LAN TELEPHONY

BY

NIRAV UCHAT

HARDIK SHAH

ROHAN DESAI

PARAG SAVANT

UNDER THE GUIDANCE OF

Prof : S.M.Toraskar

AND

Co-Guide

Prof. Mahesh Deshpande

DEPARTMENT OF COMPUTER ENGINEERING

PARSHVANATH COLLEGE OF ENGINEERING

UNIVERSITY OF MUMBAI

(2003-2004)

ACKNOWLEDGEMENT

We owe Prof. Mahesh Deshpande debt of gratitude without whose help, expert guidance and invaluable co-operation, this project would not have been possible.

We are also greatly indebted to Prof. S.M.Toraskar H.O.D. (Computer Department) who always encourages us during our project.

We make this opportunity to thank all those who helped us in this project. This project has been introduced as a part of our curriculum and has been a very useful source in amalgamating our theoretical knowledge with the practical knowledge.

It would have been very difficult for us to reach at this stage without their guidance and encouragement throughout our tenure.

NIRAV UCHAT
HARDIK SHAH
ROHAN DESAI
PARAG SAVANT

PARSHVANATH CHARITABLE TRUST'S

Parshvanath College of Engineering

Kasar-Vadavali, Ghod Bunder Road, Thane-400 601.

Phone : 541 22 02 / 542 79 53

Ref. : ..... Date : .....

CERTIFICATE

The project on LAN TELEPHONY is submitted by:

1) MR. NIRAV UCHAT co200051

2) MR. HARDIK SHAH co200088

3) MR. ROHAN DESAI co200040

4) MR. PARAG SAVANT co200041

In partial fulfillment of the degree of B.E. in Computer Engineering is approved by

Co-Guide	Examiner (1)	Examiner (2)
(Prof. Mahesh Deshpande)	( )	( )

Head of Computer Dept	Principal
(Prof. S. M. Toraskar)	(Prof. R. S. Jhagirdar)

Date: _____

CONTENTS

List of Figures

List of Flowchart

CHAPTER 1

INTRODUCTION

The local-area network is ripe for telephony, there is no doubt about it. If you are looking to save money with both internal and external communications, trying to add contact features and improve the 'touch' of the individuals sitting at the endpoints of your LAN, then you are ready for LAN telephony solutions. And why not? The long-term investment is sound, and the short term benefits are exacerbated by the simplicity with which many of these solutions can be implemented.

To send text message through LAN was always an advantage and since technology has advanced, sending voice through LAN is also possible using LAN telephony. Imagine two employees sitting at different ends of an office wanting to communicate, the only option they had was using the existing PSTN network. But now through LAN telephony, they can communicate sitting right at their desktops using existing LAN network.

A LAN-based telephony system can offer many clear benefits, including lower costs, simpler configuration and modification, and more efficient messaging by combining e-mail and voicemail in one system.

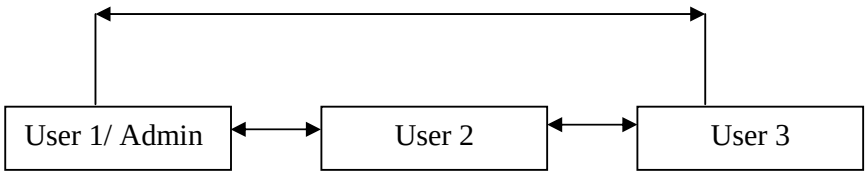


Fig 1.1 – Communication between users

(1) User 1 will act as Admin/user . This user will have control over whole database

(2) All users who are registered for the Lan Telephony service has access to whole database

(3) Any user can place call to other user by simply giving his/her name.

CHAPTER 2

REVIEW OF LITERATURE

2.1 Need Of Project And Description

2.1.1 Statement of Problem

The plain old telephone service (POTS), as it has been known for most of the twentieth century is giving way to modernization, service integration and convergence of two fundamentally different types of technologies: - voice and data. Changing needs in both the business and the residential market segments require new telephony features and capabilities that are neither simple nor economical for service providers to create on top of the PSTN infrastructure.

We propose to develop a system called LAN TELEPHONY which will provide an alternative solution to aid the people within an organization to communicate with each other actively without using the PSTN system. This would drastically reduce the telecommunication costs of any organizations as our system does not need PSTN as the backbone for communication. Our system requires multimedia computers to be connected through a LAN network. Our system will also support sending voice messages to the users who are not available for communication at that instant. These voice messages will be heard by the user as per his time availability.

2.1.2 Why does PSTN need to change?

The PSTN is collection of all the switching and networking equipments that belongs to the carriers, which are involved in providing telephone service. Voice from the telephone set in our homes to the central office exists either in analog or digital form. However once it reaches the central office voice exists in digital form of PSTN – on TDM channels of 64 Kbps each, which carry PCM samples. The QoS of speech has never been a negotiable parameter but this QoS comes at huge cost of service providers who need to tie up the network resources for the duration of entire call.

Even simple traffic analysis shows that the bandwidth is not utilized optimally. Again in the recent years enormous appetite for web access has caused noticeable problem in TDM network and resources. Clearly this is an uneconomical proposition for a telephone companies who find it difficult to satisfy growing customer needs in current network infrastructure. The culprit has been lack of integration between voice and data services.

A solution to this problem is ISDN circuits, but part of the problem with convergence in the current infrastructure is that signaling at the local access for basic service is hardware based and tailored for voice telephony, with some fundamental considerations for point to point video for those who can afford it, which is a severe limitation .The second aspect is the difficulty for bringing interactive multimedia applications such as distance learning.

Thus we propose to build a robust and scalable converged multimedia network ensuring all kinds of services expected by the users. Also it will be flexible enough for future up gradation.

2.2 Current System And It's Drawbacks

2.2.1 Evaluation of Existing PSTN System

The oldest and largest telecommunications network in existence is the public switched telephone network (PSTN) which has in excess of 700 million subscribers.For a long time, the PSTN was the only bearer network available for telephony. Today, many people choose the mobile telephone for their calls. Other bearer networks for voice transmission include integrated service digital network (ISDN), asynchronous transfer mode (ATM), frame relay and the Internet.

The PSTN's primary characteristics:

- analog access, 300-3,400 Hz;
- circuit-switched duplex connection;
- switched bandwidth, 64 kbit/s, or 300-3,400 Hz for analog exchanges;
- immobility or, at best, very limited mobility; and
- many functions in common with another bearer network: N-ISDN

The advantages of existing PSTN system are:

- Quality of sound is very good.
- Error probability is quite low.
- Reliable connection oriented service.
- Limited facilities like call forwarding, multipoint conferencing, etc. are possible.

The major disadvantages of PSTN system are:

- Because of its age-old technology it is not possible to add features according to individual client.
- The individual contact no. (or extensions) is to be remembered which is quite difficult in case of large organization.
- In case an organization is expanded on acres of land and personnel has a visiting job, to contact that person requires forwarding call from department to department till he is located. This is a quite time consuming and costly process.
- With existing system it is quite difficult and costly to add features like video conferencing, call recording, etc.
- With the existing system it is quite difficult to add new connections in an ad-hoc manner.
- Since it is a centralized connection oriented system any failure in between switch that routes the two endpoints may cause the connection be broken in between a conversation.
- Since it is a dedicated network it cannot be used for any other transmission like files, documents, images etc.

2.3 OSI ISO layer, Socket & Sound Programming

In 1978, the International Standards Organization (ISO) introduced the ISO model for Open Systems Interconnect (OSI) as a first step toward international standardization of the various protocols required for network communication.

2.3.1 The OSI ISO model

It was designed to establish data communications standards that would promote multi-vendor interoperability.

It consists of seven layers, with a specific set of network functions allocated to each layer, and guidelines for implementation of the interfaces between layers.

Details a specific set of protocols and interfaces to implement at each layer. So far, only the lowest four layers have been explicitly defined. The upper layers, and their interfaces to the lower ones, have not yet been completed. The overall model has become the basis for the government's required standard environment, GOSIP, beginning in August 1990.

Each layer of the OSI model can be viewed as an independent module. You may (theoretically) substitute one protocol for another at the same layer without affecting the operation of layers above or below.

In addition to explicitly defining protocols and interfaces at selected layers, the OSI model also serves as a concept, providing a reference for how data communication should take place. It provides a common basis for the coordination of standards development for the purpose of systems interconnection, while allowing existing standards and architectures to be placed in perspective within the overall reference model.

The principles that led to the creation of seven layers are:

- A layer should be created only where a different level of abstraction is required.
- Each layer should perform a well defined function.
- The function of each layer should be chosen with an eye toward defining internationally standardized protocols.

- The layer boundaries should be chosen to minimize the information flow across the interfaces.

- The number of layers should be large enough that distinct functions need not be thrown together in the same layer out of necessity, and small enough that the architecture does not become unwieldy.

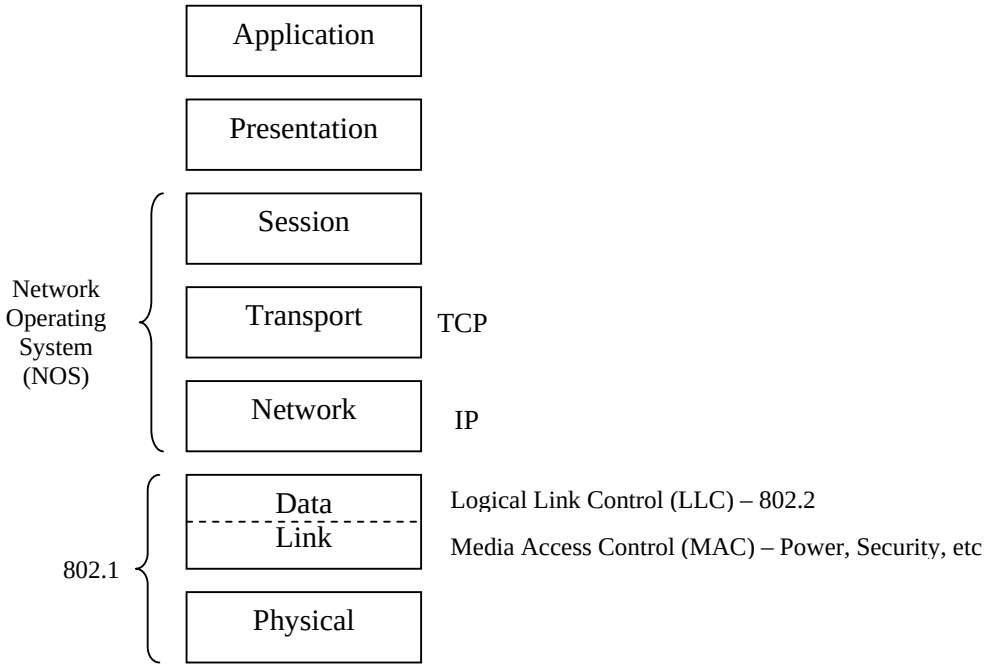


Fig 2.1 OSI ISO Reference model

2.3.2 Socket Programming

Socket is a way to speak to other programs using standard Unix file descriptors.
Unix programs do any sort of I/O, they do it by reading or writing to a file descriptor. A file descriptor is simply an integer associated with an open file. But that file can be a network connection, a FIFO, a pipe, a terminal, a real on-the-disk file, or just about anything else. Everything in Unix *is* a file! So when one wants to communicate with another program over the Internet we are going to do it through a file descriptor.

Where do one gets this file descriptor for network communication ? One makes a call to the socket() system routine. It returns the socket descriptor, and we communicate through it using the specialized send() and recv() socket calls.[1]

The communication process can be visualized from below diagram:

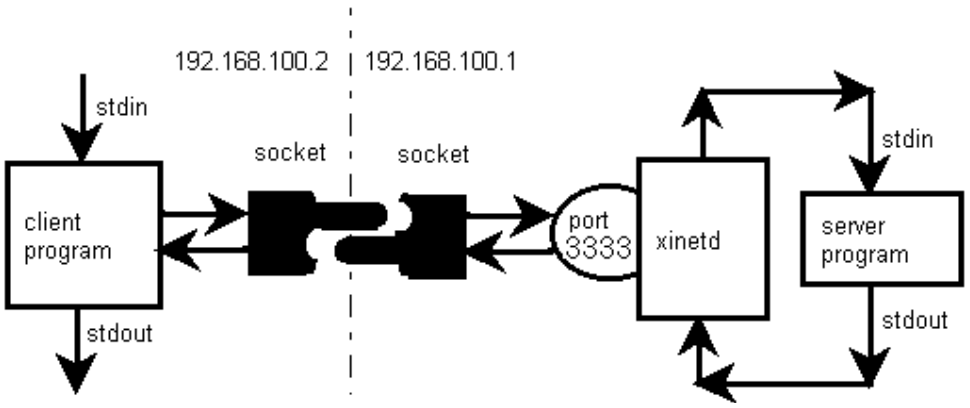


Fig 2.2 Overview of socket connection

Socket calls used in UNIX system are as follows

1. socket(): Socket Creation Using socket ()

#include <sys/types.h>

#include <sys/socket.h>

int socket(int af, int type, int protocol)

socket() is very similar to socketpair() except that only one socket is created instead of two. This is most commonly used if the process you wish to communicate with is not a child process. The af, type, and protocol fields are used just as in the socketpair() system call. On success, a file descriptor to the socket is returned. On failure, -1 is returned and errno describes the problem.

2. bind(): Giving a Socket a Name

#include <sys/types.h>

#include <sys/socket.h>
int bind(int s, struct sockaddr *name, int namelen)

Recall that, using socketpair(), sockets could only be shared between parent and child processes or children of the same parent. With a name attached to the socket, any process on the system can describe (and use) it. In a call to bind(), s is the file descriptor for the socket, obtained from the call to socket(). Name is a pointer to a structure of type sockaddr. If the address family is AF_UNIX (as specified when the socket is created), the structure is defined as follows:

```
struct sockaddr
{  u_short sa_family;
   char    sa_data[14];};
```

name.sa_family should be AF_UNIX. name.sa_data should contain up to 14 bytes of a file name which will be assigned to the socket. Namelen gives the actual length of name, that is, the length of the initialized contents of the data structure.
A value of 0 is return on success. On failure, -1 is returned with errno describing the error.

3. connect(): Specifying a Remote Socket
#include <sys/types.h>
#include <sys/socket.h>
int connect(int s, struct sockaddr *name, int namelen)

The bind () call only allows specification of a local address. To specify the remote side of an address connection the connect() call is used. In the call to connect, s is the file descriptor for the socket. Name is a pointer to a structure of type sockaddr:

```
struct sockaddr {
   u_short sa_family;
   char    sa_data[14];
};
```

As with the bind () system call, name.sa_family should be AF_UNIX. name.sa_data should contain up to 14 bytes of a file name which will be assigned to the socket. namelen gives the actual length of name. A return value of 0 indicates success, while a value of -1 indicates failure with errno describing the error.

4. sendto(): Sending to a Named Socket

int sendto(int s, char *msg, int len, int flags, struct sockaddr *to, int tolen)

This function allows a message msg of length len to be sent on a socket with descriptor s to the socket named by to and tolen, where tolen is the actual length of to. flags will always be zero for our purposes. The number of characters sent is the return value of the function. On error, -1 is returned and errno describes the error.

5. recvfrom(): Receiving on a Named Socket

#include <sys/types.h>

#include <sys/socket.h>

int recvfrom(int s, char *msg, int len, int flags, struct sockaddr *from, int *fromlen)

This function allows a message msg of maximum length len to be read from a socket with descriptor s from the socket named by from and fromlen, where fromlen is the actual length of from. The number of characters actually read from the socket is the return value of the function. On error, -1 is returned and errno describes the error. Flags may be 0, or may specify MSG_PEEK to examine a message without actually receiving it from the queue. If no message is available to be read, the process will suspend waiting for one unless the socket is set to nonblocking mode. The system I/O call read () can also be used to read data from a socket

6. close(): Disposing of a Socket

#include <stdio.h>

void close(int s).

The I/O call close () will close the socket descriptors just as it closes any open file descriptor.

7. listen(): Make a Socket a Listen-only Connection Endpoint

#include <sys/types.h>

#include <sys/socket.h>

int listen(int s, int backlog)

listen() establishes the socket as a passive endpoint of a connection. It does not suspend process execution. No messages can be sent through this socket. Incoming messages can be received is the file descriptor associated with the socket created using the socket() system call. Backlog is the size of the queue of waiting requests while the server is busy with a service request. The current system-imposed maximum value is 5.0 is returned on success, -1 on error with errno indicating the problem.

8. accept(): Connection Establishment by Server

#include <sys/types.h>

#include <sys/socket.h>

int accept(int sockfd, struct sockaddr *name, int *namelen)

The accept() call establishes a client-server connection on the server side. (The client requests the connection using the connect() system call.) The server must have created the socket using socket(), given the socket a name using bind(), and established a listen queue using listen().sockfd is the socket file descriptor returned from the socket() system call. name is a pointer to a structure of type sockaddr as described above

```
struct sockaddr {
   u_short sa_family;
   char    sa_data[14];
};
```

Upon successful return from accept(), this structure will contain the protocol address of the client's socket.The data area pointed to by namelen should be initialized to the actual length of name. Upon successful return from accept, the data area pointed to by namelen will contain the actual length of the protocol address of the client's socket.

If successful, accept() creates a new socket of the same family, type, and protocol as sockfd. The file descriptor for this new socket is the return value of accept(). This new socket is used for all communications with the client. If there is no client connection

request waiting, accept() will block until a client request is queued. accept() will fail mainly if sockfd is not a file descriptor for a socket or if the socket type is not SOCK_STREAM. In this case, accept() returns the value -1 and errno describes the problem.

9. send () & recv (): Data Transfer over Connected Sockets

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.

#include <sys/types.h>
#include <sys/socket.h>
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.

For send(), buf points to a buffer containing the data to be sent, len is the length of the data and flags will usually be 0. The return value is the number of bytes sent if successful. If not successful, -1 is returned and errno describes the error. For recv(), buf points to a data area into which the received data is copied, len is the size of this data area in bytes, and flags is usually either 0 or set to MSG_PEEK if the received data is to be retained in the system after it is received. The return value is the number of bytes received if successful. If not successful, -1 is returned and errno describes the error.[8][6]

2.3.3 Connection in TCP/IP

TCP provides sequenced, reliable, bi-directional, connection-based byte streams with transparent retransmission. In English, TCP breaks your messages up into chunks (not greater in size than 64KB) and ensures that all the chunks get to the destination without error and in the correct order. Being connection-based, a virtual connection has to be set up between one network entity and the other before they can

communicate. UDP provides connectionless, unreliable transfer of messages (of a fixed maximum length).

To allow applications to communicate with each other, either on the same machine (using loopback) or across different hosts, each application must be individually addressable. TCP/IP addresses consist of two parts--an IP address to identify the machine and a port number to identify particular applications running on that machine. The addresses are normally given in either the ``dotted-quad" notation (i.e., 127.0.0.1) or as a host name (foobar.bundy.org). The system can use either the /etc/hosts file or the *Domain Name Service* (DNS) (if available) to translate host names to host addresses. Port numbers range from 1 upwards. Ports between 1 and IPPORT_RESERVED (defined in /usr/include/netinet/in.h--typically 1024) are reserved for system use (i.e., you must be root to create a server to bind to these ports). The simplest network applications follow the *client-server* model. A server process waits for a client process to connect to it. When the connection is established, the server performs some task on behalf of the client and then usually the connection is broken. [7][11]

Using the Socket Interface

The most popular method of TCP/IP programming is to use the *BSD socket interface*. With this, network endpoints (IP address and port number) are represented as *sockets*.

The socket interprocess communication (IPC) facilities (introduced with 4.2BSD) were designed to allow network-based applications to be constructed independently of the underlying communication facilities.

Creating a Server Application

To create a server application using the BSD interface, you must follow these steps:

1. Create a new socket by typing: socket().
2. *bind* an address (IP address and port number) to the socket by typing: bind. This step identifies the server so that the client knows where to go.
3. *listen* for new connection requests on the socket by typing: listen().

4. *accept* new connections by typing: accept().

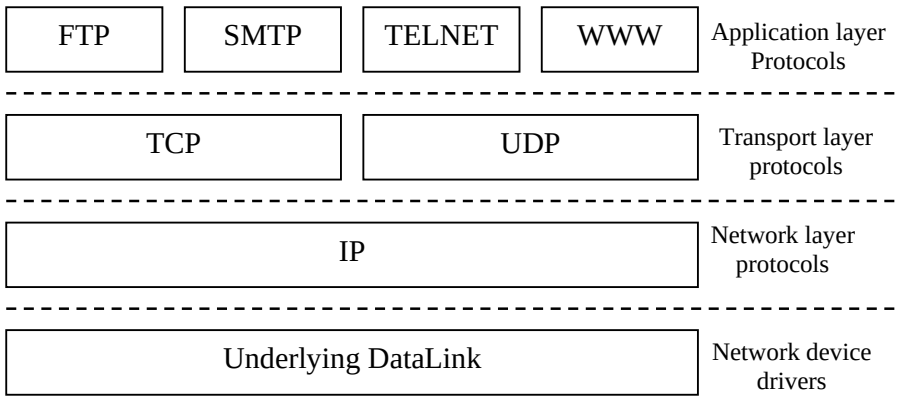
Creating the Corresponding Client

To start the client, you must provide two command-line arguments: the host name or address of the machine the server is running on and the port number the server is bound to. Obviously, the server must be running before any client can connect to it.

The first command-line argument is first assumed to be a host name for the purposes of finding the server's address. If this fails, it is then assumed to be a dotted-quad IP address. If this also fails, the client cannot resolve the server's address and will not be able to contact it.

Having located the server, an address structure is created for the client socket. No explicit call to **bind()** is needed here, as the **connect()** call handles all of this.

Once the connect() returns successfully, a duplex connection has been established. Like the server, the client can now use read() and write() calls to receive data on the connection.[12] [10]



FTP	File Transfer Protocol
SMTP	Simple Mail Transfer Protocol
TCP	Transmission control protocol
UDP	User datagram protocol
ICMP	Internet control message protocol
IP	Internet Protocol
arp	adress resolution protocol

FIG 2.3 TCP/IP Reference model

2.3.4 Programming /dev/dsp

/dev/dsp is the digital sampling and digital recording device, and probably the most important for multimedia applications. Writing to the device accesses the D/A converter to produce sound. Reading the device activates the A/D converter for sound recording and analysis. The name *DSP* comes from the term *digital signal processor*, a specialized processor chip optimized for digital signal analysis. Sound cards may use a dedicated DSP chip, or may implement the functions with a number of discrete devices. Other terms that may be used for this device are *digitized voice* and *PCM*. Some sounds cards provide more than one digital sampling device; in this case a second device is available as */dev/dsp1*. Unless noted otherwise, this device operates in the same manner as */dev/dsp*.

The DSP device is really two devices in one. Opening for read-only access allows you to use the A/D converter for sound input. Opening for write only will access the D/A converter for sound output. Generally speaking you should open the device either for read only or for write only. It is possible to perform both read and write on the device, albeit with some restrictions; this will be covered in a later section.

Only one process can have the DSP device open at a time. Attempts by another process to open it will fail with an error code of EBUSY.

Reading from the DSP device returns digital sound samples obtained from the A/D converter. Analog data is converted to digital samples by the analog to digital converter under control of the kernel sound driver and stored in a buffer internal to the kernel. When an application program invokes the read system call, the data is transferred to the calling program's data buffer. It is important to understand that the sampling rate is dependent on the kernel driver, and not the speed at which the application program reads it.[3] [4]

2.3.4.1 Code to access /dev/dsp

```
if ((speaker=open("/dev/dsp",O_WRONLY)) == -1)
{       perror("\nSpeaker Not Configured");
        printf("\nSystem Requirement Not Matched !!");
        printf("\nPlease Free The Resource !");
        printf("\nThank You");
        exit(1);
}
if ((mic=open("/dev/dsp",O_RDONLY)) == -1)
{       perror("MIC Not Configured");
        printf("System Requirement Not Matched !!");
        printf("\nPlease Free The Resource !");
        printf("\nThank You");
        exit(1);
 }
```

In above example speaker and mic are file descriptor which point to the buffer. Speaker is configured for writing to /dec/dsp and mic is configured for reading from /dev/dsp. [14][13]

2.4 Survey and Case Study of Similar Systems

2.4.1 Study Of MTNL

PSTN (public switched telephone network) is the system currently prevailing in the market. MTNL and VSNL are two of the most commonly used services coming under PSTN. MTNL (Mahanagar Telephone Nigam Limited) is the company which uses PSTN. It works on the principle of sending voice in digital form from one terminal to another.

Main factors taken in to consideration are:

Reliability:

Service provided by MTNL is very reliable. It uses modern exchange for call handling.

Cost:

As MTNL is a service provider, they charge for each call. In an organization if each employee make 10 calls a day then the overall cost is going to be very high.

Additional Facilities:

Since PSTN is used for voice transmission, File, Data, Image sharing is not possible using PSTN network. It provides only basic call interface.

Video Conferencing:

Since PSTN uses low bandwidth, Video conferencing is not possible using existing MTNL line. But now a day they provide these facilities, but customer has to pay extra money for that.

2.4.2 Study of SKYPE

Skype is a Internet Phone application currently going through a free beta-test, and generating a lot of buzz. It's been said that Internet based voice calling will replace

traditional telephone networks, the only question being how long it will take. Skype could be one of the applications that make this a reality. [15]

SKYPE is currently under development by the people, who developed peer-to-peer sharing software KAZZA. It gives following facilities

- Voice over IP
- Database to store user information
- Search engine for searching the registered user
- Miss call list
- Dialed call list
- Received call list
- Easy user interface
- Intelligent routing
- Security
- Global decentralized user directory
- No NAPT routing problem
- User Picture Display
- Incoming and outgoing sound alert

2.4.3 Study of PicoPhone

PicoPhone is a simple Internet phone application with chat. All the Internet phones around are quite heavy and use complicated protocols, such as H.323, which has problems with NAPT routers. PicoPhone uses a simple UDP-based protocol, which works very well with NAPT. The program accepts connections on UDP port 11676 and makes connection to that port. Optionally a port number can follow the address (the colon is used as a separator), if the connection has to be made to another port (the NAPT router on the receiving side should convert the port number to the default 11676). PicoPhone allows multiple concurrent calls, but the audio device should allow multiple output streams to be opened simultaneously.

On incoming calls the program plays the file ringin.wav, which should be present in the default windows location for wav files or in the same directory of PicoPhone. It gives following facilities

- Voice Compression option
- Received call log file
- Voice conferencing
- Chatting
- Offline mode
- DNS option (Only name no IP Address)

2.5 Software Requirement Specification

2.5.1 Introduction:

Goals and Objectives of Software:

- Voice Conference between two people working on lan network.
- Leaving voice mails in case if the callee is not available(i.e.e not logged in) for call.

Software Project Constraints:

- A user who has not registered cannot login.
- No other user except the administrator will have the right to use the administrator services [protection done using admin password and Pentium based local boot linux terminal ip]
- All the entries that correspond to a user should get deleted from the respective files when a user us deleted using ADD-ON utilities.
- When two users are communication no third user can interfere or stop their communication.
- The communication can be closed by client or rejected by a server.

2.5.2 Information Description:

The system accepts a request from a caller who is in client mode of operation and sends it to the callee using the latter's socket name created by the server process.

An administrator who operates one pre-assigned terminal (Pentium based local boot(Linux terminal) will only be able to register a new user. He administrator will assign a username and password and a port to a new user. The registered user will now be able to login from any terminal using this username and password. The caller will be able to initiate a call at any time and his call will reach the callee which in turn will be in server mode.

The connection between two people will be handled in such a way that no third person will interfere in it. The callee will be given indication of a call even if he is not working on the application windows, say by beep sounds. The callee will have the freedom to reject a call if he decides no to talk to the caller. The system will save a voice mail in callee's mailbox when the callee is offline.

The callee will be in server mode (or waiting mode) and when a caller calls he will hear beep sounds an will have to make a choice to accept or reject a call. A rejected call will be indicated to the caller by a Display Message. A call accepted will result into communication using headset (headphone and microphone). The caller and the callee should be physically present at each terminal (Pentium based local boot linux Terminal) for voice communication. Appropriate error should be displayed when any constraint is ruled out.

2.5.3 Functional Description:

To achieve such a system as described above we have to partition the whole project into different functions. The functions are as follows.

- A function which takes information from a new user who registers for the first time. This function will assign a username and password and a port number. This function should be only executed by the user who is an administrator. The following files would be updated.

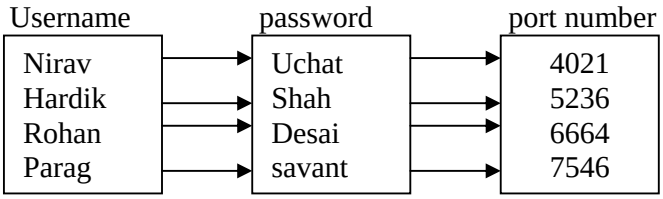


Fig 2.4 Entry in Master Database

Let us call this function as the registration function.

- A function which allows the user to become an active user by logging to the system. The user should be a registered user and so the function will check for validity of username, password (one to one correspondence). This function will also check for voice mails. Let us call this function as LOGIN FUNCTION.

- A function which helps in initiating a call to a caller. This function at the server side will create a SOCKET, which is just an O.S. resource assigned to the server process. The server process will give the socket a name. The server process then waits for a client to connect to a named socket. The clients side of a socket base system is more straight forward, The client creaks an unnamed socket by calling socket. It then calls connect to establish a connection with the server by using the server's named socket as an address. Let us call this module as CALL Funciton.

- A function that displays various add-on utilities like delete voice mails, to update various files for space purpose.

2.5.4 Behavioral Description:

The software will behave properly in heavy load conditions. If a user is deleted Then there shall be a module that will Take care of deletion of all the selected items from files.

2.5.5 Validation:

The software will contain validations like username and password should not be blank, administrator password is required to create new users, the user should be registered for placing a call, the length of the username and password should not exceed 8 characters, only 15 voice mails can be stored in any user's login, both the caller and the callee should be online if they want to communicate or else they can send voice-mails, for using the LAN TELEPHONY system the user should be a registered user.

CHAPTER 3

SYSTEM DESIGN

3.1 Module Wise Explanation

To solve the redundant cost of maintaining PSTN system we proposed to develop LAN telephony system. This method digitizes voice by sampling (using full duplex sound card) and then sends them in packets (Small chunk of data with added information) using socket programming.

The proposed system will consists of voice communication between two users. In this system every user who wants to communicate will have to login. The administrator authorizes the user request if he is already registered, else the registration process will take care of the user registration. Once the user is registered he can make a request to any other registered user. Once the user's request is approved, communication between the two users is initiated successfully. If the targeted user is not available then the requester can leave a voice message for targeted user from main menu.
When user runs the program, a five option menu is displayed.

1. Register
2. Login
3. Call
4. Send Voice Mail
5. Read Voice Mail
6. Logout
7. Exit

In this proposed system the key concept of communication between the two users is accomplished using socket programming.

So finally our System will use

1. Socket Programming for code development
2. Existing LAN network

3.1.1 Registration Module:

Files Used: masterdb.lst , userpass.lst , port.lst

This module is used for registering / creating new users for the LAN TELEPHONY SYSTEM.

Only the terminal which is acting as the server is authorized for using this module.

Say if the terminal having IP 192.168.6.34 has been configured as server then only this terminal will have access to the registration module.

Also for gaining access to this module the user needs to be an administrator.

Once the user is registered, its entry goes in the following files

1) masterdb.lst : User name is stored in this file.
2) userpass.lst : Password for the new user is stored in this file.
3) port.lst : Port No. will be stored in this file.

All the entries in the above files will have one to one correspondence.

The following directory structure will be created for the user once he is registered.

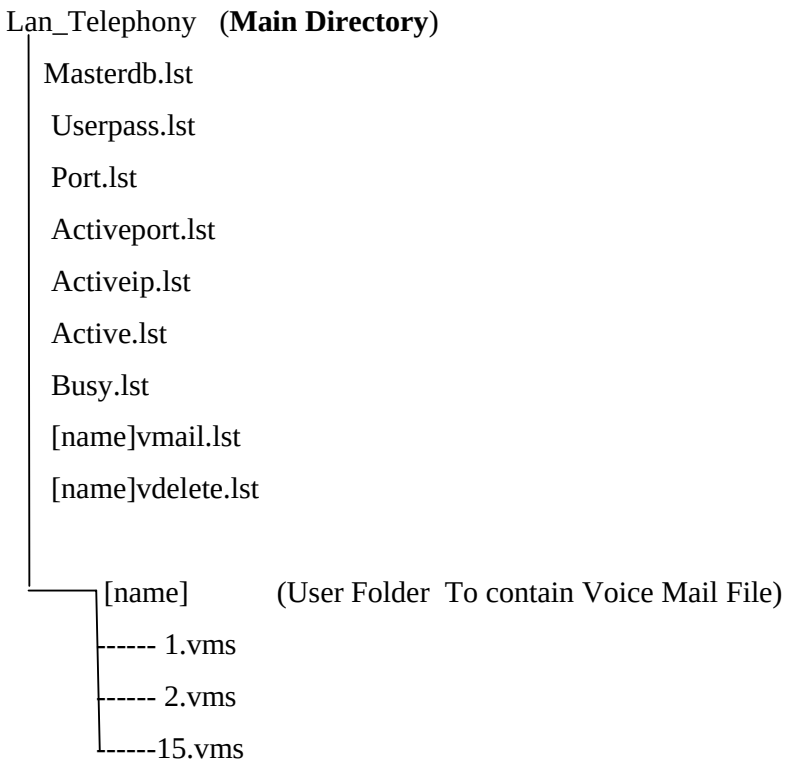


Fig 3.1 Directory structure

Also [name]vmail.lst and [name]vdelete.lst files will be created.

Validations:
Username should not be more than 8 characters.
Passwords should not be more than 8 characters.
Port should be greater than 1024 and 4 digits in length.

Initially the username is checked against the entries in the masterdb.lst. If a match is found then a message is displayed saying “User Already Exists”.

Activity Performed By this module is as follows:

1. Accepting New User Name and password
2. Check for validity of user
3. Accept Port number
4. Make directory for that user to store voice mail files (Up to 15)
5. make [name]vmail.lst and [name]vdelete.lst in main directory
6. User is ready to place call

Flowchart for registration module:

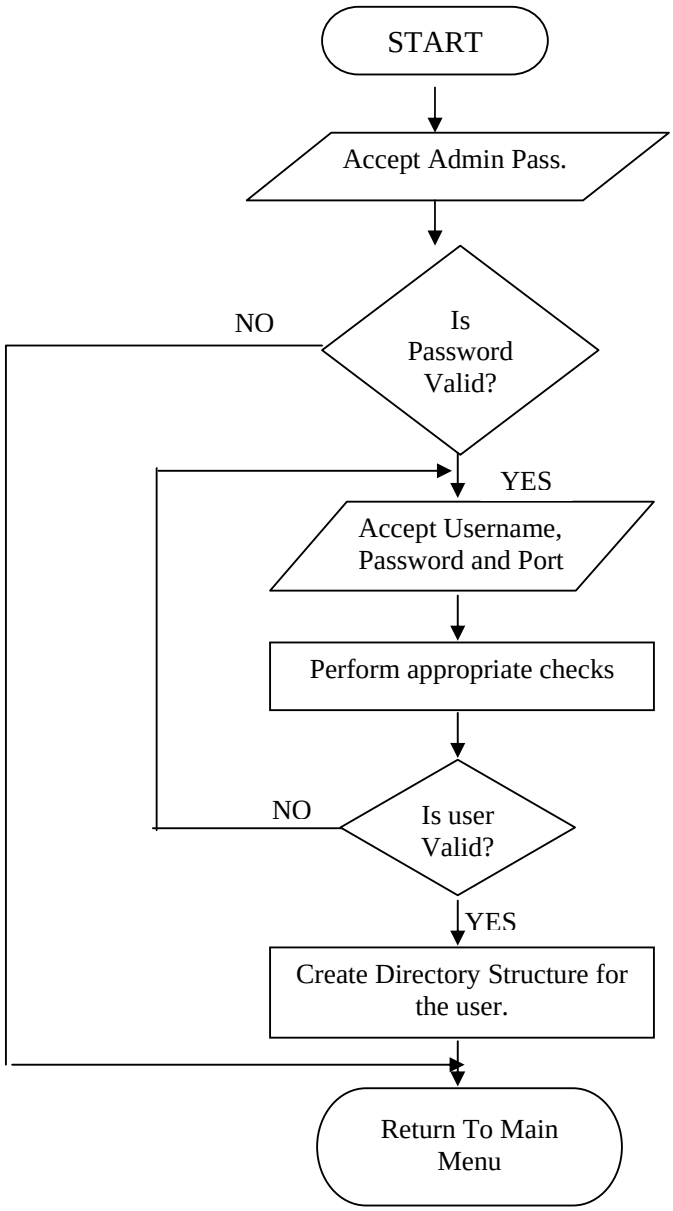


Fig 3.2 Flowchart of Registration Module

3.1.2 Login Module:

Files used: active.lst, activeip.lst, activeport.lst

This module is used for authorizing the registered users for using the services of LAN TELEPHONY.

Once the user has logged in the entries are made in the following files.

1) active.lst : Active user's username is stored in this file.
2) activeip.lst : Active user's IP (of the terminal) is stored in this file.
3) activeport.lst : Port for the corresponding user is stored in this file.

All the above files have one to one correspondence.

Also, once the user has logged in ,a check is made for his vmails by checking the [name]vmail.lst file of that user.

Information such as senders name, date, time and voice mail file name are stored in this file.

Validations:

Username should not be more than 8 characters.

Passwords should not be more than 8 characters.

Activity performed by this module is as follows:

1. Accept username
2. Accept Password
3. check validity of user (Using validity() function)
4. check [name]vmail.lst for checking voice mail status if validity() returns positive value.
5. set user as curren_user for that machine and set masterlock to 1
   (current_user and masterlock are variable used in program)

Flowchart of Login Module:

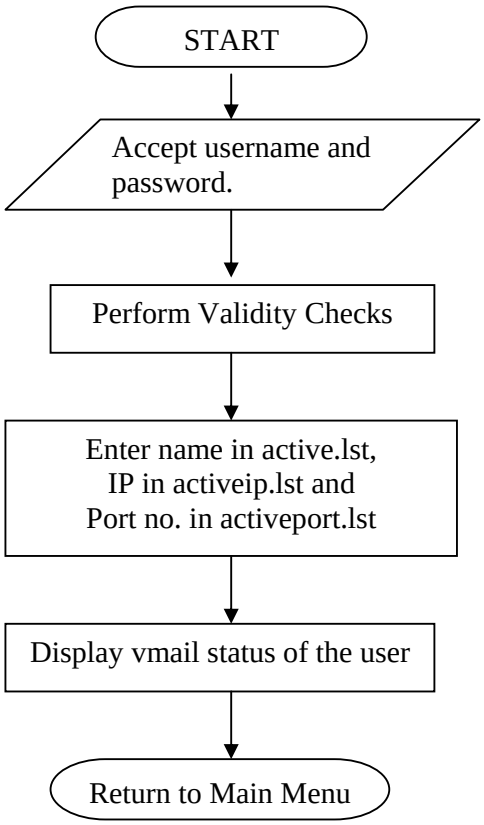


Fig 3.3 Flowchart of Login Module

3.1.3 Call Module:

Files Used: masterdb.lst, active.lst, activeip.lst, activeport.lst

This module enables communication between two logged in users.

When the CALL option is selected a check is made for the receivers name in the masterdb.lst to determine whether the user is registered or not. If not then appropriate error message is displayed. After that, a check is made in the active.lst to see whether the user is an active user.

For the connection to be established the target user should be in server mode.

Once the target user is found in server mode the call initiator (client) takes the entries from activeip.lst and activeport.lst and establishes the connection.

Server Mode:

Server Mode is waiting mode (it waits for a client to make a request for call).

To enter this mode the user has to type (ctrl +B)

When the call initiator makes the call, the target user (which is in server mode) gets 2 options starting with 5 beeps from speaker.

1. Accept the Call : This option will initiate communication between the two users. The voice communication is possible by opening and closing "/dev/dsp".

2. Reject the Call : This option will send a message to the client that "Server Has Rejected the Connection".

Validation:

1. There must be loged in user from that machine
2. Receiver name should not be greater that 8 character

Activity Performed By this module are as follows

1. To check the receiver name is valid and it is currently active.

2. check whether the receiver is in server mode or not
3. Establish a call if receiver is in server mode.

Flowchart for call module:

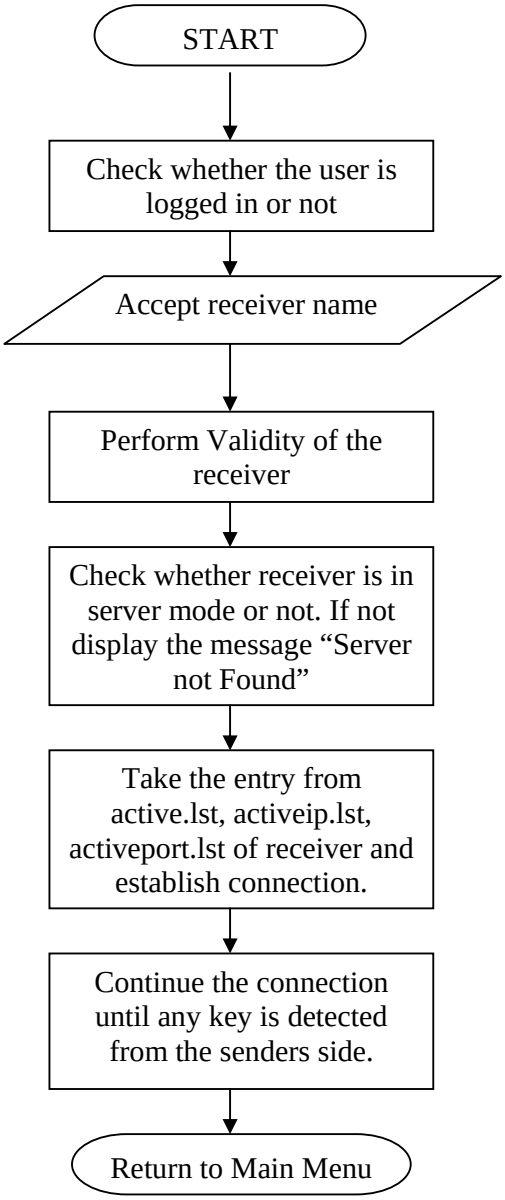


Fig 3.4 Flowchart of Call Module

3.1.4 Read V-mail Module:

Files Used : [name]vmail.lst, [name]vdelete.lst, *.vms

This module enables the user to read his voice mails from Inbox.

When user select this option the listing of voice mail is displayed (Refer [name]vmail.lst)

Upto 15 voice mail can be stored.

[name]vmail.lst contains the list of voice mails, its sender's name, date and time the voice mail was sent.

Example of [name]vmail.lst

```
1.vms_____Sender_name_____Time
2.vms_____Sender_name_____Time
3.vms_____Sender_name_____Time
.
.
.
.
.
15.vms_____Sender_name_____Time
```

Fig 3.5 V-Mail Entry for User

Validation

1. Check for loged In user

Activity Performed By this module are as follows

1. Display the voice mail of loged in user
2. Accept the name of the voicemail file
3. Play that file

Flowchart for Read V-mail module:

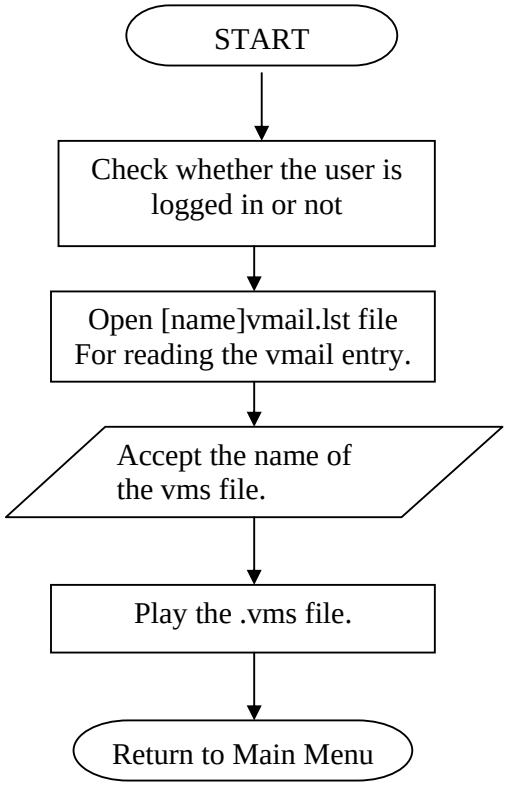


Fig 3.6 Flowchart of Read V-Mail Module

3.1.5 Send V-mail Module:

Files Used : [name]vmail.lst, [name]vdelete.lst, *.vms

This module enables the user to send the voice mail.

If the target's user account is full i.e 15 voice mail are present then initiator will get error message.

Validation

1. Check for logged In user
2. Check for deleted file list of target user

Activity Performed By this module are as follows

1. Check for target user's inbox size
2. Check [name]vdelete.lst file of target user.
3. if any entry present in [name]vdelete.lst file then get that entry and pass it to vmailentry() function.
4. on receiver side corresponding sender name and time is stored for listing.

File Content Before making send vmail option on target side.[2]

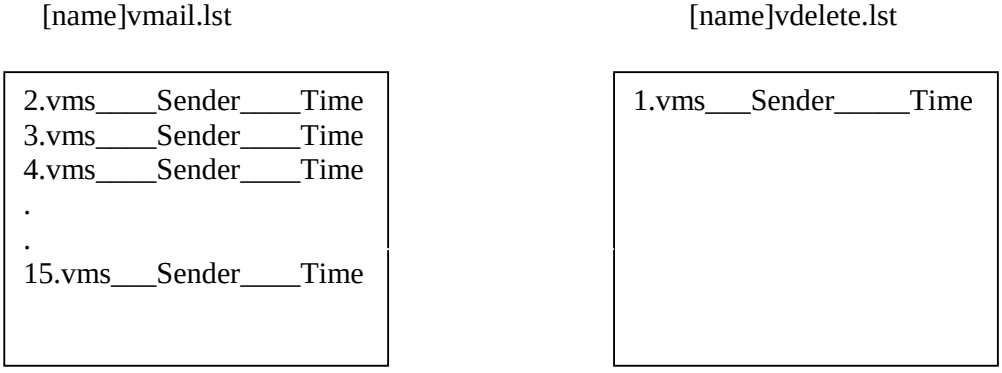


Fig 3.7 V-Mail files before update

File Content Before making send vmail option on target side

[name]vmail.lst

2.vms____Sender____Time
3.vms____Sender____Time
4.vms____Sender____Time
.
.
15.vms___Sender____Time
1.vms____Sender____Time

[name]vdelete.lst

Fig 3.8 V-Mail files after update

Always the most recent mail is listed at the end of the file.

Flowchart for Send V-mail Module:

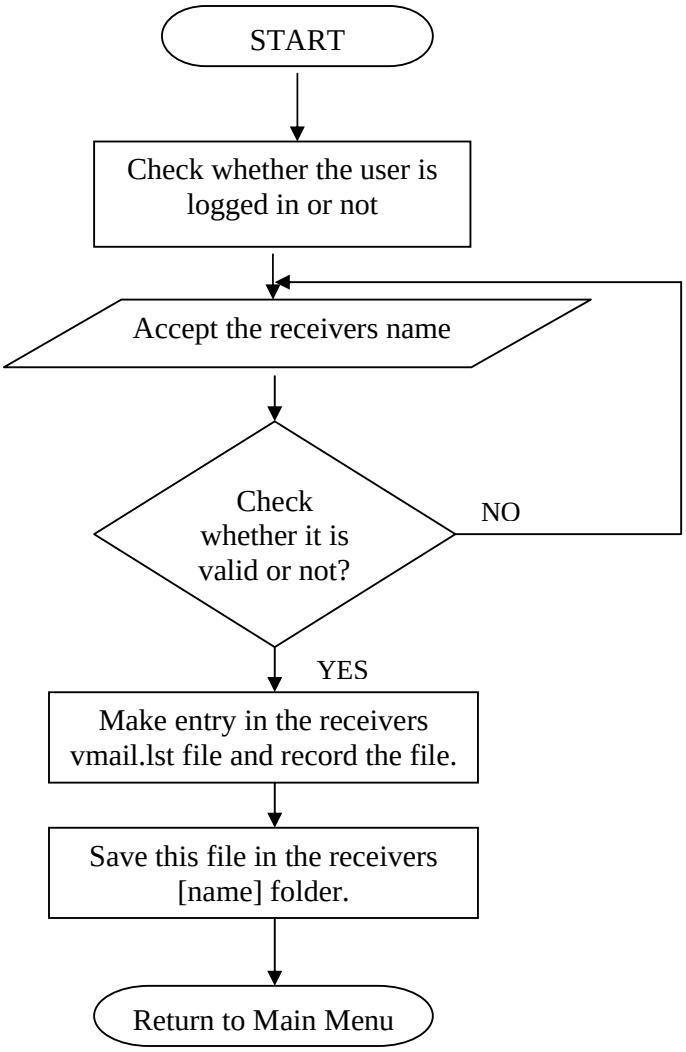


Fig 3.9 Flowchart of Send V-Mail Module

3.1.6 Logout Module:

Files Used: active.lst, activeip.lst, activeport.lst

This module is used to logout from the system so that another user can use the same terminal .

When the user selects the logout option, his entries are removed from active.lst, activeip.lst, and activeport.lst file.

This module returns to the main menu after execution.

Activity Performed By this module are as follows

1. Set Current_user to Null
2. remove logged in user name from active.lst, activeip.lst and activeport.lst
3. Set masterlock to 0

Flowchart for Logout Module:

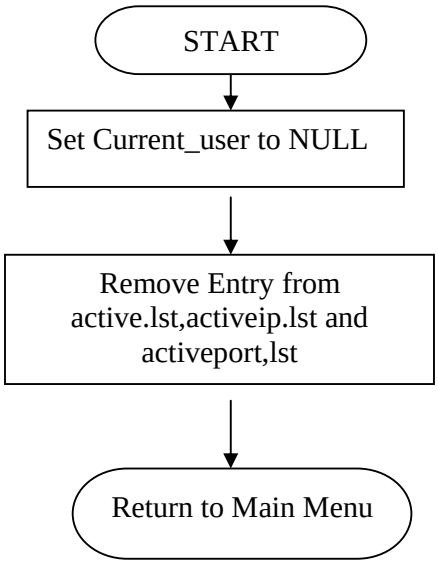


Fig 3.10 Flowchart of Logout Module

3.1.7 Exit Module:

This module performs the same function as the logout module but the difference is that it exits from the program.

3.1.8 Additional Utilities:

This option is available at the main menu screen.

When user press any of the following key corresponding option is activated

1. **Admin Utility (Ctrl+A)**

   This utility can be used only by the terminal which acts as an administrator / server.

   Once this check is performed following 4 options are displayed.

   a. Display registered Users : This option is used to display the list of registered users from the masterdb.lst file.

   b. Delete User : This option is used to delete any registered users if they are not currently active . All the entries of the user from the masterdb.lst, port.lst, and userpass.lst are removed. Also directory structure is removed.

   c. Update Master database : This option removes all the intermediate spaces In the masterdb.lst and userpass.lst. This ensure the database consistency

   d. Exit : This option returns the control back to the main menu.

Flowchart for Admin Utility:

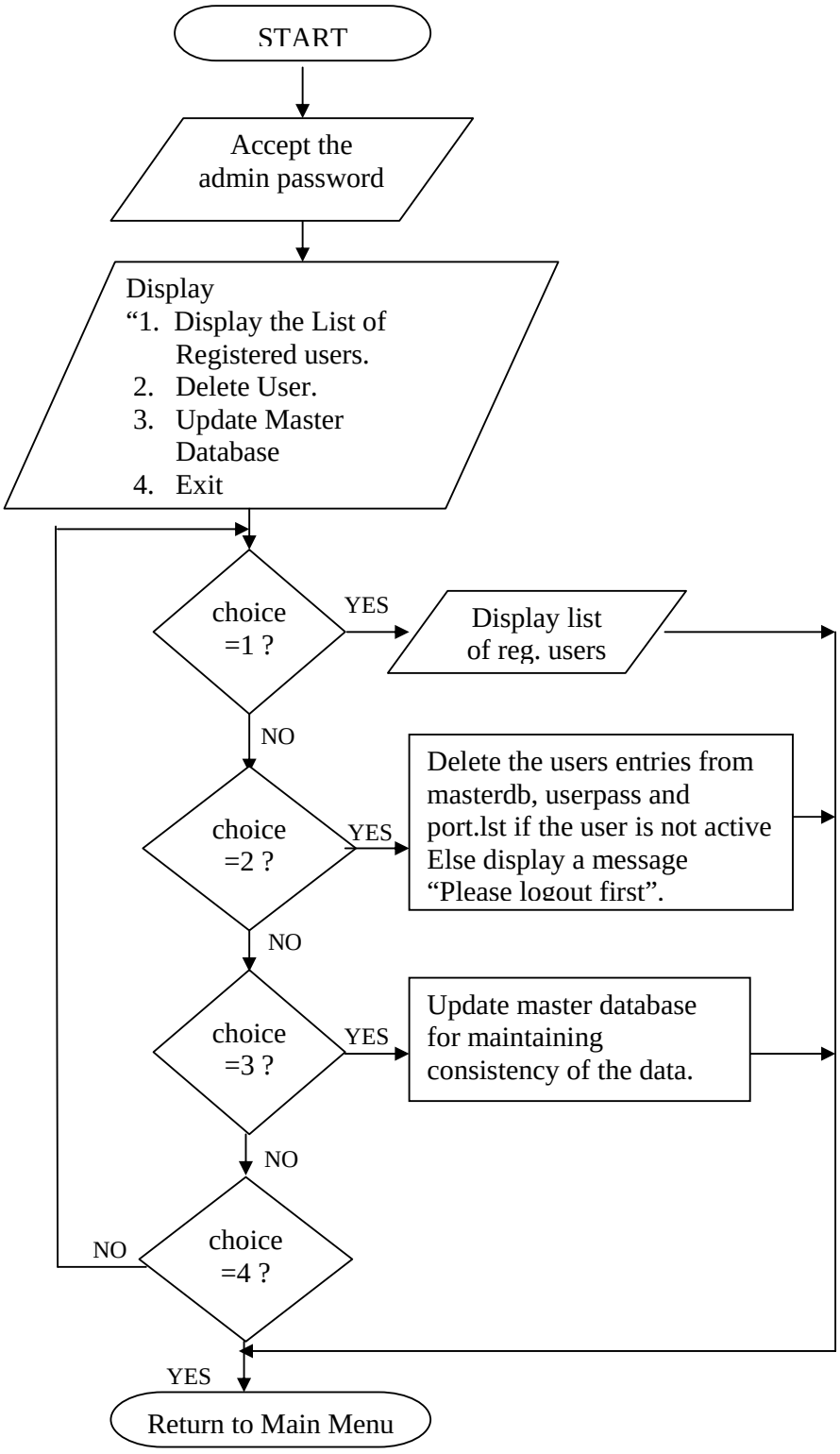


Fig 3.11 Flowchart of Admin Utility

2. To See Registered Users.(Key- R/r)

This option is used to display the list of registered users from the masterdb.lst File. No Admin password is required to access this option.

3. To See Active Users (Key – A/a)

This option displays the list of active users.

4. Delete VMS files: (Key – D/d)

This option is used to delete existing vms files. Once a particular vms file is deleted, its entry is made in the [name]vdelete.lst file. So next time when a new *.vms file is created, its file name will be assigned from the [name]vdelete.lst file entry.

3.2 Data Flow Diagram For LAN Telephony System

3.2.1 Context Level / Level 0 DFD For LAN Telephony System:

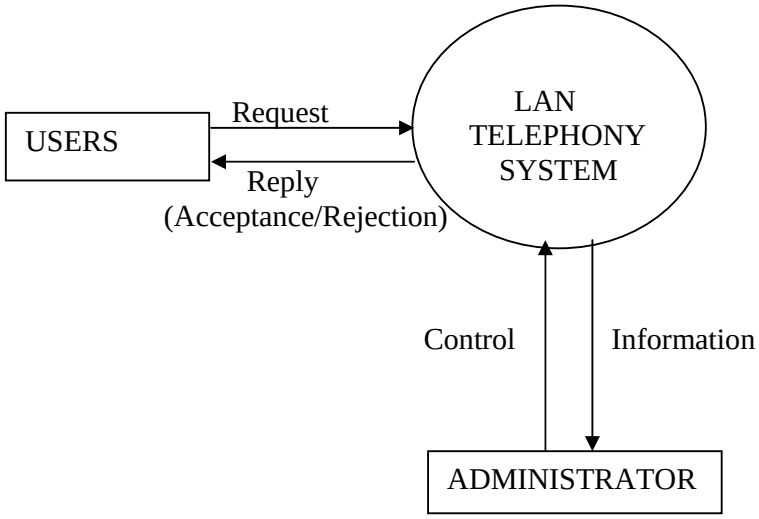


FIG 3.12 Context Diagram (LEVEL 0 DFD)

3.2.2 LEVEL 1 DFD

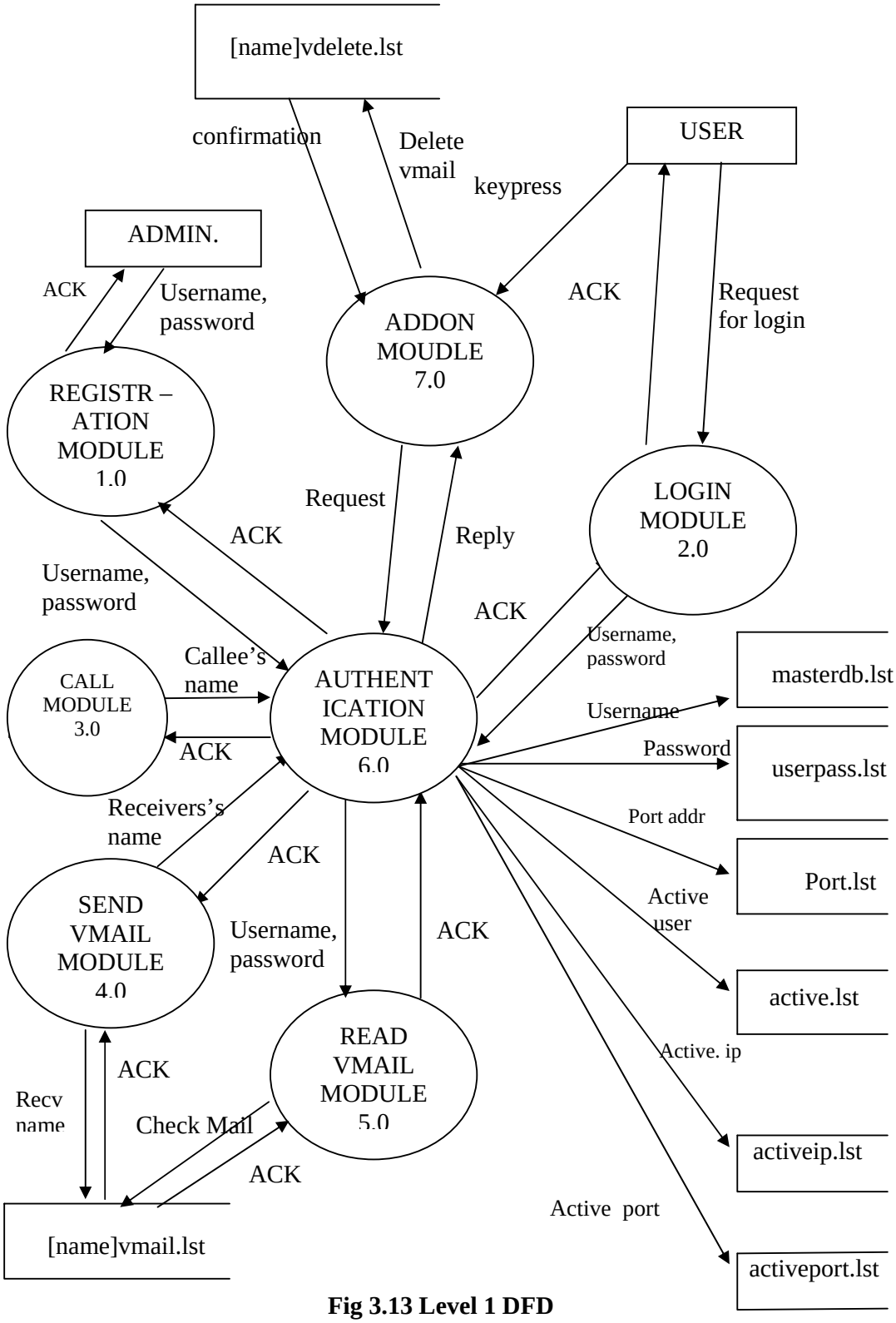


Fig 3.13 Level 1 DFD

LEVEL 1 PROCESS

Process 1.0 Registration Module :This module creates new users. The authority of creatng new user is only given to the administrator.

Process 2.0 Login Module : This module enables the registered users to login to the LAN TELEPHONY SYSTEM.

Process 3.0 Call Module : This module enables the logged in users to make a call to another registered user who is logged in.

Process 4.0 Send Vmail Module : This module enables the logged in users to send a voice message to another user who is not available.

Process 5.0 Read Vmail Module : This module enables the logged in users to check his mail box and read the voice mails he has received while he was offline.

Process 6.0 Authentication Module: This module performs various validations on the username and passwords. It also validates that any non-registered user will not be able to access the LAN TELEPHONY services.

Process 7.0 Addon Module :This module provides ADD-ON utilities and administrator utilities.

3.2.3 LEVEL 2 DFD

3.2.3.1 Registration Module:

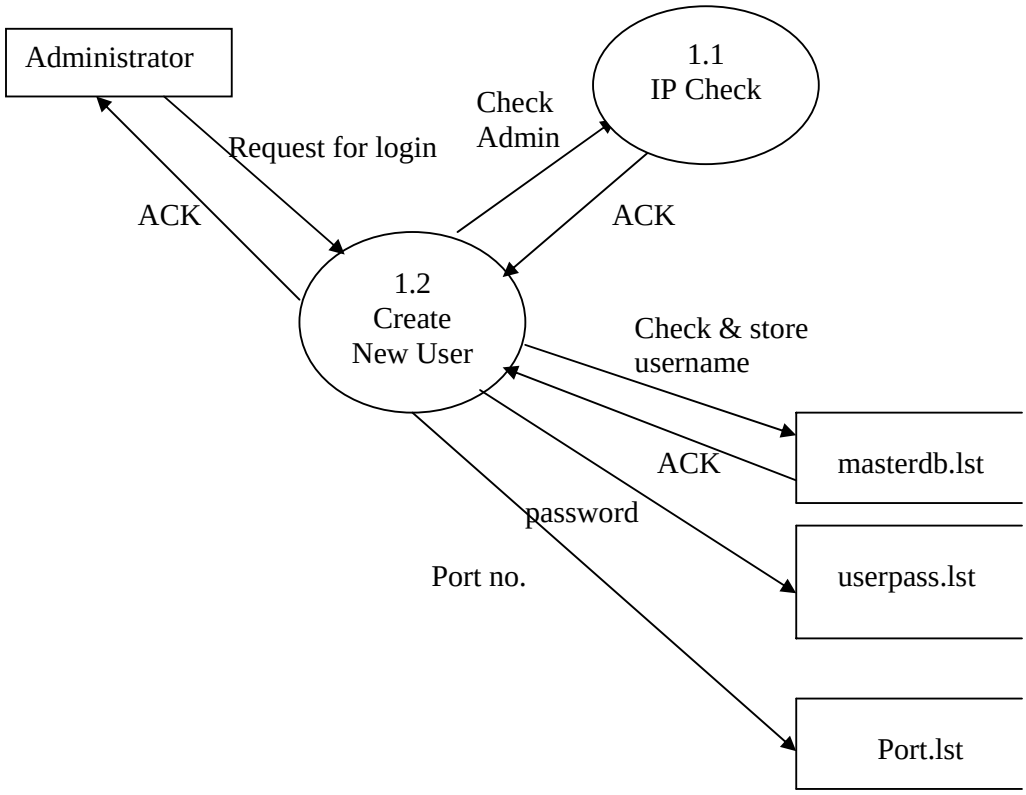


Fig 3.14 LEVEL 2 OF PROCESS 1.0

LEVEL 2 OF PROCESS 1.0

Process 1.1 IP Check : This process checks the terminal IP since the administrator is permitted to login only from the server terminal.

Process 1.2 To create new user : This process creates new users on the LAN TELEPHONY SYSTEM.

3.2.3.2 Login Module:

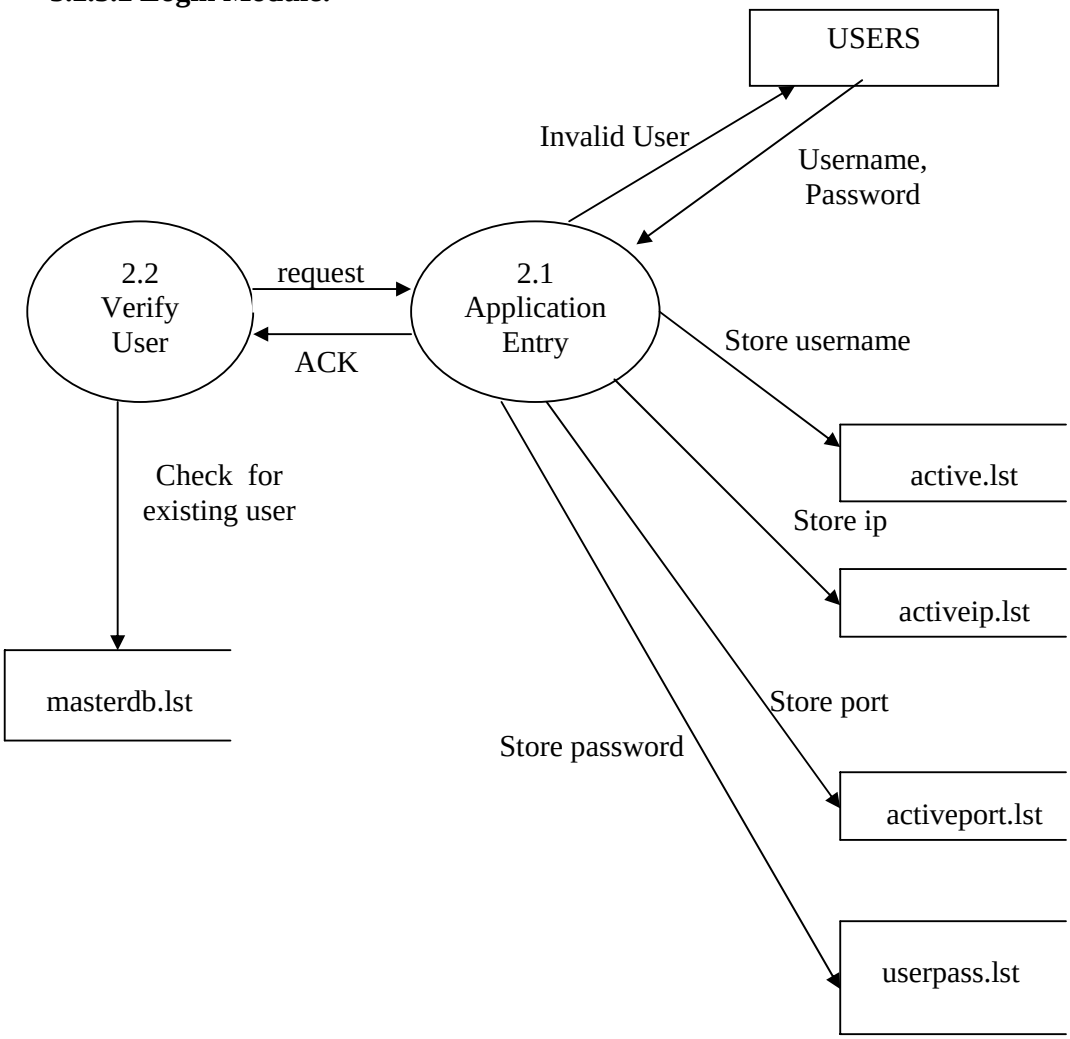


Fig 3.15 LEVEL 2 OF PROCESS 2.0

LEVEL 2 OF PROCESS 2.0

Process 2.1 Application Entry : This process makes the entries into active.lst, activeip.lst, activeport.lst, userpass.lst once the user logs in.

Process 2.2 Verify User : This process verifies whether the username and password entered by the user are present in the masterdb.lst file or not.

3.2.3.3 Call Module:

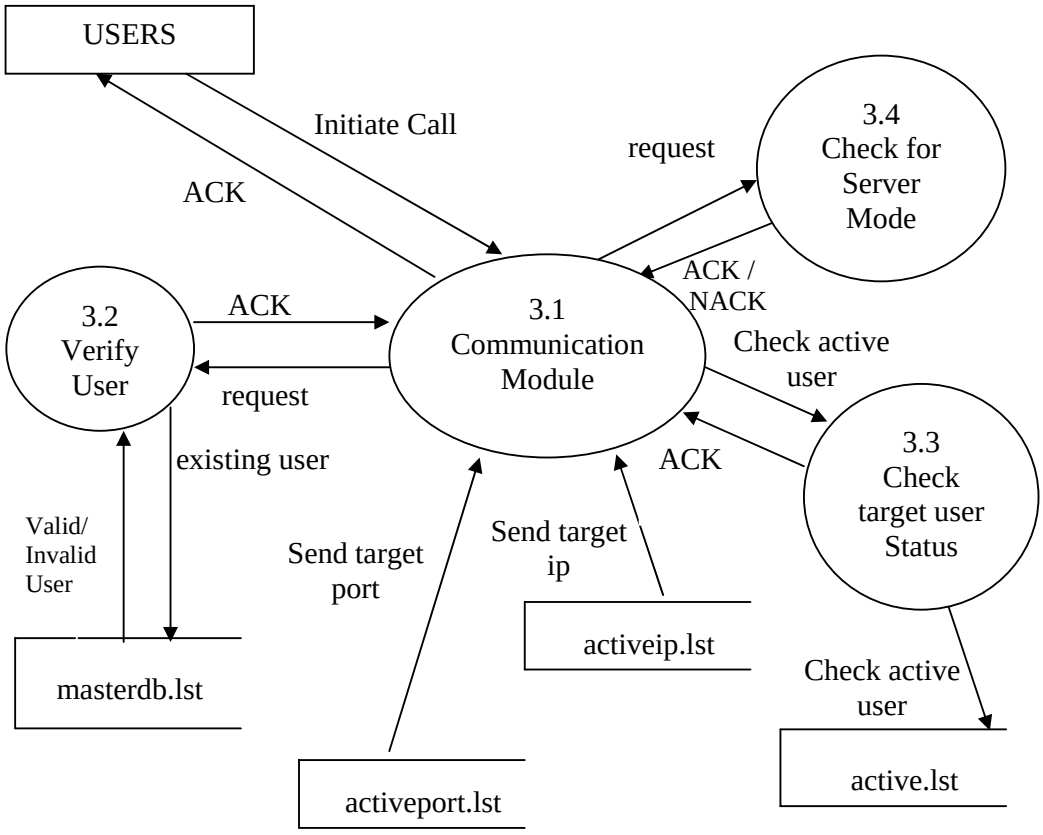


Fig 3.16 LEVEL 2 OF PROCESS 3.0

LEVEL 2 OF PROCESS 3.0

Process 3.1 Communication module : This process after performing all the validations establishes connection between the caller and the callee and enables communication.

Process 3.2 Verify User : This process verifies whether the user is authorized to use the call service of LAN TELEPHONY SYSTEM.

Process 3.3 Check Target User Status: This process checks whether the user who is being tried to call is registered and online or not.

Process 3.4 Check For Server Mode : This process checks whether the callee is in server mode of not .i.e he is ready for communication or not.

3.2.3.4 Send V-Mail Module:

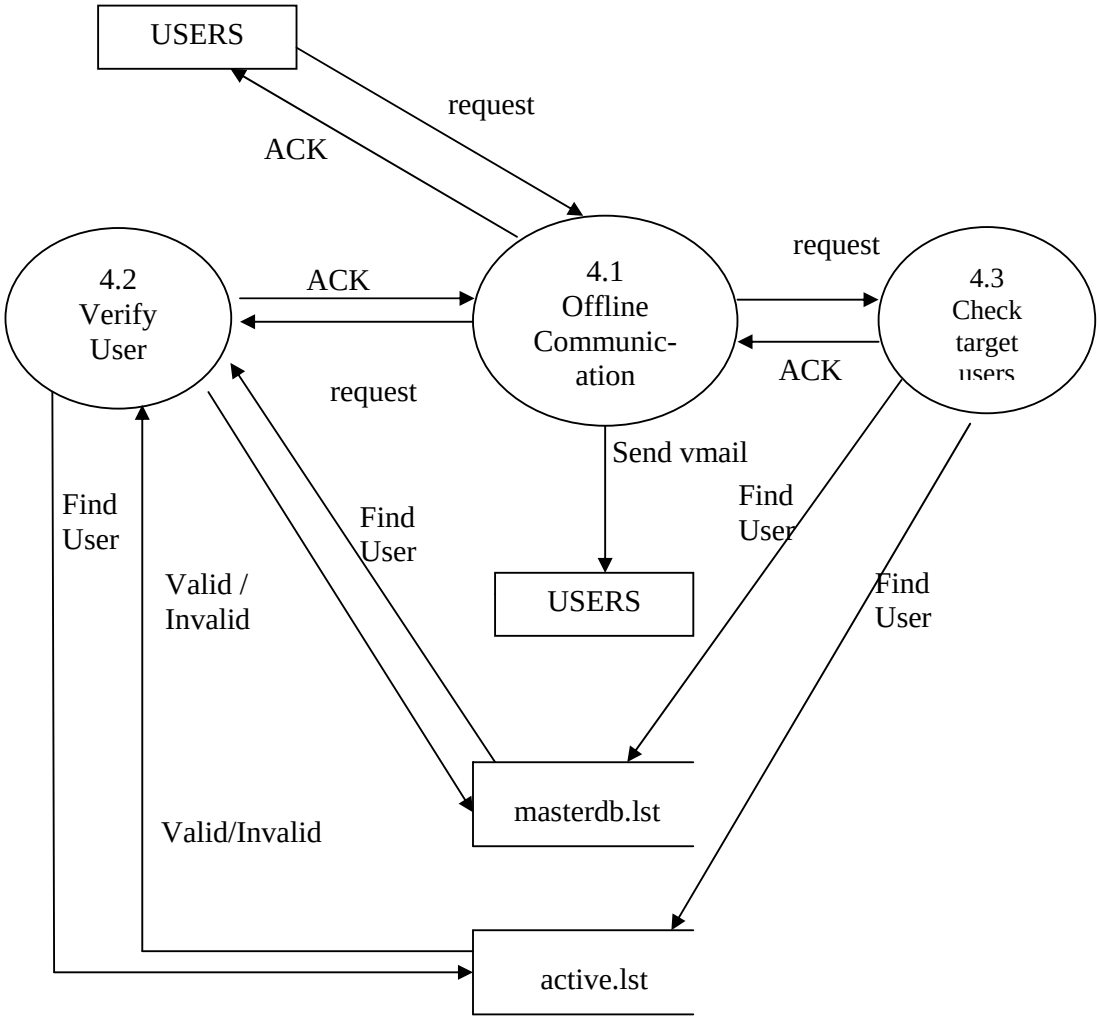


Fig 3.17 LEVEL 2 OF PROCESS 4.0

LEVEL 2 OF PROCESS 4.0

Process 4.1 Offline Communication : This process performs all validations and then permits the user to send voice mail to another registered user.

Process 4.2 Verify User : This process verifies whether the user is authorized to use the voice mail service of LAN TELEPHONY SYSTEM.

Process 4.3 Check Target User Status : This process checks whether the user who is being tried to call is registered and offline or not.

3.2.3.5 Read V-Mail Module:

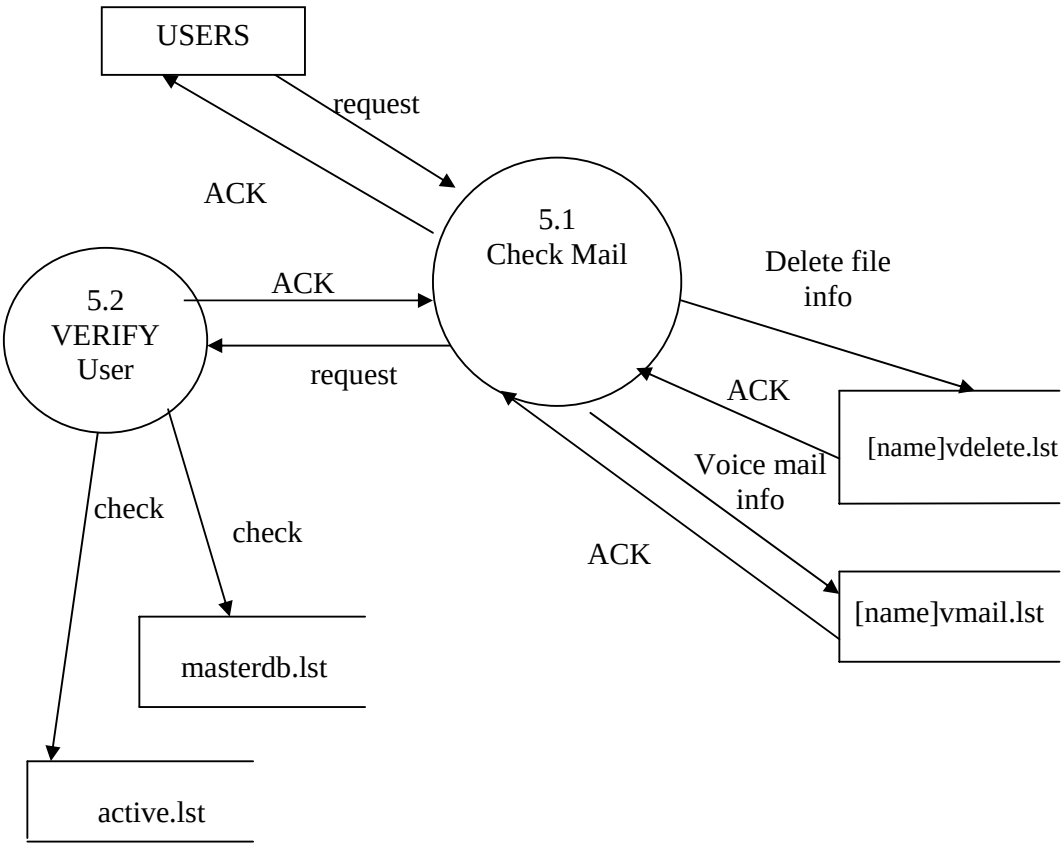


Fig 3.18 LEVEL 2 OF PROCESS 5.0

LEVEL 2 OF PROCESS 5.0

Process 5.1 Check Mail : This process enables he user to check his voice mail box.

Process 5.2 Verify User : This process verifies whether the user is authorized to use the voice mail service of LAN TELEPHONY SYSTEM.

CHAPTER 4

IMPLEMENTATION

For implementing LAN-TELEPHONY SYSTEM we have made one Pentium based local boot Linux terminal which would be acting as a server. This one terminal will act as an Administrator also and other Pentium based local boot Linux terminal will mount files from these terminals. The executable file lan_telephony will be running on all the machines.

The administrator is given the authority to create new users and delete existing users. When the administrator creates a new user he will be asked for username, password, and port no. through which the new user will communicate. After all the validations the username will be entered in masterdb.lst, password in userpass.lst and port no. in port.lst. Also a directory will be created in the name of the username in which all the voice mails will be stored. Also all the names of the voice mail files would be stored in the file [name].vmail.lst where [name] is the username. This file is stored in the root directory. A file called [name]vdelete.lst will keep a track of the voice mail files which are deleted.

For logging in to the system the user need to specify his username and password. This username and password are checked in masterdb.lst and userpass.lst respectively. If a match is found then the user is allowed to log in else he is given an error message saying “Please register first”.

Once a user logs in, after all the validations, then by pressing <Ctrl+B> he goes in to the server mode. In this mode the terminal can acts as a listener and waits for the caller to call. While the user is in server mode, he/she run any other program or application.

Once the caller calls the user, the system or software takes the caller’s username & takes its corresponding IP and port no. from active.lst, activeip.lst, activeport.lst and initiates a call. Once call is made, the callee gets 5 beeps on his machine with the caller’s username and he is left with the option of accepting or rejecting the call.

If the callee accepts the call then connection between the caller and the callee is

established and then the caller and callee can communicate. Once the callee accepts the caller's call, both the names are entered in the busy.lst file so that no third user can intervene their communication. After communication is complete the entries will be removed from busy.lst file.

The user also has an option of sending a voice mail to the callee if the callee is not available for communication at that moment to time. The callee can then hear the caller's message as per his time availability. For sending a voice mail both the user should be registered and the user who wants to send a voice mail should be logged in.

The user can read his voice mails from his Inbox. He is given a message when he logs in saying "You have a Voice Mail". At the most any user can have maximum 15 voice mails in his Inbox. Once the Inbox has 15 messages he will get a message saying "Inbox Full!!!". He will not be able to receive any new messages once his Inbox is full. He can delete his voice messages by pressing the D/d key. After pressing the D/d key his will be shown a list of all the voice mails in his Inbox with the sender's name, date and time when he sent it and the file name. The voice mails can be deleted by giving the file name shown in the list. For storing voice mails we have used .vms type files since they occupy very little space. [13]

When the user Log's out or Exit's from the LAN TELEPHONY system his username, ip and port no. entries will be removed from the active.lst, activeip.lst and activeport.lst.

The LAN TELEPHONY system also provides some additional utilities such as viewing the list of registered user (by pressing the R/r key) and viewing the list of active users (by pressing the A/a key).

Working of socket programming:

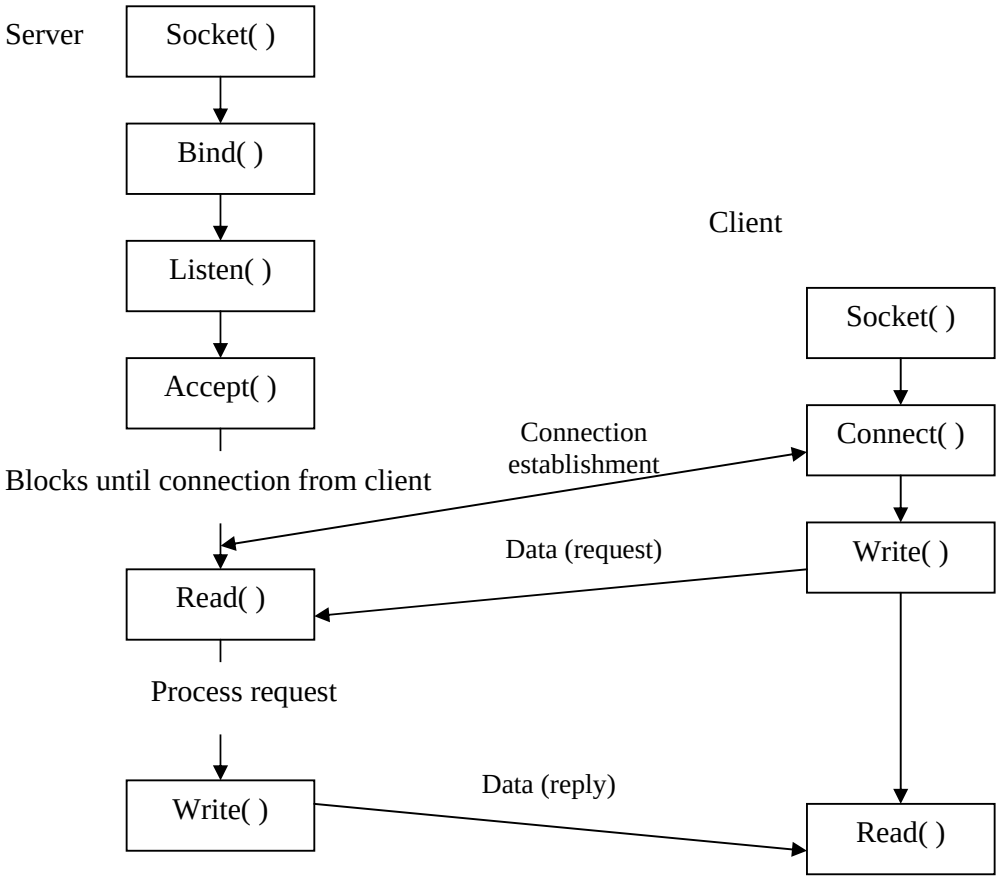


Fig 4.1 Socket Programming using connection oriented protocol

In lan telephony we have used TCP/IP protocol for reliable transfer of voice data.

Voice communication is done using /dev/dsp. When the connection is established between two user voice data is transferred using socket programming in the chunk of 1024 bytes. When callee close the connection receiver will get message that callee has close the connection. At a time only one process is allowed to use the /dev/dsp. So user has to take care that all other voice application are close and resource is free else it will show error message.

As fig 4.1 describe the steps to be followed to establish the connection. Once the connection is established, both server and client uses socket descriptor to access the socket. In lan telephony we have use newsockfd and sockfd as socket descriptorwhich

is used to transfer the voice data between server and client.

4.1 Files Used In Program :

Masterdb.lst - Contains the names of the users who are registered on the LAN TELEPHONY SYSTEM.

Userpass.lst - Contains the passwords of the users in the masterdb.lst with one to one correspondence.

Port.lst - Contains the ports which are assigned to the users in the masterdb.lst with one to one correspondence.

[name]vmail.lst - will contain the list of voice mail files which the user [name] has in his mailbox.

[name]vdelete.lst - will keep a track of the deleted voicemail files.

Active.lst - This file will contain the users who are currently logged in on LAN TELEPHONY system through various terminals.

Activeip.lst - This file will contain the ip address (xxx.xxx.xxx.xxx) of the users who are logged in on LAN TELEPHONY system

Activeport.lst - This file will contain the port address of the users who are logged in on LAN TELEPHONY system

****.vms*** - These are the actual voice messages which will be stored under the users directory.

Busy.lst - Contain the list of users who are currently chatting.

[name] - Directory per user

4.2 Minimum Hardware Requirements:

1. IBM Compatible PC (486 or higher)
2. 32 MB RAM
3. 100 MB of Hard disk Space
4. NIC (10 Mbps)
5. Super VGA (800x600)(color or monochrome) or higher resolution display.
6. Multimedia Kit (mic and speaker)

Recommended

1. IBM Compatible PC (Pentium or higher)

1 128 MB RAM

2 4 GB of Hard disk Space

3 NIC (100 Mbps)

5 Super VGA (800x600)(color or monochrome) or higher resolution display.

6 CD ROM drive

7. Multimedia Kit (mic and speaker)

4.3 Minimum Software Requirements:

1. Red Hat Linux (kernel version 2.4.x onwards)
2. TCP / IP Support
3. /dev/dsp driver for sound.

CHAPTER 5

TESTING

5.1 Testing

5.1.1 Validations for Registration module:

1) Only Administrator is authorized to register new users.

2) The username should be unique.

3) The username should not be more than 8 characters.

4) The username should not be blank.

5) The password should be maximum 8 characters.

6) The password should not be blank.

Conclusion:

All the above tests have been successfully done

5.1.2 Validations for Login module:

1) The user should be registered user.

2) The user should not have already been logged in.

3) The username should not exceed 8 characters.

4) The password should not exceed 8 characters.

5) The username and password should not be blank.

6) Only single user can login from one terminal at a time.

7) A user cannot login from different terminals at the same time.

Conclusion:

All the above tests have been successfully done

5.1.3 Validations for Call module:

1) The Users (Caller and the Callee) should already be registered.

2) Both the users should on logged in on different terminals.

3) The Callee should be in server mode else the Caller will get an error “User

not in server mode".

4) The Callee's name should not be more than 8 characters.

5) The Caller and the Callee should connect at the same port no.

6) If two user are chatting and if third user try to connect to either of the two then Callee will get an error "User Is Busy With Other user, Try again later"

Conclusion:

All the above tests have been successfully done.

5.1.3 Validations for Send V-mail module:

1) The user who is sending the vmail should be registered and logged in.

2) The target user should have enough space in his mail box, if not then callee will get an error "Target User's Mail Box is full"

Conclusion:

All the above tests have been successfully done.

5.1.4 Validations for Read V-mail module:

1) The user should be registered and logged in to read him vmails.

2) User can maximum upto 15 voice mails in his mailbox at a time.

3) If user try to enter invalid file name then display error message

Conclusion:

All the above tests have been successfully done.

5.1.5 Validations for Logout module:

1) User should be registered and logged in to log out.

2) User should logout for proper termination.

Conclusion:

All the above tests have been successfully done.

5.1.6 Test Results:

Tests for 5.1.1 through 5.1.5 have been indicated in the appendix.

CHAPTER 6

CONCLUSION & FURTHER WORK

6.1 Conclusion

LAN TELEPHONY system plays an integral part in the communication process through the network in an organization. Being restricted to a domestic network, it displays invaluable features such as low cost of implementation, reliability, less complexity, less bandwidth requirements**,** etc. Nothing feels better than hearing a voice of your friend. It gives in-person feeling rather than writing a text message to him.

LAN TELEPHONY system has been developed using the concept of socket programming. He transmitted voice is converted into digital form (using full duplex sound card) and then it is sent over the ports opened through socket programming.

Thus LAN ELEPHONY system has been successfully implemented over a private network with a variety of features such as voice mail, voice communication.

6.2 Future Work

One of the important things in the LAN TELEPHONY system is that it can be further expanded over the global network. All these features can be implemented on a larger scale using Internet. VOIP (Voice Over IP) provides all these features on a global scale. In this system voice packets are routed over the network through Internet. In addition LAN TELEPHONY system can also be expanded using protocols like H.323 and SIP. Also value added features such as video transmission, video conferencing, simple file transmission and many more can be readily extended for future modifications and implementation.[7]

BIBLIOGRAPHY

Book References:

[1] W. Richard Stevens: "UNIX Network Programming", Prentice-Hall of India, seventh Indian reprint (1997).

[2] Herbert Schidlt: "The Complete Reference C", Tata McGraw Hill ,Fourth Edition (2001).

[3] Warren W.Gay: "Teach Yourself Linux Programming in 24 Hours", SAMS Publication (1996).

[4] Mark Mitchell, Jeffrey Oldham, and Alex Samuel: "Advanced Linux Programming", New Riders Publishers(1999).

[5] Kurt Wall, Mark Watson, Mark Whitis: "Linux Programming Unleashed", SAMS Publication(1999).

[6] Jeff Tranter: "Linux Multimedia Guide", O'Reilly & Associates (1996)

Internet References:

[7] http://www.ecst.csuchico.edu/~beej/guide/net/

[8] www.vovida.org

[9] http://www.lowtek.com/sockets

[10] http://www.ericsson.com/support/telecom/part-b/b-1-1.shtml

[11] http://www.linuxjournal.com/article.php?sid=2333

[12] www.rfc.com

[13] www.catalyst.com

[14] www.tldp.org

[15] http://www.oreilly.de/catalog/multilinux/excerpt/ch14-05.htm

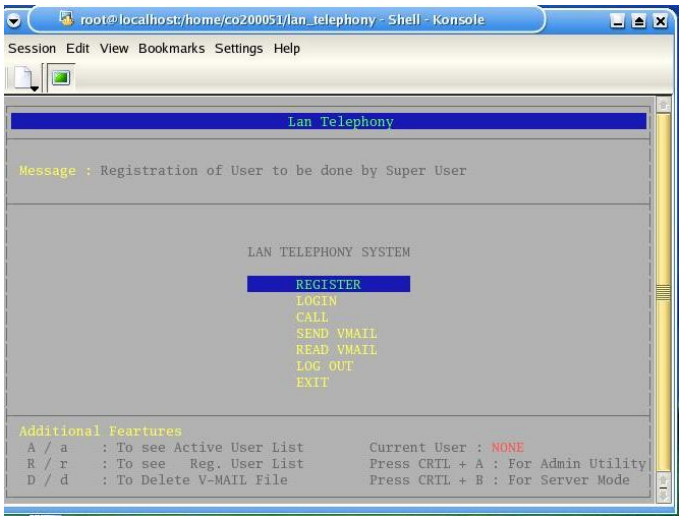
[16] www.skype.com

[17] http://www.vitez.it/picophone/

APPENDIX

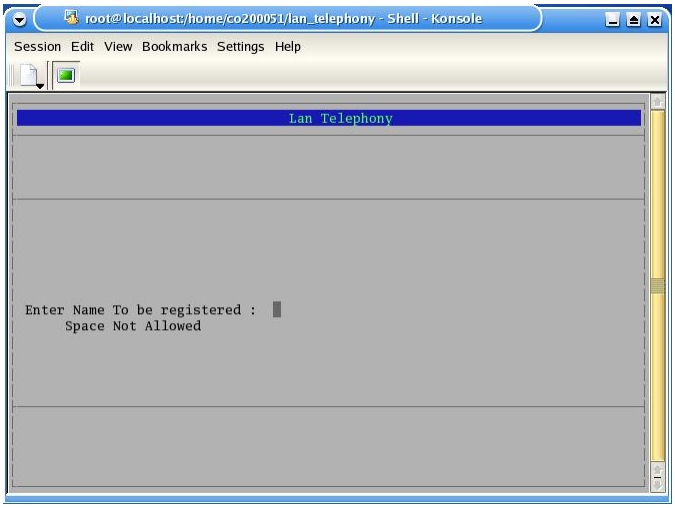
Screen Shots

Input screen shots:



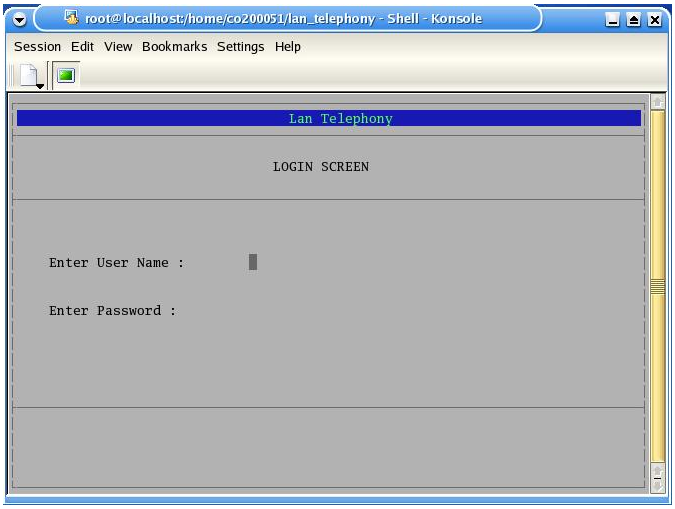
This is the screen shot of main menu

Main Menu



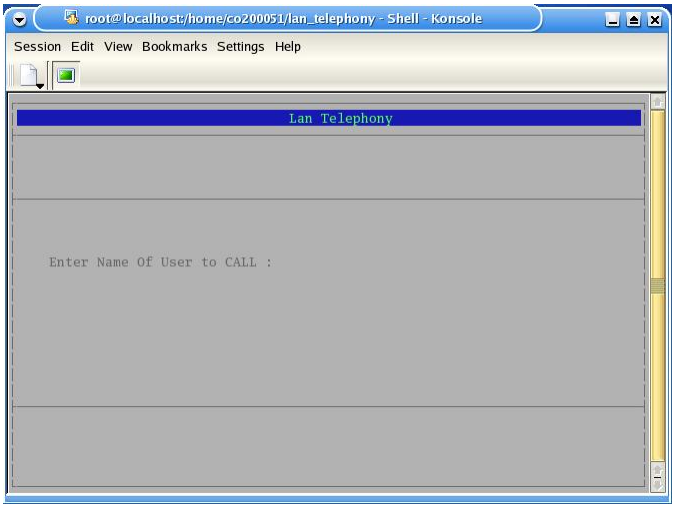
This is the registration Module. Accept name, password and port number

Register User Screen



This is the Login Screen
Accepts Username and Password.
After that will show the appropriate message as per the user input

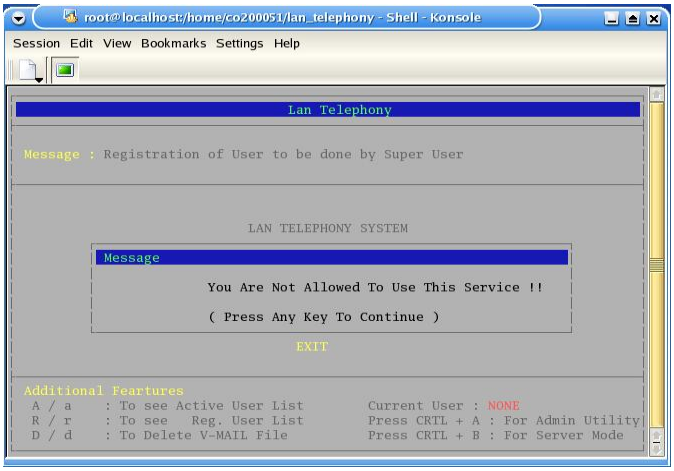
Login Screen



When user select call option this screen will be shown.

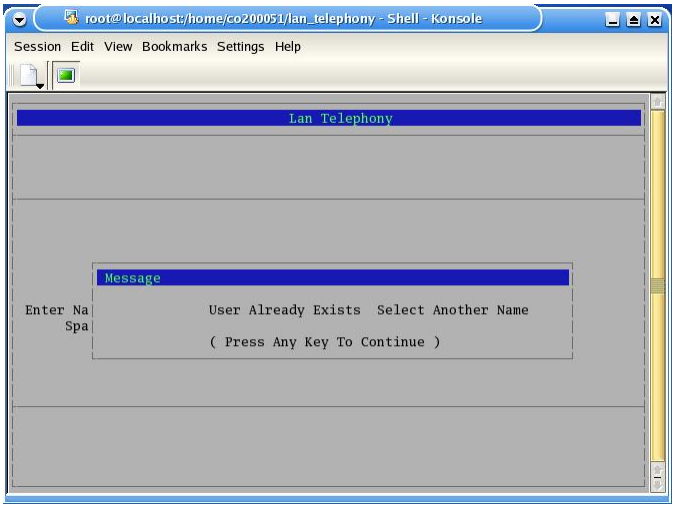
call screen

Output screen shots



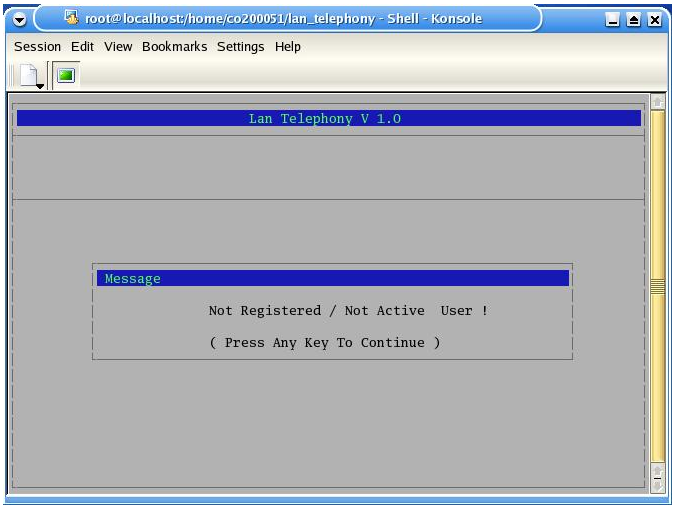
If user try to enter in to registration module and if its not server then this message will be shown

Error Display For Client other then Admin



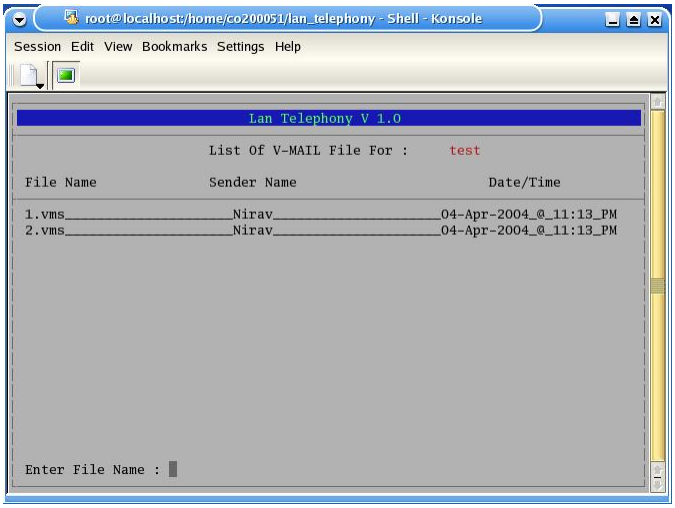
If user try to enter in to registration module and if its not server then this message will be shown

Error message at the time of registration



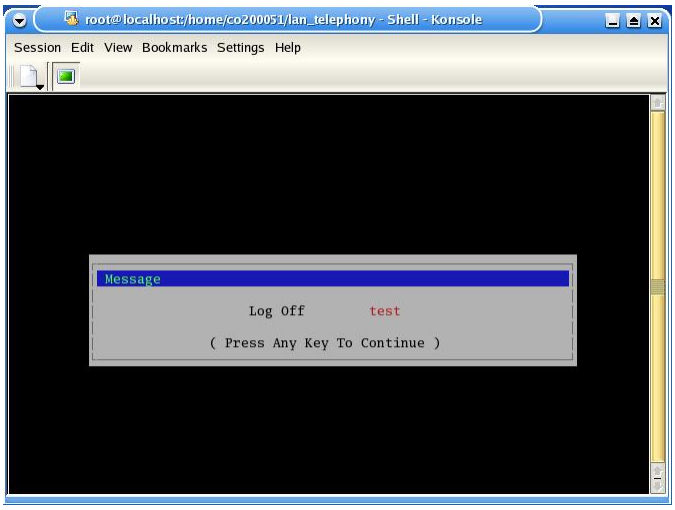
If user try to call user and if it is not registered or active then this message will be shown

Error Message at the time of call



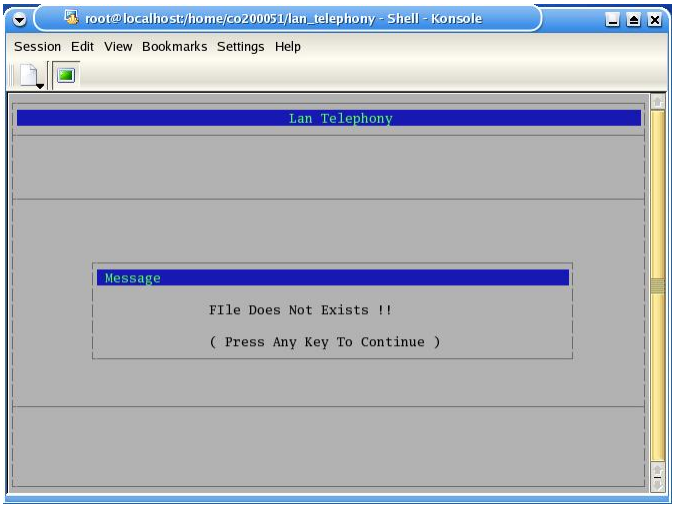
On selecting read vmail option, this screen will be displayed

V-Mail Display



On selecting logout this screen will be displayed
So that other user can login from same machine

Logout Display



If user select invalid file from list of voice mail then this screen will be displayed

Error message at the time of Read V-Mail

NAPT

Say, an organization has a private IP network and a WAN link to a service provider. The private network's stub router is assigned a globally valid address on the WAN link and the remaining nodes in the organization have IP addresses that have only local significance. In such a case, nodes on the private network could be allowed simultaneous access to the external network, using the single registered IP address with the aid of NAPT. NAPT would allow mapping of tuples of the type (local IP addresses, local TU port number) to tuples of the type (registered IP address, assigned TU port number).

This model fits the requirements of most Small Office Home Office (SOHO) groups to access external network using a single service provider assigned IP address. This model could be extended to allow inbound access by statically mapping a local node per each service TU port of the registered IP address.

In the example of figure 3 below, stub A internally uses class A address block 10.0.0.0/8. The stub router's WAN interface is assigned an IP address 138.76.28.4 by the service provider.

When stub A host 10.0.0.10 sends a telnet packet to host 138.76.29.7, it uses the globally unique address 138.76.29.7 as destination, and sends the packet to it's primary router. The stub router has a static route for the subnet 138.76.0.0/16 so the packet is forwarded to the WAN-link. However, NAPT translates the tuple of source address 10.0.0.10 and source TCP port 3017 in the IP and TCP headers into the globally unique 138.76.28.4 and a uniquely assigned TCP port, say 1024, before the packet is forwarded. Packets on the return path go through similar address and TCP port translations for the target IP address and target TCP port. Once again, notice that this requires no changes to hosts or routers. The translation is completely transparent.

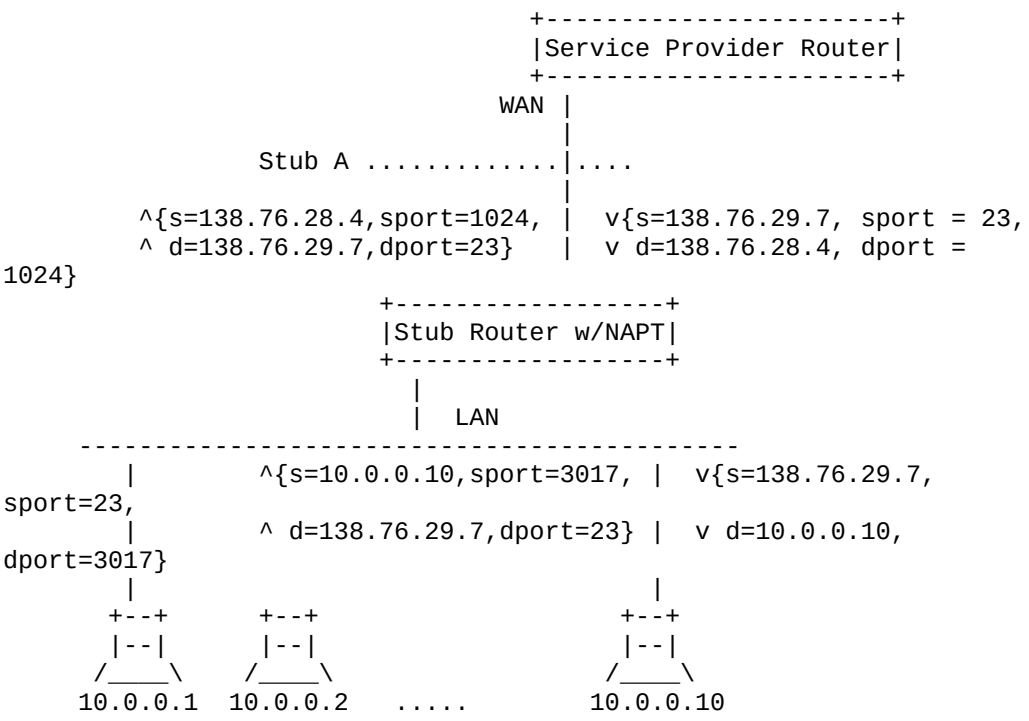


Figure 1: Network Address Port Translation (NAPT) Operation

In this setup, only TCP/UDP sessions are allowed and must originate from the local network. However, there are services such as DNS that demand inbound access. There may be other services for which an organization wishes to allow inbound session access. It is possible to statically configure a well known TU port service [RFC1700] on the stub router to be directed to a specific node in the private network.

In addition to TCP/UDP sessions, ICMP messages, with the exception of REDIRECT message type may also be monitored by NAPT router. ICMP query type packets are translated similar to that of TCP/UDP packets, in that the identifier field in ICMP message header will be uniquely mapped to a query identifier of the registered IP address. The identifier field in ICMP query messages is set by Query sender and returned unchanged in response message from the Query responder. So, the tuple of (Local IP address, local ICMP query identifier) is mapped to a tuple of (registered IP address, assigned ICMP query Identifier) by the NAPT router to uniquely identify

ICMP queries of all types from any of the local hosts. Modifications to ICMP error messages are discussed in a later section, as that involves modifications to ICMP payload as well as the IP and ICMP headers.

In NAPT setup, where the registered IP address is the same as the IP address of the stub router WAN interface, the router has to be sure to make distinction between TCP, UDP or ICMP query sessions originated from itself versus those originated from the nodes on local network. All inbound sessions (including TCP, UDP and ICMP query sessions) are assumed to be directed to the NAT router as the end node, unless the target service port is statically mapped to a different node in the local network.

Sessions other than TCP, UDP and ICMP query type are simply not permitted from local nodes, serviced by a NAPT router.

SIP Protocol

SIP is a text-based protocol with syntax much like the Hyper-Text Transfer Protocol (HTTP) and Real-Time Streaming Protocol (RTSP). A SIP *INVITE* request looks something like this:

figure=pix/siprequest.ps

Typically the payload is an SDP description of the session the caller wishes to set up. The *Via* fields indicate the path taken by the request through proxies so far, with the first *via* field indicating the most recent proxy traversed and the last one indicating the caller's host. In this case Eve sent the message from *chopin.cs.caltech.edu*, it has traversed *isi.edu* and *east.isi.edu*, and is currently being sent to *north.east.isi.edu* as indicated in the request URL.

A response to this message indicating that *mjh* does want to talk might look like:

figure=pix/sipresponse.ps

The response code *200* indicates success (many other response codes are possible). When *north.east.isi.edu* sent this response, it removed the first *via* field (*east.isi.edu*) and then sent the response to east. In this way the response retraces the path through a chain of proxies that the request took.

SIP supports several request *methods* as shown below.

Request Method	Purpose
INVITE	Invite the callee into a session
OPTIONS	Discover the capabilities of the receiver
BYE	Terminate a call or call request
CANCEL	Terminate incomplete call requests
ACK	Acknowledge a successful response
REGISTER	Register the current location of a user

In response to a request, a server sends a SIP response which gives a response code indicating how the request was processed. Response codes are divided into six categories depending on the general form of behaviour expected. These categories are shown below.

Response Class	Meaning	Example
1XX	Information about call status	180 RINGING
2XX	Success	200 OK
3XX	Redirection to another server	301 MOVED TEMPORARILY
4XX	Client did something wrong	401 UNAUTHORISED
5XX	Server did something wrong	500 INTERNAL SERVER ERROR
6XX	Global failure (don't resend same request)	606 NOT ACCEPTABLE

The normal progress of a SIP call involves sending an *INVITE* request, getting a *180 RINGING* response, followed by a ``*200 OK*'' response when the callee answers, then sending an *ACK* request to confirm the connected call.

After either party hangs up, a *BYE* request is sent, which elicits a ``*200 OK*'' response, which in turn causes a final *ACK* request to be sent.

The *CANCEL* method is used when a call is made to more than one destination at the same time. For example, a proxy server may respond indicating that someone might be at one of server locations. My client can either call these one-by-one, or call all of them simultaneously. If it does the latter, then when the callee answers at one of these locations, my client sends a CANCEL request to all the other locations to stop the phone (or whatever it is) ringing there.

ACRONYMS

NAPT : Network Address Port Translation

SIP : Session Initiation Protocol.