1513

ELECTRONIC MAIL SYSTEM ON TOP OF NETBIOS

Dissertation submitted to Jawaharlal Nehru University
in partial fulfilment of the requirements
for the award of the Degree of

MASTER OF TECHNOLOGY

66 p. + biblio + fig.

OM PRAKASH SONI

SCHOOL OF COMPUTER & SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI-110067

1989

CERTIFICATE

This is to certify that this project thesis entitled **ELECTRONIC MAIL SYSTEM ON TOP OF NETBIOS LAN** being submitted by OM PRAKASH SONI , in partial fulfillment of the requirements of the degree of Master Of Technology in Computer Science , has been completed under the supervision and guidence of Prof. K. K. Nambiar ,J.N.U. and Dr. Surendra Pal, D.O.E.

This work has been carried out at ERNET Project Group, D.O.E. New Delhi and has not been submitted to any other institution or University for award of any degree.

[illegible] 4/1/90

Dr. Surendra Pal
Joint Director
Ernet Project Group
New Delhi

[illegible]

Prof K.K. Nambiar
SC & SS, J.N.U
New Delhi

S. Ramakrishnan
4.1.90

Mr S.Ramakrishnan
Additional Director
Ernet Project Group
New Delhi

[illegible]

...OF. N... MUKHERJEE
Dean
School of C... and Systems Sciences
Jawaharlal Nehru University
NEW DELHI-110067

ACKNOLEDGEMENTS

I am pleased to express my indebteness to Mr Ramakrishnan Additional Director, ERNET Project, D.O.E. for having given me the opportunity to work on this project and allowing me to use all the resources available at ERNET Project D.O.E.

I am very much indebted to my guides Prof. K. K. Nambiar at J.N.U and Dr. Surendra Pal, Joint Director at D.O.E New Delhi who had been a constant source of encouragement and guided me throughout the course of this project work.

I express my sincere thanks to Mr. Fateh Singh (A.O.) SC & SS, JNU. for his valuable guidance throughout my M.Tech. course.

I would also like to thanks the other friends at D.O.E who had offered their critical and constructive suggestions throughout.

[illegible]

Date: 4th Jan, 90.

OM PRAKASH SONI

CONTENTS

CHAPTER 1

INTRODUCTION TO ERNET PROJECT

Education and Research in Computer Networking (ERNET) is a project of Department Of Electronics (DOE Government of India), New Delhi, funded by the United Nations Development programme (UNDP). The project is being implemented by the DOE in close cordination with seven major technical academic and research institution in India viz., five Indian Institutes of Technology at New Delhi, Kanpur, Khargpur, Madras, Bombay, Indian Institute of Science, Bangalore and the National Center for software Technology, Bombay.

The developement objective of the project is to improve national capabilities in the areas of computer networking and emerging telematic concepts in the country through establishment of experimental computer networks, upgradation of skills and training of the manpower. The immediate objectives are:

1. To design, setup, and interconnect Local Area Networks for campus coverage at eight locations mentioned above, Metropolitan Area Networks covering a few tens of kilometers and Wide Area Network with national coverage.
2. To establish capabilities at the above mentioned technical institutes to undertake research and development in the area of computer networking.
3. To establish educational programmes in computer networking at graduate and master level.
4. To lead and co-ordinate work in India on establishment of computer networks.

CHAPTER 2

ISO / OSI OPEN NET IMPLEMENTATION

2.1 INTRODUCTION

The Open System Interconnection (OSI) reference model is as shown in Fig - 2.1 below:

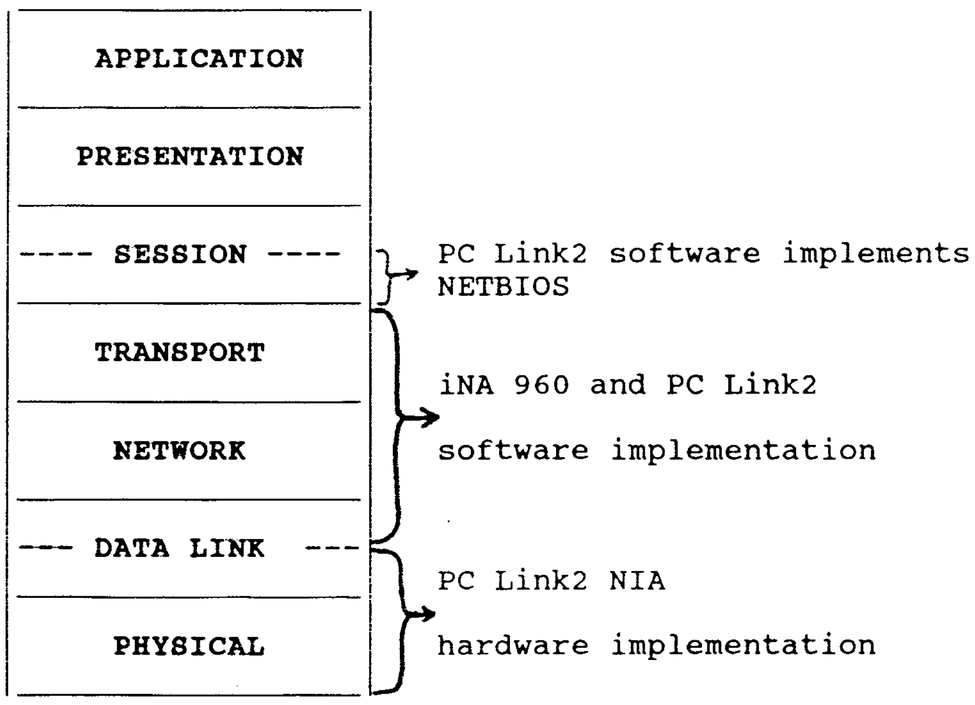

APPLICATION
PRESENTATION
---- SESSION ----
TRANSPORT
NETWORK
--- DATA LINK ---
PHYSICAL
PC Link2 software implements NETBIOS
iNA 960 and PC Link2 software implementation
PC Link2 NIA hardware implementation

FIG - 2.1 OSI Reference Model Layers

The <u>Physical</u> layer and some protocols of <u>Data Link</u> layer's are implemented by PC Link2 NIA (Network Interface Adapter) hardware. This is known as network control board. It is an adapter board that provides an Ethernet\ IEEE 802.3 connetion to the PC system. The network control board with iNA 960 and PC Link2 software performs all network communication functions for the first four layers of the ISO/OSI reference model. In addition to NETBIOS interface the

PC Link2 software provides a programmatic interface to the iNA 960 Transport software. The iNA 960 Transport software provides the services of the Data Link, Network, and Trnsport layers protocols of the OSI reference model.

2.2 SOFTWARE OVERVIEW

The iNA 960 and The PC Link2 software provides the services of the Data Link, Network And Transport Layers of the OSI reference model.

2.2.1 PC Link2/iNA Transport Layer

The transport layer ensures the integrity of the data packets and are received from and are directed to the Network Layer. The Transport Layer establishes either a gauranteed, error_free, point_to_point, (virtual circuit) connection that delivers messages in the sequence sent, or a datagram service that does not gaurantee message delivery, the order of delivery, or the integrity of the message.

2.2.2 PC Link2/iNA 960 Network Layer

The Network Layer establishes connections across a communication network. On the receiving side, frames are accepted from the Data Link Layer, network packets are created and then forwarded to the Transport Layer. Conversely, on the transmitting side, messages are accepted from the Transport Layer, converted to data packets, and then a route is established for forwarding the packets of the correct network subsystem.

Routing is accomplished via routing tables that are maintained and updated by the iNA 960 Network

Management Facility (NMF). The routing table can be built by the user application and either statically or dynamically updated using ISO routing protocol (IS 9542).

2.2.3 PC Link2/iNA 960 Data Link Layer

The Data Link Layer assembles the raw data that is received from the Physical Layer into frames. The frames are then subsequently transmitted to the Network Layer. In the same manner, the frames are received from the Network Layer are converted into the raw data bits and forwarded to the Physical Layer for transmission to the designated receiving system.

CHAPTER 3

ELECTRONIC MAIL SYSTEMS

3.1 INTRODUCTION

Electronic Mail is a general name that refers to a large class of message and document delivery equipment and services, centered on the use of computers at some point in the system. The concept of moving messages between individuals can be approached differently, from simple memeoranda to complex graphics and textual material. To some, electonic mail is simply a mean of exchanging memoranda between the microcomputers of individuals in a LAN. To others, electronic mail is a way of distributing messages graphics both on site and to widely scattered offices, using a large host computer as the controller. To still others, it is method for the routine distribution of the large amount of written material that is generated in a office.

Electronic mail System (EMS) are computer oriented systems for the electronic transmission of information, including messages, text, and graphics, between two or more selected individuals. The types of information transmitted are:

Messages, such as memoranda, new items, and reminders.

Documents, such as computer output reports generated at central computers or microcomputers, and textual material, either from electronic storage or terminal entry.

Graphics, or graphic textual mixtures, which may be microcomputer video screen, or may be transmitted to graphics output machinary.

The implication of **Electronic** in EMS are first, that the transmission will be fast. Second the transmission will be distance independent, the receipt can be almost instantaneous. Third there will be computer involvement at one or more point in the system. This may be in the switching equipment and at the terminals.

The implication of **Mail** are that this the form of end to end communication capability, where a specific message is sent from one individual to another, or to a selected group. It is a direct , interpersonal communication by any of a variety of media. A permanent record of the communication can be printed out, recorded and made available. It is usually non-simultaneous communication.

The implication of **Systems** are that electronic mail systems are combination of the electronic and non-electronic means employed in concert with understood procedures. There are usually computer programs, with protocols that must be followed.

3.2 EVOLUTION OF E_MAIL SYSTEMS

In the evolution of electronic mail systems, there were four major advances. The first was the **telegraph**, which was introduced in the early part of the nineteenth century. This was the first system to send messages by electronic mean. The problem with the telegraph system that manual code conversion was required, handled by a human operator. There was also no switching capability.

The **teletype** system was developed in the early part of twentieth century. It was the second generation of the electronic mail, and had automatic code conversion. It was also point-to-point. Both sending and receiving station had to be active simultaneously. It has circuit switching. It has a manual variety of message switching, in which the pieces of punched paper tape are manually hung on pegboards, available to be used for further transmission.

The third advance was **message switching**. this was developed in early 1960's as a store-and-forward teletype system. It has computer automated store and forward replacing the old torn-tape system. It became the basis for TELEX and TWX services. It supplies few end user services. The message are simply sent from one TELEX room to another TELEX room, with little capability of helping the user with further handling of the messages, for electronic filing, word processing and so on.

In 1970's, **Computer_Based Message Systems** (CBMS) were introduced. These systems provide variety of end user functions. The allow flexible editing and addressing, general query and retrieval capabilities, connection to databases, and so on.

From this point there will not be a simple progression to another step, but there will be a great variety of alternative procedures introduced. There were include data, text, image, and voice transmission, plus combination of all of them. Equipment and technological approaches will proliferate, and there will be no single description and categorization comman to all. We are on the threshold of rich variety of computer based message systems and their extension. In the 1980's CBMS that are available commercially have generally been called Electronics Mail or E-Mail.

3.3 TYPES OF E-MAIL SYSTEMS

In general most electronics mail system can be grouped in to four categories.

1. Keyed Entry Networks
2. Facsimile Transmission
3. Communicating Word Processors
4. Computer-Based Message System (CBMS) or E-Mail Services

Keyed Entry Networks or Record services, are those services that enable users to make station to station calls, similar to regular telephone service, where the actual communication involves electrical or electronic printing equipment rather than the voice telephone. These generally involve some sort of central or control computer message switching that has store and forward capabilities, rather than simple switching. That is, message can be received, held until they are requested, and delivered automatically to a variety of recipients on a list that may be data dependent or instruction dependent. The commonest record services are teletypewriter networks, such as TWX or Telex. These now has variety of enhancement available and there are newer services, such as Teletex and Videotex that are becoming available.

Facsimile Transmission, usually called FAX, this is technology for the transmission of an image over the communication lines. This may be a diagram, a picture, or a reproduction of a printed document. The transmission is in the form of electrical signals from one location to another, where the original image is reproduced on special equipment or paper.

Communicating Word Processors are simply text editing units with communications capabilities. These units can communicate with other communicating word processors as well as full range output devices. they are simply a way for one secretrial workstation to send and receive textual material to and from another workstation. Communicating word paocessors are frequently clustered with a small controlling computer and are used to balance secretarial loads, to cross edit, and to maintain cordinated files of finished material.

Computer-Based Message Systems (CBMS) or E-Mail Services pull together the other type of electronic mail, and allow more comlex equipment networking to be employed. They are simply any electronic mail system that networks microcomputer with a minicomputer or a host mainframe to exchange notes and memoranda at a single site or a multiple sites, with all distribution control and switching handled by an integrated computer system. They offer considerable efficiency for intraoffice communication, and can handle bulk of message switching and delivery that is required.

Most CBMS are microcomputer/host systems since they offer considerably more flexibility than dumb terminal/host systems. Automatic polling is available both from the host and micros. The information can be manipulated at the micros and returnd to the host network for further handling. The trend in local area networks to handle CBMS. These usually have less overhead and faster message passing than local networks that are simply a part of a larger system with considerable competition for switching time. A LAN system generally offers a greater variety of transmission capabilities locally, and handles the interconnection problems of a variety of equipment.

Inter Personal Message Services (IPMS) is a term to cover a range of technology involving personal computer and communicating word processors to send and receive message using a appropriate high-level software. The term is becoming widely used by the Consultative Committee for International Telephony and Telegraphy (CCITT), an international committee which is working on standards for data communications. One standard is for Message Handling System (MHS).

The CCITT message handling system standard (X.400) considers that there is a group of "message transfer agents" (MTA) to route message from originator to the receivers. These MTA essentially put messages in electronic "envelopes", and deliver them as packets through a Message Transfer Service (MTS). The MTS is designed to carry any form of electronic mail from ASCII text to voice and facsimile. The MTS allows a variety of formats for messages, but specifies a standard for one message form, the Interpersonal Message Service. The IPMS standard, which uses the MTS for delivery, specifies the internal format for messages.

3.4 GENERAL REQUIREMENTS OF AN E-MAIL SYSTEM

There are many specific functional requirment for an Electroni Mail System that depend upon specific needs. There are number of general requirments, however, which really define the flexibility of an EMS. Some of these are:

1. The EMS should provide store and forward message switching between all units on the network in the selective manner.

2. The EMS should provide a message network using standard computer terminal (such as ASCII, dial-up, synchronous) for use at all required locations.

3. The EMS should provide access to public services such as the Sources and Computer Serve, that may be desired for use.

4. The EMS should provide support an essentially unlimited number of switching nodes in the network, plus essentially unlimited number of addressable stations between the switches.

5. The EMS should be capable of single or multiple address delivery to an essentially unlimited number of stations, with a modifiable distribution list.

6. The EMS should support the possible future inclusion of a delayed voice capability, or avoice store and forward provision.

7. The EMS should provide the capability of secure message storage and transmission by means of message encryption.

8. The EMS provide a security system with a minimum effort for the verification and authentication of the sender of the message.

3.5 FUNCTIONS OF THE E-MAIL SYSTEMS

The functions of the EMS can be divided in to the five major parts, which may well be on separate pieces of equipment or with separate controller.

1. **Input** wiil normally be microprocessor at the sending end, although it may be other type of equipment. The input may be alphanumerics, graphics, digital siganl, voice, or video.
2. **Processing** may be on a central host computer, or on a dedicated minicomputer. The processing is concernt with data, text, graphics, error control, media conversion, integration of functions, and voice.
3. **Storage** will normally be attached to the processing computer, although it may be close to the terminal locations, in part at least. The storage is concernt with data, text, graphics, voice, and security.
4. **Communications** may be local area network, a private network, a public network, or a third party services. The communication deals with digital to analog conversion, protocol handling, error control, routing and rerouting, transport and distribution, and security.
5. **Output** wiil normally be microprocessor at the receiving end, although it may be other type of equipment.The output may be visual copy on screen, hard copy, graphics, digital siganl, voice, or video.

3.6 BENEFITS AND CONCERNS

The fact is that the electronic mail has many direct benefits that can be translated to cost saving if analyzed individually for specific circumstances. The general benefits are listed below.

Faster than conventional mail.

Faster than conventional electronic transmission.

Minimized the rekeying and rehandling of data.

Increased efficiency of handling data.

Increased availability of electronically stored data.

Acceptable cost of operation.

Personalized automatic filing of information.

Aids time management.

Reduces lost telephone lines.

In any discussion of benefits, the concerns should also be considered to be assure that the benefits can be realized in actual circumstances. The concerns are listed below.

Security of information difficult to control.

Reluctance of user.

Developement and maintenance of exapanded electroni files.

External information mixed with internal.

Possible incompatibility of equipment and systems.

Requirement of many intelligent terminals.

Support of large number of participants needed.

Keyboarding by managers.

CHAPTER 4

SOFTWARE : NETBIOS

4.1 INTRODUCTION

A stand alone PC requires three layers of software that work together and are quite similar to the layers in a large computer. Closest to the computer is the basic input/output system, or BIOS, that manages all serial peripherals. In the layer above BIOS is the disk operating system, DOS software. In the past, version of MSDOS and PCDOS through Version 3.0 allowed only single user application. The top software level is the applications software, which may be a DBMS, utility programs, etc.

Terminal in a LAN have the same basic software layers. The network server in the LAN must have additional utilities and layers added to this scheme. At the BIOS level, NETBIOS (Network Basic Input/Output System) is introduced. It sends and receives data from the network adapter card in the same way that the regular DOS BIOS communicates with the serial, parallel, and other communication ports. It thus allows the networking cards to communicate with the PC hardware in the network. At the next level a new disk operating system (currently DOS 3.1) is introduced. With it are programs called server and the redirector. The server allows every PC on the network to share its pheripheral devices. Redirector determines when data should go out to and be received by the network. An expanded application layer resides above the redirector level. It may conain security programs, utility programs, and menus.

4.2 THE NETBIOS SESSION LAYER

The responsibilities of NETBIOS Session Layer includes the identification of the personal computer to the network, session establishment and session termination between pairs of network systems (peer), and message transport. Network peerer are physically connected to the network and identified by names.

The **Message Delivery** for the NETBIOS session layer is performed either as a reliable virtual_circuit or as a datagram services. **Vertual_circuit** service delivers a message between pairs of users involes four basic procedures. First, network user is identified by a unique name or a group name, which is added to local name table. As an alternative to the name, the Permanent System Name (PSN) may be used. Next a virtual circuit session is established between a pair of network names. Once the session is established, message can be sent and/or received. When the messge transactions are complete, the session is terminated. The virtual circuit service messages are delivered to a specified destination in the exact order as sent from the source, without errors losses or duplication.

Datagram message service is commanly used for broadcast messages, where the destination is not specifically defined, and does not require establishment of session between pairs of users. For datagram services, the user name is traslated to a transport address and then passed to the transport layer. The datagram services do not provide the same reliability as virtual circuit services.

The Session Layer dynamically binds user names to transport addresses for both virtual_circuit and datagram services. A user is translated to a transport address by the Session Layer and then passed to the services of Transport Layers. Thus, session layer communication with a remote system requires only that the application know the user name, and not the transport address, in order to provide communication services between two users.

The NETBIOS maintains a local name table that holds a maximum of 64 user defined, ASCII character names plus one, hexadecimal permanent system name (PSN). The PSN is composed of ten bytes of zeros followed by a 6 bytes' unique adapter identification number. The PSN is always the first entry in the name table and can not be deleted.

4.3 NETBIOS COMMANDS

4.3.1 Introduction

NETBIOS commands provides an interface for programming the PC Link2 Network interface adapter. A NETBIOS command is presented to the PC Link2 NIA in the form of a data structure called a Network Control Block (NCB). Each NCB consists of different fields that provide information to the NIA about the command. The NCB also receives information from the NIA that reflects the status of the command processing. The following guidlines should be followed when issuing NCB command to the NETBIOS:

1. Ensure that the NETBIOS software is operating.
2. Ensure that a minimum stack space of 20 bytes is

available for each outstanding command that is issued.

3. Allocate the data structure for the NCB and fill all the necessary fields for the command being issued.

4. Allocate the buffer that are used by the command.

5. Load the NCB address into the ES:BX register pair, place a value of 01H in to the AH register, and issue an INT 5CH software interrupt.

6. Do not move or alter the NCB untill the command is completed processing.

7. Check the status of the completed command. Result codes are obtained from either the AL register or the return code field of the issued command.

4.3.2 Commands

The NETBIOS commands are divided in to following four categories:

1. GENERAL COMMANDS make it possible for the NETBIOS to control and monitor other commands.

reset sets the maximum number of session and commands.

cancel terminates processing of other command.

adapter_status provides information about the adapter.

2. NAME SUPPORT COMMANDS supports the identification of the personal computer on the network.

add_name validates and adds a name to the local name table.

add_group_name allowes a name to be used by other users.

delete_name removes name from the local name table.

3. SESSION SUPPORT COMMANDS establish and terminate network connections and transfer information between pairs of network systems.

call establish local or remote session.

listen waits for a call from another system.

hang_up terminates a session.

send transmits data to the session end point.

chain_send concatenates session messages.

receive_any waits for message from any network system.

session_status provides status of active session.

4. DATAGRAM SUPPORT COMMANDS are used to exchange datagram messages with other network users.

send_datagram sends a datagram to a specified name or group name.

send_broadcast_datagram sends a datagram to all network systems.

receive_datagram waits for datagrams from a name or group name.

receive_broadcast_datagram waits for datagrams from any name.

4.3.3 Command struture

The each command consists of fields that provide the information about the command. Command fields are used to send as well as receive information.Field information includes the session number, the users, and the addresses of the buffer that are associated with the processing.Command fields also information about wait or no-wait option and the status of command processing.

The wait option instructs the NETBIOS not to return control to the user program and to wait until the command is complete.The no-wait option instructs the NETBIOS to return control to the user program and continue with the next program instruction immediately after the NCB command is invoked.

the return code field of acommand provides information about the status of the command. Two types of the return codes are available for the command.

* immediate return codes
* final return codes

Immediate return codes are only for no-wait option commands.Final return codes are issued upon completion of all commands for both wait or no-wait options.

The data struture used for all Network Control Block commands is as shown below.

```
# define BYTE unsigned char
# define WORD unsigned int

struct NCB {
    BYTE    command;
    BYTE    retcode;
    BYTE    lsn;
    BYTE    num;
    char far *buffer;
    WORD    length;
    BYTE    callname[16];
    BYTE    name[16];
    BYTE    rto;
    BYTE    sto;
    WORD    off_post;
    WORD    seg_post;
    BYTE    lana_num;
    BYTE    cmd_cplt;
    BYTE    reserve[14];

};
```

Network Control Block (NCB) Struture.

4.3.4 Wait and No_wait Options

The application's interface into the NETBIOS varies according to whether the wait option or no_wait option is chosen.

The **wait** option command does not pass control to the next program instruction until the NIA completes processing of the command. When the wait option command completes, a final return code is placed in the AL register and the retcode field of the NCB.

With **no_wait** option control is immediately returned to the next program instruction and an immediate return code is placed in the AL register. When the no_wait option command completes, a final return code is sent to both AL register and the retcode field. The no_wait option allows for Queueing of multiple commands and is recommended for maximum throughput. No_wait option command can be specified to execute a posting routine upon command completion. The address of the posting routine is given in the post_addr field. No_wait posting routines are performed on an interrupt level with interrupts masked. If the post_addr field contains a value of 00H, the application is not interrupted for a posting routine.

4.3.5 Return Codes

Immediate Return Codes:

Immediate return codes are only valid for no_wait option commands and are received in AL register immediately after the NCB is invoked. An immediate return code other than 00H (cmmand complete) or FFH (pending) indicates that the command executed with an error condition. If the post_addr field is 00H, the application is not interrupted for a posting routine when the NCB completes. The cmd_cplt contains a value FFH (pending) until NCB completes. The table 2 in appendix A summarizes the immedaiate return codes that may be received in the cmd_cplt field.

Final Return Code:

For no wait option commands, a value change (from FFH) in the AL register indicates that command processing is complete. The value returned in the AL register is the same as the value returned in the retcode field. For wait option commands, the final return cade value is sent to the retcode field of the NCB when the command completes processing. The Appendix A has the complete ~~l~~isting of all return codes and recommended action.

4.3.6 NCB Field Description

NCB is used for the fixed size data structure that is allocated for each NETBIOS command. Not all fields are used for every command.

1. The command Field

This single byte field is the command code that indicates which command is to execute. Setting the high order bit to 1 selects the no_wait option. The wait option is selected by setting the high order bit 0. the remaining 7 bits determine the actual command that is executed by NETBIOS. Table 4.1 lists the command codes for each command.

Table-4.1 Command Codes

WAIT	NO_WAIT	COMMAND NAME
10H	90H	call
11H	91H	listen
12H	92H	hang_up
14H	94H	send
15H	95H	receive
16H	96H	receive_any
17H	97H	chain_send
20H	A0H	send_datagram
21H	A1H	receive_datagram
22H	A2H	send_broadcast_datagram
23H	A3H	receive_broadcast_datagram
30H	B0H	add_name
31H	B1H	delete_name
32H	.....	reset
33H	B3H	adapter_status
34H	B4H	session_status
35H	.....	cancel
36H	B6H	add_group_name

2. The <u>retcode</u> Field

The retcode field occupies one byte of memmory and indicates the completion status of an NCB command. A return code value of 00H in this field represents successful command completion, other values indicate incomplete operation or that the command executed with an error condition. In the Appendix A complete description of return code values and recommended actions.

3. The <u>lsn</u> Field

The local session number field is the 1 byte field representing a number assigned for a session with another network user. The local session is returned in this field after successful completion of a **call** and **listen** command. Ensure that correct values are used for **send** and

receive commands. When applied to the **reset** command, this field indicates the maximum number of supported sessions.

4. The num Field

The num is 1 byte field contains a value that is returned after executing **add_name** or **add_group_name** command. This number identifies the name table entry for the added name or group name and must be used for all datagram commands and receive_any command.

5. The buffer Field

This 4 bytes field pointer to a message buffer associated with a command. The maximum length of the buffer pointed to by this pointer is 64,535 bytes.

6. The length Field

the 2 bytes length field indicates the length in bytes, of the data that is to be transferred. For commands that receive data this field indicates the number of bytes to receive. After the data is received the length field is adjusted by the NETBIOS to reflects the actual number of bytes received. For commands that send data, the field indicates the number of bytes to send.

7. The callname Field

The 16 bytes callname field specifies the name of the user being called. This field may indicate a name in the name table of the local system or remote system. All 16 bytes of the callname field must be used. The **chain_send** command uses bytes 0 to 6 of the callname field to specify length and address of the second data buffer.

8. The name Field

The name 16 bytes field that indicates the local system through out the network. All 16 bytes of the name field must be used. The Permanent System Name (PSN) can also be used in the name field.

9. The rto Field

The 1 byte rto (receive time_out) field specifies, in increments of 500 milliseconds, the maximum amount of time that can elapse before a **receive** command returns the time-out error. For **call** and **listen** commands, the time-out period applies to all **receive** commands associated with the session. A value of zero indicates no time-out.

10. The sto Field

The 1 byte sto (send time-out) field specifies, in increments of 500 milliseconds, the maximum amount of time that can elapse before a send command returns a time-out error. For **call** and **listen** commands, the time-out period applies to all **send** commands associated with a session. A value od zero indicates no time-out. **Send** command time-out terminates the session on expiration and should be used with caution.

11. The post addr Field

The field is valid only with no_wait option command. The no_wait option is used to allow an application to continue processing while waiting for NCB command to complete. The post_addr field is a segment:offset address specifying a posting routine. When the command completes, the NETBIOS interrupts the application at the address specified in the post_addr field and the posting routine is executed. When the post_addr field is set to zero, the application is not interrupted for a posting routine.

12. The <u>lana num</u> Field

This 1 byte field indicates the adapter that is to receive the NCB commands. A value of 0 instructs the NETBIOS to issue the command to the primary PC Link2 NIA, the value of 1 for secondary adapter.

13. The <u>cmd cplt</u> Field

The field is useful only for no_wait option command that do not call posting routines (post_addr = 0). The cmd_cplt (command completion) is 1 byte field used to indicate that an NCB command has been completed. This field will contain an immediate return code.

14. The <u>reserve</u> Field

This 14 bytes field is reserved for NETBIOS and should not be altered.

4.3.7 NCB Commands Description

Under this topic we will describe some of the NETBIOS commands which are used in the E_MAIL System.

<u>NETBIOS GENERAL COMMANDS:</u>

1. Cancel

The cancel command issues a termination request to another command. The fields used by this command are as mentioned below:

```
BYTE     commnad              /* wait option only */
char far * buffer
BYTE     lana_num
BYTE     retcode             /* value returrned  */
```

The cancel command is used to terminate pending NCB commands. The address of the command that is to be canceled is specified in the buffer field of the cancel command. Caution has to be taken when cancelling a **send** command. Because the cancelled send command also terminates the session. the following commands can not be terminated by the cancel command:

add_group_name	reset
add_name	send_broadcast_datagram
cancel	send_datagram
delete_name	session_status

2. Adapter_status

The adapter_stattus command is used to obtain the information about alocal or remote adapter. The fields that are used for the adapter status command are mentioned below:

```
BYTE        command        /* wait or no_wait option */
char far   *buffer
WORD        length         /* value returned         */
BYTE        callname[16]
WORD        off_post       /* if no_wait option used */
WORD        seg_post       /* if no_wait option used */
BYTE        lana_num
BYTE        cmd_cplt       /* if no_wait option used */
BYTE        retcode        /* value returned         */
```

The **adapter_status** command provides status information for the adapter defined in the callname field. The status information is returned in the message buffer that is pointed to by the buffer field. When the message buffer has received the status information, a value is returned in the length field to indicate the actual number of bytes received in the buffer. If the specified length of buffer is too small to contain all the data, a return code of 06H return to the retcode field, and the excess data is lost. The actual message buffer ranges in size from 60 to 348 bytes depending on the number of names in the local name table. The PSN of the local or remote sytem can be obtained by this command. For more details refer [1].

NETBIOS SESSION SUPPORT COMMANDS:

Under this topic we will discuss NCB session support commands which are used to establish virtual circuit connection between systems, excahnge messages, terminate connections, and to provide the status of a virtual_circuit connection. The commands provides reliable data exchange for messages ranging in size from 0 to 65,535 bytes. More than one command may be active at the same time. A maximum of 32 session may be esablish at one time.

1. Call

The call command establishes a session with a local or remote name. The fields used by the call commands are mentioned as below:

```
BYTE        command           /* wait or no_wait option */
BYTE        lsn               /* session number returned*/
BYTE        callname[16]
BYTE        name[16]
BYTE        rto
BYTE        sto
WORD        off_post          /* if no_wait option used */
WORD        seg_post          /* if no_wait option used */
BYTE        lana_num
BYTE        cmd_cplt          /* if no_wait option used */
BYTE        retcode           /* value returned         */
```

The **call** command is used to establish a session connection between two users, as identified in the callname and name fields. The system specified in the callname field must have an outstanding **listen** command before the connection can be established. Virtual_circuit connection can be established for local/local or local/remote name pair. More than one session can be established between the same pair of names.

After completion of call command, a number is assigned and returned in the lsn field. This number is called the Local Session Number (LSN) and is used by other commands when referring to the established session. When the specified time_out interval expires (rto and sto fields), any unsuccessful send and receive commands that are associated with the session are aborted. The call command not aborted unless it is unsuccessful when the system time_out limits are reached.

2. Listen

The listen command enables a session between network systems waiting to establish a session. The fields used by the listen commands as mentioned below:

```
BYTE        command         /* wait or no_wait option */
BYTE        lsn             /* session number returned*/
BYTE        callname[16]
BYTE        name[16]
BYTE        rto
BYTE        sto
WORD        off_post        /* if no_wait option used */
WORD        seg_post        /* if no_wait option used */
BYTE        lana_num
BYTE        cmd_cplt        /* if no_wait option used */
BYTE        retcode         /* value returned         */
```

A session established between systems that issue **listen** and **call** command. A system that issue a call command (identified in the callname field) cannot open a session unless a name that is called (identified in the name field) has previously issued a listen command to recognize the caller. The asterisk (*) in the callname field indicates that the NIA is to listen for calls from any name on the network. Upon completion it also returns LSN in the lsn field. A listen command does not time out however it is a session entry and holds a status of "pending" for session information returned by the adapter satatus command.

3. Hang_up

The hang_up command terminates a session. The fields used by the hang up commands are as mentioned below:

```
BYTE        command         /* wait or no_wait option */
BYTE        lsn
WORD        off_post        /* if no_wait option used */
WORD        seg_post        /* if no_wait option used */
BYTE        lana_num
BYTE        retcode         /* value returned         */
```

The hang_up command closes a session with a previously assigned LSN. The hang_up command also terminates any associated pending **receive, receive_any, send,** and **chain_send** commands and sends the session closed return code (0AH) to the originator.

4. Send

The send command is used to dispatch messages to other system. The fields associated with the send commands are as followes:

```
BYTE        command     /* wait or no_wait option */
BYTE        lsn
char far    *buffer
WORD .      length
WORD        off_post    /* if no_wait option used */
WORD        seg_post    /* if no_wait option used */
BYTE        lana_num
BYTE        cmd_cplt    /* if no_wait option used */
BYTE        retcode     /* value returned         */
```

The send command transmits buffer data to other system. The transmitted data is taken from a message buffer pointed to by the buffer field. Data is transmitted on the FIFO basis when more than one **send** command is pending. The receiving system must issue a corresponding **receive** command. When the time out is reached, the command time_out return code (05H) is returned in the retcode field. Send command have priority over hang_up command. Pending local send command halt the hang_up command until completion of the send command or an elapsed time of 20 seconds, whichever occurs first.

5. Receive

The receive command accepts incomming messages. The fields used by receive command are as followes:

```
BYTE        command     /* wait or no_wait option */
BYTE        lsn
char far    *buffer
WORD        length      /* data length returned   */
WORD        off_post    /* if no_wait option used */
WORD        seg_post    /* if no_wait option used */
BYTE        lana_num
BYTE        cmd_cplt    /* if no_wait option used */
BYTE        retcode     /* value returned         */
```

The receive command accepts data transmitted by a **send** command for the session identified in the lsn field. Receive commands are processed in a FIFO order when more than one receive or receive any commands are pending. Time_out values are specified by a **call** or **listen** command. If the receive data buffer is too small to contain all the transmitted data, an incomplete message return code of 06H is assigned to the NCB. The remaining data can be retrieved by issuing another **receive** command before the time_out expires.

CHAPTER 5

E-MAIL ON NETBIOS

5.1 INTRODUCTION

This Electronic Mail System is one part of the VOICE MAIL SYSTEM ON LAN project carried out at Ernet project. The Voice Mail System consist of two parts one is the NETBIOS inteface and another is the voice degitizer card interface. This Mail System on NETBIOS has been implemented and interfaced with existing Voice Degitizer card inteface to develope the Voice Mail System on LAN.

An E-MAIL system is being implemented on NETBIOS, which requires an office environment where LAN exists. A dedicated mail server (micro computer) has been designed which is a node of LAN and has NETBIOS installed. As explained in earlier chapter that NETBIOS is required to establish a session between the server and workstation. Mail server stores all the messages and manages them for retrieval, deletion, store, and security.

The principle feature of the E-MAIL is that it operates in a store and forward manner. This means that the originator of message need not wait until the receipients indicates willingness to accept the message. Rather the originator can submit a message at any time convenient to him and equally the recepient can choose the time when he will actually read it. A further feature of this is that it does not require the originator computing system to be have a session to the recepient system. Instead the message is routed via some intermediate system which is capable of having a session to any system in the LAN. The originator can send a message to multiple recepients. It also has the following features:

--- Automatic time tagging.
--- Permanent storage on user request.
--- Variable length messages.
--- Friendly user interface.

Each mail system user is provided with a mail box. This mail box has all the message headers of the messages were sent to the user in a fixed format. The user can access his mail box via user ID and password. For additional security, user may encrypt the messages. Addition and deletion of the user name from this mail system can be performed by another routine at mail server end. All user ID and passwords are stored in a perticular file.

Only successful completion of authentic check will provide access to the real message system. For smooth handling of the user's all request related to a mail we have to send a corresponding message code at the server end and server sends back the response of the requested command. If user is not in interaction since a perticular time period (It can be changed) then server will hangup that perticular session.

This system can handle multiple user (fixed) simultaneously. To make multiple user environment we have used "polling" technique.

5.2 PLANNING AND OBJECTIVES

Electronic mail system represents a major change in the way managers run their daily routines and desk work. They are not just a quicker way of sending message but a new way of feeling retrieving and discussing the informations. It has been pointed out that there are variety of approaches to electronic mail and each has its set of benifits and concerns. The selection of a system therefore should not be arbitrary decision, but should be based on which features are most helpful and which type of system most likely to be popular.

The following are the objectives of the E-MAIL system:

Easier preparation of information.

Faster distribution of informations.

More complete and timely information available.

Reduced efferts in filling (read, store, and delete) and retrieving information.

More effective supervision and management.

More effective security.

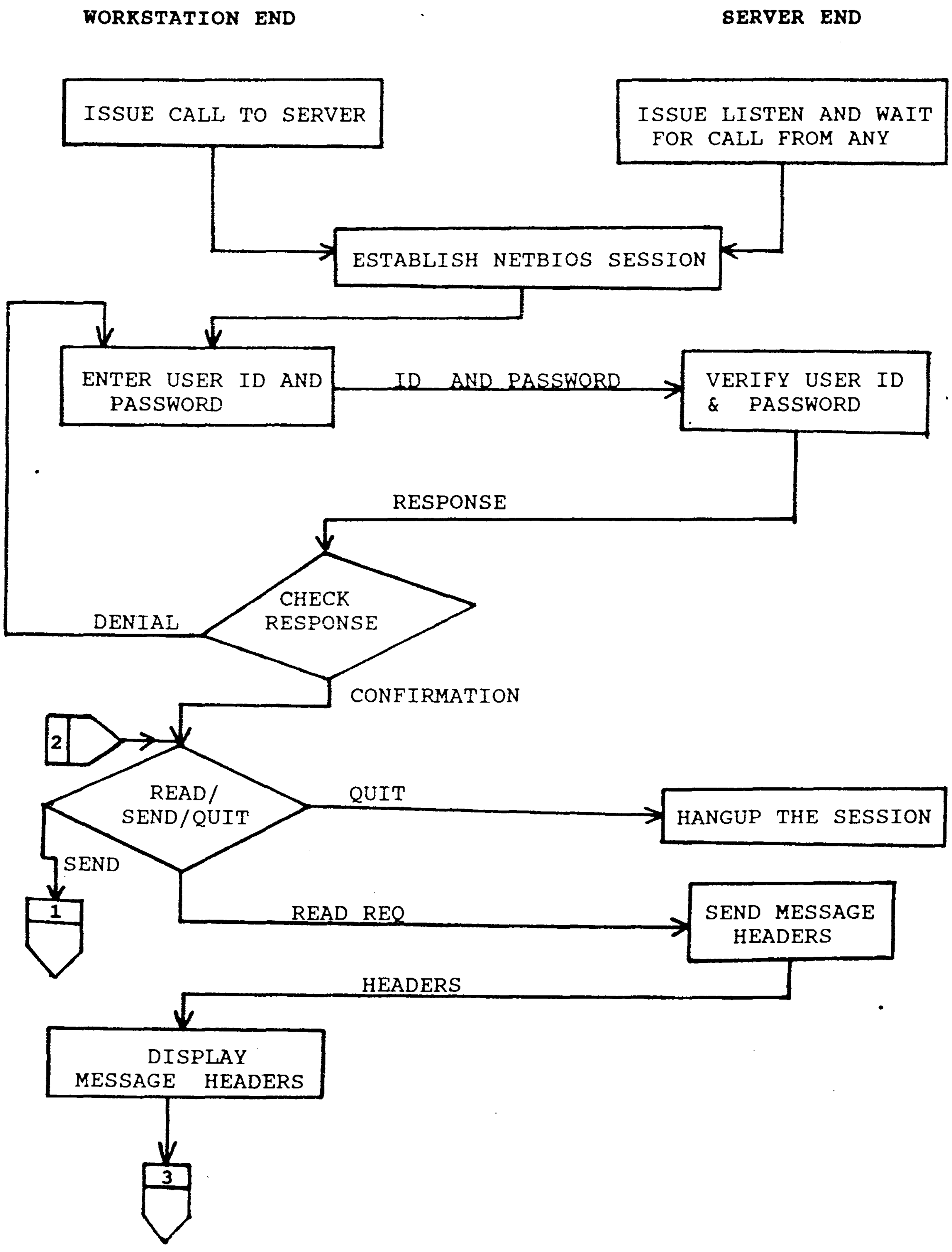
Voice mail facility.

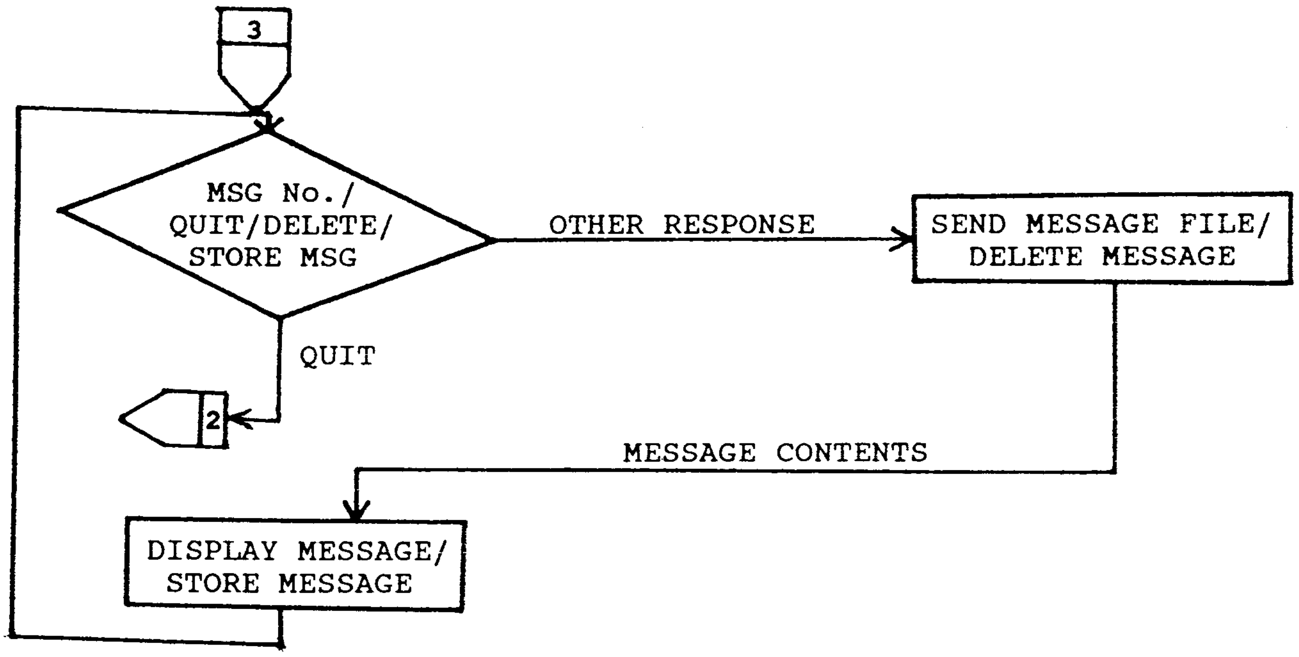
5.3 **E-MAIL SYSTEM** (FUNCTIONAL BLOCK DIAGRAM)

In this E-MAIL System there is **mail server** which manages all the messages. Only the mail server has the hard copy of each message. To send a message and to read a message to/from any system or user a session has to be establish between mail server and user's computing system. The functional block diagram of the E-MAIL system is as shown in the Fig-5.1.

As shown in the block diagram the server wait for a call from any system in network. After establishing the session the mail server issues a pending receive command for

FIG - 5.1 FUNCTIONAL BLOCK DIAGRAM OF THE MAIL SYSTEM




3
MSG No./ QUIT/DELETE/ STORE MSG
OTHER RESPONSE
SEND MESSAGE FILE/ DELETE MESSAGE
QUIT
2
MESSAGE CONTENTS
DISPLAY MESSAGE/ STORE MESSAGE

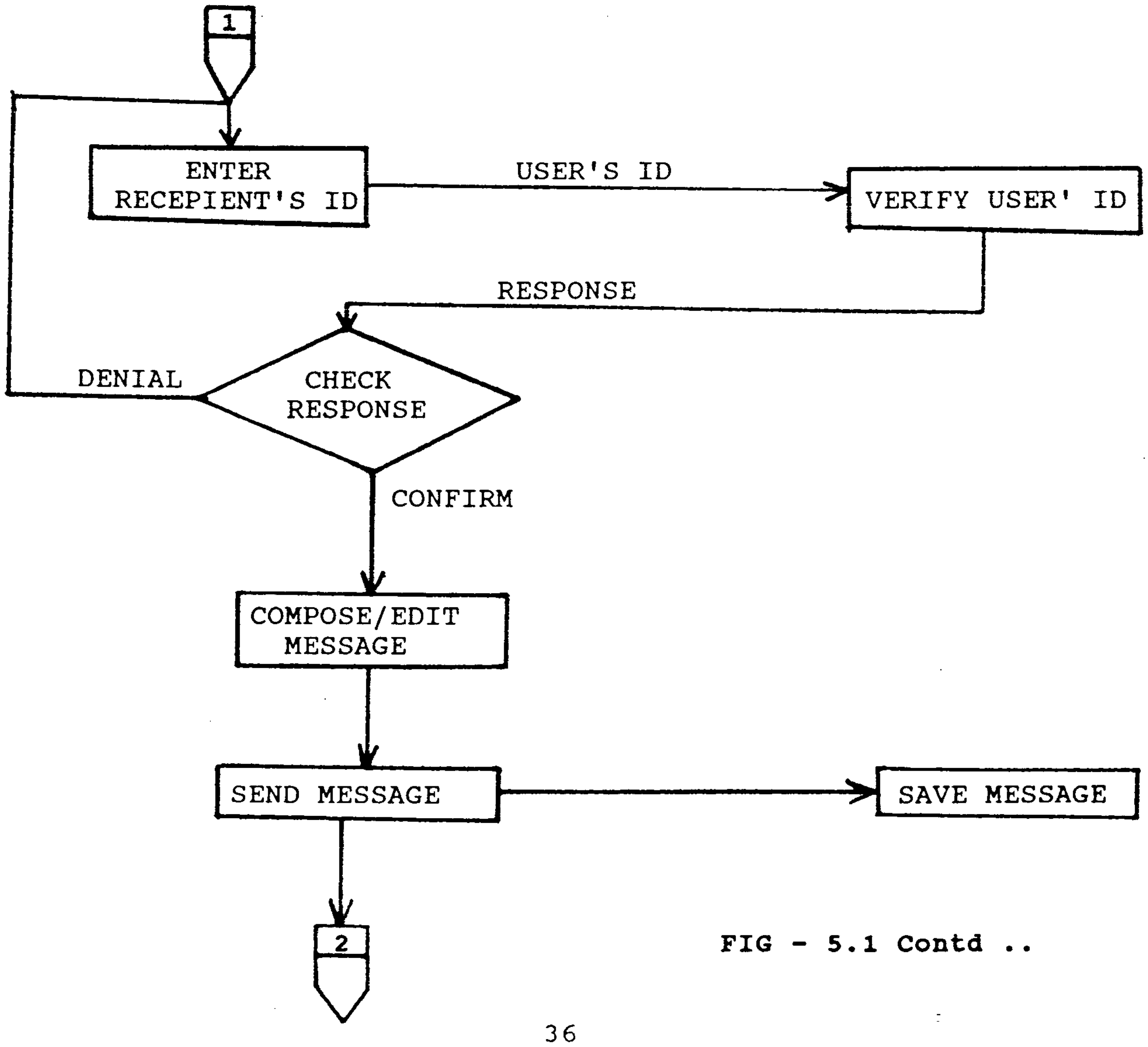

1
ENTER RECEPIENT'S ID
USER'S ID
VERIFY USER' ID
RESPONSE
DENIAL
CHECK RESPONSE
CONFIRM
COMPOSE/EDIT MESSAGE
SEND MESSAGE
SAVE MESSAGE
2

FIG - 5.1 Contd ..

this session, the user enters its ID and password. The user ID and password sends to the server in the perticular format. Mail server always has a pending receive for each active session. After receiving the user ID and password the mail server checks the the user ID and password with the already stored user ID and password. If it is not correct then user has to retype the ID and password to proceed further.

Only successful completion of authentication allow to access the mail system further. After all this check user will have three option with him.

1. READ messages.
2. SEND a mail.
3. QUIT (From the mail system).

If user wants to read a message then read request is sent to the mail server and issue a pending receive command to receive the response of the read request command. In response to this read request the server reads message header from the user's mail box and sends them at the user end. At the user end the message headers are displayed in a perticular format with sender's ID, time of submission and Subject.

After displaying the headers the user can read, delete, and store a perticular message by typing the message number which were displayed along with the message headers. From this message number we derive the message file name and send it to the mail server. If the message command is to read a message (mail) then server reads the message file and sends the contents of the this file to the user end. At the user end it will be displayed and then stored if the user requested to store the message at local drive. If the user request to delete a perticular message then server will delete the required message from the user's mail box. After quit from the read option user again comes to main options.

If user want to send a mail then he has to give the recepient(s) ID. These recepients ID are send to the mail server for recepients confirmation. Server checks the recepients ID and sends the results at the user end. Server also assign a message file name and creates a file with this name. After confirmation user is allowed to edit the subject or text. The file naming strategy and structure is such that we can have voice mail in between text part. This is explained later. In between editing of text we sends dummy commands at server to avoid time_out of pending receive command at the mail server. After completion of the text editing we sends the text along with the all verified recepients, to store the message at the mail server. Mail server prepares header and appends it to all recepient's mail box and stores the message in to the message file which created earlier. After completion of send a mail user again comes to the main options.

If the user option is to quit from the mail system. Then we send the quit request to the mail server and hangup the session at user end. After receiving this quit request the mail server cancels if any pending commands related to this session and issue a command to hangup the session.

All the session support commands at the mail server are with "no_wait" option to make this system as multiuser. Means more than one user can use this mail system at the same time. At the user end all the commands are issued with "wait" option.

5.4 MAIL SERVER SYSTEM

5.4.1 Introduction

This is one part of the E-MAIL on NETBIOS. This system can be install and run on any micro computer in the network. This system makes the computer as mail server. Before starting to execute this system the manager should check the following points:

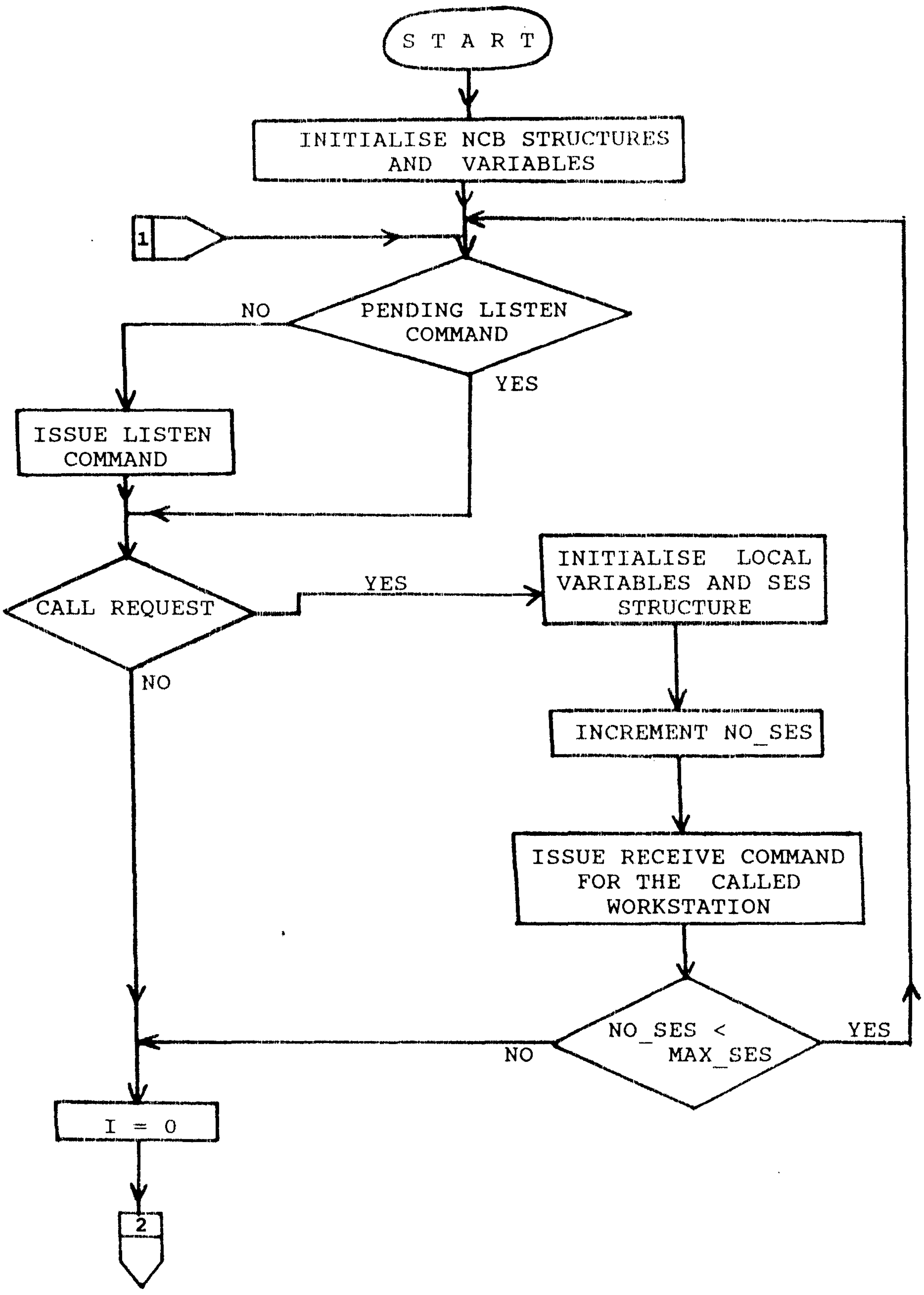
1. Ensure that NETBIOS software is operating.
2. Ensure that latest version of all the files related with this mail system should be resides in the current directory i.e. userlist, all the mail boxes and all the message files should exist.

This system will run continuously until some one wants to stop it or abnormal condition occures. This system handles all the message request, related to the mail system. A special care has been taken in designing this system that the system should not enter to an infinite loop except the main loop. It should not serve a perticular user for long time to reduce the response time in multiuser case. After responding one message for an active session it responds to a pending message of another active session. There is a maximum limit of the number of active session supported by this system at the same time.

5.4.2 Block Diagram Description

The block diagram of the main loop of the mail server is as shown in the Fig-5.2.

FIG - 5.2 FLOW CHART OF MAIL SERVER MAIN LOOP


START
INITIALISE NCB STRUCTURES AND VARIABLES
1
PENDING LISTEN COMMAND
NO
YES
ISSUE LISTEN COMMAND
CALL REQUEST
YES
NO
INITIALISE LOCAL VARIABLES AND SES STRUCTURE
INCREMENT NO_SES
ISSUE RECEIVE COMMAND FOR THE CALLED WORKSTATION
NO_SES < MAX_SES
YES
NO
I = 0
2

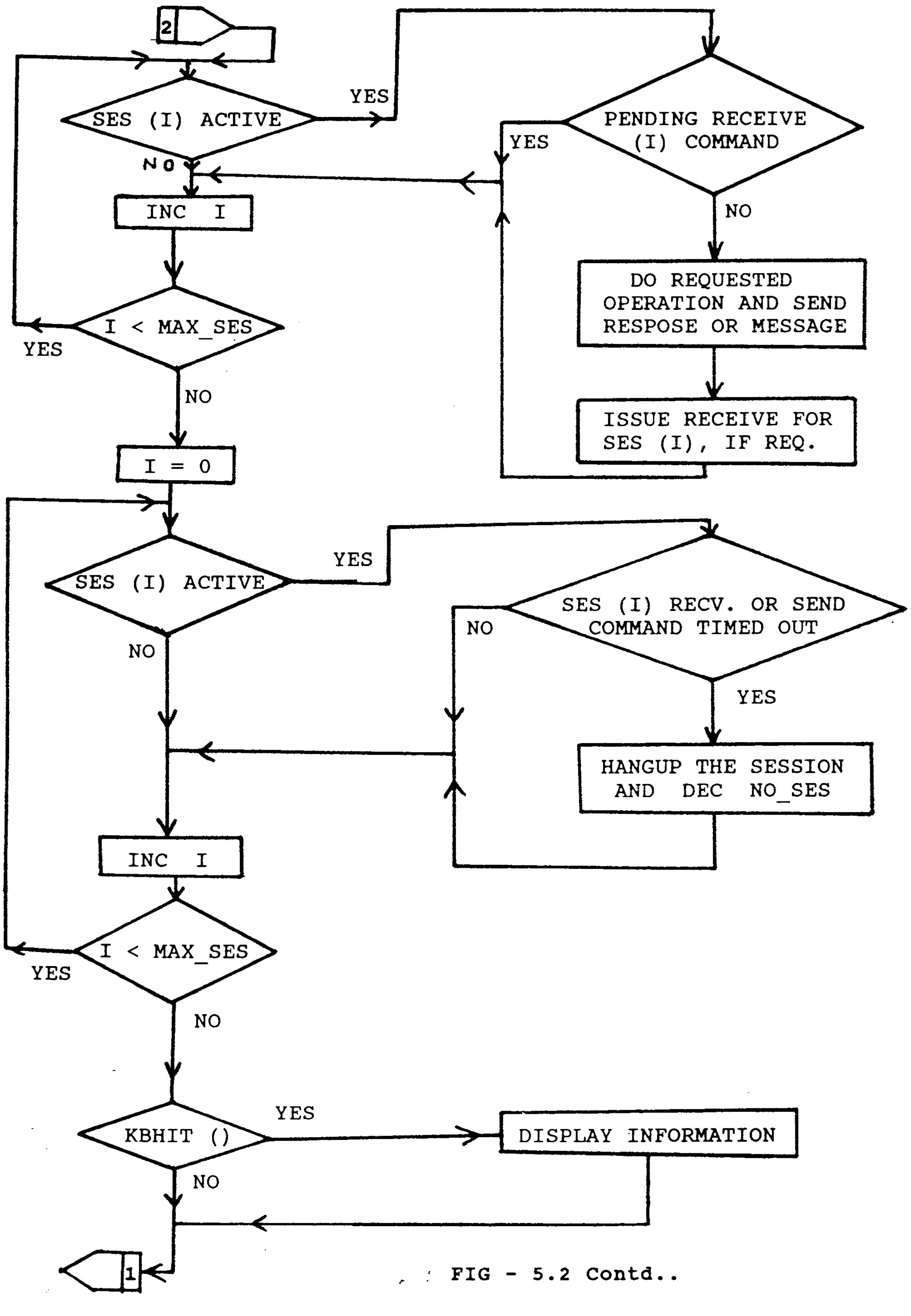

2
SES (I) ACTIVE
YES
NO
PENDING RECEIVE (I) COMMAND
YES
NO
INC I
DO REQUESTED OPERATION AND SEND RESPOSE OR MESSAGE
I < MAX_SES
YES
NO
ISSUE RECEIVE FOR SES (I), IF REQ.
I = 0
SES (I) ACTIVE
YES
NO
SES (I) RECV. OR SEND COMMAND TIMED OUT
NO
YES
HANGUP THE SESSION AND DEC NO_SES
INC I
I < MAX_SES
YES
NO
KBHIT ()
YES
DISPLAY INFORMATION
NO
1

FIG - 5.2 Contd..

Before starting the description we should discuss about the variables, fixed numbers and the structure used in the main loop of the mail server system block diagram.

I is an integer variable.

MAX_SES is the number of the maximum active session that can exist at a time.

no_ses number of active session existing.

There is one session structure alloted to each active session with server. The following are the fields of the session structure.

```
struct session {

    int flag;              /* session active or not          */
    int fflag;             /* file pointer legal or illegal */
    int fptr;              /* file pointer                   */
    struct NCB rncb;       /* NCB for receive command        */
    struct NCB sncb;       /* NCB for send command           */
    char user[9];          /* user ID who is ungaged         */
    char dir[15];          /* message header file name       */
    char text_file[8];     /* message file name without ext */

} ses[MAX_SES];
```

Before entering to the main loop the server initialises the session structure and NCB structure and load userlist file in to the memory. After initialisation it enters into the main loop. It issues a listen NETBIOS command for any node if there is no pending listen. This listen will remain pending till some one executes call command to the server. After successful completion of the listen and call commands server reserves one session structure for that session and initialize it. Then increase the no_ses by one and then issue one receive (no_wait option) command in pending for the called workstaion. Now if the no_ses is less than MAX_SES then server will issue another listen command, in this way there will be always a listen command in pending when no_ses is less than the MAX_SES.

Now it enters to a loop in which it checks all the session structure one by one. If the session structure is associated with an active session then it checks the status of the **receive** command. If the receive command related to this session is not pending then do operation according to the message received from the workstation. The list of the message code and operation to be done is given in artical 5.4.3. After this, send (no_wait option) the response or message to the workstation and then issue another receive command for this session to receive next message code. In this way there will be always a pending **receive** command for each active session.

After this it enters to a second loop in which it checks for all active session one by one, if the receive or send command timed_out then hangup that session, decrease the no_ses by one and flagged off the session structure so it can be reserve for another new session. The time of time_out of these pending commands is decided during the completion of **listen** and **call** commands. It is same for all session because we are using same NCB structure for all listen commands.

After this it checks for any key board entry. If the typed character is legal then server will diaplay the corresponding information. Like we can get the mail address (user ID) of all the users of this mail system. We can also get the list of all the currently interactive users. We can shutdown the server by typing relevant command at the server side. After responding key board hit it start from the starting position of the main loop, if the server is not stopped. In this way it wiil remain in the main loop and serves to all the interactive mail users.

5.4.3 Message Codes And Operation

As shown in the functional diagram FIG - 5.1 if server has received some message from any node then it has to do some operation acccording to the request of the mail user. For the smooth functioning of the system we have defined some codes and the operation to be done at the server end. The codes and corresponding operation are as listed below:

Commande Code **'A'** Verify User ID and Password.

This command is send when user wants to login to the mail system. The server has to verify the user ID and password and then has to send the respose at user end.

Command Code **'B'** Read Request.

This command is used when the user wants to read his mail. In response to this command server reads the mail box and forwards the message headers to the user computing system.

Command Code **'C'** Send Message Number.

This command send to the user when user wants to read a perticular message. The server will send the contents of the message having the message number asked by the user.

Command Code **'D'** Play Back Request.

Command Code **'E'** Start Voice Play Back.

The above mentioned both the command codes are used to listen the voice mail during reading of the mail.

Command Code **'F'** Send Request.

This command tells to server that user wants to send a mail. The server verifies the recepient's IDs which are along with the send request command.

Command Code **'G'** Voice Record Request.

Command Code **'H'** Voice Record Start.

Command Code **'I'** Review The Just Recorded Voice.

Command Code **'J'** Delete The Just Recorded Voice.

All the above four mentioned commands are used to record the voice at the server end, for a voice mail.

Command Code **'K'** Save The Message.

This commands informs to server to store the message contents for the recepient(s). The format of this command is described in the # 5.4.4.

Command Code **'M'** Quit.

This command informs to server that user wants to quit from the mail system. The server hangup the session after receiving this command.

Command Code **'O'** Change Password.

This command send to server when a user wants to change his password. The server will check old password and if find correct then replaces old password with new password.

Command Code **'P'** Voice Record Stop.

Commnd Code **'Q'** Create Text File.

On receiving this command code server assigns a unique file name to the message and creates the file with the same name.

Command Code **'R'** Dummy Command.

This command is used to avoide the time_out of the pending receive command at the server side. The server again isuue a receive command.

Command Code **'S'** Delete Message.

This code is send when user wants to delete a message. The server will remove the message header from the mail box and then delete the text file if no other user has the same mail.

Command Code **'T'** Delete Text File.

This command is used when the message length is zero. The server will delete the corresponding text_file which was created earlier.

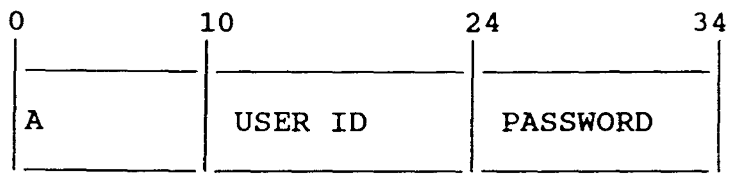
After every operation (except quit) the server issues a receive command.

5.4.4 Important Message Structures

The message code and the related data will be received in the buffer associated with "rncb" of the session structure. The first byte of this buffer will be always the message code and the remaining bytes will be the data associated with this message code. The structure of these data is different for different mesage code. Out of these message structures some important message structures are discussed below.

1. Verify User ID and Password

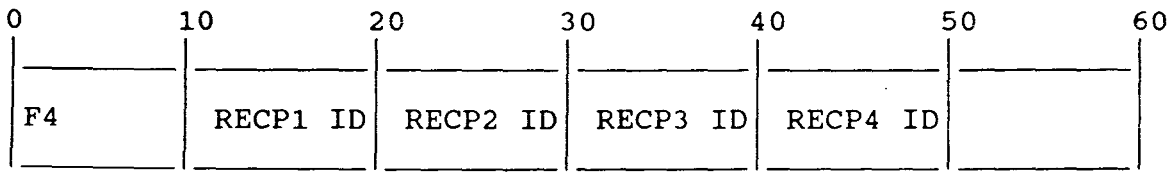
This message is send to the server when user wants to login to the mail system. The message structure is as shown below:



The 0th location of buffer has the command code 'A'. The user ID is stored from 10th to 24th location and password is stored at the 25th to 34th location of the buffer. The message buffer length will be 35 bytes.

2. **Send Request** (Recepient's ID Verification)

This message is send to server when a mail user wants to send a mail to other user(s). The sender have to enter recepient(s) ID. The format of this message is as follows:



At the 0th location the command code 'F' is stored. The 1th location will have the number of recepients to be verified, this nuber can be maximum upto 10. The recepient ID will be stored from 10th location of the buffer. Ten bytes are used for each recepient ID i.e. 1st recepient ID is stored at 10th to 19th location, 2nd recepient ID is stored at 20th to 29th location and so on. The length of buffer to be send is depend on the number of recepients to be verified. The length can be calculated as follow:

LENGTH = (R * 10) + 10 Bytes

Where 'R' is the number of recepients.

3. Save The Text

This message is send after the completion of composition of the message. After completion of the message the server has to prepare a message header and then append it to the mail box of each recepient and then store the message into the text_file which has been created on send request message command. So with the message we have to send all the verified recepint's ID along with the composed message.

If the size of message is smaller than 900 Bytes then we can send the complete message in a single **send** command. But if the mesage size is greater than 900 Bytes than we have to sent remaining message in next **send** command, but recepient's ID is need not be send again. We have to send the information that the buffer contains recepient's ID & 1st part of the message or last part of the message or the other part of the message. So that according to the message received the appropriate operation can be done on the message text_file.

All these informations are send through the 2 Bytes in the sbuff. The message stored in the sbuff will be as follows:

```
sbuff[0] = 'K'      /* Command Code */
sbuff[2] = 0/1
```

If sbuff[2] = 0 means the buffer contains the recepient's ID also and the message part is the first part of the message and message starts from the 110th location onward. In this case server has to append message headers in the recepient's mail box and open the text_file to store the message.

If sbuff[2] = 1 means the buffer contains the message part only and message start from the 10th location of the buffer.

sbuff[3] = 0/1

If sbuff[3] = 1 indicates that the message is uncomplete and more message is there to send.

If sbuff[3] = 0 means the buffer has the last part of the message and after storing it into the text_file the file should be closed.

sbuff[4] location contains the number of the verified recepints whom the message were sent. The length of the buffer to be send depend on the size of the message.

4. Delete Message

This message is send to the server when user has come out from the read mail option and deleted some messages. The format of this message contains only the total number of messages to be deleted and the message numbers. From the message numbers the server derives the location of the message header in the mail box. The format is as below:

```
sbuff[0] = 'S'     /* Commnad Code */
sbuff[2] = Number of messages to delete.
sbuff[11] and onward will have the message number.
```

The message number will be less than 256 so we can store in a single byte. The length of this message will be 11 more than the number of the messages to be deleted.

5.4.5 Mail System File Structures And Management

As mentioned earlier that there exist a file named as userlist, it contains all the user ID their related password. This file is read at the starting of the mail server and stored into the memory in a doubly linked list data structure. This link list is used during the users authentication. The addition and deletion of any user name is done through an another software (pass) before loading the mail server.

At the time of adding a new user name into the mail system creates a file for each user that is called as mail box. The name of this file will be same as user ID with extension ".HDR". Each user will have its own mail box. This mail box will contain only header part of a mail which was sent for the user. The length of one message header is 80 bytes. The structure of each message header is as follows:

8 bytes	--19 bytes--	--- 40 bytes ---	--12 bytes --
sn's ID	date & time	Subject	M. file name

The first 8 bytes of the message header will have the sender's address or the ID. The next 19 bytes are for the time and date of the submission of the message. The next 40 bytes are reserved to store the subject of the message, and the remaining 12 bytes are reserved for the name (without extension) of the message file. This header is prepared at the server end and appended to mail box each recepient,s of the message. As mentioned earlier that one mail can be send to more than one recepient. All the recepient which are going to be share will have the same message header and message file.

To built a message header we need a message file name in which the message will be stored. At the server side we have a routine which derives a unique file name which is not already exist in the mail directory. After getting the name of the file it creates a file with this name having the extension ".Tnn" where the 'T' denotes the text file and 'nn' is the number of the users, will share this file. The message will be store in this file. This naming technique supports the voice mail in between the text. The voice file may have the same name with the extension ".Vnn". Where 'V' indicates that it is a voice file and 'nn' gives the number of voice file in the message file having the same name with different extension.

This whole structure support easy deletion of a message from mail box. If a user wants to delete a perticular message then derive the message text file name from the header in the mail box. Then from a routine we gets the full name of the text message file now if the extension ".Tnn" has the value of 'nn' as one then delete all the files with this name else rename this file with same name having extension as ".Tmm", where 'mm' = 'nn' - 1. Which means that the number of users which will share this file has reduced by one.

5.5 E-MAIL SYSTEM AT WORKSTATION

5.5.1 Introduction

This part of the mail system is complement of the mail server which is discussed earlier in # 5.4. This system has to be run at the users computing system which should be a node of the same LAN in which the mail server is connected. If any mail system user wants to send or read a message then he has to run this system. This system is bound to establish a session between the mail server and work station.

After establishing a session with the mail server the user can have a series of conversation with mail server until user do not want to quit from the mail system or user timed_out due to non interactive with the mail server since a long time (3 minutes approx).

5.5.2 Login To The Mail System

Just after completion of call and listen command or establishing the session between the server. The system ask for the user ID and password. This user ID and password are send to the mail server for verification. If the user ID or password is not correct then user will get two more chances to reenter the ID and password. If user fails to enter the correct user ID and password then this system sends an information to the mail server to hangup the session and then exit from further execution. This system provides protection from the false messages from the unauthorised user, and access of messages to unauthorisd user. Only the user who knows the correct user ID and corresponding password can send and read a message or login to the mail system. After login user have one choice out of the three options (read, send, and quit from the mail system).

5.5.3 Message Composition And Send A Mail

If the user wants to send a mail then system ask the destination addresses or the recepients ID, whom the mail is to be send. User can enter a list of recepients ID in a single line (not more than 10). After typing of the list, this system sends this list at the server end for verification of the recepient's ID. If there is any incorrect recepient ID then system will give a chance to retype the correct recepient ID before starting the composition of the message. After completion of the recepient's ID verification the system provides a small editor to edit the message. This editor is not much efficient but user can enter a line, delete a line, insert some text in between two lines, and listing of the edited text.

User may have bigger message to type and it may take longer time than timed_out duration of the <u>receive</u> command of the mail server, then mail server will hangup the session. To encounter this problem we sends a dummy message to the mail server at every hit of return key. This dummy message simply use the pending **receive** command at the server and issue a new <u>receive</u> commnd. So the user may not timed_out during editing of message.

User have to type **^D** at the and of message. After end of the editing of the message, the text along with verified recepients ID and some necessary information transfer to a send buffer for sending it to the mail server for storing this message. After completion of message send, the system again ask users option.

5.5.4 Read or Actions On Receipt Of Mail

If user have option to read his mail then this system will send a read request message to the mail server to receive the mail headers from the user's mail box. After receiving the message headers this system will diaplay the message headers after numbering them.

Now user can have one of the following choice on the received mails:

1. Read a mail.
2. Store the mail. (s)
3. Delete a mail. (d)
4. Quit from read. (q)

For **reading** a message user have to type only the message number. From this message number system derives the message file name from the message headers and send a request to the mail server to receive the contents of the message. Now the contents of the message are displayed along with the message senders name, time and date on which the mail was send and the subject.

The another action that may be taken after reading a message is to **store** the message on to the local storage. So that the recepient can have a hard copy of the recveived mail. To store the mail at local storage, user has to type **'s'** then system will ask the file name where the message (which was read latest) to be store. After getting the file name the system creates the file and store the message with header in that file.

The another action that may be taken on receipt of electronic mail is to **delete** it. The recepient may not want to cluttering of messages or even filling the holding storage for delayed delivery of the message. Therefore there must be a possibility of deleting message, if it is found of no interest or it has past its time of interest. For deleting the message recepient has to type **'d'** then system will ask the message number to be delete. Then system flagged that message as discarded message. If user wants to delete another message then has to type again 'd' and then the message number. The information to delete some message is send to the mail server after user have **quit** from this read mail option by typing **'q'**.

5.5.5 Quit From The Mail System

It is recepient controlled system. If recepient wants to quit from the mail system then user can select the quit option from the main menu. After selection of this option the system sends an information to the mail server to hangup the session and then exit from the further execution.

5.6 OTHER RELATED SOFTWARE

Along with the mail system some necessary and important software routine has been developed. Which are related with mail management and status informations.

5.6.1 The pass

In a mail system mail manager should have a facility that he can add a new user name or delete an existing user name from the mail system, and the list of all the existing user of the mail system. This routine provides all these facility to the mail manager. On execution of this routine along with add name, delete a name and list of user the manager has 7 options but these three are of important use.

Add a new user name option asks the user ID and then password. After reading user name and password it is stored into the userlist file, and creates a mail box for the newly added user name. All the user ID are stored in this file in sorted doublly linked list.

To delete an existing user from the mail system not an easy task as add a user name. Because to delete a user means to delete all the messages related to the corresponding user. Before deleting a user name we have to read the user's mail box and from mail box access the message file name and then delete all the message files related to the user. On deleting the message files we have to take care of the message files which are shared by other users. After deleting the message files delete the mail box and then remove the user ID from the 'userlist'.

With the list of user option the mail manager can have the list of all the users of the mail system. In this option system reads the userlist file and then displays the all users of the mail system.

This routine should only be used by the mail manager or an authorised person. No other user should have access to this routine otherwise any body can delete or add a name. This routine should be executed at the server system.

5.6.2 The passwd

This routine is related to personal security.

The user of the mail system may wants to cahnge his password after some time, or after disclosing of existing password then user should have a facility to change the password. This routine provides the facility to cahnge the password from any node and for any user.

This routine establishes a session between user workstation and mail server, then ask user's ID, old password, new password, and again new password for confirmation. The user name along with old password and new password send to the mail server then after verification of autorised user the old password is replaced by new password and then stores in to the userlist file.

5.6.3 The adastat

There are two types adastat routine one is local adapter satatus **ladastat**, and another is remote adapter status **radastat** . As we knouw each node in the network has NIA (Network Interface Adapter) card. With these routines user can get the status of any adapter (local/remote) card in the network.

With the remote adapter status routine user has to specify the node name as the command line parameter. Along with the node PSN (Permanent System Name), number of session, and node's local name table, these routine gives more information about adapter status. More detail to about adapter status refer [1].

5.6.4 The sesstat

This routine gives the session status of the local node. this routine uses session status METBIOS command. This routine lists all the active sessions along with session number, session status, local and remote system names, and total number of outsatnding receive and send commands. For more detail about session status refer [1].

5.6.5 The test

This routine is used to test the NETBIOS installation. This obtained by issuing an INT 2FH software interrupt with a value of B951H in AX and 00FFH in BX register. If the NETBIOS interface is installed the BX register will change to a value of 01H. Otherwise the value unchanged to 0.

CHAPTER 6

CONCLUSION AND FUTURE IMPROVEMENTS

The main objective to develope an E-MAIL system on top of NETBIOS is to provide system which can support the **Voice Mail** facility. So this system is designed to provide backbone for a Voice Mail System. This system can developed as Voice Mail System (with PABX) in which you can have voice mail in between the text part of a message.

Although this E-MAIL System has all basic features of an Electronic Mail System i.e. send a mail, read a mail, delete a mail, store the mail at loacl storage, security etc. but there are few things which can be improved or implement to increase the features of this present E-MAIL System.

1. Replacement of the line editor with a screen editor. So the mail sender feel comfortable during composition of message.
2. Implementation of forwarding of received message. Means a mail recepients can forward the same mail to another user after receipt of the mail.
3. Implementation of a feature that a mail sender can send a stored record as a message.
4. To improve the security in the system the information stored in the "userlist" file can be encrypted. This file contains all the mail user's ID and their password. Further the message to be stored can also be encrypted before

sending it to the server and during display of the message at recepient end it should be decrypted.

5. The maximum number of total message files this system can support is 676. This figure can be increase by a small modification in the "newfile" routine in mail server. The system can handle maximum 125 messages for a single user, this can also be increase if required.
6. This system interpret user ID as mail address. We can have mail address different from user ID to make it more flexible.
7. Implementation of the feature that the mail manager can add new user name or can delete a user from any node in the network without stoping server.
8. Graphics textual mixture, which may be microcomputer video screen or may be which transmitted to graphic output machinary.

APPENDIX

This appendix lists all the return code and recommended actions. The codes are listed include both immediate codse and final return codes.

00H Successful command completion.

The execution of command has completed successfully. No action is required.

01H Illegal buffer length.

This message indicates that an invalid length is specified for a buffer that is associated with one of the following commands **adapter_status, session_status, send_datagram,** or **send_broadcast_datagram.**

Reissue the command after specifying the correct buffer length.

03H Command not valid.

The command code is not valid, reissue the command using the correct code.

05H Command timed_out.

If this return code is received for a **call** or **adapter_status** command, the system time_out period has expired. Ensure that correct name is used and reissue the command.

For a **receive, receive_any,** or **send** command, the 05H return code is received when the command time_out period has expired. For receive or receive_any command, reissue the command. For send command this return code means that the session has been abnormally terminated. Establish the aother session and reissue the command.

If the return code is received for **hang_up** command means that the 20 seconds time-out period has expired. The session has been terminated abnormally. No further action is required.

06H Incomplete message.

Only part of the message is received. This is because the specified buffer is not large enough to receive the full message.

08H Illegal local session number.

The specified session number is not an active session number. Specify the active session number and reissue the command.

09H No resources available.

This return code indicates that there is insufficient adapter space for the session. Reissue the command at a later time.

0AH Session closed.

The session is closed by the local or remote system. This notification is received for pending **send, receive,** or **receive_any** commands. If this return code is received for a **hang_up** command it means that session has been closed by remote system.

0BH Command cancelled.

If the cancelled command was a **send** or a **chain_send** command, then the session was abnormally terminated. No action is required.

0DH Name duplicated in local name table.

This return code occurs in response to an attempt to specify a name that already exist in the local name table.

0EH Local name table is full.

The local name table can hold a maximum of 16 names. The maximum number of names already been added in name table. Reissue the **add_name** or **add_group_name** after a **delete_name** command is issued.

0FH Command completed. Name has active sessions, now de_registered.

This message means the name is to be deleted is currently active in a session, but is flagged as de_registered. When the active session is ended, the name is removed from the name table.

11H Local session table full.

This means that insufficient table entry space exist in the session table. Wait for a session to close and thereby provide the available space.

12H Request for session open is rejected.

The remote system does not have an outstanding **listen** command. Wait until the listen command is issued at the remote system.

13H Illegal name number.

The name table entry number, as specified in the <u>num</u> field, is incorrect. The original name number must be used.

14H Cannot find callname or no answer.

The specified callnmae cannot be found or did not answer.

Ensure that specified call name is correct, then reissue the command using the correct or different callname.

15H Name not found, cannot specify * or 00H.

One of the following condition exists:

The specified name is not in the name table.

The specified name contains an asterisk (*) in column one of the name field.

The name is specified as 00H.

Reissue the command using the correct name.

16H Name is being used at remote adapter.

This return code indicates that the name is already in use. Specify another name and reissue the command.

17H Name deleted.

This return code is received when a name is deleted and no outstanding **listen, receive_any, receive_datagram,** or **receive_broadcast_datagram** commands exists for the specified name. No action is required.

18H Session abnormally terminated.

This return code means one of the following condition is true.
The remote system is powered off.
The Ethernet cable link is broken.
The session **send** or **chain_send** command has timed_out or cancelled.

A **hang_up** command is timed_out.

Check the status of the remote systenm and check the cable connection.

19H Detected name conflict.

The network protocol detects two or more identical names on the network.Duplicated names should be deleted immediately.

1AH Incompatible remote system.

An unexpected protocol packet was received. Ensure that all network system agree and use the same protocol.

21H Interface busy.

This return code indicates that the NETBIOS is busy processing an interrupt handler routine. Return and try again later.

22H Too many outstanding commands.

The maximum number of outstanding commands has reached. Retry at a later time when less commands are outstanding.

23H Invalid number in the lana num field.

This return code means that an attempt was made to specify a number other than 00H or 01H. Specify the correct value and reissue the command.

24H Command completed during cancel.

The command being cancelled has already completed or does not exist. No action is required.

25H Reserved name specified.

The name that has been specified is a reserved name and cannot be used. Specify another name and reissue the command.

26H Command not valid for cancel.

The following commands cannot be cancelled:

add_name	add_group_name
cancel	delete_name
reset	send_datagram
session_status	and send_broadcast_datagram.

4xH Unusual network condition, x can be any number.

The NETBIOS is unable to define the condition encountered. Try the command again.

50H - FEH Adapter malfunction.

The Intel PC Link 2 Network Interface Adapter detects an internal problem. Retry the operation or contact an authorized service representative.

FFH Command pending.

The command is pending. No action is required.

Table - 2 Immediate Return Codes

CODE	MEANING
00H	Successful command completion
03H	Command not valid
21H	Interface busy
22H	Too many outstanding commands
23H	Invalid lana_num value
24H	Completed during cancel command
26H	Not a valid command to cancel
40H-4FH	Unusual network condition
50H-FEH	NIA malfunction
FFH	Pending

BIBLIOGRAPHY

1. PC Link2 Software Developer's Manual

   Intel corporation.

2. Electronic Mail By:- John A. Bauckland

   Critical Issues in Information Processing Management And Technology Volume - 4.

3. Micro Computer Networking By:- John A. Bauckland

   Critical Issues in Information Processing Management And Technology Volume - 4.

4. Innovation In Electronic Mail

   By :- H. M. Vervest

5. Current Trends In Electronic Mail Security

   By :- K. V. Murphy

   185 - 196, Message Handling System 1987.

6. iNA 960 Programmers Reference Manual

   Intel Corporation

```
/**********************************************************************
*
*       This program is to check the NETBIOS instalation.
*       If NETBIOS installed then it returns BX = 1,
*       otherwise returns BX = 0.
*/
#include <stdio.h>
#include <dos.h>

main()
{
    union REGS r;

    r.x.ax = 0xB951;
    r.x.bx = 0x00FF;

    int86 (0x2F, &r, &r);

    printf ("BX=%X",r.x.bx);
}
```

```
/*******************************************************************
*
*       This program is to find the session status of any node in
*       the network.
*/
#include <stdio.h>
#include <string.h>
#include <dos.h>
#include "ncbs.h"

#define Success 0
#define Failure -1
#define Pending 0xFF
#define status  0x34

struct  NCB ncb;                /* The NCB */
byte    buff[347];      /* Data Buffer for the send NCB */

union   REGS    reg;
struct  SREGS   seg;

char    signon[] = "\n SESSION STATUS\n\n";

main()
{
    int i,len,p;

    fprintf(stderr,"%s",signon);
    init ();
    ses_status ();
    len = ncb.length;
    if ((ncb.lsn != 0) || (ncb.lsn != 0xFF)) {
        printf ("Name table reference Num       : %2X\n",buff[0]);
        printf ("Num of session associated      : %2X\n",buff[1]);
        printf ("Num of RCV DGM & RCV BCST DGM  : %2X\n",buff[2]);
        printf ("Num of outstading RCV_ANYs     : %2X\n",buff[3]);
        i = 4;
        printf ("----------------------------------------------\n");
        while ((i < len) && (i < 76)) {
            p = 0;
            printf ("Session number                 : %2X\n",buff[i++]);
            printf ("Session status                 : %2X\n",buff[i++]);
            printf ("Local name                     :");
            while ((p < 16) && (i < len)) {
                printf ("%c",buff[i]);
                i++;
                p++;
            }
            p = 0;
            printf ("\n");
            printf ("Remote name                     :");
            while ((p < 16) && (i < len)) {
                printf ("%c",buff[i]);
                i++;
                p++;
            }
            printf ("\n");
            printf ("Num of receive outstading      :%2X\n",buff[i++]);
            printf ("Num of send & csend outstanding:%2X\n",buff[i++]);
        }
    }
}

/****************** sendncb (ncb) ************************************
*
*       Issues the NCB. If an error occurs, it exits after displaying
*       the error.
*
*/

sendncb(ncb)
struct NCB *ncb;
{
    char far *fptr;

    fptr = (char far *) ncb;

    reg.x.bx = FP_OFF (fptr);
    seg.es = FP_SEG (fptr);
    reg.h.ah = 0x01;

    int86x (0x5C, &reg, &reg, &seg);
    while (ncb->retcode == Pending)
        if (kbhit()) getch();

    if ((ncb->retcode != Success) && (ncb->retcode !=Pending))
            {
        printf (" \nError: NCB Command is %2.2X and retcode
                is %2.2X\n", ncb->command, ncb->retcode);
        exit(1);
    }
}

/*********** ses_status()**********************************************
*/
ses_status()
{
    char *lname = "*";

    strncpy(ncb.name, lname, strlen(lname));
    ncb.command = status;
    sendncb(&ncb);
}

/**************** init () ********************************************
*
*       Initialises the fields in the NCB structure.
*/
init()
{
    char *name = "                ";

    ncb.buffer = (char far *) buff;
    strncpy(ncb.name,name,16);
    ncb.lana_num = 0;
    ncb.length = 340;
}
```

```
    ncb.retcode = Success;
    local = strupr(strdup(lname));
    for (i = 0; i <= 15; i++) {
        ncb.callname[i] = 0x20;
    }
    strncpy(ncb.callname, local, strlen(lname));
    ncb.lana_num = 0;
    ncb.length = 348;
}
```

```
/*****************************************************************
 *
 *       This program is used to find the NIA status of any node
 *       in the network. You have to specify node name in command
 *       line.
 */
#include <stdio.h>
#include <string.h>
#include <dos.h>
#include "ncbs.h"

#define Success 0
#define Failure -1
#define Pending 0xFF
#define status  0x33

struct  NCB ncb;                /* The NCB */
byte    buff[400];      /* Data Buffer for the send NCB */

union   REGS    reg;
struct  SREGS   seg;

char    signon[] = "\n ADAPTER STATUS\n\n";

main(ac,av)
int ac;
char **av;
{
    int i,len,p;

    if (ac < 2 ) {
        printf("Usage  : COMMAND  <Local or Remote name> \n");
        printf("Example: COMMAND      lab1xt    <enter>\n");
        exit(1);
    }

    fprintf(stderr,"%s",signon);
    init (av[1]);
    ncb_status ();
    len = ncb.length - 1;
    printf ("LENGTH : %d\n ", ncb.length);
    if ((ncb.lsn != 0) || (ncb.lsn != 0xFF)) {
        printf ("\nUnit ID number              : ");
        for (i = 0; i <= 5; i++)
            printf ("%2X ",buff[i]);
        printf ("\nExt jumper status     : %2X\n",buff[6]);
        printf ("Result of POST test     : %2X\n",buff[7]);
        printf ("Software Version No.    :%2X.%2X\n",buff[8],buff[9]);
        printf ("Actual Num of free NCBs :%2X%2X\n",buff[41],buff[40]);
        printf ("Max configured NCBs     : %2X%2X\n",buff[43],buff[42]);
        printf ("Max free NCBs           : %2X%2X\n",buff[45],buff[44]);
        printf ("Pending sessions        : %2X%2X\n",buff[51],buff[50]);
        printf ("Max Pending sessions    : %2X%2X\n",buff[53],buff[52]);
        printf ("Max possible sessions   : %2X%2X\n",buff[55],buff[54]);
        printf ("Max sess data pkt size  : %2X%2X\n",buff[57],buff[56]);
        printf ("Num of names            : %2X%2X\n",buff[59],buff[58]);
        i = 60;
        printf ("\n    NAMES          Num   STATUS  \n");
        printf ("--------------------------------\n");
        while ((i < len) && (i < 347)) {
            p = 0;
            while ((p < 16 ) && ( i < len )) {
                printf ("%c",buff[i]);
                i++;
                p++;
            }
        if (i < len) printf ("   %2X      %2X\n",buff[i],buff[++i]);
        i++;
        }
        printf ("---------------------------------\n");

    }
}

/********************* sendncb (ncb) *********************************
 *
 *      This function issues the NCB. If an error occurs,
 *      it exits after displaying the error.
 */
sendncb(ncb)
struct NCB *ncb;
{
    char far *fptr;

    fptr = (char far *) ncb;
    reg.x.bx = FP_OFF (fptr);
    seg.es = FP_SEG (fptr);
    reg.h.ah = 0x01;

    int86x (0x5C, &reg, &reg, &seg);

    while (ncb->retcode == Pending)
        if (kbhit()) getch();
    if ((ncb->retcode != Success) && (ncb->retcode !=Pending))  {
        printf (" \nError: NCB Command is %2.2X and retcode
                        is %2.2X\n", ncb->command, ncb->retcode);
        exit(1);
    }
}


/******************* ncb_status()*************************************/

ncb_status()
{
    ncb.command = status;
    sendncb(&ncb);
}

/**************** init () *******************************************
 *
 *      Initialises the fields in the NCB structure.
 */
init(lname)
char *lname;
{
    int         i;
    char        *local;

    ncb.buffer = buff;
```

```
/*********************************************************************
 *
 *       This program is to be run at the server end for
 *       administravitive  work  regarding  addition and
 *       deletion of mail user.
 */


#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <malloc.h>
#include <string.h>
#include <sys\types.h>
#include <sys\stat.h>

struct user {

   char user_name[9];
   char pass_word[10];
   char directory[15];
   unsigned int sess_no;
   struct user *next;          /* pointer to next entry */
   struct user *prior;         /* pointer to previous record */
} list_entry;

struct find_t fbuf;
struct user *start;         /* pointer to first entry in the list */
struct user *last;                       /* pointer to last entry */
struct user *find(char *);

void enter(void),search(void),save(void);
void load(void),list(void);
void delete(struct user  **,struct user **);
void dls_store(struct user *i,
                struct user **start,
                struct user **last );
void inputs(char *,char *, int),display(struct user *);

int menu_select(void);
char    maildir[] = "";

main()
{

   start = last = NULL;         /* initialize top & bottom pointers */
   load();
   for(;;) {
       switch(menu_select()) {

           case 1: enter();
                   printf("\n");
                   break;
           case 2: delete(&start,&last);
                   printf("\n");
                   break;
           case 3: list();
                   printf("\n");
                   break;
           case 4: search();
```

```
                printf("\n");
                break;
        case 5: save();
                printf("\n");
                break;
        case 6: load();
                printf("\n");
                break;
        case 7: return (0);
      }
   }
}

/************ menu_select () *****************************************
 *
 *      Display the menu and ask for choice.
 *
 *      returns :  (int) your choice.
 */
menu_select(void)
{

   char s[80];
   int c;

   printf("\t1: ENTER A USER NAME\n");
   printf("\t2: DELETE A USER NAME\n");
   printf("\t3: LIST THE FILE\n");
   printf("\t4: SEARCH THE FILE FOR A USER\n");
   printf("\t5: SAVE THE FILE\n");
   printf("\t6: LOAD THE FILE\n");
   printf("\t7: QUIT\n");

   do {
       printf("\n\tenter your choice:  ");
       gets(s);
       c = atoi(s);
   } while (c < 0 || c > 7);
   return (c);
}


/*********** enter () ************************************************
 *
 *      Enter a new user in the node. And creats users HDR file
 *      to store messages headers.
 */
void enter(void)
{
    struct user *info;
    char result[15];
    int handle;

    for (;;) {
        info = (struct user *) malloc(sizeof(list_entry));
        if(!info) {
            printf("\n\tout of memory ");
            return;
        }
        inputs("\tEnter name: ",info->user_name,8);
        if (!info->user_name[0]) break;
        inputs("\tEnter pass_word: ",info->pass_word,9);
        strcpy(result,info->user_name);
        strcpy(info->directory ,strcat(result,".hdr"));
        if ((handle = creat (info->directory,
                                S_IREAD|S_IWRITE )) != -1) {
            close (handle);
            dls_store(info,&start,&last);
            save ();
        }
        else printf("Name not added.\n");
    }
}
/************** inputs (*prompt, *s, count) *************************
*
*       Reads a string from keyboard of length less than or equal
*       to the count.
*/
void inputs(char *prompt,char *s, int count)
{
   char p[255];

   do {
       printf(prompt);
       gets(p);
       if(strlen(p) > count) printf("\n too long\n");
   } while(strlen(p) > count);
   strcpy(s,p);
}
/*************** dls_store (*i, **start, **last) *******************
*
*       It stores the new element '*i' in the doubly link list at
*       proper position to sort the list.
*/
void dls_store(struct user *i,          /* new element */
                 struct user **start,   /* first element in list */
                    struct user **last) /* last element in list */
{
   struct user *old,*p;

   if(*last == NULL) {                  /* first element in list */
       i->next = NULL;
       i->prior = NULL;
       *last = i;
       *start = i;
       return;
   }

   p = *start;                          /* start at top of list */

   old = NULL;
   while(p) {
       if(strcmp(p->user_name,i->user_name) < 0) {
           old = p;
           p = p->next;
       }
       else {
           if(p->prior) {
               p->prior->next = i;
               i->next = p;
```

```
                i->prior = p->prior;
                p->prior = i;

                return;
            }
            i->next = p;                /* new first element */
            i->prior = NULL;
            p->prior = i;
            *start = i;
            return;
        }
    }
    old->next = i;                      /* put an end */
    i->next = NULL;
    i->prior = old;
    *last = i;
}

/************* delete (**start, **last) ****************************
 *
 *      Remove an user from the list. It also calls remove_hdr
 *      to delete the hdr_file.
 */
void delete(struct user **start, struct user **last)
{
    struct user *info, *find();
    char s[80];

    printf ("\tEnter the user_name:\n");
    gets(s);
    info = find(s);
    if (info) {
        remove_hdr (info->directory); /*To delete message hdr file.*/
        if (*start == info) {
            *start = info->next;
            if (*start) (*start)->prior = NULL;
            else *last = NULL;
        }
        else {
            info->prior->next = info->next;
            if (info != *last)
                info->next->prior = info->prior;
            else
                *last = info->prior;
        }
        free(info);                     /* Return memory to system */
        save();
    }
}


/*********** find (user_name) ****************************************
 *
 *      It finds the node in the list with user_name.
 *      returns : A pointer to that node if exists.
 *                NULL pointer on non existance.
 */
struct user *find( char *user_name)
{
    struct user *info;
    info = start;
    while (info) {
        if (!strcmp(user_name, info->user_name)) return info;
        info = info->next; /* Get next user */
    }
    printf ("\tName not found.\n\n");
    return NULL;
}

/************* list ()**********************************************
*
*       It displays the entire list.
*/
void list(void)
{
    struct user *info;

    info = start;
    while (info) {
        display(info);
        info = info->next;                          /* Get next user */
    }
    printf("\n");
}

/*************** display (info) ***************************************
*
*       This function actually prints fields of the node with
*       pointer info.
*/
void display(struct user *info)
{
    printf ("\tuser      : %s\n", info->user_name);
    printf ("\tpassword  : %s\n", info->pass_word);
    printf ("\n");
}

/*********** search () *********************************************
*
*       Look for a user_name in the list.
*/
void search(void)
{

    char user_name[15];
    struct user *info, *find();

    printf("\tEnter user_name to find\n");
    gets(user_name);
    info = find (user_name);
    if (!info) printf("\tNot found\n");
    else
          printf("\tvalid user\n");
}

/************* save () ********************************************
*
*       It save the link list in a file on the  disk.
*/
```

```
void save(void)
{
    struct user *info;

    FILE *fp;

    fp = fopen ("userlist","wb");
    if (!fp) {
        printf ("\tcan't open the file userlist\n");
        exit(1);
    }
    printf("\tSaving file on disk.\n\n ");
    info = start;
    while (info) {
        fwrite(info, sizeof(struct user), 1, fp);
        info = info->next;                /* Get next user */
    }
    fclose(fp);
}

/************* load () ***********************************************
 *
 *       It loads the (userlist) file in to memory in the
 *       form of a link list.
 */

void load()
{
    struct user *info;
    FILE *fp;

    fp = fopen("userlist","rb");
    if (!fp) {
        printf ("\tCan't open file userlist.\n");
        exit(1);
    }

    /* Free any previously allocated memory */

    while (start) {
        info = start->next;
        free(info);
        start = info;
    }

    /* Reset top and bottom pointer */
    start = last = NULL;
    printf ("\tLoading file .\n\n");
    while (!feof(fp)) {
        info = (struct user *) malloc(sizeof(struct user));
        if (!info) {
                printf ("\tOut of memory.\n");
                return;
        }
        if (1 != fread (info, sizeof(struct user), 1, fp)) break;
        dls_store(info, &start, &last);
    }
    fclose (fp);
}
/************* remove_hdr (hdr_file) *********************************
```

```
 *
 *       It deletes all the message files releted to the hdr_file
 *       before deleting the hdr_file.
 *
 *       returns : -ve File not found.
 *                  0 Successful deletion of the hdr_file
 */
remove_hdr (hdr_file)
char *hdr_file;
{
    int n_bytes,fptr,p,i,j;
    char hdr[810], fname[10];

    if ((fptr = open (hdr_file,0)) < 0) {
        printf ("ERROR : In opening %s file.\n", hdr_file);
        return (-1);
    }
    else {
        n_bytes = read (fptr, &hdr[0],800);
        while (n_bytes > 0) {
            p  = 67;
            while ( p < n_bytes ) {
                i = p;
                j = 0;
                while (hdr[i] != 0x20) {
                     fname[j] = hdr[i];
                     j++; i++;
                }
                fname[j] = '\0';
                delmail (fname);
                p = p + 80;
            }
            n_bytes = read (fptr, &hdr[0],800);
        }
    }
    close (fptr);
    if (remove (hdr_file)) return (-2);
    else return(0);
}

/**********  delmail  ****************************************************
 *
 *      Delete mail files
 *
 *      Returns: -1  File not found
 *                0  Deleted
 *               >0  Number of links remaining
 */
delmail (name)
char *name;      /* File Name without extension   */
{
    int i, nlink, handle;
    char fname[100], oldname[100], newname[100];

    strcpy (fname, maildir);
    strcpy (oldname, maildir);
    strcpy (newname, maildir);

    strcat (fname, name);  strcat (fname, ".T*");
    if (_dos_findfirst (fname, _A_NORMAL, &fbuf))  return (-1);

    strcat (oldname, fbuf.name);  strcat (newname, fbuf.name);
    for (i=strlen (oldname); i>0; i--)  if (oldname[i] == '.')  break;
    if (i < 1) return (-1);

    nlink = 0;
    sscanf (&oldname[i+2],"%d",&nlink);
    if (nlink-- > 1) {                          /* Rename File */
        sprintf (&newname[i+2],"%d",nlink);
        if (rename (oldname, newname) != 0) return (-2);
        return (nlink);
    }
    else {
        strcpy (fname, maildir);  strcat (fname, name);
        strcat (fname, ".*");
        if (_dos_findfirst (fname, _A_NORMAL, &fbuf))  return (-3);
        while (1) {
            strcpy (oldname, maildir);   strcat (oldname, fbuf.name);
            if (remove (oldname)) return (-4);
            if (_dos_findnext (&fbuf))  return (0);
```

```
/*********************************************************************
 *
 *      This program is to change the password of any user, from
 *      any node of this mail system.
 */
#include <stdio.h>
#include <string.h>
#include <dos.h>
#include "ncbs.h"

#define Success 0
#define Failure -1
#define Pending 0xFF
#define Timeout 0x05
#define call    0x10
#define send    0x14
#define listen  0x11
#define recv    0x15
#define hangup  0x12
#define status  0x33

struct  NCB ncb,ncb1;   /* The NCB */
byte    sbuffer[Bufsize];       /* Data Buffer for the send NCB */
byte    rbuffer[Bufsize];       /* Data Buffer for the recv NCB */
byte    server[16] = "LAB1XT          ";
byte    buff[400];
byte    local[16] = "*               ";

union   REGS    reg;
struct  SREGS   seg;

char    signon[] = "\n CHANGING PASWORD.\n\n";

main()
{
    int i,len;

    fprintf(stderr,"%s",signon);
    ncb_call ();
    if ((ncb.lsn != 0) || (ncb.lsn != 0xFF))
        change_pwd ();
}

/*************** change_pwd () ****************************************
 *
 *      This function sends user_name & password at
 *      server & receives respose.
 *
 */
change_pwd()
{
    int i,p;
    byte lsn;
    char  user[15], opass[10], npass[10];

    lsn = ncb.lsn;
    for (i = 0; i < 45; i++) {
        buff[i] = 0x20;
    }
    buff[0] = 'O';
    do {
        printf ("\tUSER: ");
        gets(user);
        if(strlen(user) > 8) printf("\n too long\n");
    } while(strlen(user) > 8);
    printf ("\tOLD PASSWORD: ");
    getpwd(opass,9);
    printf ("\n\tNEW PASSWORD: ");
    getpwd(npass,9);
    strncpy(&buff[10], user, 9);
    strncpy(&buff[25], opass, 10);
    strncpy(&buff[35], npass, 10);
    printf ("\n\tNEW PASSWORD AGAIN: ");
    for (i = 0; i < 10; i++)
        npass[i] = 0x20;
    getpwd(npass,9);
    if (!strncmp(npass, &buff[35], 10)) {

        for (i = 0; i < 45; ++i)
            printf ("%c",buff[i]);
        ncb.length = 45;
        ncb_send ();
        ncb_recv (lsn);
        if (rbuffer[0] == 'o') {
            switch (rbuffer[10]) {

                case 0:{ printf ("\nPassword changed.\n");
                        break;
                }
                case 1:{ printf ("\nUser name not found.\n");
                        break;
                }
                case 2:{ printf ("\nUnauthorised user.\n");
                        break;
                }
            }
        send_quit ();
        }
    }
    else printf ("\n\tCan't change the password.");
}

/************* quit () ***********************************************
*
*       Hangup the session.
*/
quit ()
{

    ncb.command = hangup;
    sendlncb (&ncb);

}
/************ send_quit () ******************************************
*
*       Sends message to server to hang up the session.
*/
send_quit()
{
    buff[0] = 'M';
    ncb_send ();
    quit ();
}

/****************** load_ncb (ncb) ************************************
*
*       This function issues the NCB. If an error occurs,
*       it exits after displaying the error.
*/
load_ncb(ncb)
struct NCB *ncb;
{
    char far *fptr;

    while (ncb->retcode == Pending) if (kbhit()) getch();
    if (ncb->retcode != Success) {
        printf (" \nError: NCB Command is %2.2X and retcode is %2.2X\n"
                    , ncb->command, ncb->retcode);
        if ( ncb->retcode == Timeout) {
            if (ncb->command == recv) send_quit ();
            else quit ();
        }
        exit(1);
    }
    fptr = (char far *) ncb;

    reg.x.bx = FP_OFF (fptr);
    seg.es = FP_SEG (fptr);
    reg.h.ah = 0x01;

    int86x (0x5C, &reg, &reg, &seg);
    if ((ncb->retcode != Success) && (ncb->retcode !=Pending)) {
        printf (" \nError: NCB Command is %2.2X and retcode is %2.2X\n"
                    , ncb->command, ncb->retcode);
        if ( ncb->retcode == Timeout) {
            if (ncb->command == recv) send_quit ();
            else quit ();
        }
        exit(1);
    }
}

/************  sendlncb () ********************************************
*
*       Issues the NCB. It does not do any error display
*       or checking.
*/

sendlncb(ncb)
struct NCB *ncb;
{
    char far *fptr;

    fptr = (char far *) ncb;
    reg.x.bx = FP_OFF (fptr);
    seg.es = FP_SEG (fptr);
    reg.h.ah = 0x01;
    int86x (0x5C, &reg, &reg, &seg);
```

```
/************** ncb_call () ******************************************
 *
 *      This procedure performes a call to the remote node
 *      specified after having added the local name.
 */

ncb_call()
{
    int i,p,len;

    initl ();
    ncb_status ();
    len = ncb.length;
    i = 60;
    p = 0;
    while ((p < 16 ) && ( i < len )) {
        ncb.name[p] = buff[i];
        i++;
        p++;
    }
    strncpy(ncb.callname, server, 16);
    ncb.command = call;
    load_ncb(&ncb);
}

/***************  ncbsend () *****************************************/

ncb_send()
{
    ncb.command = send;
    load_ncb(&ncb);
}
/************* ncb_recv(lsno) ****************************************
 *
 *      This function initialises the ncb to recv where
 *      lsno is the corresponding session number.
 */
ncb_recv(lsno)
byte lsno;
{
    ncb1.lsn = lsno;
    ncb1.buffer = (char far *) rbuffer;
    ncb1.length = Bufsize;
    ncb1.lana_num = 0;
    ncb1.command = recv;
    load_ncb(&ncb1);
}

/********** ncb_status ()*********************************************
 *
 *      It issues the adapter status command to get local name.
 */
ncb_status()
{

    strncpy(ncb.callname, local, 16);
    ncb.command = status;
    load_ncb(&ncb);
}
```

```
/************ initl () *************************************************
*
*       Predefines the fields in the NCB structure.
*/
initl()
{

    ncb.buffer = (char far *) buff;
    ncb.retcode = Success;
    ncb.lana_num = 0;
    ncb.length = 348;
    ncb.sto = 60;
    ncb.rto = 60;
}

/*********** getpwd  (linebuf, maxch) *********************************
*
*       This function reads atmost maxch character(s) from
*       the keyboard for password and store in linebuf.
*
*       returns : Number of character read.
*/
getpwd(linebuf, maxch)
char linebuf[];
int maxch;
{
    int ch, nch = 0;

    while ((ch = getch()) != 0x0d) {
        if (ch >= 0x20) {
            if (nch < maxch) {
                linebuf[nch] = ch; nch++;
            }
        }
        else if (ch == 8) {
            if (nch != 0) nch--;
        }
        else if (ch == 0) getch();
    }
    linebuf[nch] = 0;
    return(nch);
}
```

```
/**********************************************************************
*
*       Header file for Network Control Block ( NCB ).
*/
#define byte unsigned char
#define word unsigned int

#define Bufsize 1000

struct  NCB     {
        byte    command;
        byte    retcode;
        byte    lsn;
        byte    num;
        char far *buffer;
        word    length;
        byte    callname[16];
        byte    name[16];
        byte    rto;
        byte    sto;
        word    off_post;
        word    seg_post;
        byte    lana_num;
        byte    cmd_cplt;
        byte    reserve[14];
} ;
```

```
/***********************************************************************
*
*       This program is the VMS server program which is to be run
*       at the server end.
*/
#include <stdio.h>
#include <string.h>
#include <dos.h>
#include <stdlib.h>
#include <malloc.h>
#include "ncbs.h"
#include <sys\types.h>
#include <sys\stat.h>
#include <time.h>
#include <fcntl.h>
#include <io.h>

#define Success 0
#define Pending 0xFF
#define Closed  0x0A
#define Timeout 0x05
#define listen  0x11
#define recv    0x15
#define send    0x14
#define no_wsend 0x94
#define no_wlisten 0x91
#define no_wrecv 0x95
#define hangup  0x12
#define no_whangup 0x92
#define status  0x33
#define cancel  0x35
#define MAX_SES 4

#define VMS_BLEN 8000

struct  NCB ncb,ncbc;                  /* The NCBs */
struct  find_t fbuf;
byte    rbuff[MAX_SES][Bufsize];/* Data Buffer for the receive NCB */
byte    sbuff[MAX_SES][Bufsize];/* Data Buffer for the send    NCB */
union   REGS    reg;
struct  SREGS   seg;
char    server[16] = "LAB1XT          ";
char    maildir[] = "";

struct user {
   char user_name[9];
   char pass_word[10];
   char directory[15];
   unsigned int sess_no;
   struct user *next;           /* pointer to next entry */
   struct user *prior;          /* pointer to previous record */
};

struct session {
    int flag;
    int fflag;
    int fptr;
    struct NCB rncb;
    struct NCB sncb;
    char user[9];
    char dir[15];
    char text_file[8];
}ses[MAX_SES];

struct user *start;     /* pointer to first entry in the list */
struct user *last;      /* pointer to last entry */
struct user *find(char *);

void load_list(void),list(void),save(void);
void dls_store(struct user *i,
                struct user **start,
                struct user **last );
void display(struct user *);

char    signon[] = "VMS_Server: ";
int     no_ses = 0;             /* Number session existing. */
int     sign = 1;
int     line_flag = 0;

unsigned char vms_buf[VMS_BLEN*2+100], *vms_buf1, *vms_buf2;
int  vms_count, max_count;
unsigned char vms_irq, vms_flag, *vms_bobuf, *vms_eobuf, *vms_bufptr;

extern void vms_start();
extern void vms_stop();
extern void voiceon();
extern void voiceoff();

main()
{
    int         i,len,num,signal;
    char        ch,com[10];
    byte        lsn;

    fprintf(stderr,"%s",signon);
    signal = 1;
    start = last = NULL;
    load_list ();
    init ();
    num = 0;
    while (1) {
        start:if ((ncb.retcode == Success) && (sign)) {
            ncb_listen ();
            sign = 0;
        }
        if ((ncb.cmd_cplt == Success ) && (!sign)) {
            if (ncb.retcode == Success) {
                lsn = ncb.lsn;
                if ((num = find_ses(lsn,num)) < MAX_SES) {
                    no_ses++;
                    sign = 1;
                    ncb_recv(num);
                    if (no_ses < MAX_SES) goto start;
                    else sign = 0;
                }
            }
        }
        for (i = 0; i < MAX_SES; i++) {
            if (ses[i].flag)
                if (ses[i].rncb.cmd cplt == Success)
        if(ses[i].rncb.retcode == Success) {
            switch (rbuff[i][0]) {

                         case 'A': chkpwd (i);
                                   ncb_recv (i);
                                   break;

                         case 'B': read_file (i);
                                   ncb_recv (i);
                                   break;

                         case 'C': read_file (i);
                                   ncb_recv (i);
                                   break;

                         case 'F': recepient (i);
                                   ncb_recv (i);
                                   break;

                         case 'G': process_voice_req(i);
                                   ncb_recv (i);
                                   break;

                         case 'H': ncb_recv(i);
                                   process_start_key(i);
                                   while(rbuff[num][0] != 'P')
                                     dovoice(ses[num].fptr,2);
                                   process_stop_key(i);
                                   ncb_recv (i);
                                   break;

                         case 'I': process_pback_req(i);
                                   ncb_recv (i);
                                   break;

                         case 'J': process_del_vfile(i);
                                   ncb_recv (i);
                                   break;

                         case 'K': save_text (i);
                                   ncb_recv (i);
                                   break;

                         case 'M': quit(i);
                                   break;

                         case 'O': change_pwd(i);
                                   ncb_recv (i);
                                   break;

                         case 'Q': file_name (i);
                                   ncb_recv (i);
                                   break;

                         case 'R': ncb_recv (i);
                                   break;

                         case 'S': del_msg (i);
                                   ncb_recv (i);
                                   break;
```

```
                            case 'U': process_pback_key(i);
                                      ncb_recv (i);
                                      break;

                            case 'V': process_hang_key(i);
                                      ncb_recv (i);
                                      break;

                            case 'W': process_pback_quit(i);
                                      ncb_recv (i);
                                      break;

                      }
                  }
      }
      for ( i = 0; i < MAX_SES; i++) {
          if (ses[i].flag) {
              if (ses[i].rncb.retcode == Timeout) {
                  quit (i);
                  ses[i].rncb.retcode = Success;
              }
              if ((ses[i].sncb.retcode == Timeout) &&
                      (ses[i].sncb.command == no_wsend)) {
                      quit (i);
                      ses[i].sncb.retcode = Success;
              }
              if (ses[i].sncb.command == no_whangup) {
                  if (ses[i].sncb.cmd_cplt == Success) {
                      if (ses[i].sncb.retcode == Success) {
                          ses[i].sncb.command = no_wsend;
                          ses[i].sncb.retcode = Success;
                          ses[i].sncb.cmd_cplt = Success;
                          ses[i].flag = 0;
                          if ((no_ses == MAX_SES) && (!sign))
                              sign = 1;
                          no_ses--;
                      }
                  }
                  else {
                      if ((ses[i].sncb.cmd_cplt == Timeout) ||
                              (ses[i].sncb.cmd_cplt == Closed )) {
                          ses[i].sncb.command = no_wsend;
                          ses[i].sncb.retcode = Success;
                          ses[i].sncb.cmd_cplt = Success;
                          ses[i].flag = 0;
                          if ((no_ses == MAX_SES) && (!sign))
                              sign = 1;
                          no_ses--;
                      }
                  }
              }
          }
      }
      if (kbhit()) {
          gets(com);
          signal = 0;
          if (strcmp(com, NULL) != 0) {
              sscanf(com, "%c", &ch);
              tolower (ch);
```

```
            switch (ch) {

                    case '?': help ();
                              break;

                    case 'u': list ();
                              break;

                    case 'w': c_user ();
                              break;

                    case 's': shut_down ();
                              break;

                    default:  printf ("Unrecognized command.");
                              printf (" Type ? for HELP.\n");
            }
        }
    }
    if (!signal) {
        fprintf (stderr, "%s", signon);
        signal = 1;
    }
  }
}

/********** process voice request ***************************************
 *
 *      processes the voice compose request from any user.
 *      checks if line available if so Creats and Opens a voice file
 *      Also initialises the digitiser card.
 *      Returns:  0  server available
 *                1  server not available
 */

process_voice_req(num)
int num;
{
    int voice_no,i;
    char vf_name[12];

    rbuff[num][0] = 0x20;
    sbuff[num][0] = 'g';
    voice_no = rbuff[num][10];

    if(line_flag)  sbuff[num][10] = 1;
    else {
        line_flag = 1;                          /* hold the line */
        vms_card_on(2);                   /* initialize at card */
        if(voicefile(ses[num].text_file, voice_no) == 0) {
            strcpy (vf_name, maildir);
            strcat (vf_name, ses[num].text_file);
            strcat (vf_name, ".V");
            i = strlen (vf_name);
            sprintf (&vf_name[i],"%d",voice_no);
            if (!ses[num].fflag) {
                if ((ses[num].fptr = open(vf_name,O_RDWR|O_BINARY)) < 0) {
                    sbuff[num][10] = 1;
                }
                else {
                    ses[num].fflag = 1;
                    sbuff[num][10] = Success;
                }
            }
        }
        else sbuff[num][10] = 1;
    }
    ncb_send(num,15);
}

/********** process start key for digitisation   *******************
 *
 *      initiates digitisation.
 *       Returns: 0 success always
 */
process_start_key(num)          /***** TO DIGITIZE VOICE    *******/
int num;
{
    rbuff[num][0] = 0x20;
    sbuff[num][0] = 'h';
    sbuff[num][10] = Success;
    ncb_send(num,15);
    voice_on(15);                       /*start_digitization();*/
}

/********** process stop key  ***************************************
 *
 *      processes user initiated stop key for stopping digitisation.
 *      vms card is turned off. Voice file is closed and line
 *      released
 *      Returns:  0 Success
 *                1 Failure
 */

process_stop_key(num)
int num;
{

    rbuff[num][0] = 0x20;
    sbuff[num][0] = 'p';
    voice_off();
    vms_card_off();
    if (ses[num].fflag) {
        close(ses[num].fptr);
        ses[num].fflag = 0;
        sbuff[num][10] = Success;
    }
    else sbuff[num][10] = 1;
    line_flag = 0;                      /** RELEASE LINE********/
    ncb_send (num, 15);
}
/********** process hang up  key ************************************
 *
 *      processes quit key to hang up when server is found busy.
 *      Closes and deletes the voice-file,releases the line.
 *      Returns:  0 Success
 *                1 Failure
 */

process_hang_key(num)                   /* ESC PRESSED TO ABORT */
```

```
int num;
{
    int voice_no;
    int i = 0;
    char vf_name[12];

    voice_no = rbuff[num][10];
    rbuff[num][0] = 0x20;
    sbuff[num][0] = 'v';
    if (ses[num].fflag) {
        close(ses[num].fptr);
        ses[num].fflag = 0;
    }
    else sbuff[num][10] = 1;

    strcpy (vf_name, maildir);
    strcat (vf_name, ses[num].text_file);
    strcat (vf_name, ".V");
    i = strlen (vf_name);
    sprintf (&vf_name[i],"%d",voice_no);i = 0;
    i = remove(vf_name);
    if(i == -1) printf(" ERROR MESSAGE :COULD NOT DELETE FILE\n");
    sbuff[num][10] = Success;

    line_flag = 0;
    ncb_send (num, 15);
}

/********** process play back request*********************************
 *
 *      initialses vms card for playback if line available.
 *      Returns:  0 Success
 *                1 Failure
 */

process_pback_req(num)
int num;
{

    rbuff[num][0] = 0x20;
    sbuff[num][0] = 'i';

    if(line_flag)  sbuff[num][10] = 1;
    else {
        line_flag = 1;                          /* assigning line */
        vms_card_on(1);
        sbuff[num][10] = Success;
    }
    ncb_send(num,15);
}

/********** process play back start key *****************************
 *
 *      opens the voice file for read. closes it after read  is
 *      complete and resets vms card. Then closes the file and
 *      releases the line.
 *      Returns:  0 Success
 *                1 Failure
 */
process_pback_key(num)
```

```
 int num;
 {
     int voice_no;
     int i = 0;
     char vf_name[12];

     voice_no = rbuff[num][10];
     rbuff[num][0] = 0x20;
     sbuff[num][0] = 'u';

     strcpy (vf_name, maildir);
     strcat (vf_name, ses[num].text_file);
     strcat (vf_name, ".V");
     i = strlen (vf_name);
     sprintf (&vf_name[i],"%d",voice_no);

     if (!ses[num].fflag) {
         if ((ses[num].fptr = open(vf_name,O_RDONLY|O_BINARY)) < 0) {
             sbuff[num][10] = 1;
         }
         else {
            ses[num].fflag = 1;
            sbuff[num][10] = Success;
            voice_on(15);
            while(!eof(ses[num].fptr))            /* AND RBUFF[0] != XXX */
            dovoice(ses[num].fptr,1);
            voice_off();
            vms_card_off();
            if (ses[num].fflag) {                 /* close file , session etc. */
                 close(ses[num].fptr);
                 ses[num].fflag = 0;
                 line_flag = 0;
            }
            else sbuff[num][10] = 1;
         }
    }
    else sbuff[num][10] = 1;
    line_flag = 0;                         /* release line */
    ncb_send(num,15);
}

/********** play back quit  ******************************************
 *
 *      If line is busy then close voice file,release line flag.
 *      Returns:  0 Success
 *                1 Failure
 */

process_pback_quit(num)
int num;
{
    rbuff[num][0] = 0x20;
    sbuff[num][0] = 'w';
    if (ses[num].fflag) {
        close(ses[num].fptr);          /*  CLOSE VOICE FILE */
        ses[num].fflag = 0;
        sbuff[num][10] = Success;
    }
    else sbuff[num][10] = 1;
    line_flag = 0;                     /** RELEASE LINE********/
    ncb_send (num, 15);
 }

 /********** delete voice file *****************************************
  *
  *      deletes the specified voice file. file no available in
  *      rbuff[num][10].
  *      Returns:  0 Success
  *                1 Failure
  */

 process_del_vfile(num)
 int num;
 {
     int voice_no;
     int i = 0;
     char vf_name[12];

     voice_no = rbuff[num][10];
     rbuff[num][0] = 0x20;
     sbuff[num][0] = 'j';
     strcpy (vf_name, maildir);
     strcat (vf_name, ses[num].text_file);
     strcat (vf_name, ".V");
     i = strlen (vf_name);
     sprintf (&vf_name[i],"%d",voice_no);i = 0;
     i = remove(vf_name);
     if(i == -1){
         sbuff[num][10] = 1;
     }
     else {
         sbuff[num][10] = Success;
     }
     ncb_send(num,15);
 }

 /********* vms_card_on  **********************************************
  *
  *      Initialise the voice card and set redirect interrupt
  */
 vms_card_on (mode)
 int mode;
 {
     vms_buf1   = vms_buf;   vms_buf2  = vms_buf1+VMS_BLEN;
     vms_bobuf  = vms_buf1;  vms_eobuf = vms_buf1 + VMS_BLEN*2;
     vms_bufptr = vms_bobuf; vms_irq   = 0x0c;

     if ((mode < 1) || (mode > 2)) return (-1);
     vms_flag = mode;  vms_start ();

     return (0);
 }

 /**********  voice_on  ***********************************************
  *
  *      Start Digitisation/Paly Back of Voice
  */
 voice_on (maxtime)
 int  maxtime;           /* max time in Seconds */
 {
    vms_count  = 0;
    max_count = (maxtime * 8000l) / VMS_BLEN;
    voiceon ();
    return (0);
}

/**********  voice_off  *********************************************
 *
 *      Stops Digitisation/Play back of Voice
 */
voice_off ()
{
    int time;

    voiceoff ();
    time = ((1l * vms_count) * VMS_BLEN) / 8000l;
    return (time+1);
}

/**********  vms_card_off  ******************************************
 *
 *      Disable voice card and switch relay off.
 */

vms_card_off ()
{
    vms_stop ();
    return (0);
}

/**********  dovoice  ***********************************************
 *
 *      Process voice bufferes
 */

dovoice (fd, mode)
int fd, mode;
{
    if (((vms_count%2) == 0) && (vms_bufptr >= vms_buf2)) {
        if (mode == 1) {
            if (++vms_count >= max_count)   voice_off();
            if (read (fd, vms_buf1, VMS_BLEN) >= VMS_BLEN)
                return (0);
        }
        else if (mode == 2) {
            if (++vms_count >= max_count)   voice_off();
            if (write (fd, vms_buf1, VMS_BLEN) >= VMS_BLEN)
                return (0);
        }
        return (-1);
    }
    if (((vms_count%2) == 1) && (vms_bufptr < vms_buf2)) {
        if (mode == 1) {
            if (++vms_count >= max_count)   voice_off();
            if (read (fd, vms_buf2, VMS_BLEN) >= VMS_BLEN)
                return (0);
        }
        else if (mode == 2) {
            if (++vms_count >= max_count)   voice_off();
            if (write (fd, vms_buf2, VMS_BLEN) >= VMS_BLEN)
```

```
                return (0);
            }
            return (-1);
        }
        return (0);
}

/*********** help () ************************************************
*
*       This function prints the server commands.
*/
help ()
{
    printf("\n\n\t________________HELP_____________\n");
    printf ("\n\tW : Who are currently interactive.\n");
    printf ("\tU : Users list.\n");
    printf ("\tS : Shutdown the server.\n");
    printf ("\t? : Help.\n");
}

/********** c_user () ***********************************************
*
*       This function prints the currently interactive user(s) list.
*/
c_user ()
{
    int i, num;

    num = 0;
    for (i = 0; i < MAX_SES; i++) {
        if (ses[i].flag) {
            num++;
            printf ("\n\t%d. %s\n",num,ses[i].user);
        }
    }
    if ( num == 0) printf ("\n\tNo entries in the list.\n");
}
/********** shut_down () ********************************************
*
*       This function confirm shut down of server.
*
*/
shut_down ()
{
    char ch;

    printf ("\n\tSHUT DOWN SERVER ?  PRESS___ Y/N.  ");
    ch = getche ();printf("\n");
    if ((ch == 'y') || (ch == 'Y')) exit(1);
}

/************* quit(num) ********************************************
*
*       This funcyion hangup the session having ses[num].
*
*/
quit(num)
int num;
{
    if ((ses[num].rncb.cmd_cplt == Pending) ||
          (ses[num].rncb.retcode == Pending)) {
           ncb_cancel(&ses[num].rncb);
      }
      if ((ses[num].sncb.cmd_cplt == Pending) ||
          (ses[num].sncb.retcode == Pending)) {
           ncb_cancel(&ses[num].sncb);
      }
      rbuff[num][0] = 0x20;
      ses[num].sncb.command = no_whangup;
      load_ncb(&ses[num].sncb);
 }

 /*********** find_ses(lsn, num) ***************************************
  *       formal parameter :
  *                        lsn  is the local session number.
  *                        num  is the ses index.
  *
  *       This function search session array.
  *       returns :The index if session available.
  *                :MAX_SES otherwise.
  */

 find_ses(lsn,num)
 byte lsn;
 int num;
 {
     int i,j;
     for (j = num; j < j + MAX_SES; j++) {
         i = (j % MAX_SES);
         if ((!ses[i].flag) && (ses[i].sncb.cmd_cplt == Success)) {
             ses[i].flag = 1;
             ses[i].rncb.lsn = lsn;
             ses[i].sncb.lsn = lsn;
             return (i);
         }
     }
     return(MAX_SES);
 }

 /************* ncb_cancel(ncb) ****************************************
 *
 *       formal parameter :ncb is the pointer to pending command NCB.
 *       This function the cancel pending the command.
 */

 ncb_cancel(ncb)
 struct NCB *ncb;
 {
     ncbc.buffer = (char far *) ncb;
     load_ncb(&ncbc);
 }

 /************** load_ncb (ncb) ****************************************
 *
 *       formal parameter : ncb  pointer to NCB.
 *       This function loads the NETBIOS command for execution.
 */

 load_ncb(ncb)
 struct NCB *ncb;
```

```
{
    char far *fptr;

    fptr = (char far *) ncb;

    reg.x.bx = FP_OFF (fptr);
    seg.es = FP_SEG (fptr);
    reg.h.ah = 0x01;

    int86x (0x5C, &reg, &reg, &seg);
}

/************** ncb_listen () ****************************************
*
*       This function performes a listen command for any node.
*/
ncb_listen()
{
    char *rname = "*";
    int         i;

    ncb.retcode = Success;
    ncb.cmd_cplt = Success;
    ncb.buffer = (char far *) rbuff;
    for (i = 0; i <= 15; i++) {
        ncb.callname[i] = 0x20;
        ncb.name[i] = 0x20;
    }
    ncb.rto = 240;
    ncb.sto = 120;
    ncb.off_post = 0;
    ncb.seg_post = 0;
    ncb.lana_num = 0;
    strncpy (ncb.name, server, 16);
    strncpy(ncb.callname, rname, strlen(rname));
    ncb.command = no_wlisten;
    load_ncb(&ncb);
}
/************ ncb_receive (num) **************************************
*
*       Issue a receive command for ses[num]
*/

ncb_recv(num)
int num;
{
    ses[num].rncb.length = Bufsize;
    rbuff[num][0] = 0x20;
    ses[num].rncb.command = no_wrecv;
    load_ncb(&ses[num].rncb);
}

/************* dls_store ********************************************
*
*       This function store the new user name in the list of
*       users in sorted form.
*
*/

void dls_store(struct user *i,          /* new element */
                  struct user **start,   /* first element in list */
                     struct user **last) /* last element in list */
  {
     struct user *old,*p;

     if(*last == NULL) {                  /* first element in list */
         i->next = NULL;
         i->prior = NULL;
         *last = i;
         *start = i;
         return;
     }

     p = *start;                          /* start at top of list */

     old = NULL;
     while(p) {
         if(strcmp(p->user_name,i->user_name) < 0) {
             old = p;
             p = p->next;
         }
         else {
             if(p->prior) {
                 p->prior->next = i;
                 i->next = p;
                 i->prior = p->prior;
                 p->prior = i;

                 return;
             }
             i->next = p;                 /* new first element */
             i->prior = NULL;
             p->prior = i;
             *start = i;
             return;
         }
     }
     old->next = i;                       /* put an end */
     i->next = NULL;
     i->prior = old;
     *last = i;
}

/************ find ****************************************************
*
*       formal parameter: pointer to user_name.
*       This function search user list for user_name.
*       returns: Pointer to the node if name exist.
*               NULL otherwise.
*/

struct user *find( char *user_name)
{
    struct user *info;

    info = start;
    while (info) {
        if (!strcmp(user_name, info->user_name)) return info;
        info = info->next; /* Get next user */
    }
    return NULL;
}

/************** list ()**********************************************
*
*       This function prints the name all users.
*
*/
void list(void)
{
    struct user *info;

    info = start;
    printf ("\n\tLIST OF AUTHORISED USERS\n");
    printf ("\t________________________\n");
    while (info) {
        printf ("\t%s\n", info->user_name);
        info = info->next;                      /* Get next user */
    }
    printf("\n");
}

/************ load_list () ******************************************
*
*       This function load the user_list to a file named as
*       userlist.
*/
void load_list()
{
    struct user *info;
    FILE *fp;

    fp = fopen("userlist","rb");
    if (!fp) {
        printf ("\tCan't open file userlist.\n");
        exit(1);
    }
    /* Free any previously allocated memory */
    while (start) {
        info = start->next;
        free(info);
        start = info;
    }
    /* Reset top and bottom pointer */
    start = last = NULL;
    while (!feof(fp)) {
        info = (struct user *) malloc(sizeof(struct user));
        if (!info) {
                printf ("\tOut of memory.\n");
                return;
        }
        if (1 != fread (info, sizeof(struct user), 1, fp)) break;
        dls_store(info, &start, &last);
    }
    fclose (fp);
}
/*************** chkpwd (num)*****************************************
*
*       This function checks the user  and password
*       and sends results to the workstation.
```

```
*/
chkpwd (num)
int num;
{
    struct user *info;
    int lsn,i;

    i = 0;
    rbuff[num][0] = 0x20;
    info = find (&rbuff[num][10]);
    sbuff[num][0] = 'a';
    if (!info)
        sbuff[num][10] = 1;
```

```
  ses[num].sncb.command = no_wsend;
  load_ncb(&ses[num].sncb);

}
/******************* Save ()*******************************************
*       This function save the link list in to the userlist file.
*/
void save(void)
{
    struct user *info;
    FILE *fp;

    fp = fopen ("userlist","wb");
    if (!fp) {
        printf ("\tcan't open the file userlist\n");
        exit(1);
    }
    info = start;
    while (info) {
        fwrite(info, sizeof(struct user), 1, fp);
        info = info->next;                       /* Get next user */
    }
    fclose(fp);
}
/******************** init ********************************************
*   This function initialises the structures.
*/
init()
{
    int i,p;

    for (i = 0; i < MAX_SES; i++) {
        ses[i].rncb.retcode  = Success;
        ses[i].rncb.cmd_cplt = Success;
        ses[i].rncb.length   = Bufsize;
        ses[i].rncb.off_post = 0;
        ses[i].rncb.seg_post = 0;
        ses[i].rncb.lana_num = 0;
        ses[i].rncb.buffer   = (char far *) &rbuff[i][0];
        ses[i].rncb.command  = no_wrecv;
        ses[i].sncb.retcode  = Success;
        ses[i].sncb.cmd_cplt = Success;
        ses[i].sncb.off_post = 0;
        ses[i].sncb.seg_post = 0;
        ses[i].sncb.lana_num = 0;
        ses[i].sncb.buffer   = (char far *) &sbuff[i][0];
        ses[i].sncb.command  = no_wsend;
        ses[i].flag = 0;
        ses[i].fflag = 0;
    }
    ncbc.lana_num = 0;
    ncbc.command = cancel;
}

/************** recepient ( num) **********************************
*
*       This function checks valid users whom mail is to be send.
*/
recepient (num)
int num;
```

```
     else
          if (!strcmp(info->pass_word, &rbuff[num][25])) {
               sbuff[num][10] = 0;
               strcpy(ses[num].user, info->user_name);
               strcpy(ses[num].dir, info->directory);
          }
          else sbuff[num][10] = 2;
     ncb_send (num, 15);
}
/************* change_pwd (num)***************************************
*
*       This function change the users password if old password
*       is verified and send information to the workstation.
*/
change_pwd(num)
int num;
{
     struct user *info;
     int lsn,i;
     rbuff[num][0] = 0x20;
          info = find (&rbuff[num][10]);
          sbuff[num][0] = 'o';
          if (!info) {
               sbuff[num][10] = 1;
          }
          else {
               if (!strcmp(info->pass_word, &rbuff[num][25])) {
                    sbuff[num][10] = 0;
                    strncpy(info->pass_word, &rbuff[num][35], 10);
                    save ();
                    load_list ();
               }
               else sbuff[num][10] = 2;
          }
          ncb_send (num, 15);
}
/************ ncb_send (num, len) ***********************************
*
*       Formal parameter: 'len' is the length of sbuff to be send.
*       This function initialises ncb to send information to the
*       work station.
*/
ncb_send(num,len)
int num;
word len;
{
  ses[num].sncb.length = len;
```

```
{
    struct user *info;
    int  i,len,no;
    char name[9];

    rbuff[num][0] = 0x20;
    len = ses[num].rncb.length;
    sbuff[num][0] = 'f';
    no = len/10 - 1;
    for (i = 0; i < no; i++) {
        strncpy(name,&rbuff[num][(i+1)*10],9);
        info = find (name);
        if (!info)
            sbuff[num][i+10] = 1;
        else sbuff[num][i+10] = 0;
    }
    ncb_send (num, 20);
}
/********************  read_file (num) ****************************
*
* This function reads the headers or massage file and send it to user
*/
read_file (num)
int num;
{
    int n_bytes, len;
    char fnewname[12];

    if (rbuff[num][0] == 'B') {
        sbuff[num][0] = 'b';
        if (!ses[num].fflag) {
            if ((ses[num].fptr = open (ses[num].dir,0)) < 0) {
                sbuff[num][2] = 2;
                ncb_send (num, 10);
            }
            else ses[num].fflag = 1;
        }
    }
    if (rbuff[num][0] == 'C') {
        sbuff[num][0] = 'c';
        textfile (&rbuff[num][10], fnewname);
        if (!ses[num].fflag) {
            if ((ses[num].fptr = open (fnewname, 0)) < 0) {
                sbuff[num][2] = 2;
                ncb_send (num, 10);
            }
            else ses[num].fflag = 1;
        }
    }
    rbuff[num][0] = 0x20;
    if (ses[num].fflag) {
        n_bytes = read (ses[num].fptr, &sbuff[num][10], 980);
        if (n_bytes > 0) {
            sbuff[num][2] = 1;
            len = n_bytes + 10;
            ncb_send (num, len);
        }
        else {
            sbuff[num][2] = 0;
            close (ses[num].fptr);
            ses[num].fflag = 0;
            ncb_send (num, 10);
        }
    }
}
/*********  filesize  ***********************************************
 *
 *      Calculates and returns the size of the named file.
 *
 *      Returns:  0 (long)   if File not found
 *                size       Otherwise
 */

long
filesize (fname)
char *fname;             /* Name of file to get size of */
{
    if (_dos_findfirst (fname, _A_NORMAL, &fbuf) )  return (0l);
    while (1) {
        if ((31 & fbuf.attrib) == _A_NORMAL)  return (fbuf.size);
        if (_dos_findnext (&fbuf))  return (0l);
    }
}

/**********  newtextfile  ********************************************
 *
 *      Create a new text file with given number of users.
 *
 *      Returns: -1  if Unable to Create
 *                0  Otherwise
 */
newfile (name, nlink)
char *name;              /* String where file name will be returned */
int  nlink;              /* Number of links                         */
{
    char fname[100];
    int i, handle;

    if ((nlink < 0) || (nlink > 99)) return (-1);

    strcpy (fname, maildir); strcat (fname,"AA.*");
    i= strlen (maildir);

    for (fname[i]='A'; fname[i]<='Z'; fname[i]=fname[i]+1)
        for (fname[i+1]='A'; fname[i+1]<='Z'; fname[i+1]=fname[i+1]+1){
            if (_dos_findfirst (fname, _A_NORMAL, &fbuf) != 0) {
                strcpy (&fname[i+2], ".T");
                sprintf (&fname[i+4],"%d",nlink);
                if ((handle = creat (fname, S_IREAD|S_IWRITE)) != -1){
                    close (handle);
                    *name++ = fname[i]; *name++ = fname[i+1];
                    *name = 0;
                    return (0);
                }
            }
        }
    *name = 0;
    return (-1);
}
/**********  textfile  **********************************************
```

```
 *
 *      Return full name of the text file
 *
 *      Returns: -1  File not found
 *                0  O.K.
 */

textfile (name, newname)
char *name, *newname;           /* Strings used for file name   */
{
    char fname[100];

    strcpy (fname, maildir); strcat (fname,name);
    strcat (fname, ".T*");
    if (_dos_findfirst (fname, _A_NORMAL, &fbuf))  return (-1);
    strcpy (newname, maildir);    strcat (newname, fbuf.name);
    return (0);
}
/**********  voicefile  *********************************************
 *
 *      Create a new voice file with given name and msgid.
 *
 *      Returns: -1  if Unable to Create
 *                0  Otherwise
 */
voicefile (name, msgid)
char *name;                 /* File Name without extension            */
int  msgid;                 /* Message Number                         */
{
    int i, handle;
    char fname[100];

    if ((msgid < 0) || (msgid > 99)) return (-1);

    strcpy (fname, maildir);  strcat (fname, name);
    strcat (fname, ".V");     i = strlen (fname);
    sprintf (&fname[i],"%d",msgid);
    if ((handle = creat (fname, S_IREAD|S_IWRITE)) != -1) {
        close (handle);
        return (0);
    }
    return (-1);
}
/**********  delmail  *******************************************************
 *
 *      Delete mail files
 *
 *      Returns: -1  File not found
 *                0  Deleted
 *               >0  Number of links remaining
 */

delmail (name)
char *name;                 /* File Name without extension            */
{
    int i, nlink, handle;
    char fname[100], oldname[100], newname[100];

    strcpy (fname, maildir);
    strcpy (oldname, maildir);
    strcpy (newname, maildir);

    strcat (fname, name);  strcat (fname, ".T*");

    if (_dos_findfirst (fname, _A_NORMAL, &fbuf))  return (-1);

    strcat (oldname, fbuf.name);  strcat (newname, fbuf.name);
    for (i=strlen (oldname); i>0; i--)  if (oldname[i] == '.') break;
    if (i < 1) return (-1);

    nlink = 0;
    sscanf (&oldname[i+2],"%d",&nlink);
    if (nlink-- > 1) {                          /* Rename File */
        sprintf (&newname[i+2],"%d",nlink);
        if (rename (oldname, newname) != 0) return (-2);
        return (nlink);
    }
    else {
        strcpy (fname, maildir);  strcat (fname, name);
        strcat (fname, ".*");
        if (_dos_findfirst (fname, _A_NORMAL, &fbuf))  return (-3);
        while (1) {
            strcpy (oldname, maildir);   strcat (oldname, fbuf.name);
            if (remove (oldname)) return (-4);
            if (_dos_findnext (&fbuf))  return (0);
        }
    }
}
/********************  file_name (num) ******************************
 *
 * This function calls newfile to get a unique text file name.
 */

file_name(num)
int num;
{
    int link,ret;

    rbuff[num][0] = 0x20;
    link = rbuff[num][10];
    sbuff[num][0] = 'q';
    if ((ret = newfile(ses[num].text_file, link)) == 0) {
        sbuff[num][10] = 0;
    }
    else sbuff[num][10] = 1;
    ncb_send (num, 15);

}
/**************************  time_f(ctime) **************************
* This function gets the system time and date and returns in ctime.
*/

time_f(ctime)
char *ctime;
{
    struct tm *newtime;
    time_t long_time;

    time(&long_time);
    newtime  = localtime(&long_time);
    strncpy (&ctime[0], asctime(newtime), 16);
    sprintf(&ctime[16], " %d", newtime->tm_year);
    return (0);
}
/****************************  save_text(num) ***********************
* This function save the message text and writes headers in
* corresponding user's headers file.
*/
save_text (num)
int num;
{
    char hdr[81], c_time[20], hf_name[12], tf_name[12];
    int recp_no, ms_pnt, i, u_len, m_len;
    FILE *fopen(), *fp;

    if (rbuff[num][2] == 0) {
        if (!ses[num].fflag) {
            textfile (ses[num].text_file, tf_name);
            if ((ses[num].fptr = open(tf_name,1)) < 0) {
                sbuff[num][0] = 'k';
                sbuff[num][10] = 1;
                ncb_send(num, 15);
            }
            else {
                ses[num].fflag = 1;
                recp_no = rbuff[num][4];
                ms_pnt = ((recp_no + 1) * 10);
                u_len = strlen(ses[num].user);
                time_f (c_time);
                strncpy(&hdr[0], ses[num].user, u_len);
                while (u_len <= 8) {
                     hdr[u_len] = 0x20;
                     u_len++;
                }
                strncpy (&hdr[8], c_time, 19);
                i = 0;
                while ((rbuff[num][ms_pnt+i] != '\n') && (i < 40)) {
                     hdr[i+27] = rbuff[num][ms_pnt+i];
                     i++;
                }
                while (i < 40) {
                     hdr[i+27] = 0x20;
                     i++;
                }
                strcpy(&hdr[67], ses[num].text_file);
                u_len = strlen(hdr);
                for(i = u_len; i < 80; i++)
                    hdr[i] = 0x20;
                hdr[80] = '\0';
                while (recp_no > 0) {
                     strcpy (hf_name, strupr(&rbuff[num][recp_no*10]));
                     strcat (hf_name, ".HDR");
                     if ((fp = fopen (hf_name, "a+")) == NULL ) {
                        sbuff[num][10+recp_no] = 1;
                     }
                     else {
                        fputs (hdr , fp);
                        fclose (fp);
                        sbuff[num][10 + recp_no] = 0;
                     }
                    recp_no--;
                }
            }
            if (ses[num].fflag) {
                m_len = ses[num].rncb.length - ms_pnt;
                write (ses[num].fptr, &rbuff[num][ms_pnt], m_len);
                if (rbuff[num][3] == 0) {
                    close(ses[num].fptr);
                    ses[num].fflag = 0;
                }
                sbuff[num][0] = 'k';
                sbuff[num][10] = 0;
                ncb_send (num, 21);
            }
        }
    }
    else {
        if (rbuff[num][2] == 1) {
            if (rbuff[num][3] != 0) {
                write (ses[num].fptr, &rbuff[num][10], 990);
            }
            else {
                m_len = ses[num].rncb.length - 10;
                write (ses[num].fptr, &rbuff[num][10], m_len);
                close(ses[num].fptr);
                ses[num].fflag = 0;
                sbuff[num][0] = 'k';
                sbuff[num][10] = 0;
                ncb_send (num, 15);
            }
        }
    }
}
/**************** del_msg (num) **************************************
*
*       This function deletes the message(s) from the header file
*       and calls delmail(name) to delete the corresponding
*       text file.
*/
del_msg (num)
int num;
{
    int fpold, fpnew, handle;
    int n_bytes, no_msg, msg_no, no, i, p;
    char hdr[81], fname[12];

    rbuff[num][0] = 0x20;
    sbuff[num][0] = 's';
    no_msg = rbuff[num][2];
    if ((fpold = open(ses[num].dir, 0)) < 0) {
        sbuff[num][10] = 1;
        ncb_send (num, 15);
        return (-1);
    }
    if ((handle = creat ("hdr.hdr", S_IREAD|S_IWRITE)) != -1)
        close (handle);

    if ((fpnew = open("hdr.hdr", 1)) < 0) {
        sbuff[num][10] = 1;
        ncb_send (num, 15);
        return (-1);
    }
    no = 1;
    while (no_msg > 0) {
        msg_no = rbuff[num][10+no_msg];
        while (no < msg_no) {
            read (fpold, hdr, 80);
            write (fpnew, hdr, 80);
            no++;
        }
        if ( no == msg_no ) {
            read (fpold, hdr, 80);
            no++;
            i = 67;
            p = 0;
            while (hdr[i] != 0x20) {
                fname[p] = hdr[i];
                p++;
                i++;
            }
            delmail (fname);
        }
        no_msg--;
    }
    while ((n_bytes = read (fpold, hdr, 80)) > 0) {
        write (fpnew, hdr, n_bytes);
    }
    close(fpold);
    close(fpnew);
    if (!remove (ses[num].dir))
        if (!rename ("hdr.hdr", ses[num].dir))
            sbuff[num][10] = 0;
        else sbuff[num][10] = 1;
    else sbuff[num][10] = 1;
    ncb_send (num, 15);
    return (0);
}
```

```
/*********************************************************************
*
*                 PROGRAM TO BE RUN AT THE WORKSTATION
*
*/

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <string.h>
#include <dos.h>
#include "ncbs.h"

#define Success 0
#define Pending 0xFF
#define Timeout 0x05
#define call    0x10
#define send    0x14
#define listen  0x11
#define recv    0x15
#define hangup  0x12
#define status  0x33

struct line {

   int  num;
   char text[120];
   struct line *next;   /* pointer to next entry */
   struct line *prior;  /* pointer to previous record */
} ;

struct line *start;     /* pointer to first entry in the list */
struct line *last;      /* pointer to last entry */

struct line *dls_store(struct line *),*find(int);

int menu_select(void),enter(int);
void patchup(int,int),delete(void),list(void);
void save(char *),load(char *);

char wbuff[1000];

struct  NCB sncb,rncb;  /* The NCB */
byte    sbuff[Bufsize]; /* Data Buffer for the send NCB */
byte    rbuff[Bufsize]; /* Data Buffer for the recv NCB */
byte    server[16] = "LAB1XT          ";
byte    hbuff[10000],mbuff[10000];
byte    local[16] = "*               ";

struct recepient {
        int flag;
        char name[9];
}users[10];

union   REGS    reg;
struct  SREGS   seg;

int del_no;
int voice_no = 0;;
char    signon[] = "\n MESSAGE SEND.\n\n";
```

```
static int start_line,end_line;        /* for cursor size mgt */

main()
{
    int i,len;

    fprintf(stderr,"%s",signon);

    del_no = 0;
    cur_norm();
    ncb_call ();
    printf ("LSN: %x\n",sncb.lsn);

    if ((sncb.lsn != 0) || (sncb.lsn != 0xFF)) {
        rncb.lsn = sncb.lsn;
        if (ver_pass () == 0 ) action ();
    }
}

/******************************  quit *******************************
*/

quit ()
{
    sncb.command = hangup;
    sendlncb (&sncb);
}

/******************************  send quit **************************
*/
send_quit()
{
    sbuff[0] = 'M';
    ncb_send ();
    quit ();
}

/************************** load_ncb *******************************
* issues the NCB. If an error occurs, it exits showing the error.
*/

load_ncb(ncb)
struct NCB *ncb;
{
    char far *fptr;

    while (ncb->retcode == Pending) if (kbhit()) getch();
    if (ncb->retcode != Success) {
      printf (" \nError: NCB Command is %2.2X and retcode is %2.2X\n"
                      , ncb->command, ncb->retcode);
        if ( ncb->retcode == Timeout) {
            if (ncb->command == recv) send_quit ();
            else quit ();
        }
        exit(1);
    }
    ncb->retcode = 4;
    fptr = (char far *) ncb;

    reg.x.bx = FP_OFF (fptr);
```

```
     seg.es = FP_SEG (fptr);
     reg.h.ah = 0x01;

     int86x (0x5C, &reg, &reg, &seg);
     if ((ncb->retcode != Success) && (ncb->retcode !=Pending)) {
       printf (" \nError: NCB Command is %2.2X and retcode is %2.2X\n"
                      , ncb->command, ncb->retcode);
         if ( ncb->retcode == Timeout) {
             if (ncb->command == recv ) send_quit ();
             else quit ();
         }
         exit(1);
     }
}

/***************************** sendlncb *****************************
* issues the NCB. It does not do any error display or checking.
*/

sendlncb(ncb)
struct NCB *ncb;
{
    char far *fptr;

    fptr = (char far *) ncb;
    reg.x.bx = FP_OFF (fptr);
    seg.es = FP_SEG (fptr);
    reg.h.ah = 0x01;
    int86x (0x5C, &reg, &reg, &seg);
}

/****************************** Ncb_call ****************************
* This procedure performes a call to the remote node specified after
* having added the local name.
*/

ncb_call()
{
    int i,p,len;

    init1 ();
    ncb_status ();
    len = sncb.length;
    i = 60;
    p = 0;
    while ((p < 16 ) && ( i < len )) {
        sncb.name[p] = sbuff[i];
        i++;
        p++;
    }
    strncpy(sncb.callname, server, 16);
    sncb.command = call;
    load_ncb(&sncb);
}

/**************************************** Ncbsend *****************/

ncb_send()
{
    sncb.command = send;
    load_ncb(&sncb);
}


/*************************** ncb_recv(lsno) ********************
*  This function initialises the ncb to recv where lsno is the
*  corresponding session number.
*/

ncb_recv()
{
    rncb.buffer = (char far *) rbuff;
    rncb.length = Bufsize;
    rncb.lana_num = 0;
    rncb.command = recv;
    load_ncb(&rncb);
}

/******************************** Ncb_status ***************** */

ncb_status()
{

    strncpy(sncb.callname, local, 16);
    sncb.command = status;
    load_ncb(&sncb);
}

/************************ initl ******************************
*  predefines the fields in the NCB structure.
*/

initl()
{

    sncb.buffer = (char far *) sbuff;
    sncb.retcode = Success;
    sncb.lana_num = 0;
    sncb.length = 348;
    sncb.sto = 60;
    sncb.rto = 60;
}

/*********************** Ver_pass ******************************
*  This function sends user_name & password at server & receives
*  respose.
*/

ver_pass()
{
    int i,p;
    byte lsn;
    char  user[15], pass[10];

    lsn = sncb.lsn;
    p = 0;
    rbuff[0] = 'a';
    while ((p < 3) && (rbuff[0] == 'a')) {
        for (i = 0; i < 35; i++) {
            sbuff[i] = 0x20;
            if (i < 10) pass[i] = 0x20;
        }
        sbuff[0] = 'A';
        do {
            printf ("USER: ");
            gets(user);
            if(strlen(user) > 8) printf("\n too long\n");
        } while(strlen(user) > 8);
        printf ("PASSWORD: ");
        getpwd(pass,9);
        strncpy(&sbuff[10], user, 9);
        strncpy(&sbuff[25], pass, 10);
        sncb.length = 35;
        ncb_send ();
        p++;
        rbuff[0] = 0x20;
        ncb_recv ();
        if (rbuff[0] == 'a') {
            switch (rbuff[10]) {

                case 0:{
                        rbuff[0] = 0x20;
                        return (0);
                }
                case 1:{ printf ("\nUser name not found.\n");
                        if (p == 3) {
                             send_quit ();
                             return (1);
                        }
                        break;
                }
                case 2:{ printf ("\nUnauthorised user.\n");
                        if (p == 3) {
                            send_quit ();
                            return (1);
                        }
                        break;
                }
            }
        }


    }
}

/******************************  getpwd  *************************
*  This function reads character(s) from the keyboard for password.
*/

getpwd(linebuf, maxch)
char linebuf[];
int maxch;
{
    int         ch, nch = 0;

    while ((ch = getch()) != 0x0d) {
        if (ch >= 0x20) {
            if (nch < maxch) {
                linebuf[nch] = ch; nch++;
            }
        }
        else if (ch == 8) {
            if (nch != 0) nch--;
        }
        else if (ch == 0) getch();
    }
    linebuf[nch] = 0;
    return(nch);
}

/****************************  main proceedures start here *******/

action()
{
   for(;;) {
       switch(choice()) {

           case 1: cur_block();
                   read_mail ();
                   cur_norm();
                   printf("\n");
                   break;

           case 2: cur_block();
                   voice_no = 0;
                   send_mail ();
                   printf("\n");
                   cur_norm();
                   break;

           case 3: send_quit ();
                   return (0);

       }
   }
}

choice(void)
{

   char s[80];
   int c;

   printf("\n\t1: READ THE MAIL.\n");
   printf("\t2: SEND A MAIL.\n");
   printf("\t3: QUIT\n");

   do {
       printf("\n\tEnter your choice:  ");
       cur_block();
       gets(s);
       c = atoi(s);
       cur_norm();
   } while (c < 0 || c > 3);
   return (c);
}

/*****************************  send_mail *************************
* This function checks recepients ID and sends mail to for them.
*/
```

```
send_mail ()
{
    int len,recp_no;

    recp_no  = 0;
    recp_no = recv_user ();
    if (recp_no > 0) {
        len = 0;
        len = editor();
        if (len > 0 ) {
            send_text (len);
        }
    }
}

/***************************  read_mail ()  ***********************/

read_mail ()
{
    int fine,flag,msg_no,tmsg_no,r_flag;
    char ch,resp[20];

    r_flag = 0;
    if ((fine = recv_hdr ()) > 0) {
        tmsg_no = display_hdr (fine);
        flag = 1;
        while (flag) {
            gets(resp);
            if (strcmp(resp ,NULL)) {
                sscanf (resp, "%c", &ch);
                switch (ch) {

                    case 'q': if (del_no > 0)
                              {
                                  send_del (del_no);
                                  del_no = 0;
                              }
                              return (0);

                    case 'd': del_msg (tmsg_no);
                              break;
                    case 's': if (r_flag) {
                                r_flag = 0;
                                store_msg (msg_no);
                              }
                              else
                              printf("No message has been read.\n");
                    default:msg_no = atoi(resp);
                            if ((msg_no > 0) && (msg_no <= tmsg_no)){
                                if (hbuff[(msg_no-1)*80] == '*') {
                                    printf ("Deleted message.\n");
                                }
                                else {
                                    get_msg (msg_no);
                                    r_flag = 1;
                                }
                            }
                            else
                            printf("Invalid message number.\n");
```

```
                        break;
                    }
                }
            }
        }
    else {
        printf ("NO MAIL\n");
        return (-1);
    }
}

/*********************  recv_user () ******************************
* This function takes the recepient(s) name and verify them.
* It returns the number of verified recepients.
*/

recv_user ()
{
    char recp[100];
    int buflen,slen, i,j,k,s,p,row,col;
    int ver_no,fault;

    ver_no = 0;
    printf ("Give recepient(s) name.\n");
    for (i = 0; i < 10; i++) {
        users[i].flag = 0;
        for (j = 0; j < 9; j++)
            users[i].name[j] = 0x20;
    }
  again: gets(recp);
   if (strcmp (recp, NULL)) {
       s = 0;
       fault = 0;
       row = 0;
       strlwr(recp);
       slen = strlen(recp);
       while (s < slen) {
           while (recp[s] == 0x20) s++;
           col = 0;
           while ((col < 8) && (recp[s] != 0x20) && (s < slen)) {
               users[ver_no + row].name[col] = recp[s];
               col++;
               s++;
           }
           if (col != 0) users[ver_no + row].name[col] = '\0';
           if (col >= 8) {
               printf ("Too long name %s\n",users[row].name);
               printf ("Give all the name again.\n");
               for (row = 0; row < 10; row++) {
                   users[i].flag = 0;
                   for (col = 0; col < 10; col++)
                       users[row].name[col] = 0x20;
               }
               s = 0;
               ver_no = 0;
               goto again;
           }
           row++ ;
           while (recp[s] == 0x20) s++;
       }
```

```
        buflen = (row + 1)*10;
        for (i = 0; i < buflen; i++)
            sbuff[i] = 0x20;
        sbuff[0] = 'F';
        for (i = 0; i < row; i++) {
            p = i + ver_no;
            strcpy(&sbuff[(i+1)*10], &users[p].name[0]);
        }
        sncb.length = buflen;
        ncb_send ();
        rbuff[0] = 0x20;
        ncb_recv ();
        if (rbuff[0] == 'f') {
            p = ver_no;
            for (i = 0; i < row; i++) {
                if (rbuff[i + 10] == 0) {
                    ver_no++;
                    users[i + p].flag = 1;
                }
                else {
                    fault++;
                    users[i + p].flag = 0;
                }
            }
            if (fault > 0) {
                for ( i = 0; i < row; i++) {
                    if (users[i + p].flag == 0) {
                    printf ("\nName '%s' not found. Enter again.\n",
                                users[p + i].name);
                        for (j = 0; j < 9; j++)
                            users[p + i].name[j] = 0x20;
                    }
                }
                for (i = p; i < (p + row); i++) {
                    if (users[i].flag == 0) {
                        for (j = i; j < (p + row -1); j++) {
                            strcpy (users[j].name, users[j+1].name);
                            users[j].flag = users[j+1].flag;
                        }
                        users[j].flag = 0;
                        for (k = 0; k < 9; k++)
                            users[j].name[k] = 0x20;
                    }

                }
                for (i = 0; i < 20; i++)
                    rbuff[i] = 0x20;
                goto again;
            }
        }
    }
    if (ver_no > 0) {
        printf ("VERIFIED USERS: \n");
        for (i = 0; i < ver_no; i++) {
            if (users[i].flag)
                printf ("%d. %s\n",i+1,users[i].name);
        }
        sbuff[0] = 'Q';
        sbuff[10] = ver_no;
        sncb.length = 15;
```

```
          ncb_send ();
          rbuff[0] = 0x20;
          ncb_recv ();
          if (rbuff[0] = 'q') {
              if (rbuff[10] != 0) {
                  printf ("TEXT FILE CAN'T BE CREATED.\n");
                  return (0);
              }
          }
      }
      return (ver_no);
  }
  /****************************  recv_hdr ()  ************************
  * This function receives the message header file.
  */

  recv_hdr ()
  {
      int flag, len,i;

      flag = 1;
      i = 0;
      sbuff[0] = 'B';
      sncb.length = 10;
      ncb_send ();
      rbuff[0] = 0x20;
      rbuff[0] = 0x20;
      ncb_recv ();

      while (flag) {
          if (rbuff[0] = 'b') {
              switch (rbuff[2]) {

                  case 0: { hbuff[i] = '\0';
                          flag = 0;
                          return (i);
                  }
                  case 1: { len = (rncb.length - 10);
                          strncpy (&hbuff[i], &rbuff[10], len);
                          i = i + len;
                          ncb_send ();
                          rbuff[0] = 0x20;
                          rbuff[0] = 0x20;
                          ncb_recv ();
                          break;
                  }
                  case 2: { printf ("Header file can't be open.\n");
                          return (-1);
                  }
              }
          }
      }
}

/****************************  display_hdr(len) *****************
* This function display the message headers from hbuff.
* It will return number of total messages.
*/

display_hdr (len)
```

```
int len;
{
    int i, no,p;

    i = 0;
    no = 0;
    while (i < (len-1)) {
        printf ("%d. ",(no+1));
        for (p = 0; p < 8; ++p) {
            printf("%c",hbuff[i]);
            ++i;
        }
        printf (" ");
        for (p = 0; p <19; ++p) {
            printf("%c",hbuff[i]);
            ++i;
        }
        printf ("\"");
        for (p = 0; (p <40); ++p) {
                printf("%c",hbuff[i]);
                ++i;
        }
        while (p < 53) {
            i++;
            p++;
        }
        printf ("\"\n");
        no++;
    }
    return(no);
}

/****************************  get_msg (msg_no) *****************
* This function find the text file name from hbuff and receive the
* message corresponding to 'msg_no' and display on the screen.
*/

get_msg (msg_no)
int msg_no;
{
    int i,p,ret;

    char fname[12];
    i = 0;
    i = (msg_no-1)*80 + 67;
    p = 0;
    while (hbuff[i] != 0x20) {
        fname[p] = hbuff[i];
        p++;
        i++;
    }
    fname[p] = '\C';
    ret = recv_msg(fname);
    if (ret > 0) {
        i = (msg_no-1)*80;
        printf ("\nFrom: ");
        for (p = 0; p < 8; ++p) {
            printf("%c",hbuff[i]);
            ++i;
        }
        printf ("\nDate: ");
        for (p = 0; p < 19; ++p) {
            printf("%c",hbuff[i]);
            ++i;
        }
        printf ("\nSubject: ");
        for (p = 0; ((p <40) && (hbuff[i] != '\0')); ++p) {
                printf("%c",hbuff[i]);
                ++i;
        }
        printf ("\n");
        printf ("______________________________________________\n");
        display_msg ();
    }
}

/**************************** recv_msg (fnmae) **********************
* This function receives the message  file and stores in mbuff.
*/

recv_msg (fname)
char fname[12];
{
    int flag, len,i;

    flag = 1;
    i = 0;
    sbuff[0] = 'C';
    strcpy(&sbuff[10], fname);
    sncb.length = 22;
    ncb_send ();
    rbuff[0] = 0x20;
    rbuff[0] = 0x20;
    ncb_recv ();

    while (flag) {
        if (rbuff[0] = 'c') {
            switch (rbuff[2]) {

                case 0: { mbuff[i] = '\0';
                        flag = 0;
                        return (i);
                }
                case 1: { len = (rncb.length - 10);
                        strncpy (&mbuff[i], &rbuff[10], len);
                        i = i + len;
                        ncb_send ();
                        rbuff[0] = 0x20;
                        rbuff[0] = 0x20;
                        ncb_recv ();
                        break;
                }
                case 2: { printf ("Message file can't be open.\n");
                        return (-1);
                }
            }
        }
    }
}
```

```
/*****************************  display_msg () *********************
* This function displays the message received in mbuff.
*/

display_msg ()
{
    int i,line,res,ret;
    char ch;

    i = 0;
    line = 0;
    while (mbuff[i] != '\0') {
        while (mbuff[i] != '\n') {
            printf("%c",mbuff[i]);
            i++;
        }
        if (mbuff[i] == '\n') {
                line++;
                printf("\n");
                i++;
        }

        if (line == 24) {
            line = 0;
            printf ("PRESS ANY KEY OR F1 TO READ VOICE FILE \n");
            while (!kbhit());
        }

        if((mbuff[i] = '\0') || (line == 24))
        {
          res = getch();
          if(res == 0)
          {
              res = getch();
              ret = read_menu();
              switch (ret) {

                 case 1: playback_voice_req();
                         break;

                 case 2: break;
              }
          }
        }
    }
}

read_menu()
{
   char s[80];
   int c;

   drawbox(2,19,40,4,38,1);
   setcurpos(20,43);
   printf("1: READ VOICE FILE ");
   setcurpos(2144);
   printf("2: QUIT ");

   do {
       setcurpos(22,44);
```

```
        bright();
        printf("enter your choice here: ");
        normal();
        gets(s);
        c = atoi(s);
    } while (c < 0 || c > 2);
    return (c);
}

/******************************  send_text (len) ******************
* This function sends the text of given length len to server.
*/

send_text(len)
unsigned int len;
{
    unsigned int  temp,i, ini_len, ms_pnt;

    temp = len;
    sbuff[0] = 'K';
    i = 0;
    while (users[i].flag) {
        strcpy(&sbuff[(i+1)*10], users[i].name);
        i++;
    }
    ms_pnt = (i+1)*10;
    ini_len = 1000 - ms_pnt;
    sbuff[4] = i;
    if (temp < ini_len) {
        sbuff[2] = 0;
        sbuff[3] = 0;
        strncpy (&sbuff[ms_pnt], &mbuff[0], temp);
        sncb.length = temp + ms_pnt + 1;
        ncb_send ();
        rbuff[0] = 0x20;
        ncb_recv ();
        if (rbuff[0] == 'k') {
            if (rbuff[10] == 0) {
                printf ("MESSAGE SEND COMPLETED.\n");
                for (i = 0; users[i].flag; i++)
                    if (rbuff[10+i] != 0)
                        printf ("ERROR: Message not send to '%s'\n",
                                users[i].name);
            }
            else printf ("ERROR: MESAGE NOT STORED.\n");
        }
    }
    else {
        sbuff[2] = 0;
        sbuff[3] = 1;
        strncpy (&sbuff[ms_pnt], &mbuff[0], ini_len);
        sncb.length = Bufsize;
        ncb_send ();
        rbuff[0] = 0x20;
        ncb_recv ();
        if (rbuff[0] == 'k') {
            if (rbuff[10] != 0) {
                printf ("ERROR IN OPENING TEXT FILE.\n");
                return (0);
            }
            else {
                temp = temp - ini_len;
                while (temp > 990) {
                    sbuff[2] = 1;
                    sbuff[3] = 1;
                    strncpy (&sbuff[10], &mbuff[ini_len], 990);
                    sncb.length = Bufsize;
                    ncb_send ();
                    temp = temp - 990;
                    ini_len = ini_len + 990;
                }
                if (temp >= 0) {
                    sbuff[2] = 1;
                    sbuff[3] = 0;
                    strncpy (&sbuff[10], &mbuff[ini_len], temp);
                    sncb.length = temp + 11;
                    ncb_send ();
                    rbuff[0] = 0x20;
                    ncb_recv ();
                    if (rbuff[0] == 'k') {
                        if (rbuff[10] == 0)
                          printf ("MESSAGE SEND COMPLETED.\n");
                        else printf ("ERROR: MESAGE NOT STORED.\n");
                    }
                }
            }
        }
    }
}

/************************** del_msg (tmsg_no) ********************
* This function ask for message number to delete and mark the
* message deleted. The 'tmsg_no' is the total number of message.
*/

del_msg(tmsg_no)
int tmsg_no;
{
    int   msg_no, index;
    char resp[20];

    printf("What ?\n");
    gets (resp);
    msg_no = atoi(resp);
    if ((msg_no > 0) && (msg_no <= tmsg_no)) {
        index = (msg_no - 1) * 80;
        if (hbuff[index] == '*') {
            printf ("Deleted message.\n");
        }
        else {
            hbuff[index] = '*';
            del_no++;
        }
    }
    else printf("Invalid message number.\n");
}

/**************************** send_del(no_msg) *******************
* This function sends the message to serever for deleting
* 'no_msg' number of message.
```

```
 */

 send_del (no_msg)
 int no_msg;            /* Number of message(s) to delete.*/
 {
     int temp;

     temp = 0;
     sbuff[0] = 'S';
     sbuff[2] = no_msg;
     sncb.length = 10 + no_msg + 2;
     while (no_msg > 0) {
         if (hbuff[temp*80] == '*') {
             temp++;
             sbuff[10 + no_msg] = temp;
             printf ("MSG-NO: %d\n",temp);
             no_msg--;
         }
         else temp++;
     }
     ncb_send ();
     rbuff[0] = 0x20;
     ncb_recv ();
     if (rbuff[0] = 's') {
         if (rbuff[10] != 0) {
             printf ("ERROR: Message(s) not deleted.\n");
         }
     }
}
/******************** store_msg (msg_no) ***************************
*
*       It store the message in to the current drive at work
*       station end.
*/
store_msg (msg_no)
int msg_no;
{
    int i,p,fptr;
    char resp[20];
    FILE *stream;

    printf ("Type the file name.\n");
    gets (resp);
    if (strcmp (resp, NULL)) {
        if ((fptr = creat(resp, 777)) < 0) {
            printf ("ERROR in creating %s file.\n",resp);
            return (-1);
        }
        else {
            if ((stream = fopen(resp, "w")) == NULL ) {
                printf ("ERROR in %s opening.\n",resp);
                return (-1);
            }
            else {
                i = 0;
                i = (msg_no-1)*80;
                fprintf (stream, "From: ");
                for (p = 0; p < 8; ++p) {
                    fprintf (stream, "%c",hbuff[i]);
                    ++i;
                }
                fprintf (stream, "\nDate: ");
                for (p = 0; p < 19; ++p) {
                    fprintf (stream, "%c",hbuff[i]);
                    ++i;
                }
                fprintf (stream, "\nSubject: ");
                for (p = 0; ((p <40) && (hbuff[i] != '\0')); ++p) {
                    fprintf (stream, "%c",hbuff[i]);
                    ++i;
                }
                fprintf (stream, "\n____________________________\n");
                i = 0;
                while (mbuff[i] != '\0') {
                    fprintf (stream, "%c",mbuff[i]);
                    i++;
                }
            }
        }
        fclose (stream);
    }
}

/************************* text editor *****************************
* text and voice composing starts here.
*/
editor()
{

   char s[20],choice,fname[20];
   int len, linenum = 1;
   char string[20],dir_str[15],ss[30];
   char *result;

   start = last = NULL;      /* initialize top & bottom pointers */
   cls();cur_norm();
   drawbox(2,10,25,4,30,1);
   setcurpos(11,27);
   printf("WELCOME TO THE VMS PACKAGE");
   setcurpos(12,27);
   printf("      version 1.0         ");
   setcurpos(24,1);
   printf("Press any key to begin..");
   while(!kbhit());getch();
   cls();
   len = voice_no = 0;
   for(;;) {
       choice = menu_select();
       switch(choice) {
           case 1: cur_block();
                   printf("Enter line number: ");
                   gets(s);
                   if (!strcmp(s, NULL)) linenum = last->num + 1;
                   else linenum = atoi(s);
                   setcurpos(3,0);
                   list();
                   setcurpos(last->num + 3, 0);
                   enter(linenum);
                   cur_norm();
                   break;
            case 2: delete();
                    cls();
                    break;

            case 3: voice_no++;
                    voice_request(voice_no);
                    pastescrn(18,39,6,40,wbuff);
                    break;

            case 4: bright();
                    printf("enter file name :");
                    normal();
                    gets(fname);
                    save(fname);
                    cls();
                    break;

            case 5: bright();
                    printf("enter file name: ");
                    normal();
                    gets(fname);
                    load(fname);
                    list();
                    break;

            case 6: cls();
                    setcurpos (3,0);
                    list();
                    printf("\n");
                    break;

            case 7: del_voice_file ();
                    break;

            case 8: len = save_buff();
                    cls();
                    return (len);
        }
    }
}

/*****************************************************************
*       main menu to be displaced and selected from.
*/

menu_select(void)
{

   char s[80];
   int c;

   setcurpos(0,0);
   reverse();
   printf(" 1: EDIT ");
   printf("2: DEL LINE ");
   printf("3: VOICE ");
   printf("4: SAVE ");
   printf("5: LOAD ");
   printf("6: LIST ");
```

```
    printf("7: DEL VOICE ");
    printf("8: QUIT ");
    normal();
    clrline(1);
    do {
        setcurpos(0,0);
        bright();
        printf("\nenter your choice: ");
        normal();
        gets(s);
        c = atoi(s);
        clrline(1);
        setcurpos(0,0);
        printf("\nenter your choice: %s\n",s);
    } while (c < 0 || c > 8);
    return (c);
}


/* link list being created --enter info in the node */

enter(int linenum)
{
     int in;
     char cmd;
     struct line *info;

     for (;;) {
         info =  malloc(sizeof(struct line));
         if(!info) {
             printf("\n\tout of memory ");
             exit(1);
         }
         printf("%d : ",linenum);
         gets(info->text);
         sbuff[0] = 'R';          /** TO  KEEP SERVER LISTENING */
         sncb.length = 10;
         ncb_send ();
         in = scan_line(info->text);
         info->num = linenum;
         if(in) {
           if(find(linenum))
                 patchup(linenum,1);      /* fix up old line no */
           if(in) start = dls_store(info);
         }
         else {
               break;
         }
         linenum++;
      }
     return linenum;
}

/******* to reorder the line numbers after insertion/deletion  ****/

void patchup(int n, int incr)
{
    struct line *i;

    i = find(n);
    while(i) {
      i->num = i->num+incr;
      i = i->next;
    }
}
/******* double link sorted storage function ***********************/

struct line *dls_store(struct line *i)          /* new element */
{

   struct line *old,*p;

   if(!last) {                      /* first element in list */
       i->next = NULL;
       i->prior = NULL;
       last = i;
       return i;
   }
   p = start;                       /* start at top of list */
   old = NULL;
   while(p) {
       if(p->num < i->num) {
           old = p;
           p = p->next;
       }
       else {
           if(p->prior) {
               p->prior->next = i;
               i->next = p;
               p->prior = i;

               return start;
           }
           i->next = p;           /* new first element */
           i->prior = NULL;
           p->prior = i;
           return i;
       }
   }
   old->next = i;                           /* put an end */
   i->next = NULL;
   i->prior = old;
   last = i;
   return start;
}

/* delete a line */
void delete(void)
{
    struct line *info;
    char s[10];
    int linenum;

    bright();
    printf ("Enter the line number :");
    normal();
    gets(s);
    linenum = atoi(s);
    info = find(linenum);
    if ((info) && (strncmp(info->text, "##### ", 6) != 0)) {
        if (start == info) {
            start = info->next;
            if (start) start->prior = NULL;
            else last = NULL;
        }
        else {
            info->prior->next = info->next;
            if (info != last)
                info->next->prior = info->prior;
            else
                last = info->prior;
        }
        printf("%d: %s\n",info->num,info->text);
        free(info);                          /* Return memory to system */
        patchup(linenum + 1, -1);                    /* dec line num */
    }
}

/* Find a line of text */
struct line *find( int linenum)
{
    struct line *info;

    info = start;
    while (info) {        if (linenum == info->num)  return info;
            info = info->next;           /* Get next line */
    }
    return NULL;
}

/* Display the entire text */
void list(void)
{
    struct line *info;
    int count;

    count = 1;
    info = start;
    while (info) {
        printf("%d: %s \n",info->num,info->text);
        if(count == 23) {
                      count = 0;
                      bright();
                      setcurpos(24,1);
                      printf("Press any key.......\n");
                      while(!kbhit());getch();
                      normal();
        }
        info = info->next;count++;               /* Get next line */
    }
}

/* Save the file to the disk*/
void save(char *fname)
{
    struct line *info;
    char *p;

    FILE *fp;
    fp = fopen (fname,"wb");
    if (!fp) {
        printf ("can't open the file \n");
        exit(1);
    }
    printf("Saving file on disk.\n\n ");
    info = start;
    while (info) {
        p = info->text;
        while(*p) putc(*p++ , fp);      /*save byte at a time */
        putc('\r',fp);                  /* terminator */
        putc('\n',fp);                  /* terminator */
        info = info->next;              /* Get next line */
    }
    fclose(fp);
}

/* Load the  file.*/
void load(char *fname)
{

    register int size , lnct;
    struct line *info,*temp;
    char *p;
    FILE *fp;

    fp = fopen(fname,"rb");
    if (!fp) {
        printf ("Can't open file \n");
        return;
    }

    /* Free any previously allocated memory */

    while (start) {
        temp = start;
        start = start->next;
        free(temp);
    }

    printf ("Loading file .\n\n");
    size = sizeof(struct line);
    start = malloc(size);
    if(!start) {
       printf("out of memory \n");
       return;
    }

    info = start;
    p = info->text;
    lnct = 1;
    while ((*p = getc(fp)) != EOF) {
        p++;
        while(((*p = getc(fp)) != '\r') && (*p != EOF))
         p++;

        if( *p != EOF) getc(fp);        /*throw away the \n    */
        *p = '\0';
        info->num = lnct++;
        info->next = malloc(size);      /* get away with next */
        if(!info->next) {
           printf("out of memory:\n");
           return;
        }
        info->prior = temp;
        temp = info;
        info = info->next;
        p = info->text;
     }
     temp->next = NULL;
     last = temp;
     free(info);
     start->prior = NULL;
     fclose(fp);
}
/*********************************************************************
 *
 *       voice request from user processed here.Takes voice number as
 *       argument and passes on to server for creation of appropriate
 *       file.
 */

voice_request(v_num)
int v_num;
{

    sbuff[0] = 'G';
    sbuff[10] = v_num;
    sncb.length = 15;
    ncb_send();
    rbuff[0] = 0x20;

    ncb_recv();
    capturetext (18,39,6,40,wbuff);
    drawbox(2,19,40,4,38,1);
    if(rbuff[0] == 'g') {
         switch(rbuff[10]) {

                case 0:  instructions();
                         user_response();
                         break;

                case 1: setcurpos(21,45);
                         printf("SERVER ENGAGED TRY LATER ");
                         setcurpos(22,45);
                         printf("Press any key...........");
                         while(!kbhit());getch();
                         pastescrn(18,39,6,40,wbuff);
                         return 0;
               default: printf("something wrong somewhere\n");
                         break;
         }
    }
}

/*********************************************************************
 *       instructions to the user before speaking
 */
```

```
instructions()
{
    setcurpos(19,56);
    bright();
    printf("DIAL");
    normal();
    setcurpos(21,41);
    printf("PLEASE RING UP THE SERVER NO: 4383");
    setcurpos(22,41);
    printf("PRESS");
    reverse();
    printf("  F1  ");
    normal();
    printf("TO SPEAK ");
    reverse();
    printf("  ESC  ");
    normal();
    printf(" TO ABORT");
    curoff();
}

/*************** user response for above **************************/
user_response()
{
    int res,test = 1;
    while(test) {
        res = getch();
        if(res == 0) res = getch();
        switch(res)
        {

            case 59: test = 0;                /*  F1 PRESSED  */
                     start_voice_key();
                     break;
            case 27: test = 0;                /*  ESC PRESSED */
                     hang_up();
                     break;
            default: setcurpos(24,0);
                     printf("PLEASE TRY AGAIN");
                     break;
        }
    }
}

/*******************************************************************
*       when the connection is established with the server the user
*       starts speaking after pressing key F1. this function handles
*       the response to above. After voice composing a sub menu is
*       displayed for replaying what was stored.
*/

start_voice_key()
{
    int res,choice,test;

    sbuff[0] = 'H';
    sncb.length = 10;
    ncb_send();
    rbuff[0] = 0x20;
    ncb_recv();
```

```
drawbox(2,19,40,4,38,1);
setcurpos(21,43);
if(rbuff[0] == 'h') {
  switch(rbuff[10]) {

     case 0: printf("   PRESS");
             reverse();
             printf("  F5  ");
             normal();
             printf("TO STOP SPEAKING");
             curoff();
             do {
                res = ((int)getch());
              } while (res != 63);

             sbuff[0] = 'P';
             sncb.length = 10;
             ncb_send();
             rbuff[0] = 0x20;
             ncb_recv();
             drawbox(2,19,40,4,38,1);
             setcurpos(21,43);
             if(rbuff[0] == 'p') {
               switch(rbuff[10]) {

                  case  0: printf("      COMPOSING SUCCESSFUL");
                           append_vfile ();
                           setcurpos(3,0);
                           list();
                           setcurpos(22,43);
                           printf("      Press any key.......");
                           while(!kbhit());getch();

                           test = 1;
                           while(test) {
                           switch(choice = sub_menu()) {

                                case 1: playback_voice_req();
                                        break;

                                case 2: test = 0;
                                        break;
                           }
                           }
                           break;

                  case 1:  printf("      ");
                           printf("UNABLE TO CLOSE VOICE FILE");
                           setcurpos(22,43);
                           printf("      Press any key.......");
                           while(!kbhit());getch();
                           break;           /*return;*/
               }
             }
             break;

    case 1: printf("UNEXPECTED ERROR_BUFF PLEASE HANG UP");
             setcurpos(22,43);
             printf("Press any key ......");
             while(!kbhit());getch();
```

```
                break;
        }
      }
      curon();
}

/****************************************************************
*       If the server is found busy or the line is not available
*       the user can hang up by pressing ESC key.
*/

hang_up()
{

    sbuff[0] = 'V';
    sbuff[10] = voice_no;
    sncb.length = 15;
    ncb_send();
    voice_no--;
    rbuff[0] = 0x20;
    ncb_recv();
    drawbox(2,19,40,4,38,1);
    setcurpos(21,43);
    if(rbuff[0] == 'v') {
      switch(rbuff[10]) {

          case 0: printf("    THANKS FOR HANGING UP");
                  break;

          case 1:printf("    UNEXPECTED ERROR HANGING UP");
                  break;
      }
    }
    setcurpos(22,43);
    printf("    Press any key........");
    while(!kbhit());getch();
}

/****************** sub menu for playback / quit ****************/
sub_menu()
{
   char s[80];
   int c;

   drawbox(2,19,40,5,38,1);
   setcurpos(19,52);
   reverse();
   printf(" MENU SELECT ");
   normal();
   setcurpos(21,44);
   printf("1: PLAYBACK RECORDED VOICE ");
   setcurpos(22,44);
   printf("2: QUIT ");

   do {
       setcurpos(23,44);
       bright();
       printf("enter your choice here: ");
       normal();
       gets(s);
      c = atoi(s);
   } while (c < 0 || c > 3);
   clrline(24);
   return (c);
}

/*********************************************************************
*
*/

playback_voice_req()
{
    int ret;

    sbuff[0] = 'I';
    sncb.length = 10;
    ncb_send();
    rbuff[0] = 0x20;
    ncb_recv();
    drawbox(2,19,40,4,38,1);
    if(rbuff[0] == 'i') {
      switch(rbuff[10]) {

        case 0:  setcurpos(19,56);
                 bright();
                 printf("DIAL");
                 normal();
                 setcurpos(21,41);
                 printf("PLEASE RING UP THE SERVER NO: --4383");
                 setcurpos(22,41);
                 printf("PRESS");
                 reverse();
                 printf("  F10  ");
                 normal();
                 printf("TO LISTEN");
                 reverse();
                 printf("  ESC  ");
                 normal();
                 printf("TO ABORT");
                 curoff();
                 ret = play_back_response();
                 switch(ret)
                 {

                     case 1: start_play_back();
                             break;
                     case 0: quit_play_back();
                             break;
                 }
                 break;

       case 1: setcurpos(21,43);
                 printf("LINE NOT AVAILABLE PLEASE TRY LATER ");
                 setcurpos(22,43);
                 printf("Press any key.... ");
                 while(!kbhit()); getch();
                 break;
     }
   }
   curon();
```

```
}

play_back_response()
{
    int res,test = 1;
    while(test) {
        res = getch();
        if(res == 0) res = getch();
        switch(res)
        {

             case 68: return 1;              /* PRESS F10 */
             case 27: return 0;              /* PRESS ESC */
             default: setcurpos(24,0);
                      printf("PLEASE TRY AGAIN");
                      break;
        }
    }
}

start_play_back()
{
    char file_no[5];
    int voice_no;

    drawbox(2,19,40,4,38,1);
    setcurpos(21,43);
    printf("ENTER VOICE FILE NO:  ");
    gets(file_no);
    voice_no = atoi(file_no);
    sbuff[0] = 'U';
    sbuff[10] = voice_no;
    sncb.length = 15;
    ncb_send();
    rbuff[0] = 0x20;
    ncb_recv();
    drawbox(2,19,40,4,38,1);
    setcurpos(21,45);
    if(rbuff[0] == 'u') {
      switch(rbuff[10]) {

           case 0: printf("PLAYING  BACK  VOICE  FILE ");
                   break;

           case 1:printf("UNEXPECTED PLAY BACK ERROR ");
                   break;
      }
    }
    setcurpos(22,45);
    printf("Press any key when over..... ");
    while(!kbhit()); getch();
}

quit_play_back()
{

    sbuff[0] = 'W';
    sncb.length = 10;
    ncb_send();
    rbuff[0] = 0x20;
    ncb_recv();
    drawbox(2,19,40,4,38,1);
    setcurpos(21,43);
    if(rbuff[0] == 'w') {
      switch(rbuff[10]) {

          case 0: printf("    QUITTING PLAYBACK SESSION  ");
                  break;

          case 1: printf("    UNEXPECTED ERROR WHILE PLAYING BACK");
                  break;
      }
    }
    setcurpos(22,43);
    printf("    Press any key........");
    while(!kbhit());getch();
}

/*****************************************************************
*       This function is for deleting any voice file from the users
*       mailbox.
*/

del_voice_file()
{
    char file_no[5];
    int vf_no;

    drawbox(2,19,40,4,38,1);
    setcurpos(21,42);
    printf(" ENTER VOICE FILE NO:  < 1 to %d > ",voice_no);

    gets(file_no);
    vf_no = atoi(file_no);

    sbuff[0] = 'J';
    sbuff[10] = vf_no;
    sncb.length = 15;
    ncb_send();
    rbuff[0] = 0x20;
    ncb_recv();
    drawbox(2,19,40,4,38,1);
    setcurpos(21,50);
    if(rbuff[0] == 'j') {
      switch(rbuff[10]) {

          case 0: printf("   DELETION COMPLETE ");
                  del_vfnum (vf_no);
                  break;

          case 1:printf("    CAN'T DELETE FILE  ");
                  break;
      }
   }
   setcurpos(22 ,50);
   printf("    Press any key.... ");
   while(!kbhit()); getch();
}
```

```
/*******************************************************************
 * capturetext - capture text from screen from the cursor position
 *
 *       row1/col1     - position of first character read
 *       nrows/ncols   - window size
 *       buff          - buffer to store character read with attribute
 */

capturetext (row1, col1, nrows, ncols, buff)
int row1, col1, nrows, ncols;
unsigned int *buff;
{
    int i,j;
    union REGS r;

    for( i=row1; i<(nrows+row1); i++){
        for(j=col1; j<(ncols+col1); j++){
            r.h.ah = 0x0f; int86(0x10, &r, &r);
            r.h.ah = 02;
            r.h.dh = i;
            r.h.dl = j;
            int86(0x10, &r, &r);   /* set the cursor position */

            r.h.ah = 8;
            int86 (0x10, &r, &r);/* read the character & attribute */
            *buff = r.x.ax;
            buff++;
        }
    }
}

/*******************************************************************
*   pastescrn - write text with attribute on screen
*
*       row1/col1    - position of first character of window
*       nrows/ncols  - window soze
*       buff         - buffer from where data is displayed
*/

pastescrn (row1,col1,nrows,ncols,buff)
int row1, col1, nrows, ncols;
unsigned int *buff;
{
    int i,j;
    union REGS r;

    for( i=row1; i<(nrows+row1); i++){
        for(j=col1; j<(ncols+col1); j++){
            r.h.ah = 0x0f; int86(0x10, &r, &r);
            r.h.ah = 02;
            r.h.dh = i;
            r.h.dl = j;
            int86(0x10, &r, &r);    /* set cursor position */

            r.h.ah = 9;
            r.x.dx = *buff;
            buff++;
            r.h.al = r.h.dl;       /* character to be written */
            r.h.bl = r.h.dh;       /* attribute */
            r.x.cx = 1;    /* no of times the character is written */
            int86 (0x10, &r, &r);
        }
    }
}

/*********************************************************************
*
*       When the composing is over and the user exits the editor
*       the link list content is transferred on to a buffer which
*       is later sent to the server.
*/

save_buff()
{
    int i,len;
    struct line *info;

    i = 0;
    len = 0;

    info = start;
    while (info) {
        len = strlen(info->text);
        strcpy(&mbuff[i], info->text);
        i = i + len;
        mbuff[i] = '\n';
        i++;
        info = info->next;              /* Get next line */
    }
    if ( i > 0) mbuff[i++] = '\0';
    printf("%s",mbuff);
    return (i);
}

/****** cursor size management.. user interface. ****************/
cur_size(start,end)
int start,end;
{
    union REGS u;

    u.h.ah = 1;
    u.h.ch = start_line = start;
    u.h.cl = end_line = end;
    int86(0x10,&u,&u);
}

cur_norm()                          /* SET CURSOR TO NORMAL */
{
    if(get_crt() == 7) cur_size(11,12);
      else cur_size(6,7);
}

cur_block()                         /* SET CURSOR TO BLOCK SIZE */
{
    cur_size(0,end_line);
}

get_crt()
{
    union REGS u;
    u.h.ah = 15;
    int86(0x10,&u,&u);
    return((int)u.h.al);
}

/**** scans the node for ctrl-d char which is for END OF TEXT. ***/
scan_line(xyz)                              /* scan line for ctrl-d */
char xyz[120];                              /* ie for end of text   */
{
    int i;

    for(i = 0; i < 120; i++)
    {
        if(xyz[i] == 0x04) return 0;break;  /* if found return0 */
    }
    return 1;
}

/************** append_vfile () ***********************************
*
*       This function append the voice file number to the text file.
*
*/

append_vfile ()
{
    int in,linenum;
    char cmd;
    struct line *info;

        info =  malloc(sizeof(struct line));
        if(!info) {
            printf("\n\tout of memory ");
            exit(1);
        }
        linenum = last->num + 1;
        printf("%d : ",linenum);
        info->num = linenum;
        strcpy(info->text, "########## THIS IS A VOICE FILE NO:  ");
        sprintf(&info->text[38], "%d", voice_no);
        strcpy(&info->text[40], " ##########");
        start = dls_store(info);
    return linenum;
}

/*********************** del_vfnum(vfnum)*************************
*    This function remove the voice file number from the text file.
*/

del_vfnum(vfnum)
int vfnum;
{
    struct line *info;
    int ret,vf_no,linenum;
    char file_no[5],s[80];

    strcpy(s,"########## THIS IS A VOICE FILE NO:  ");
    sprintf(&s[38], "%d", vfnum);
    strcpy(&s[40]," ##########");
    info = start;
    while (info) {
        if ((ret = strncmp(info->text, s, 50)) == 0) {
            linenum = info->num;
            printf("%d: %s\n",info->num,info->text);
            if (start == info) {
                start = info->next;
                if (start) start->prior = NULL;
                else last = NULL;
            }
            else {
                info->prior->next = info->next;
                if (info != last)
                    info->next->prior = info->prior;
                else
                    last = info->prior;
            }
            free(info);                    /* Return memory to system */
            patchup(linenum + 1, -1);              /* dec line num */
            setcurpos(3,0);
            list();
            return (0);
        }
        info = info->next;                 /* Get next line */
    }
    return (-1);
}
```