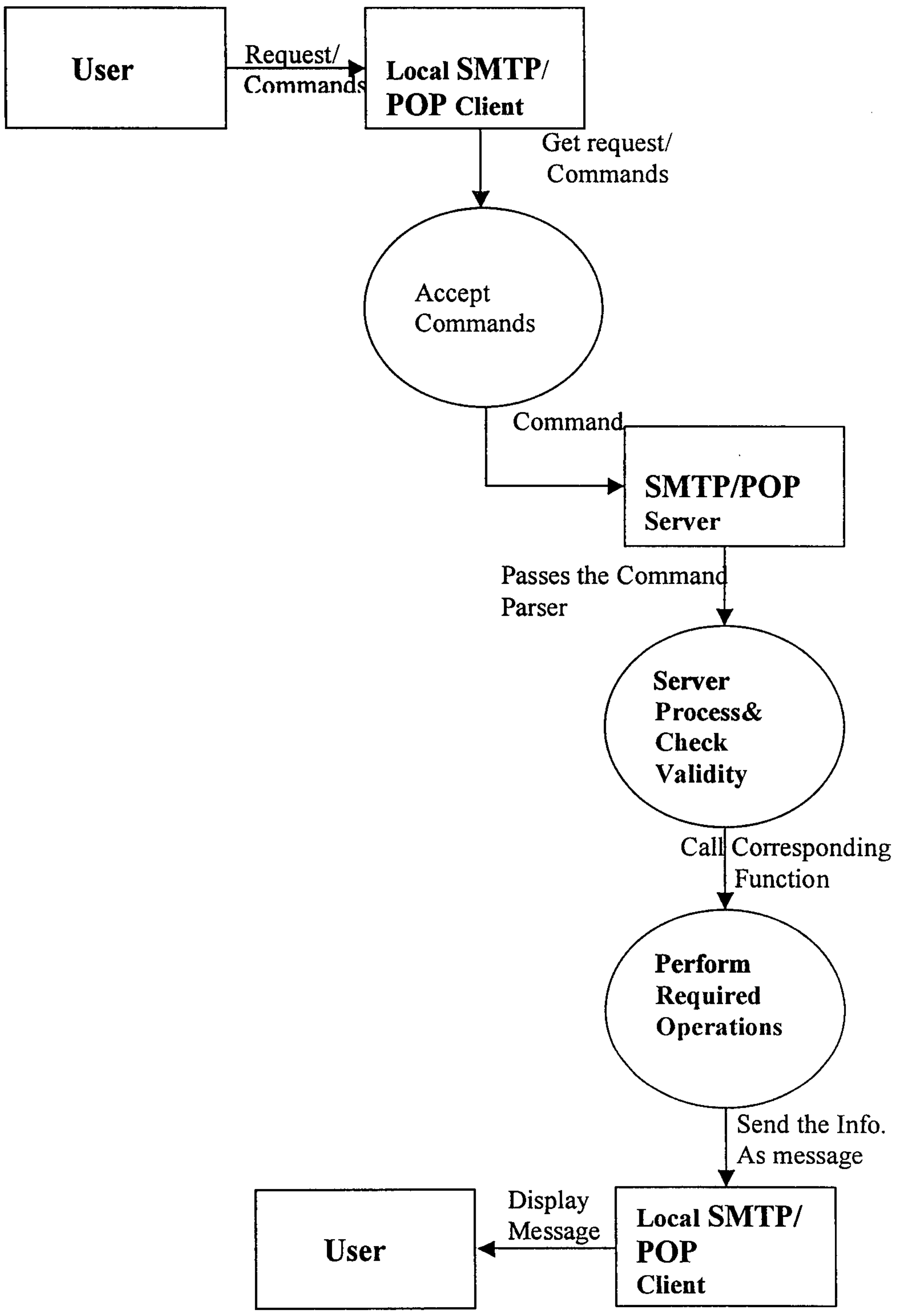


Fig 5.1 DFD of Mail Server

Application 2:

The main objective of the system is to provide the client with the requested application if it is present in the network. The design phase starts with identifying the entities present in the system - server, client that places request, other clients present in the network, user who provides the client with the name of the application. The application, if found, is transferred to the client and then automatically invoked by the kernel of that system.

DFD for Client -A module

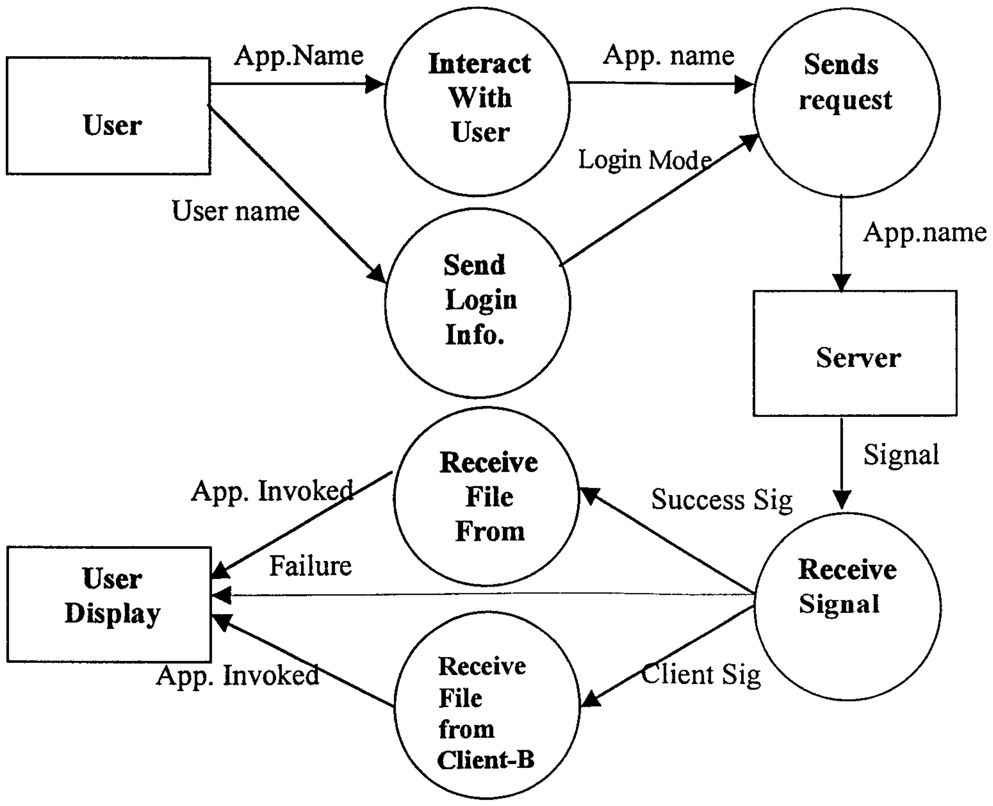


Fig 5.2 DFD of Client-A

DFD for Server Module

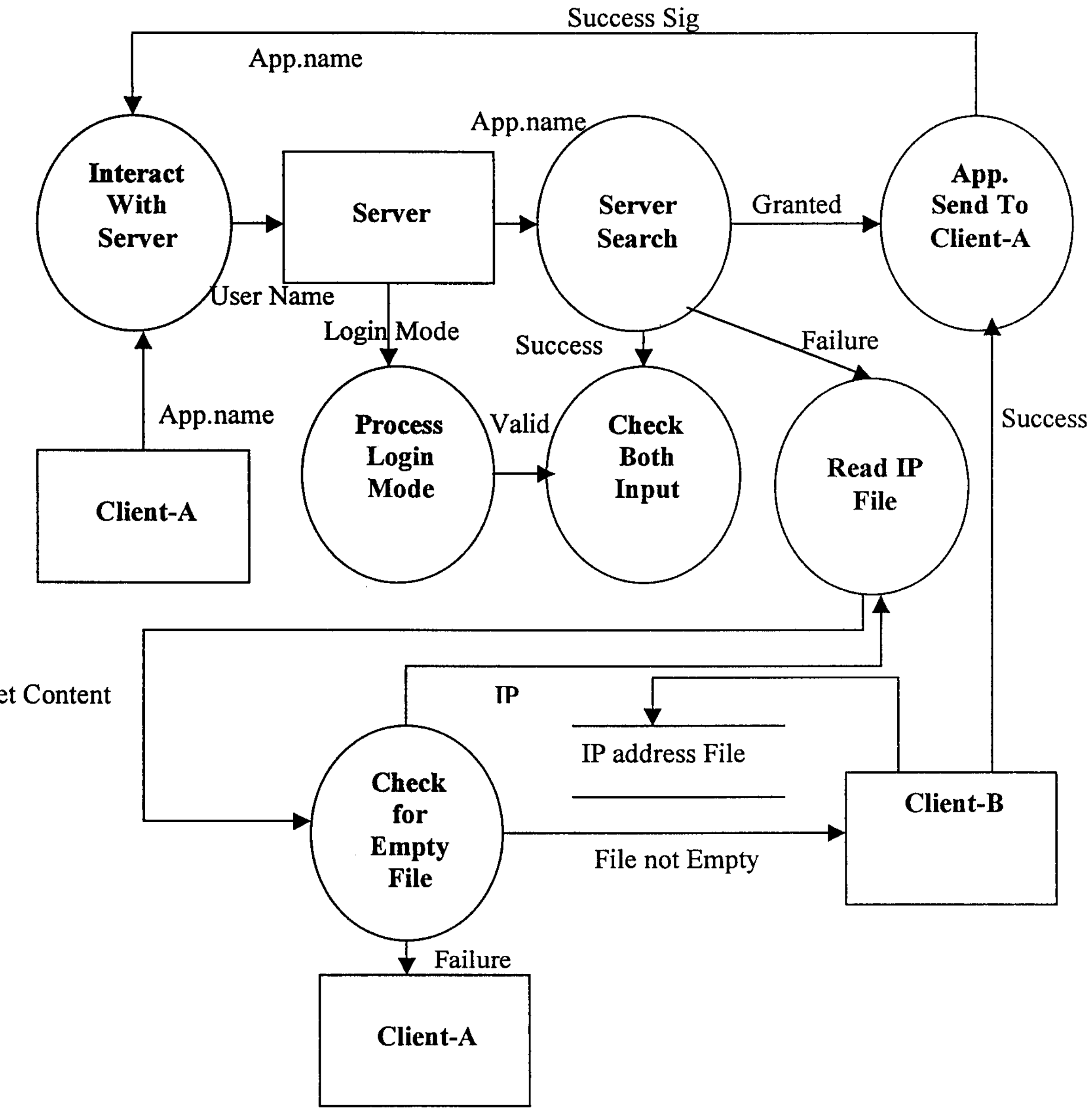


Fig 5.3 DFD of Server

DFD for Client -B Module

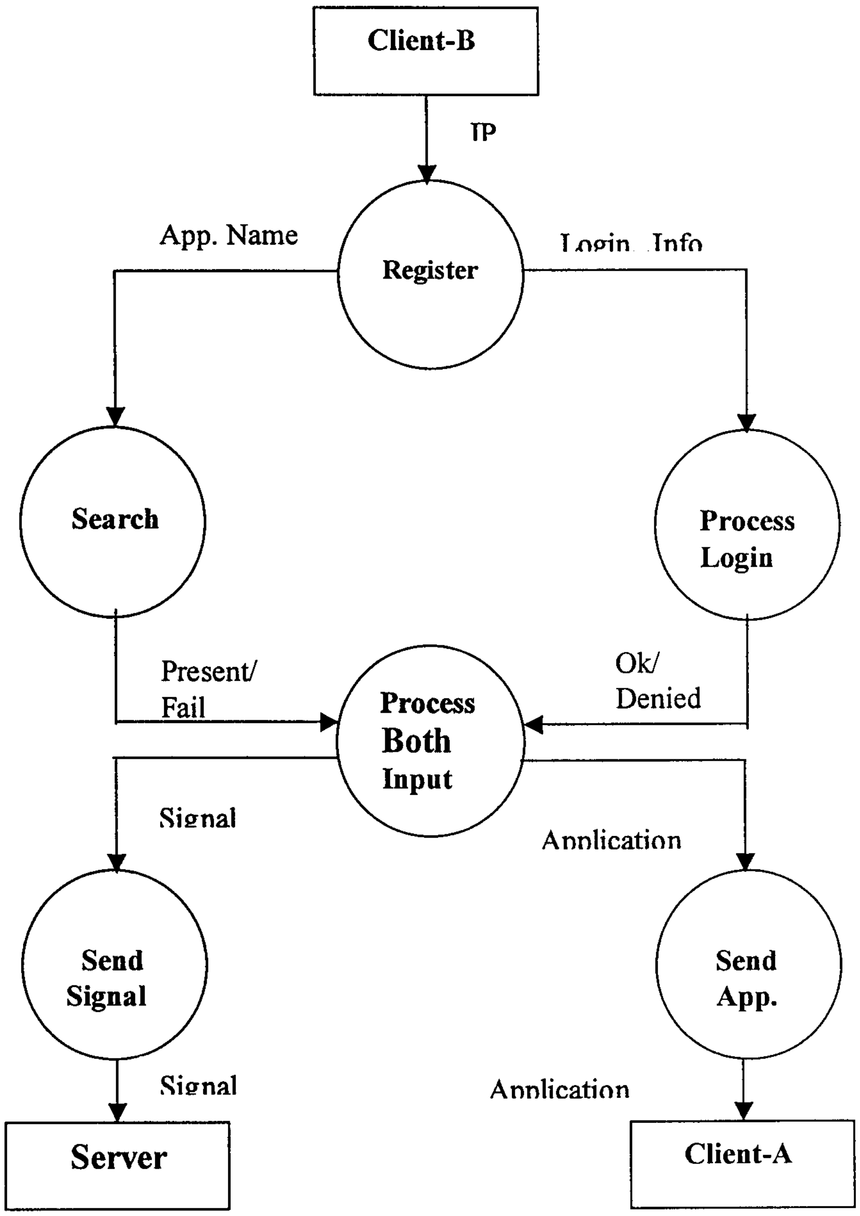


Fig 5.4 DFD of Client-B

5.3 Modular Design

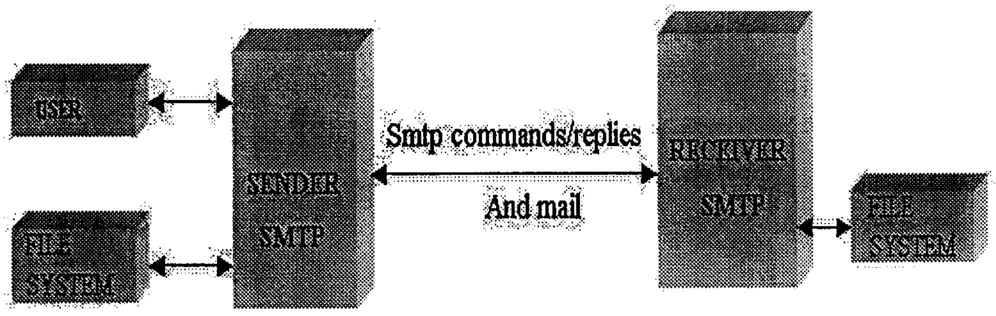
Application 1:

The modules that have been extracted from the above data flow diagram and its functionality is given in the following tabular column.

MODULE	FUNCTION
SMTP Client	Establish a connection with the SMTP sever and specifies the sender's, receiver's mail addresses and the message data to be send to
SMTP Server	Identifies the sender (SMTP client) and his domain and receives the message from sender and transfers it to the receiver's inbox. It also checks the validity of the receiver
POP Client	Establish a connection with the POP server and logs in as a user of the mail service and gets the request from user. The request is send to
POP Server	Identifies the user and checks the validity of his address and password. Then checks the request from user and replies with error
Hashing	Hashing is used to check the validity of receiver's mail address during the sending of mail from SMTP client to server. It is also

The SMTP design is based on the following model of communication: as the result of a user mail request, the sender-SMTP establishes a two-way transmission channel to a receiver-SMTP. The receiver-SMTP may be either the ultimate destination or an intermediate. SMTP commands are generated by the sender-SMTP and sent to the receiver-SMTP. SMTP replies are sent from the receiver-SMTP to the sender-SMTP in response to the commands. Once the transmission channel is established, the SMTP-sender sends a **MAIL** command indicating the sender of the mail. If the SMTP-receiver can accept mail it responds with an OK reply.

The SMTP-sender then sends a **RCPT** command identifying a recipient of the mail. If the SMTP-receiver can accept mail for that recipient it responds with an OK reply; V if not, it responds with a reply rejecting that recipient (but not the whole mail transaction). The SMTP-sender and SMTP-receiver may negotiate several recipients. When the recipients have been negotiated the SMTP-sender sends the mail data, terminating with a special sequence. If the SMTP-receiver successfully processes the mail data it responds with an OK reply. The dialog is purposely lock step, one-at-a-time.



Module for SMTP use

Fig 5.5 Functional Diagram of SMTP

The SMTP provides mechanisms for the transmission of mail; directly from the sending user's host to the receiving user's host when the two host are connected to the same transport service, or via one or more relay SMTP-servers when the source and destination hosts are not connected to the same transport service.

To be able to provide the relay capability the SMTP-server must be supplied with the name of the ultimate destination host as well as the destination mailbox name.
The argument to the MAIL command is a reverse-path, which specifies who the mail is from. The argument to the RCPT command is a forward-path, which specifies whom the mail is to. The forward-path is a source route, while the reverse-path is a return route (which may be used to return a message to the sender when an error occurs with a relayed message).

When the same message is sent to multiple recipients the SMTP encourages the transmission of only one copy of the data for all the recipients at the same destination host.

The mail commands and replies have a rigid syntax. Replies also have a numeric code. In the following, examples appear which use actual commands and replies. The complete lists of commands and replies appear in Section 4 on specifications.

Commands and replies are not case sensitive. That is, a command or reply word may be upper case, lower case, or any mixture of upper and lower case. Note that this is not true of mailbox user names. For some hosts the user name is case sensitive, and SMTP implementations must take case to preserve the case of user names as they appear in mailbox arguments.

Functional Block Diagram

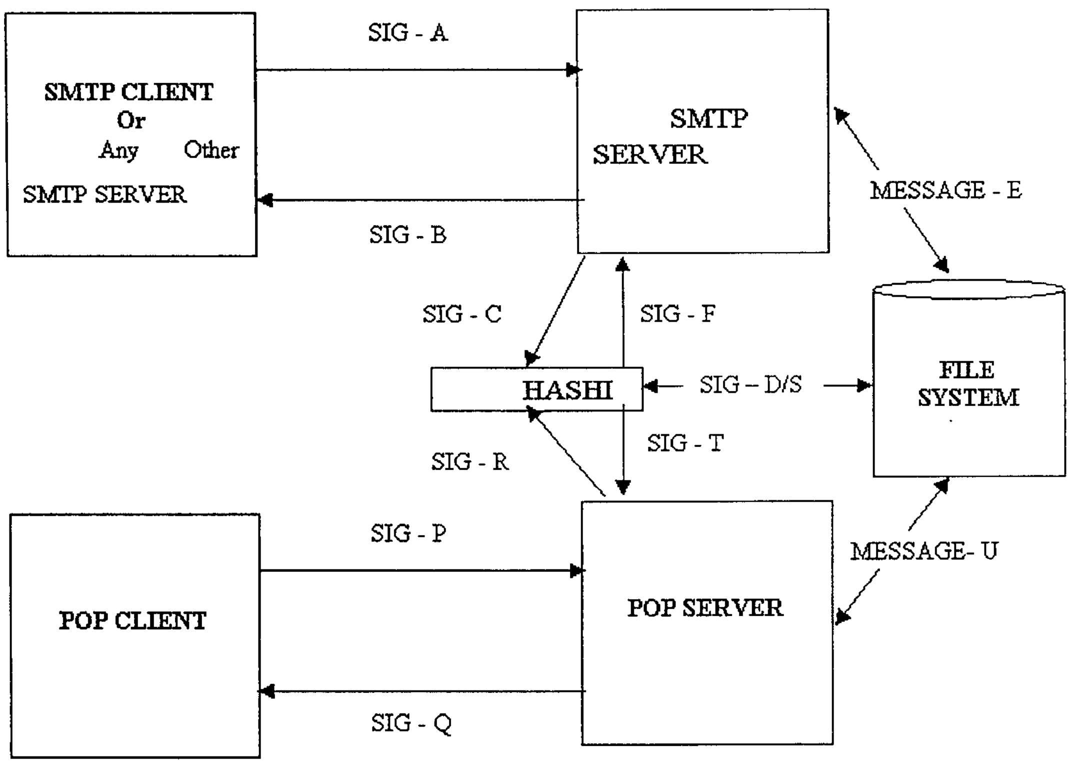


Fig. 5.6 Functional Diagram of Mail Server

SIG - A	Sends mail and other commands.
SIG - B	Sends reply message (error or positive replies).
SIG - C	Sends receiver's mail address for validity check.
SIG - D	Hashing checks password file for validity of receiver's mail address
MESSAGE- E	Sends the data (message) for storing in receiver's inbox.

SIG - F	Sends positive or negative reply depending on receiver's mail address
SIG - P	User sends his request of mail to be read, deleted or login details.
SIG - R	Sends the user mail address and password for login check-up.
SIG - S	Hashing checks password file for receiver to authenticate
SIG - T	Sends positive or negative reply.
MESSAGE - U	Sends the data (message) form user's inbox for reading.

Application 2:

The modules that have been extracted from the above data flow diagrams and its functionality is given in the following tabular column.

MODULE	**FUNCTION**
Client-A	Interacts with the user, gets application name an places request to the server. Invokes the application received from the server or client system.
Client-B	Receives application name from the server, looks for the application in itself and if found, it is transferred to the requesting client - A
Server	Receives request from the client - A, looks for the application in itself and if found, it is transferred to the requesting client - A. If the application is not present, it sends signal to client - B requesting file search in that system.
Search	Performs breadth-first searching using circular queue.
File Transfer	Performs file transfer operations using IP addresses of the systems connected to the network.

Functional Block Diagram:

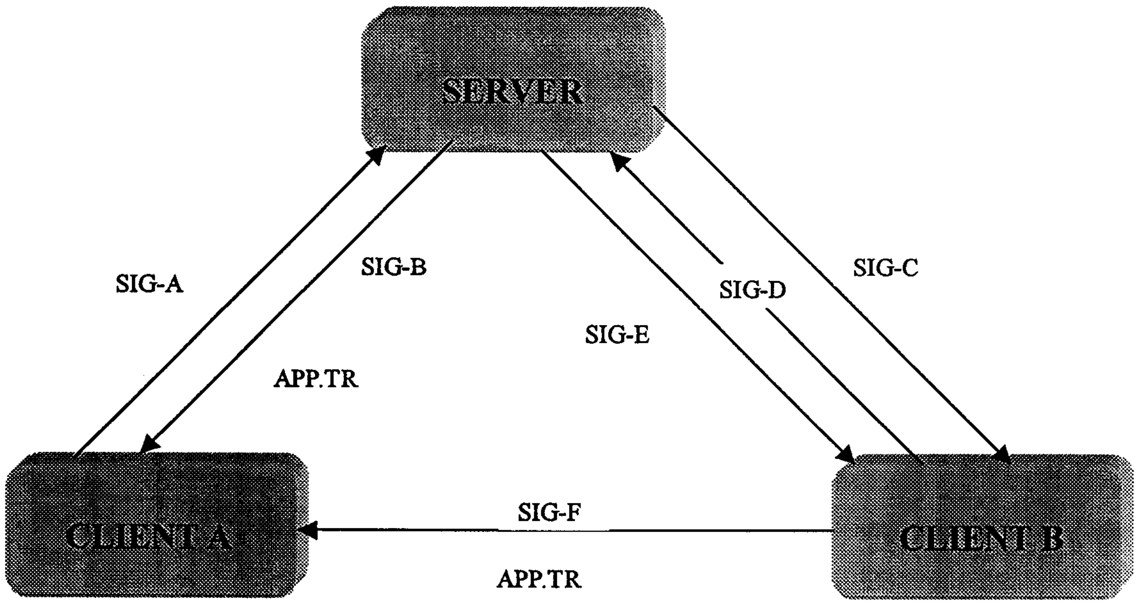


Fig. 5.7 Functional Diagram of Distributed Application Management

SIG - A	Sends name of the required application
SIG - B	Sends SUCCESS or AD or CLIENT message
SIG - C	Sends application name to other clients
SIG - D	Sends SUCCESS or FAILURE message
SIG - E	Sends IP address of the requested client, if D = SUCCESS
SIG - F	Connection establishment between clients
APP.TR	The application is transferred

5.4 Logical Data Design

Application 1:

File system

Files are used for storing user mail address and its password in the *"pass.txt"* file. Domains of various servers are stored in *"domin.txt"* file. It is also used for storing details about the messages received in *"user_detail.txt"* file. And also about the messages send in *"user_send.txt"* file. The user's address book is also maintained as a file named *"user_adbook.txt"*. Messages received by the user are stored in the name of "user01.txt", "user02.txt" … files for corresponding messages.

Password file:

Password file is used for two purposes.

- ❖ Firstly it is used for checking the validity of receiver's mail address during mail send from SMTP client to SMTP server.
- ❖ Secondly it is used during the user login through POP client.

In both the cases hashing module handles the check-up for servers. Its header structure is shown in the file structure diagram.

Domains:

The various servers' domains are stored in this file. This is used for checking the validity of out side SMTP servers sending mails to our server.

Message received file and Message send file:

These two files contain the details about the user's send mails and received mails. The header of the messages received or send are stored in these files. So the header stored in both the files are same. Then the total number of files and the total memory occupied by them are stored at its end. The header structure is shown in the file structure diagram Scheduling file and address book.

This file has the details about the user's relatives or friends name, mail address, their birthday, and anniversary day. This file is used by the SMTP server for sending a reminding mail for the user about his relative's birthday or anniversary day. Its address book header structure is shown in the file system diagram.

Messages:

Messages received by the user are stored in a queue. So the mail, which reaches him, first is given a number less than the one reaching him later. In this file the message data is stored first and then followed by its message header. Its message header structure is shown in the file system diagram.

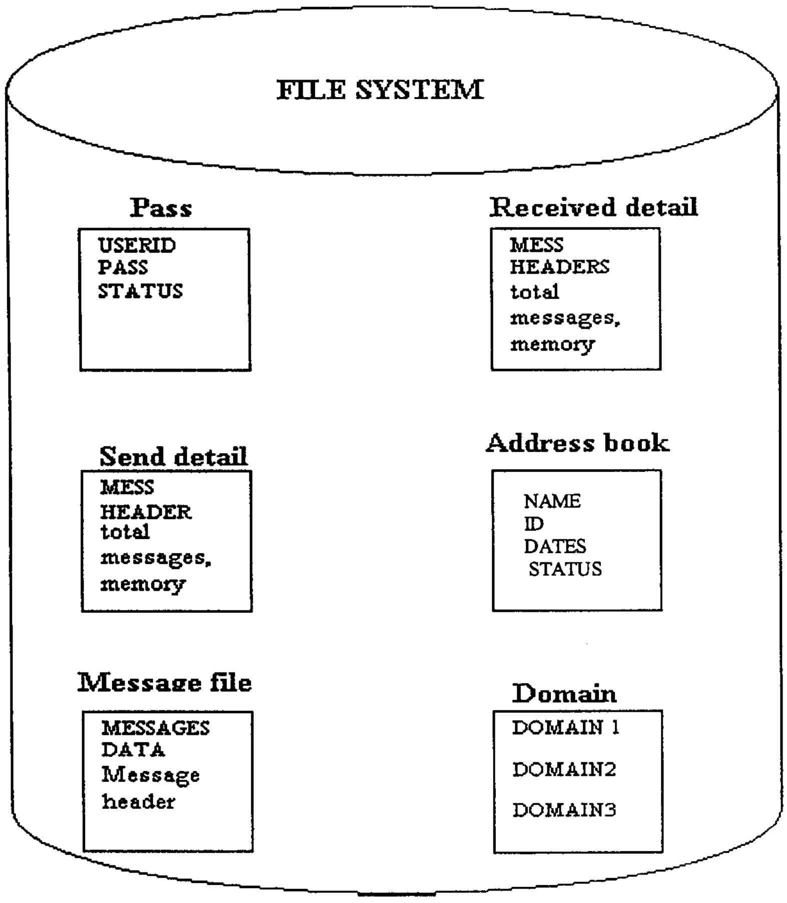


Fig. 5.8 File Components Of Mail Server

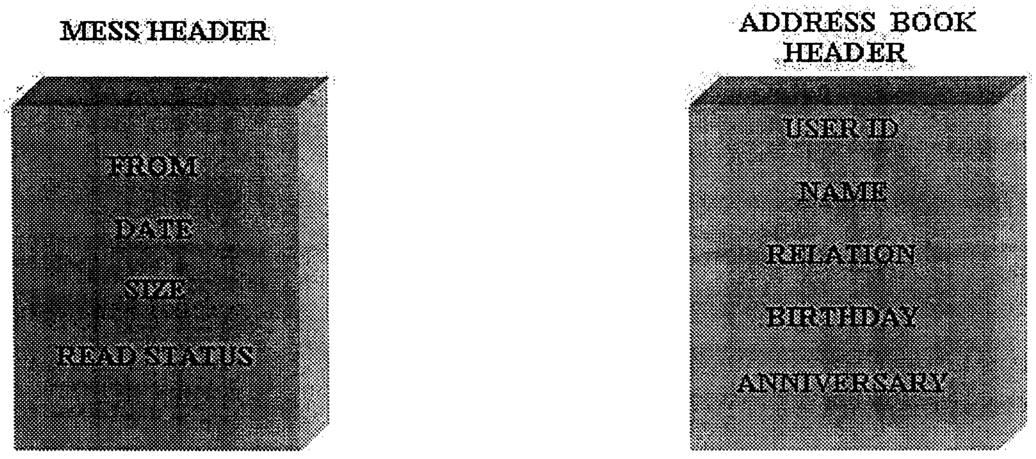


Fig. 5.9 Header Diagram

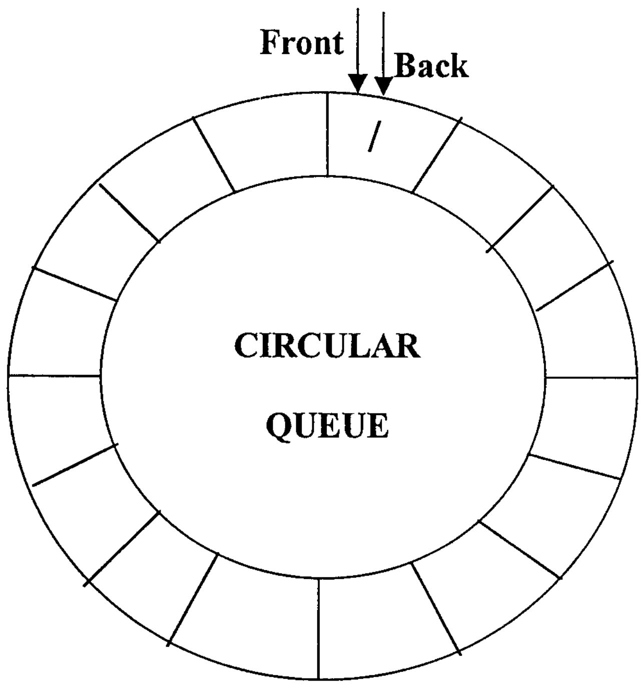
Queue

A queue is maintained for storing messages received by the user. So the messages come and join the queue of messages already present. This is done by naming the message file name as the in the next to last message received. And the corresponding message header in either send detail file or receive detail file is appended with increasing the number of total messages by one and the total memory correspondingly at the end of these files. While deleting also this queue is maintained and messages are named in that order.

Application 2:

Circular Queue

The searching module involves the usage of circular queue. The directory entries are classified into subdirectories and files. The subdirectories are inserted into queue at its rear end. The entry in the queue at the front end is deleted and its subdirectories are in turn inserted into the queue. When an element is inserted into the queue, the front-end pointer is incremented. When an element is deleted from the queue, the rear end pointer is incremented. Comparing the front and rear pointers with the size of the queue checks the queue full condition. The circular queue also has a front and rear end pointers. The pointers work in the same fashion as the ordinary queue. The front-end pointer is incremented during every deletion and it gets updated to *front-end mod queue-size*. Similarly the rear end pointer also incremented during every insertion and it gets updated to *rear-end mod queue-size.* The searching is done by taking an element each time from the queue and finding whether the file is present inside that directory. The searching stops when the file is found or the circular queue is empty. The circular queue is thus used as a logical data model for breadth first searching mechanism.



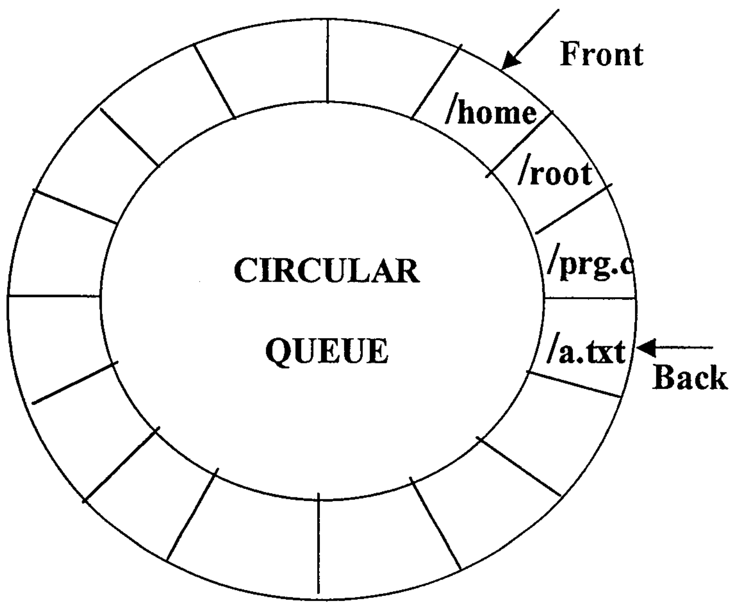


Fig. 5.10 Circular Queues

5.5 Process Design

Application 1:

SMTP CLIENT

The client first establishes a connection with the SMTP server and receives an introduction message from that server. Then first of all it has to tell the server about its domain through HELO command. Then the sequences of commands are as follows,

- MAIL FROM command is used to tell the server the sender's mail address.
- RCPT TO command is used to tell the server about the receiver's mail address
- DATA is used to tell the user about the starting of the message, followed by the message header and message content. Whenever a line is typed it is send to the server. Finally the message is ended by a line starting with "./n" sequence.
- The client can quit the connection by QUIT command.

For each command the server sends either a positive or negative message. If it happens to be a negative reply then the whole process has to be started from the MAIL FROM command.

SMTP SERVER

The server has to run before the client attempts for connection. So the server waits for the client in the accept mode. And when a client connects a separate socket is created for communication. Before that when the server is started each day it first checks the address book of each user and sends a reminding mail if there are some of his relatives or friends having birthday or anniversary on the next day.

Then the server waits for client to connect. A separate thread is created by the server in-order to handle each client individually. Then the serve sends an introduction address to its client. Then it waits for any possible commands from its client. And it performs the corresponding tasks as follows

- ❖ If it were a HELO command it identifies the domain of the client or other SMTP server. And sends a reply depending on the domain's validity.

- ❖ If it were a MAIL FROM command it stores the sender's mail address. And sends a positive reply.

- ❖ If it were a RCPT TO command it verifies whether it has one such mail address and sends a reply correspondingly. It calls the hashing function to verify the mail address. This function reads the password file from the location obtained from hashing the mail address. And checks the match for both password and mail address stored in that location.

- ❖ If it were a DATA command it opens the sender's send message detail file and receiver's receive message detail file and appends this message's header. Then it increases its total number of messages and memory correspondingly. Finally the message is stored in the message file with its message header in its end. This is done in order to find the message size and add it in the message header.

- ❖ If it were a QUIT command the server sends a concluding message and closes the socket connection.

After each message the server send either a positive or negative reply to the client. Whenever a negative reply is send the server accepts command from the beginning of MIL FROM sequence.

When the client quits the thread meant for it detaches automatically.

Since the SMTP server follows the standard any other SMTP server can also connect with this server.

POP CLIENT

The POP client first establishes a connection with its server. Then the user inputs his authentication details and sends it to the server. After getting a positive reply from the server he can check his inbox, outbox, delete messages etc. So the client waits for the user for his request. And it performs the following operation depending on the reply from server.

- ❖ If it were a RETR command it sends the read command to server and waits for server to reply. And reads the corresponding message if there is positive reply from server.

- ❖ If it were a DELE command it sends the delete command with a massage number to the server and waits for reply.

- ❖ If it was a NOOP or RESET commands it send this command and waits for reply.

- ❖ If it were a QUIT command it gets a concluding message from server and logs out.

POP SERVER

The POP server must run before its client and wait for its client to connect to it. After a connection is established a separate thread is created for each client. And the serve checks the user authentication details. After logging in the server responds to client in the following way for each request.

- ❖ If it were a STAT command then the server opens the users receive message detail file and sends the total number of files and its total memory.

- ❖ If it were a LIST command the server reads the users receive message detail file and sends each message's size.

- ❖ If it were a RETR command the server reads the receive message file and checks for the validity of message number and sends the message if it's a valid message number.

- ❖ If it were a DELE command the server again checks the validity of message number and adds that number in its marked deletion files list if valid.

- ❖ If it were a CKMB or CKOB the server sends the users receive or send message detail file to client in the same way as data is send for read command.

- ❖ If it were a REST command the server clears the marked file list for deletion.

- ❖ If it were a QUIT command the server first deletes all the marked files for deletion and renames the successive files. Then sends a concluding message before closing the socket.

The thread automatically detaches and the socket connection is closed.

Scheduling

User to add entries to his address book uses scheduling. The SMTP server to remind the users about their relative's birthdays or anniversaries uses this address book. The server first reads a use from the password file one by one. Then his address book is checked for any birthdays or anniversaries occurring on the next day. And send a reminding mail if there is a match. This is done for all the users. This function is called when a server is started on that day.

Send

The SMTP server receives the valid domain name from the client and then receives the valid sender's mail address. This is used to add it into the send message header. And similarly the receivers mail address is also received and stored.

The data (message) is obtained line-by-line from the client and appended in the buffer. Later it is stored in the message file when the end of message file is reached "./n". After this the message header is appended in the message file and also in the message send, receive detail file. It also adds the total number of messages and memory correspondingly.

If any error occurs a negative reply is send. All the files are closed before the positive message is send to the client.

Read

The POP sever gets the read command and checks the validity of the message number by reading the receive message detail file. First it reads the message header and sends it to client. It also changes the read status in the sender's outbox. And finally reads the message and sends it line-by-line to the client.

If any error occurs a negative reply is send. All the files are closed before the positive message is send to the client.

Check inbox and outbox

When the user requests for checking his inbox or outbox the POP server just opens that file and sends it similar to a message read command's data delivery operation. At the end of the file it sends the end of data sequence "./n" to indicate the client about the file end.

If any error occurs a negative reply is send. All the files are closed before the positive message is send to the client.

Delete and reset Quit

The POP server first reads the receive message file and checks the validity if delete message number. If valid it is added in the marked list for deletion. Else an error message is send.

In case of a reset before quit the POP server clears the marks on the files for deletion. But if there is no reset and the user logs out the POP server deletes that message file and removes the message header from receiver's message file. Then the successive message files are renames a number less than before. This is done separately for each deletion of messages. And a bunch of messages are deleted single-by-single deletion operation.

If any error occurs a negative reply is send. All the files are closed before the positive message is send to the client.

HASHING - verify mail address, login, login creation.

The hashing function is used in both the SMTP and POP server. In the SMTP server it is used to verify whether there is a valid receiver mail address in its domain. Hashing the mail address and finding the location in password file is done. Then a password structure is read from that location. And the status of that is checked for a value of 1(which indicates a occupied location) and the mail address stored in that location should match with the given one for a valid mail address.

In the POP server it is used twice. Once in login creation and then every time during login. During logging in the same process is done like verifying a mail address. Along with this the password should also match for logging in. And during login creation the hash function checks whether a location is present with status 0(which indicates the free location). If not it rehashes using double hashing function to find any other free location. And the user can be added in this location. If not the there is no space for user to login, and the password file needs updating. Enlarging the hashing functions domain does this.

If any error occurs a negative reply is send. All the files are closed before the positive message is send to the client.

Application 2:

Client-A Module

The user interacts with the system through this entity. The user works in the client system and types in the application name that he is in need of. The application name is then sent to the server (sig-A) using appropriate socket. The login mode of the user in client-A system is obtained using apposite system calls and it is also sent to the server.

The server sends a signal (sig-B) to this system (client-A). The signal is then processed. If it is a SUCCESS signal, it indicates that the requested application is present in the server itself and the login mode is also correct. If sig-B is AD, then it reveals that the application is present in the server but the login mode is irrelevant (i.e., ordinary user asks for root-only usable application). If sig-B is CLIENT, then the client establishes a connection with the request-satisfying client through another socket.

If sig-B is any other signal (e.g. FAILURE), it shows that the requested application is not present in any of the systems in the network.

If sig-B is SUCCESS or CLIENT, then the client receives and invokes the application that was transferred either from server or from any other clients in the network, by using apposite system calls.

Client-B Module

The client-B module first registers its IP addresses with the server. After registration, it waits in a blocking mode for any signal from server. It receives a signal sig-C containing the name of the application from the server. The system then starts searching for application. If the search is successful, then as in the case of server, login mode is checked and then correspondingly the signal sig-D = AD or SUCCESS is sent. In the alternate case sig-D = FAILURE is sent.

Server Module

The server registers the IP addresses of all the clients in the network as a child process, separately. These IP addresses are stored in a file for future use.

The server system receives the application name, requested by a client in the network through the socket (sig-A). It also receives the login mode of the user in the client-A system. The server then searches for requested application within the server system.

If the search is successful i.e., file is present in the server, it checks for login mode of the user in the client-A system. Suppose the user has been logged on as an ordinary user and asks for special files that are owned by user of super-user equivalence, then access permission is denied to the application and the signal sig-B is send as AD (Access Denied). In all other cases, the requested file is transferred to the requested client, before which sig-B= SUCCESS is sent to the client.

If the search is unsuccessful i.e., application is not present in the server, then the request is sent to every other clients (sig-C) in the network that were previously registered with the server, through a separate socket. This can be achieved by reading the contents of IP addresses file, line-by-line. After every reading and sending of signal sig-C, sig-D is checked. If sig-D = SUCCESS, then the signal sig-E would be send to the successful client, which has the address of the requesting client. The server also sends another signal sig-B = CLIENT to the requesting client indication that the request would be satisfied by a client in the network.

Another case of failure raised by having sig-D = FAILURE, could be handled by checking the end of file, which if true then sig-B = FAILURE would be send to the requested client (client-A), otherwise another entry from the file is read and the process would continue.

Search Module

The first step in the search module is to allocate required memory i.e., maximum size of queue and initialize the front end pointer of the queue to point to the root directory i.e., queue [front] = "/".

The field at the front of the queue is removed and the directory is opened using appropriate system call with removed queue element as the parameter. The front end pointer is manipulated. The contents of the directory are read sequentially using the apposite system call. After every read operation, checking whether any subdirectories are present is done. If it is so, then they are inserted at the rear end of the queue. The rear end pointer is manipulated. If it is a file, then it is compared with the requested file name. If it matches the location is returned and the search is successful.

If it is a file but not the one that was required or after manipulating the rear pointer, the emptiness of the directory is checked. If still more records have to be read, then the process of reading the directory is continued.

Considering the case, that all records in the directory have been read, it is checked whether front-end pointer is not equal to rear end pointer of the circular queue. If both pointers are equal, then the process is terminated resulting in unsuccessful search. If they are equal the process is prolonged right from removing the front field in the queue.

File Transfer Module

File Transmission

The application is opened and its permissions are sent to the requested client. The application is read in a predefined size of 512 bytes. It is sent to the client through appropriate socket. If the size of file read is less than 512 bytes the process is terminated or else the process is continued until the condition is satisfied.

File Acceptance

The permission sends either by the server or by the satisfying client is received by the requesting client. A new file is created with the received permission in the requesting client system. The contents of the file is then received and written into the created file in a predefined size of 512 bytes. If size goes less than 512 bytes the process is terminated or else the process is continued until the condition is satisfied.

CHAPTER 6

IMPLEMENTATION ISSUES

The inbuilt data structures are used in finding the required information such as file permissions, file ownership, network address i.e. IP address of the host system, etc. These data structures are the input or output of system calls used. Some of the data structures used are as explained below.

6.1 Data Structures

The functions that convert IP address of a system to and from the ``dotted'' addresses as in 137.92.11.1 to 32 bit integer addresses are:

- unsigned long inet_addr (char * ptr)
- char *inet_ntoa (struct in_addr in)

The structure in_addr is used to store the IP address in network byte order. Its description is given below:

```
struct in_addr {
    unsigned long int s_addr;
}
```

The BSD library provides some functions for finding host names.

- char * gethostname (char * name, int size) - finds the ordinary host name.
- struct hostent * gethostbyname (char * name) - returns a pointer to a structure with two important fields: "char * h_name" which is the "official" network name of the host and "char * * h_addr_list" which is a list of TCP/IP addresses.

The structure hostent is used to store details about host information such as host name, list of IP addresses, etc. Its description is given below.

```
struct hostent {
   char * h_name; /* official name of host */
   char ** h_aliases; /* alias list */
   int h_addrtype; /* host address type */
   int h_length; /* length of address */
   char ** h_addr_list;  /* list of addresses */
}
```

To handle byte ordering for non-standard size integers, there are conversion functions

- htonl - host to network, long int
- htons - host to network, short int
- ntohl - network to host, long int
- ntohs - network to host, short int

The address of an IP service given is using a structure

```
struct sockaddr_in {
  short   sin_family;
  u_short sin_port;
```

```
    struct  in_addr sin_addr;
    char    sin_zero[8];
  }
```

Creating sockets and binding it to the application using it establish the communication between systems. There are system calls for creating a socket, binding it to a particular physical port, making it to listen to a port, accepting connection request from remote systems and connecting it to other remote systems. The list of system calls for the above-mentioned services are as follows:

- int socket (int family, int type, int protocol);
- int bind (int s, struct sockaddr * name, int namelen);
- int listen (int s, int backlog);
- int accept (int sockfd, struct sockaddr * name, int * namelen);
- int connect (int s, struct sockaddr * name, int namelen);

Two additional data transfer library calls, namely send() and recv(), are available if the sockets are connected. They correspond very closely to the read() and write() functions used for I/O on ordinary file descriptors.

- int send (int sd, char * buf, int len, int flags);
- int recv (int sd, char * buf, int len, int flags);

In both cases, sd is the socket descriptor.

6.2 Implementation Description

Application 1:

HEADER FILE	FUNCTION	DESCRIPTION	INPUT	OUTPUT	LOC
Header.h	-none-	Includes header library files and structure declarations	-none-	-none-	77
Password.h	Login_addnew	Used during logging in and signing-up	Option, mail address and password	Positive or negative reply code	130
	Search	Searches the given user's mail address to check its validity	Mail address	Positive or negative reply code	44
	Hashing1	Finds the equivalent unique location for mail address	Mail address	Unique memory location	6
	Hashing2	Used as like hashing1 during collision	Mail address	Unique memory location	6
	Double_hash	Called during collision, this in turn calls hashing2 to get a free location	Mail address	Unique memory location	17
	Double_hash_ search	Called during retrieving a double hashed	Mail address	Positive or negative	5

		mail address		reply.	
Smtp_client .h	Send_mail	Accepts the command from user and passes to SMTP server. Also sends message. And reads reply codes	Socket descriptor to communicate with the server	-none-	358
Smtp_server .h	Receive	Receives the commands from client and parses it and replies appropriate code. And receives messages and stores it in user's inbox	Socket descriptor to communicate with the client	-none-	611
Pop_client. h	Pop_client	Receives the request and sends it POP server. Reads back the reply code.	Socket descriptor to communicate with the server	-none-	213
	Read_long	Reads the messages and inbox , outbox, address book details	Mail address of user	-none-	45
Pop_server. h	Pop_server	Receives the commands and parses and sends reply codes. Required function is called	Socket descriptor to communicate with the client	-none-	375
	Status	Send the inbox details	socket descriptor, mail	-none-	175

			address, option, message number		
	Outbox	Reads the outbox details	Mail address of user	-none-	41
	Adbk	Reads , displays, writes the address book.	Mail address of user	-none-	43
	Retr_mail	Sends the message to client	Mail address, socket descriptor, message number	-none-	166
	Del_check	Checks the validity of message number for deletion	Mail address and message number	Positive or negative reply code	24
	Quit	Deletes the files marked for deletion before logging out.	Socket descriptor, mail address deletion numbers and deletion list	-none-	114
Smtp_addressbook.h	Remind_mail_send	Checks for the occurrence of birthdays or anniversaries on the next day.	-none- but uses address book of each users as input.	-none-	165

Data_send.h	Remind_send	Sends the reminding messages, mails to users	User mail address and message to be send	-none-	135

In addition to the above header files, the main files for smtp_client.h, smtp_server.h, pop_client.h, pop_server.h are also present that accounts for approximately 105 LOC.

Application 2:

HEADER FILE	FUNCTION	DESCRIPTION	INPUT	OUTPUT	LOC
Server.h	Register_client	Stores the IP address of clients in the network	-none-	-none-	110
Server.h	Do_service	Service the client and sent appropriate signals	App. Name, socket descript or IP address	Status of service	120
Server.h	Sends 1	Send, receive and process signal from client - B	App. Name	Socket descriptor	70
Clia.h	Get_gile_client	Establishes connection with request satisfying client and prepare to receive	App. Name to be received .	Status of receiving of file.	50

		application from it.			
Clib.h	Mini_server	Prepare for receive signal from server.	-none-	-none-	85
Clib.h	Child_handle	Search for the requested app. And send appropriate signal for the server.	Socket descript or	Status of service	70
Clib.h	ctoc	Establishes connection with requesting client and prepare to send application to it.	App. Name, location of app., permissi ons of the app.	Status of service	50
Search.h	Search	Searches for the app. And build the queue	App. Name	Location of app.	85
Search.h	Is_file_present	Checks for the presence of app. In the specified directory	App. Name, director y name	Success or failure value	85
Filter.h	Send_file_to_ client	Send the app. And the permission of it.	App. Name, socket descript or permissi on	Success or failure of transfer	35
Filter.h	Get_file	Receive the app. And the permission of it.	Socket descript or app. Name	Success or failure of transfer	35
Header.h	-None-	Includes library headers	-None-	-None-	22

In addition to the above header files, the files that contain the main() functions for the modules Client-A (clia.c), Client-B (cliab.c), Server (server.c) accounts for 350 LOC (~approx).

NUMERIC ORDER LIST OF REPLY CODES

211 System status, or system help reply
214 Help message
[Information on how to use the receiver or the meaning of a particular non-standard command; this reply is useful only to the human user]
220 <domain> Service ready
221 <domain> Service closing transmission channel
250 Requested mail action okay, completed
251 User not local; will forward to <forward-path>
354 Start mail input; end with <CRLF>.<CRLF>
421 <domain> Service not available, closing transmission channel [This may be a reply to any command if the service knows it must shut down]
450 Requested mail action not taken: mailbox unavailable [E.g., mailbox busy]
451 Requested action aborted: local error in processing
452 Requested action not taken: insufficient system storage
500 Syntax error, command unrecognized [This may include errors such as command line too long]
501 Syntax error in parameters or arguments
502 Command not implemented
503 Bad sequence of commands
504 Command parameter not implemented
550 Requested action not taken: mailbox unavailable
[E.g., mailbox not found, no access]
551 User not local; please try <forward-path>
552 Requested mail action aborted: exceeded storage allocation
553 Requested action not taken: mailbox name not allowed [E.g., mailbox syntax incorrect]
554 Transaction failed

CHAPTER 7

TESTING TECHNIQUES

Good software testing ensures the reliability of the software. The testing techniques that were followed when testing this software are as follows:
Software was tested from two different perspectives. They were:

- ❖ Internal program logic was exercised using "white-box" test case design techniques.

- ❖ Software requirements were exercised using "black-box" test case design techniques.

When the testing phase of the project began, the point of view on the software had been changed to "break" the software design test cases in a disciplined fashion and review the test cases that were created for thoroughness. White-box and black box testing was done to test the software.

7.1 White-Box Testing

Basis path testing, a widely used white-box testing, was followed as the testing technique. The steps that include when white box testing was done were as follows:

- ❖ All independent paths within a module were exercised once. Searching a user's login details in password file, checking the authentication matches for logging in, sending messages or data across network, displaying the passed message or data in user readable form, servicing more number of clients at a time by server, etc. were some of the independent paths present within the system.

- All logical decisions on their true and false sides were exercised.

- All loops present were exercised at their boundary conditions and also within their operational bounds.

- All internal data structures were exercised to ensure their validity.

7.2 Black-Box Testing

The black box testing was focused on functional requirements of the software. The steps that include when black box testing was done were as follows:

- Validity of all functions used was tested.

- System behavior and its performance were tested.

- Classes of input that make good test cases were determined.

- Boundaries of input data to functions were determined.

- Sensitive input values for the functions involved were determined.

The boundary of input data to the system calls must lie in positive number range. It is usually greater than 0. Providing a negative value for socket-descriptor to the above function tested this and the result was examined. The system produced error message indicating negative value for socket descriptor. Like this, the valid input range for every function was determined.

CHAPTER 8

TESTING STRATEGY

The testing strategy includes performing unit testing followed by system testing. The system testing includes security testing and stress testing.

8.1 Unit Testing

The performance unit testing is common for both applications.

In unit testing, the control flow is mainly exercised and verified that all statements have been executed at least once. Providing various test cases has tested the interfaces between the functions present in different modules.

8.2 System Testing

8.2.1 Security testing

This testing attempts to verify the protection mechanisms built into a system will protect it from improper penetration.

Application 1:

The user logging in is tested with protected password. This password file is checked for any possible information leak when opening in text editors. The possibilities of hacking the password is ruled out, as there is no option like forget password. And trying the possible combinations of six characters can only do it. Only the server can delete any user accounts, send reminding mails to all its users etc are checked. The security of mails of one user is checked by trying to open it through any other means from client through the network.

Application 2:

This testing attempts to verify the protection mechanisms built into a system will protect it from improper penetration. The user when logged as an ordinary user in the client system cannot make request for applications that are owned by root. Another rule is that, when the user logged as root in the client system ha has the rights to make request for any application owned by root or other users in the server as well as other client systems. The different cases dealt and results of the testing are follows.

LOG MODE	FILE PRESENT/ABSENT		OWNER	STATUS	RESULT
	SERVER	CLIENT-B			
User	Present	-	Root	Not OK	Success
User	Absent	Present	Root	Not OK	Success
User	Present	-	User	OK	Success
User	Absent	Present	User	OK	Success
Root	Present	-	Root	OK	Success
Root	Absent	Present	Root	OK	Success
Root	Present	-	User	OK	Success
Root	Absent	Present	User	OK	Success

8.2.2 Stress Testing

Stress testing executes a system in a manner that demands resource in abnormal quantity, frequency, or volume.

Application 1:

The stress testing is performed on this system by making the software to demand more memory space. Space required to build message files is specified as a system environment variable. Around twenty users accounts were created. And an average of twenty-five messages for each user is send and tested. The outcome of this testing is that there can't be more than twenty-seven users for a password file of length sixty-one. This provides more stress on the POP server and thus limits the system from requesting users as the password file is filled. This is used as the upper limit for number of uses and can be increased by enlarging the size of password file. Increasing the key range of hashing function does this.

Application 2:

The stress testing is performed on this system by making the software to demand more memory space. Space required to build circular queue is specified as a system environment variable. Around 15,00,000 character pointers that constitute the circular queue were allocated and the system is tested for its performance. When more character pointers are allo9cated above and above -said limit, the performance of the system gets decelerated. This provides more stress on the system and thus limits the system from requesting more resources. This array is used for the logical data structure used for searching I.e., circular queue.

CHAPTER 9

FUTURE ENHANCEMENTS

The future enhancement for the Mail Server software is to implement a well-designed user interface for POP client, transfer attachments through SMTP server and other important enhancements are facilities like change password for user accounts, folder options, filters in his inbox.

The future enhancement for the Distributed Application Management software is to implement a well-designed user interface, another enrichment is to invoke remote applications related to X windows from shell prompt itself.

The implementation of application management using cache buffer is another enhancement can be done to the software in future time. This includes storing of frequently used application that was not in server but in any client in the network into the memory of the server.

The usage of mobile agent-like server is yet another prosperity that can be done in upcoming period. This prosperity is an alternative to the concurrent server used in this software.

CONCLUSION

The key objective of the project is to provide users with mailing service and to use the application present in the network environment. In the Mail Server application user scheduling, message read status and important mail informer adds the service provided to users. Thus the project fulfills the above objectives and ensures security in both applications. The performance of the system complies with its surroundings and thus it makes the application management more versatile. The scalability of the mail service application can also be enhanced makes it more adaptive easy to up-grade. The perceptibility of the second application is suited for decentralized systems and its application.

BIBLIOGRAPHY

Stevens, Richard W. Unix Network Programming. New Delhi: Prentice Hall of India, 2002.

Stevens, Richard W. Advanced Programming in the Unix Environment. New Delhi: Addison Wesley Longman, 2001.

Gandhi, Meeta, et. al. The 'C' Odyssey: Unix - The Open Boundless C. New Delhi: BPB Publications, 1992.

Pressman, Roger S. Software Engineering: A Practitioner's Approach. New Delhi: McGraw-Hill, 2001.

Tenenbaum, Aaron M, et. al. Data Structures Using C and C++. New Delhi: Prentice Hall Of India, 2000.

Internet websites:

http://pandonia.canberra.edu.au

http://www.ecst.csuchico.edu

Www.cis.ohio-state.edu/cgi-bin/rfc/

P-814.