An Electronic Bulletin Board for UNIX Based Systems

by

Gerald F. McManigal

B. S. Northern Illinois University, 1973

A Master's Report

submitted in partial fulfillment of the

requirements for the degree

Master of Science

Department of Computer Science

Kansas State University
Manhattan, Kansas

1986

Approved by:

Major Professor

LD
2668
.R4
1986
M35
c.2

A11202 663847

CONTENTS

1. Introduction..... 1
1.1 Overview..... 1
1.2 Literature Review..... 2
1.2.1 Definitions..... 3
1.2.2 Standards..... 4
1.2.3 PLATO..... 7
1.2.4 Netnews..... 10
1.2.5 PMS - Chicago Public Library 13
1.2.6 Independent Investors Forum..... 15
1.2.7 The Virtual Journal..... 16
1.2.8 The Yellow Brick Road (TYBR)..... 17
1.2.8.1 User Interface..... 18
1.2.8.2 File Organization..... 20

2. Requirements..... 20
2.1 Functional Capabilities..... 21
2.2 User Interface and Documentation..... 22
2.3 Maintenance Function..... 23

3. TYBR System Design..... 23
3.1 TYBR Design of Functional Capabilities..... 23
3.1.1 Entry Level..... 23
3.1.2 Interest Group Level..... 26
3.2 File Structure Design..... 34
3.3 Maintenance Function Design..... 37
3.4 Setting Up a New Interest Group..... 39
3.5 Summary of the Design..... 40

4. Conclusions..... 40
4.1 Future Enhancements..... 40
4.2 Value of TYBR..... 42
Bibliography..... 45
Appendix 1..... 46
Appendix 2..... 49

THIS BOOK CONTAINS NUMEROUS PAGES WITH DIAGRAMS THAT ARE CROOKED COMPARED TO THE REST OF THE INFORMATION ON THE PAGE.

THIS IS AS RECEIVED FROM CUSTOMER.

1. *Introduction*

1.1 *Overview*

The term bulletin board brings to mind a vision of a cork backed board, strewn with notes roughly divided into areas of special interest and mounted in a prominent location. In many ways electronic bulletin boards are similar to the board just mentioned. This paper is the basis for the construction and implementation of The Yellow Brick Road (TYBR) an electronic bulletin board. Users of TYBR have the ability to browse through the system reading messages left by others or posting new messages of their own, much the same as their pedestrian counterparts. The electronic bulletin board described in this paper has as its prominent location a UNIX based computer. It too is divided into areas of special interest, where like minded users express ideas and receive feedback. The goal of TYBR is to provide a basic message handling tool that will fulfill a variety of useful functions.

A computer based bulletin board system provides many more capabilities than one of their wall mounted ancestors. All of the organizational advantages normally associated with computer based systems over manual systems apply to electronic bulletin board applications. On TYBR, messages are automatically posted, time stamped and authors identified. Users of the system may even be notified when notes of interest have been added. One disadvantage of electronic bulletin boards is that they are very transitory, any telephone number list of these systems soon becomes out of date. This may be because many of

these systems are established by computer hobbiests who have no long term commitment to their system.

To show how existing message systems influenced the design of TYBR the literature review section presents five examples. Systems were chosen because they represented alternate solutions to basic message system design problems or because of an unusual application that could be implemented on an electronic bulletin board.

Much of the system comparison is devoted to the user interface and message file organization. Also the effect that terminal hardware has on these systems is considered. A study by PANKO [6], concerning message system standards, provided information that is the basis for many of the conclusions drawn with regard to the user interface.

This paper also presents a summary of TYBR in relation to the other systems. In the later sections of this paper the requirements and design issues are discussed. In the final section some future enhancements are suggested. A user scenario is contained in Appendix 1 and the source code for TYBR is found in Appendix 2.

1.2 *Literature Review*

Message handling systems include such systems as: on line conferencing systems, information retrieval services,

electronic editing and publishing systems as well as bulletin board systems. By adding or deleting one or more capabilities, one type of system may function as another. Because of these similarities a study of other message systems may be useful in developing a set of requirements for a new bulletin board system.

1.2.1 *Definitions* To avoid confusion and to provide a more accurate description of the various bulletin board systems the following definitions are given:

Article --- An article is the term used by the Netnews bulletin board system to describe user messages [13].

Base message/note --- a base message or a base note is a message that is not a direct response to another message. The term base note is used by PLATO's bulletin board system [8].

Follow up --- A follow up is a message posted in response to another message [13].

Menu --- a menu is a list of available user commands and a short description of each command, usually in tabular form.

Message --- A message is simply a note ,typically text, left on the bulletin board system by a user.

News group --- News group is the term used by Netnews for special interest groups [13].

Notice --- A notice is a message posted on a BBS, usually by the system operator, for which no response is possible.

Notices are typically printed on entry to the BBS or an interest group.

Post --- To post a message is the act of leaving a message or a notice on a bulletin board system. This action may also be referred to as entering or writing a message.

Prompt --- a Prompt is a character or string of characters output to indicate that the system is waiting for user input. In many cases prompts contain hints as to valid input.

Response --- A response is a note left by a user concerning the message just read. This is a result of reading a message and then exercising the command to enter a response.

Scan -- To scan messages is to exercise a command that outputs only the header information for each message.

Special interest group --- The term special interest group refers to a group of BBS users who have access to the same topic within the BBS.

Topic --- A topic is the subject or area of interest for a special interest group. The terms topic and special interest group are sometimes used interchangeably.

1.2.2 *Standards*

There are three major areas of standardization within computer aided message systems [6]: 1) message structure, 2) delivery and 3) user interface. Because of the local nature of the messages in TYBR, the user interface will have the greatest

influence on the system design.

Studies by Panko [6] show that two factors have the greatest impact on the user interface. These factors are the kind of equipment used in the system's implementation and the level of sophistication of the intended users.

The equipment used to access a message system has the most significant impact on the form of the user interface. High speed terminals with special graphics capabilities favor a verbose user interface, one with many menus and user prompts. Quick responses, wordy prompts, and short user input characterize systems designed for high speed terminals. Low speed or hard copy terminals show a tendency to have short prompts and long user commands with many options. This reduces the system user interface and speeds up the functioning of the system. Terminals with multiple "windows" or "touch sensitive" screens open up other options to message system designers. Because of the different types of terminal hardware in use, standards in the area of user interface form, ie., menu vs. commands, may not be possible.

Even the simplest bulletin board systems have to account for the vast differences in the types of terminals in use today. Virtual terminal protocols [14] attempt to hide these differences from a functioning system. The virtual terminal protocol for UNIX [12] uses a data structure to describe basic terminal input control, output treatment, hardware control, line discipline and special control characters. Because TYBR

is based on UNIX and its initial design uses only simple text files, any terminal suitable for UNIX will be adequate; however TYBR will look its best on a CRT with at least a 1200 baud modem.

The level of user sophistication has as much influence on the content of the user interface as the equipment has on the form of the commands and prompts. Panko [6] described three levels of users:

1. sophisticated - demanding, use facilities frequently
2. simple - use only a small subset of commands
3. indirect - delegate system use to others

Bulletin boards are in the domain of those who enjoy computers, and because most users are sophisticated these systems tend to grow more complex over time. With each new enhancement added to the system comes a new optional command or menu. Of the systems studied, the most complex and powerful were those systems that were in existence the longest.

Functional capabilities is in the area where user interface standards are most likely to exist. A message system must be able to perform basic message handling services such as: retrieving messages from a data base, adding new messages to a data base, and scanning the contents of a message system. All of these functions are found on the simplest bulletin board systems.

In spite of their differences, de facto standards do exist for bulletin board systems. No matter what system one is on, a "?"

input will almost certainly return additional information concerning the options available. In most systems, command options are displayed as menus and prompts are usually a string of characters representing appropriate command options. Publically accessible bulletin board systems have some introductory notices for those entering the system. Typically, there is a command for leaving messages to the system operator. These common elements help users move from one system to another and promote the spread of information.

A discussion of TYBR's user interface and how the standard requirements of this message system are met is contained in the requirements section of this paper.

1.2.3 *PLATO*

More than 20 years ago the University of Illinois Computer-based Education Research Library (CERL), began development of the instructional computing system PLATO [8] [11]. The system was designed to take advantage of the individualized learning and self paced attributes of computer aided education. Because of the system's age and world wide distribution, PLATO is the repository of an enormous amount of educational software.

Perhaps the most distinctive aspect of PLATO is its terminal and the uncommon capabilities made possible by a well defined and powerful user interface. The high resolution graphics, touch-sensitive screen and special function keys make the PLATO terminal an important part of the overall system design. The touch-sensitive screen opens up PLATO to the very young who

have not yet learned to read or type [11]. High resolution graphics brings a visual depth that helps maintain the interest of students and conveys concepts difficult or impossible to express otherwise.

As one might expect, PLATO's bulletin board is well organized and massive. It has over ten thousand interest groups (notesfiles) in operation [8]. These notesfiles have varying levels of activity ranging from active, requiring daily "trimming" to maintain a manageable number of messages, to inactive where messages may be several years old. Within PLATO's bulletin board a strict three level hierarchy is evident, with elements of notesfiles, base notes and responses. The organization scheme of linking responses to base messages seems simple and natural when compared with the message to response relationship used by various other systems.

On entering a notesfile, a directory is displayed showing message numbers and descriptive titles as well as the number of responses each note has accumulated. To read a note, one enters the note's number when prompted by "What Note?". To read a response the user simply pushes a special function key and each time the key is pushed the next response is shown. Responses are linked to base notes only and may not be added in direct response to another response. Notesfiles can be thought of as a chronologically ordered list of linked lists, whereby each base note within the notesfile is the root node on it's own linked list of responses (see Figure 1).

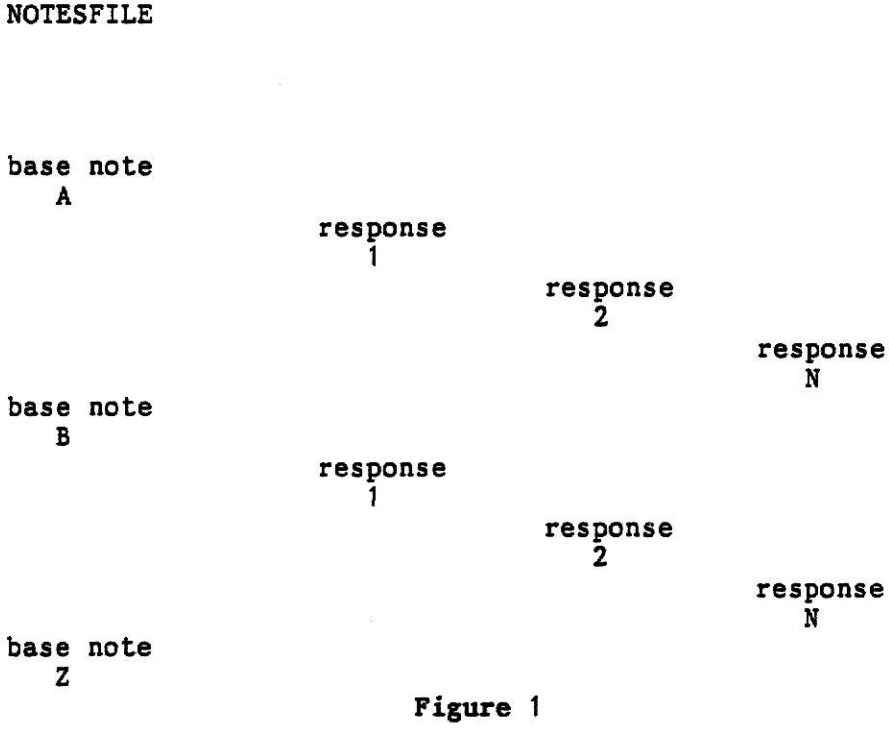

Figure 1

Though the systems differ in many, ways both PLATO and TYBR share a similar organizational structure for their interest groups, messages and responses. In TYBR, responses are added at the time messages are being read and they are associated via a linked list to the base message. Unlike PLATO, TYBR does not have a directory of messages in an interest group but it does provide several ways of scanning message headers to aid the users in choosing messages to read. PLATO's notes and responses are limited to one page in length and each note may accumulate a maximum of 99 responses. On TYBR there is no practical limit to a message's length or to the number of responses it may have .

1.2.4 *Netnews*

Netnews is a UNIX based bulletin board in use at many locations [13]. Each of the systems is linked through the UNIX mail so all users may contribute information. This bulletin board system has over 200 interest groups (news groups) listed at the location studied, however, not all the news groups are necessarily available to the network of other UNIX systems. The aforementioned allows varying levels of privacy for news groups. Netnews' diversity and distribution makes it a lively and informative bulletin board.

The ability to scan or peruse the articles on Netnews is provided by Readnews. Most commands relating to Readnews are displayed in a single menu that is printed after entering a "?". Commands consist of a single letter followed by a carriage return. In some cases, commands may be grouped together or joined with optional parameters. Figure 2 below shows the Readnews menu.

Command	Meaning
y	Yes. (Or just hit return.) Prints this article and goes on.
n	No. Goes on to next article without printing current one.
d	Digest. Breaks a digest article up into separate articles.
q	Quit. Update .newsrc if -l or -x not used.
U	Unsubscribe. You won't be shown this newsgroup any more.
c	Cancel an article you posted.
r	Reply. Reply to article's author via mail.
f [title]	Submit a follow up article.
N [newsgroup]	Go to next newsgroup or named newsgroup.
s [file]	Save. Article is appended to file (default is "Articles").
s |program	Run program with article as standard input.
e	Erase. Forget that an article was read.
h	Print verbose header. Use H for extremely verbose header.
!	Shell escape.
<number>	Go to message #<number> in this newsgroup.
-	Go back to last article.
b	Back up one article in the current group.
K	Mark the rest of the articles in current group as read.
x	Exit. Don't update .newsrc.
v	Version. Print current news version number.

c, f, r, e, h, and s can be followed by -'s to refer to the previous article.

Figure 2

All messages on Netnews are organized into news groups. Within each newsgroup there is a single list of messages stored in separate UNIX files under a directory named for the newsgroup. The message file names are simply the message number. For example, message number 1066 in the "general" newsgroup is a file named 1066, and it is stored in the subdirectory general.

Responses to messages in Netnews take two forms: direct reply to the author via mail or open follow up messages posted to the news group. When users choose the follow up option they are put into an editor with a copy of the original article. Each

line of the text from the original article is prefixed with the ">" symbol and is thereby differentiated from the text entered by the reader as a follow up. When the reader is finished with the follow up article, he or she exits the editor, and the follow up message is posted. This method of responding to messages can be very confusing to new users. Because multiple copies of messages, some heavily altered and commented, may be scattered through the news group, users find themselves reading and rereading the same information several times.

To enter an article into Netnews, the Postnews command is provided. The Postnews program prompts the user for relevant header and distribution information. Then, it invokes the editor; exiting the editor causes the article to be posted.

"Newsgroups" is the command that prints a listing of the available areas of interest and a short description of each.

Even though Netnews and TYBR are based on the UNIX operating system and have access to the same utilities, they do not make use of these utilities in the same way. TYBR uses the UNIX mail system to notify users that a new message has been added to the interest group to which they subscribe. Netnews on the other hand uses mail to allow readers to privately respond to the author of a message. In Netnews commands Readnews, Postnews and Newsgroups are all executed from the Shell level of UNIX. This makes the Netnews system seem disjoint, like an ad hoc collection of useful commands.

1.2.5 *PMS - Chicago Public Library*

The People's Message System (PMS) is a home computer based bulletin board system in use by the Chicago Public Library, it was written by Bill Blue, a pioneer in the development of this type of bulletin board. A 48k Apple II+ computer, Hays Micromodem II and two disk drives fulfill this bulletin board's hardware requirements. The system was written in Apple Soft Basic with some machine language routines where speed was necessary to improve the system performance. Chicago Public Library's bulletin board was chosen for study because it is typical of many home computer based bulletin board systems. Also, because its affiliation with a civic institution enhanced its likelihood of staying in operation long enough to study.

Small home computer based bulletin boards are generally available to the public and as such require a simple user interface. PMS is no exception, it has one command level with a single list of 1 or 2 letter commands. To aid users there are two levels of help available. First, by entering a "?" a menu of commands with single phrase descriptions is printed. If more help is needed, a user may type H followed by a command, and the system will print a verbose description of the command along with some examples of how that command is used. As one would expect of a public bulletin board, all the functions of PMS are fully documented within the system, so that no previous experience or users manuals are needed. The following is the menu of commands available on this bulletin board.

```
---->> System Commands

B  = Bulletins - repeat sign-on tips
E  = Enter a message into system.
F  = Features, articles, excerpts.
G  = Goodbye. Leave system.
H  = Help with various functions.
I  = Info about our system hardware etc
K  = Kill a message from the files.
M  = Message alert. Messages for you?
N  = News -- System news, rules, tips.
Q  = Quickscan - public message headers
QP = Quickscan of all message headers.
R  = Retrieve a message from the files.
S  = Scan of public message headers
SP = Scan of all message headers
SR = Selective message retrieval.
T  = Time, date and connect time.
U  = User modifiable system functions.
X  = eXpert user mode. (on/off toggle)
Z  = Continue message entry after abort
?  = Repeats this list of commands.
*  = Retrieves messages flagged in scan
TEST  = Modem continuous test loop.
NEWCALL  = Information for new callers.
```

figure 3

There are two types of messages available on PMS, feature articles and user provided messages. Feature articles are writings on various subjects and are posted on PMS by the system operator. These articles may also be small programs that users may down load to their own home computers. The feature articles are separate from the user provided messages and may be accessed through the "F" option.

There are no interest-group like divisions for user messages. There is only one collection of user messages and it is accessible from the main menu. All user messages are alike, there is no special provision for adding responses to messages read. The system provides a variety of ways to scan message headers so that users do not have to read all the messages on

the system. The advantage to this very simple message system is that it is easy for first time users to access a message system and enjoy using their home computer to communicate with others.

The elements of PMS that are also found in TYBR are those common to most bulletin board systems. Simple menus with online documentation are fundamental requirements for any bulletin board system. The examples of prompts, menus and introductory notices provided by PMS served as a starting point in the design of TYBR

1.2.6 *Independent Investors Forum*

The Independent Investors Forum (IIF) was the first commercial application of computer conferencing techniques to news letter publishing [7] The intent of IIF is to provide a small investment news letter where the subscriber has input to the editorial process. This interaction provides the research staff direction for their work and fulfills the needs of the subscribers in a timely manner.

The Independent Investors Forum is based on the ConferII conferencing system on the Michigan Merit network and is accessible via GTE's Telenet network or leased outside lines.

Many of the functions of ConferII are reminiscent of those on bulletin boards studied. IIF's agenda organization function is essentially the act of creating special interest groups. IIF offers it's readers several options in terms of message handling. These options include the ability to skip, scan or

read articles, summaries of articles are also provided. Through the response mechanism the reader may ask questions of the researchers in open response messages or by private mail. All these capabilities have their functional equivalents on many bulletin boards.

There are also functions not normally found on small bulletin board systems that are useful for this application. The voting mechanism of ConferII conferencing systems may be useful in joint investment decisions for a model portfolio. An up-loading option is available for posting messages on the IIF. This option allows subscribers to prepare messages on their own home computer, at much lower cost, before signing on to the system. Still these capabilities are not out of the range of possibility for small bulletin board systems.

1.2.7 *The Virtual Journal*

The Virtual Journal suggested by Roistacher [4] is the most formal of the applications discussed. It's function is to provide rapid review and distribution of papers to the scientific community.

This proposed Virtual Journal is of interest to the designer of bulletin boards due to some of the features included. The best way to describe these features is by following the flow of a document through the system. First an author submits a file containing the article to the editor. Then, the editor chooses reviewers and makes the article available to them. After reviewing the article a point score is assigned thus

establishing the articles level of prestige. Authors may withdraw articles if it was felt they had an insufficient score; thus the system avoids the author rejection process. Articles are published by making them available to the other Journal subscribers, adding the title to the table of contents and sending the abstract to relevant abstracting services. Once the article is available counts are kept on the number of times it is read. Readers may assign point scores to the articles, this provides valuable feedback to the editor, reviewers and authors.

Several features not normally found on bulletin board systems are important to the Virtual Journal's usefulness. First is the ability to assign a different subset of users to review each article. Reviewer and reader scoring mechanisms are important elements in establishing the quality of the articles in the journal. The table of contents appears to be a new twist to the fast scan feature seen on some bulletin boards.

The Virtual Journal serves as an example of how customizing features to a specific application can improve the usefulness of a basic message handling system.

1.2.8 *The Yellow Brick Road* (*TYBR*)

Since TYBR is a new system and is being introduced in this paper, it has no history to describe; still, it has many features in common with the bulletin boards just discussed. This section will describe TYBR's user interface and file structure in relation to the other systems studied.

1.2.8.1 *User Interface*

In Netnews and the Chicago Public Library's PMS, all of the available commands are shown in a single menu, see Figures 2 and 3. While this organization is simple and straight forward, the number of options available may confuse the first time user. In an effort to simplify TYBR for new users, system commands and message related commands are separated into two menus. Also, the two level command structure tends to fit well into the message file organization used by TYBR. At the systems entry level, commands relate to TYBR as a whole system or act as an interface to the system operator. The commands at this level provide the capabilities necessary to join, list, enter and quit interest groups. Figure 4 shows TYBR's system level menu.

```
Enter the letter corresponding to the option desired

    d = Display System Entry Notice
    l = List Bulletin Board topics
    x = Exit the Bulletin Board System
    a = Request to be added to an interest group
    r = Request to be removed from an interest group
    n = Request to establish a new interest group
    e = Enter an interest group Bulletin Board
    ? = Display this menu
```

Figure 4

In TYBR, when a user *enters* a selected interest group, the second command level comes into play. Some of the commands found here were influenced by PMS (i.e., the scan and fast scan of headers). At this level, commands are mainly concerned with message access. Commands for reading messages and entering

messages have small menus of their own. For reading messages the options include:

= Start reading with the message number input.

f = Start reading with the first message for this topic.

n = Start reading with the message following the last message read.

When a message has been read, the user is given the option to respond to that message or to read others' responses to that message. For entering messages into the system, the options are associated with a choice of input method. The choices are: the "vi" editor, the "ed" editor or input from a file. The command options for the interest group level are shown in Figure 5 below.

```
Enter the letter corresponding to the option desired

    k  = Kill a message or response (author only)
    e  = Enter a message
    s  = Scan of message header
    sr = Scan of messages and response headers
    f  = Fast scan of message headers
    fr = Fast scan of messages and response headers
    r  = Read messages and or responses
    rm = Read marked messages from 's' or 'sr'
    m  = List members of this interest group
    q  = Exit - leave this interest group
    x  = Exit the Bulletin Board System
    ?  = Display this menu
```

Figure 5

1.2.8.2 *File Organization*

TYBR's file structure is organized in typical UNIX hierarchical fashion. At the top or root node is a directory containing all data files needed for system operation and the subdirectories for each interest group. At the next level, within the interest group subdirectory, are all the data files needed to access the pertinent messages and check the users' accessibility to the group. The messages for an interest group are all stored in a single direct access file. Those interested in details of how messages are stored and accessed should refer to the system design, section three, of this paper.

From a user's viewpoint, the file structure used by TYBR for its messages is similar to that used by PLATO (see Figure 1). Unlike Netnews, TYBR's responses are associated directly with the base message through a linked list. Each new response to a base message is added to the end of the doubly linked list of responses for that message. If a base note is deleted, then the responses are also deleted. The intention is that a response message be related closely to its base message, and it is assumed that responses would serve no purpose alone.

2. *Requirements*

For TYBR, there are three areas of requirements: functional capabilities, user interface and maintenance. Though not one of the stated requirements, efficiency of storage utilization and processing speed should be considered where significant

gains can be made without extraordinary effort.

2.1 *Functional Capabilities*

TYBR's most basic requirement is to provide its users with a means of exchanging ideas. The functional capabilities described below must be developed to satisfy this requirement. This section lists these capabilities, and by grouping related functions together, it suggests a system architecture. The first collection of functional capabilities are related to the message system as a whole or involve communication with the system operator. As in most electronic bulletin board systems, TYBR's messages are divided into separate interest groups. This organization requires that users be able to list topics available and choose the interest group they wish to enter. Interest groups may be open to all or only to select users. This requirement is important to a number of possible nonbulletin board applications discussed in the last section. Because not all interest groups are open to everyone, users need a method of requesting that they be allowed to join or quit an interest group. A mechanism is also needed that allows users to submit a request for establishing a new interest group. On entering this bulletin board system, users are shown a system notice posted by the system operator. This notice is simply to announce the system to the user and to provide the system operator a chance to communicate to those using TYBR.

Within an interest group, functional capabilities are, in general, related to message handling. Required functions

include the following:

Read messages

Enter new messages

Delete old messages (author and system operator only)

Scan message headers

Read messages marked for retrieval during scan

Respond openly to messages read

Read responses to messages

Within each of the above functions other subfunctions may exist. These subfunctions are concerned with the implementation of their main function and will be discussed in the design section. Many other useful functions can be added, but only the aforementioned capabilities are required.

2.2 *User Interface and Documentation*

The user interface and documentation requirements of TYBR are closely related. The goal of these requirements is to allow the uninitiated user to enjoy the facilities of TYBR with a minimum of difficulty. To attain this goal, it is required that all user options be included in some menu and that the menu be displayed whenever a '?' is input. Menus contain a list of one or two letter commands followed by a brief explanation of the command. Commands must be functionally grouped so that options available at any level relate to that level.

2.3 *Maintenance Function*

For system maintenance, a 'clean up' program or procedure will be needed. Its primary function will be to remove old messages and recover file space. Additional maintenance functions should also be considered. These may include adding new interest groups and initializing its message file with an introductory message. Available UNIX commands can be used for much of the maintenance.

3. *TYBR System Design*

The design will be divided into three parts: functional capabilities, file structure and maintenance functions. Each of these areas will be described from a level that is abstract enough to avoid unimportant detail. Where details will help in understanding the system, diagrams will be provided.

3.1 *TYBR Design of Functional Capabilities*

The requirements described functional capabilities at two levels, the system entry level and the interest group level. This design section will further define the functions at each level. After a brief introduction to the design of the entry level capabilities, the remainder of the functional design will concentrate on the interest group level.

3.1.1 *Entry Level*

As stated in the requirements, functions at the entry level of TYBR are related to the system as a whole. The first function

encountered is the system introductory notice. This notice is printed simply by listing the contents of the file designated for that purpose. The remaining functions at this level are controlled by TYBR's main program. The main program prompts the user with a string of characters indicating available choices and requests the user to enter a command.

This prompt is:

```
d,l,a,r,n,e,x,?
ENTER COMMAND --->
```

The state diagram shown in Figure 6 describes actions taken for each of the available options.

State Diagram for TYBR's Entry Level

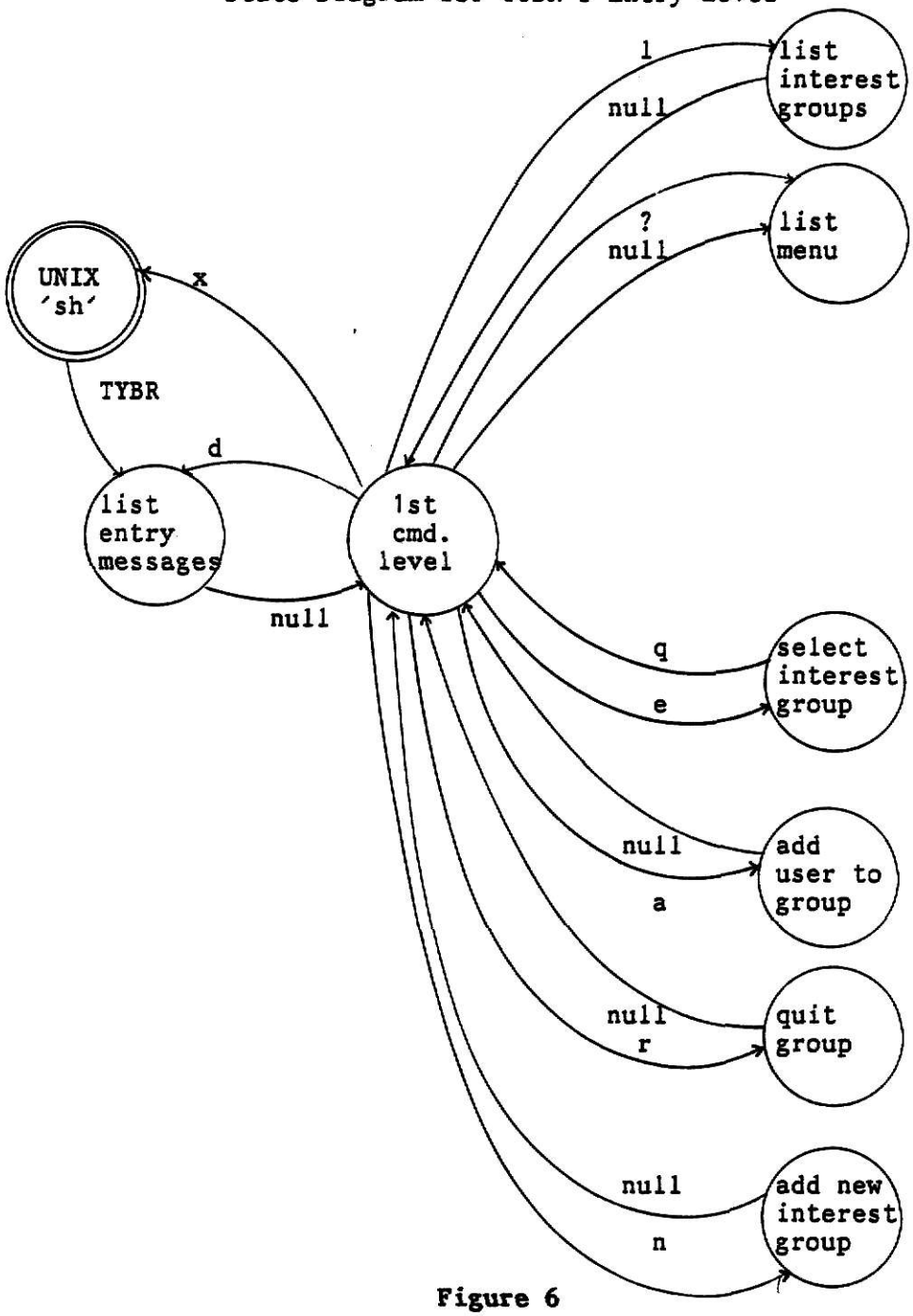

Figure 6

Each of the functions at this level are small, and all except the "e" and "x" options return to the "1st cmd." level (main program) when they complete.

To complete the design at this level a short description of each option is given. The function accessed through option "1" prints a list of available interest groups by executing a small function written for this purpose. As with almost all bulletin boards the "?" option prints the menu of available commands for the current level. To select and enter an interest group the "e" option must be exercised. The user is then given a choice of interest groups. After a selection is made, the user's login ID is checked against a list of those eligible to enter this interest group. If the user is allowed to enter this group, the second level of commands then becomes available. The "a", "r" and "n" options are provided to request that one be admitted to an interest group, that one be removed from an interest group or that a new interest group be established respectively. These funcukons *ma)lthe*user'*srequest*to*the*systemoperatorso that appropriate action may be taken.

3.1.2 *Interest Group Level*

At the interest group level of TYBR, the focus is on message handling. In order to fulfill TYBR's most basic requirement, that of providing a means of exchanging ideas, an entire message handling system must be developed. It is the intent of this section to describe the message handling routines and how these routines relate to one another.

After a user has entered the interest group, via the "e" command, a prompt much like the one shown at the entry level is displayed. The prompt, shown below, lists all the valid

options for this level.

```
k,e,s,sr,q,qr,r,rm,m,g,x,?
ENTER COMMAND--->
```

The menu associated with the prompt is shown in Figure 5. The state diagram in Figure 7 shows actions taken for each of the available commands.

State Diagram For Interest Group Main Program

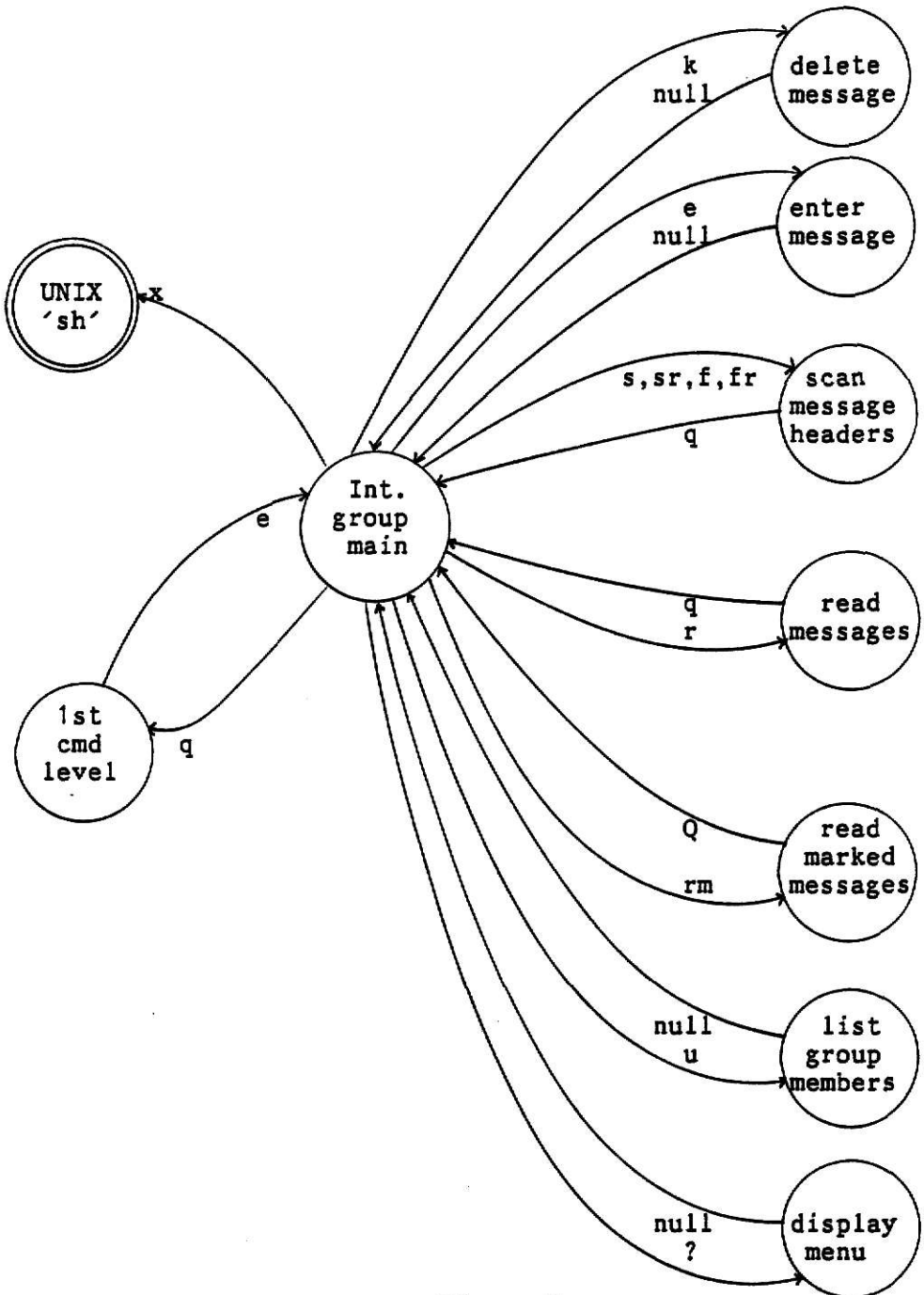

Figure 7

Unlike the functions found at the entry level, functions at this level are far more complicated. Some of these functions

even have different command levels of their own. The relationships between the functional modules of TYBR are shown in the structure diagram, Figure 8.

Structure Diagram for TYBR's Interest Group Level

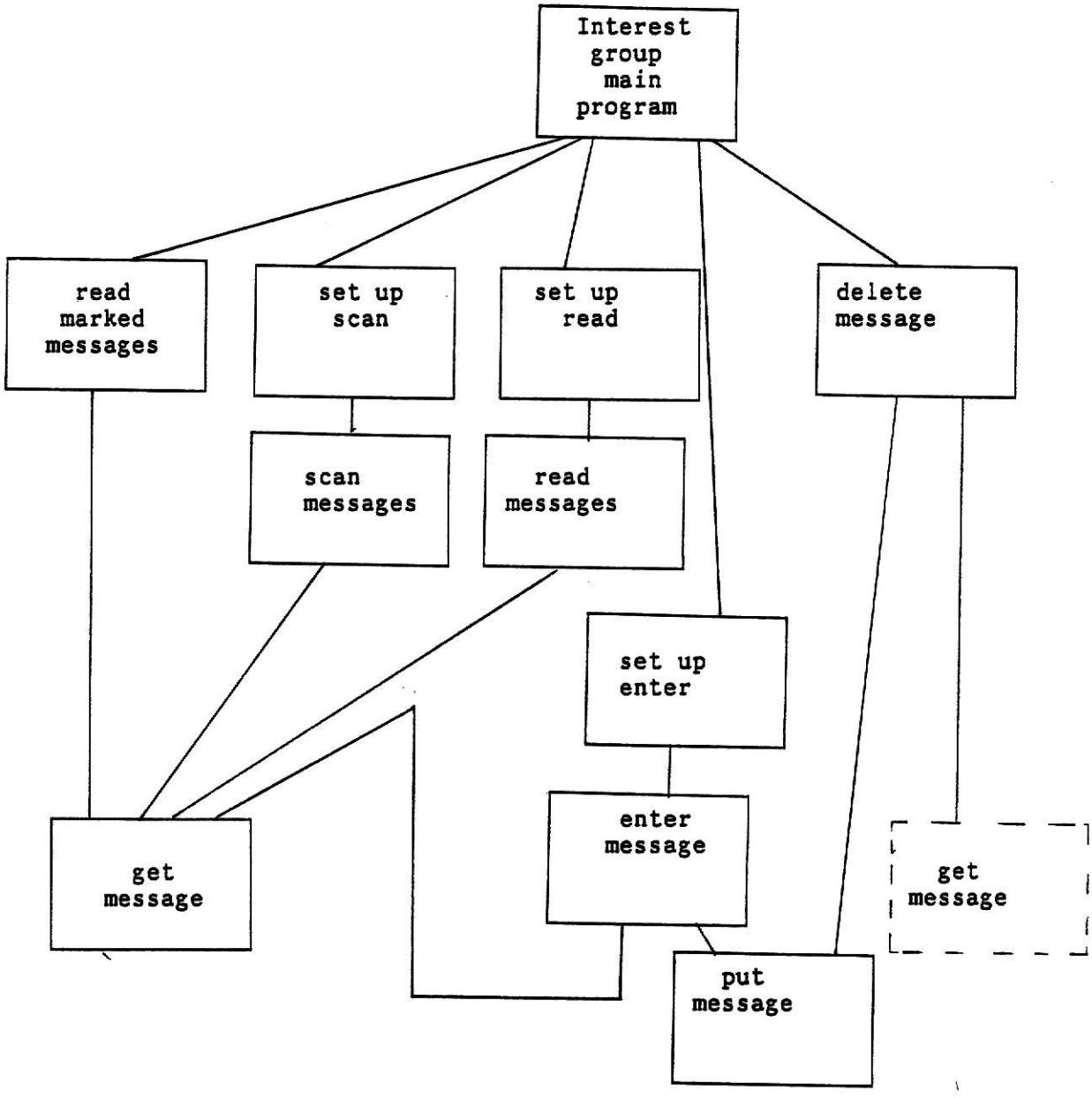

Figure 8

Interest group main program -- The interest group main program controls processing at the interest group level by providing prompts, interperting user commands and transferring control to the selected function.

Read marked messages-- During the execution of the scan command "s" or "sr", messages may be marked for later retrieval. The read-marked-messages module reads messages marked by prior scanning. The scan process builds a temporary file of message numbers which the read-marked-messages program then uses to determine what messages it will read.

Set up scan-- The set-up-scan module requests from the user a starting point for the scan. The scan may start from the first message in the interest group, the message following the last message read by the user or start with a given message number. After some minor validation effort, the starting point of the scan is passed to the scan-message-headings module.

Scan message headings-- The scan-message-headings module can print message headings for base messages only or for both base messages and their responses. For scan types "s" and "sr", a prompt is given after the heading of each message is printed. The prompt and commands are:

```
n,b,m,q,x,?
ENTER COMMAND-->

n = Print next header.
b = Print previous header.
m = Mark this message to be read later.
q = quit scan of headers.
x = Exit the bulletin board system.
? = print this menu.
```

This pause in scanning after each header gives the user an opportunity to mark messages for later recovery. The fast scans, command "f" and "fr", prints message headers nonstop, from the starting point indicated to the end of the message file. A fast scan is intended to produce a quick list of messages available on the interest group.

Set up read-- Like the set-up-scan module, the set-up-read module prompts the user for a starting point, then calls the read-message module.

Read messages-- The read-messages module prints messages stored on the interest group message file to the terminal. If a message is too large to fit on terminal a screen typically (23 lines by 72 columns), the message is shown in parts. These message segments represent the physical records stored on the interest group direct access file. If more than one segment is required the following prompt is printed after all but the last segment:

```
(MORE) "cr",q,x,? --->

"cr" = Carriage return will print more of the remaining text
       for this message.
  q  = Quit reading this message.
  x  = Exit the bulletin board system.
  ?  = Print this menu.
```

When no further segments remain to display, another prompt is displayed for the user:

```
n,b,r,N,R,q,x,?
ENTER COMMAND--->

n = Print next message.
    (Next base or response message in the linked list)
b = Print the previous message.
r = Print the response to the message if any.
N = Next - Quit reading responses and go to the next
    base message.
R = Respond to this message.
q = Quit reading messages.
? = Print this menu.
```

Set up enter-- Module set-up-enter can be used to build new base messages (if called from the interest group's main program) or to build a response message (if called from the read-message module). In either case this module presents the user with a choice of two editors, "vi" and "ed" for entering message information. The contents of a file may also be used to build a message.

No matter what choice is made, a temporary file containing the message title on the first line and the body of the message on the remaining lines is passed to the enter-message module. Also, if the message is a response, the number of the base message being responded to is passed as a parameter.

enter message-- Module enter-message must take the message

input and logically place the message in its proper location. This requires linking the new message into the existing structure. If the message is greater than some fixed size, it must be segmented into blocks that will fit the message file structure. Each of these blocks must then be written to the message file and linked to first block.

Delete message-- Module delete-message removes messages from the file structure by unlinking the message from the other messages in the system and setting the message number field to zero. In the case of a base message, all associated response messages are also deleted. No attempt is made at this time to recover any of the space made available by deleting messages. This function is left to a clean-up program.

Get message and put message-- TYBR's lowest level modules are the get-message and put-message functions. These modules retrieve messages from and store messages in the interest group message file. The put-message module can replace existing messages to the message file and thereby provide an updating function. The formula used to access messages in the message file is: (message number - first message number) * record length. This formula is used to calculate the offset in bytes from the beginning of the file prior to reading or writing of the desired message.

3.2 *File Structure Design*

The files needed to operate an interest group are: a user-list

file, an interest group information file, a message file and a user information file. The user-list file contains a complete list of users that are allowed access to the interest group. If this file does not exist, the interest group is available to all users of the UNIX system on which TYBR is located. The interest-group-information file contains the first and last message numbers available in the interest group. The user-information file has the list of interest groups a particular user subscribes to and the last message read by that user from each group. This file is stored in the user's "HOME" directory. The message file is the repository for all the messages relating to this interest group. For the message file the following criteria must be met:

1. Messages must be randomly accessible,

2. Messages have no practical limit to their length,

3. Response messages must be directly associated to a base message,

4. The message file needs to minimize UNIX system overhead associated with each message.

Random access is necessary because few restrictions are placed on TYBR users as to how they might read messages, and therefore, any message may be accessed at any time. TYBR has no restriction on the length of messages in it's system, and since UNIX direct access files are byte addressable, it is far more convenient to have fixed length records. Because fixed

length records are used, special accommodation must be made for messages longer than a certain length. The solution to this problem is to have a doubly linked list of messages whose sole purpose is to accommodate the additional text from a original message which is longer than the cutoff length.

The single message file scheme is employed by TYBR because it is assumed that most messages on the system are small and providing separate files for each message, as in Netnews, would include too much overhead. Each file on UNIX has an associated "inode" structure that contains all the information pertaining to the file. If TYBR was to use a separate file for every message and every response, the "inode" related overhead may become significant if the system were greatly used.

By associating a doubly linked list of responses messages with each base message, response messages can be directly associated with base messages. The message file with all its doubly linked lists can be best described using a diagram, as shown in Figure 9.

Message File Organization

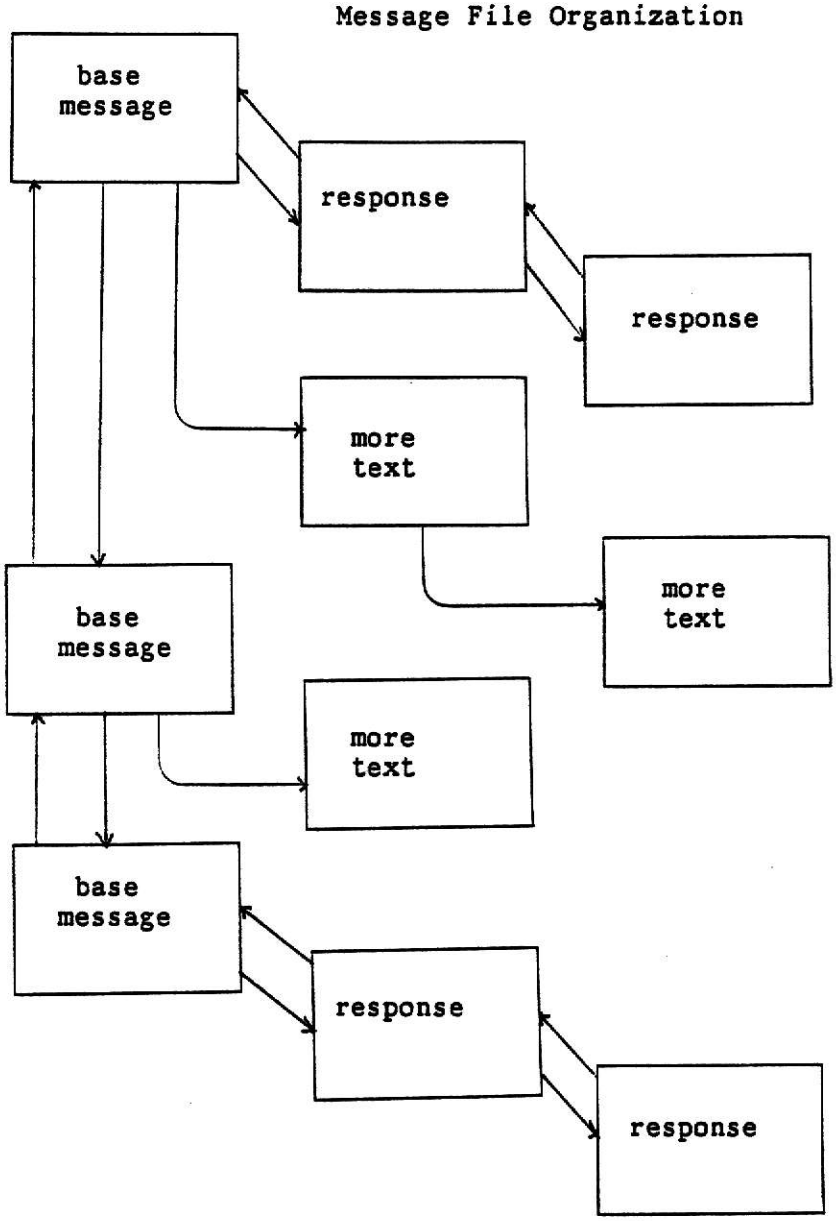

Figure 9

3.3 *Maintenance Function Design*

This section describes the utility programs that make it possible to carry on the day to day activities required of a bulletin board system operator. For the most part, existing

UNIX facilities provide the tools necessary to do many of TYBR's maintenance functions. For example, to add a user to a group or remove a user from a group existing editors can be used. More complex tasks of message file maintenance must be handled by a special purpose program. M_TYBR is a TYBR like maintenance program provided as a tool for the TYBR system operator. This program allows the system operator to enter all interest groups and delete messages. Two additional functional capabilities are available, they are message file compression and message file data dump. The function of compression is to recover space lost in the message file due to deleting old messages. This utility command copies messages that have not been deleted to a new message file and preserves the links to other messages. Messages are renumbered and the dates of the last compresses are updated in the topic information file. Since the messages are renumbered during a compress an equivalence table is kept so that users can have their last message read information updated. Compression of the message file should be done at intervals of once a month or less often so that user last message read data does not become out of date. If the last message read information does become out of date it will be reset to the first message in the interest group. The other function provided will print all the messages in the message file in the order that they are stored. This utility program is a necessity for program debugging because it shows all the links to other messages with in the file. Figure 10 shows one menu from the TYBR maintenance program.

```
c,d,k,e,s,sr,f,fr,r,rm,m,q,x,?
 YES MASTER -->?
 Enter the letter corresponding to the option desired


c  = Compress the interest group message file
d  = Dump all messages in order showing links
k  = Kill a message or response
e  = Enter a message
s  = Scan of message header
sr = Scan of messages and response headers
f  = Fast scan of message headers
fr = Fast scan of messages and response headers
r  = Read messages and or responses
rm = Read marked messages from 's' or 'sr'
m  = List members of this interest group
q  = Quit-Exit this interest group secession
x  = Exit the Bulletin Board System
?  = Display this menu
```

Figure 10

3.4 *Setting Up a New Interest Group*

Two small 'C' language programs must be modifyed and recompiled each time new interest groups are established. The program enter_topics.c moves the user into the directory for the topic selected. Any new topics must be added to case statement in this program . The other program needing modification when adding a new interest group is the program that lists the topics available. New topics must be added to those topics already being printed.

After the program modifications have been made and compiled, the new directory for the interest group must be made. By executing the utility program, first_msg, from within the new interest group directory and specifying the first message for the interest group the necessary files and data will be established. To restrict the interest group to a limited number of users, the system operator list the user IDs of those

who may use the interest group in the file user.list.

3.5 *Summary of the Design*

The overall design of TYBR emphasized many functional levels. By having many levels each relating to a specific function, the user is always provided with a manageable number of commands. Included with the list of commands for each level are commands that relate to all levels. At any level the "q" command moves the user to the previous command level, the "x" always allows the user to exit the system and the "?" will produce a menu of the valid commands for the current level. A user scenario, showing many of the command levels, is contained in Appendix 1. The design attempted to provide a comfortable atmosphere for the user while not abusing the host machine's resources.

4. *Conclusions*

4.1 *Future Enhancements*

Panko [6] stated that typically message systems "begin relatively simply" and grow more complex with the addition of each new function and expanded capability. Because TYBR is a new system, there are areas where capabilities can be expanded. The following extensions to TYBR have the potential of significantly improving the system.

Extending TYBR so that it can function on more than one machine is an enhancement that would improve the usefulness of the system by making it available to more users. The users of the different machines could then share ideas across a wider area.

Linking TYBR to USENET [13], the logical network on which Netnews is based, would provide the users of TYBR with a vast amount of information. Structural differences between Netnews and TYBR are significant, but not insurmountable. For example, the logical differences between follow-up messages on Netnews and response messages on TYBR could provide a possible sticking point. When moving response messages from TYBR to Netnews, one solution to the structural difference problem is to join the base message with each response message. To distinguish response messages from base messages each line of the base message should be prefixed with ">" so as to conform with the practice in use by Netnews. The modified response message could then be moved to Netnews as a follow-up article. All articles moving from Netnews to TYBR, follow-up and otherwise, could be handled as base messages.

For the interest group message file maintenance program, a garbage collection and data compaction enhancement may be useful. This may best be developed after some system usage patterns emerge. In the early years of TYBR's life any waste of space in the message file due to the deletion of response messages along with base messages is expected to be small and not worth recovering. As more interest groups become available and their use becomes more active, this space recovery may become more important.

Until the system has been in use for some time, it may be difficult to determine just what user interface or data base enhancements would be most beneficial. For this reason,

enhancements in these areas will not be suggested at this time.

4.2 *Value of TYBR*

As stated at the start of this paper, the goal of TYBR is to provide a basic message handling tool that will fulfill a variety of useful functions. It is the intention of this section to show examples of how TYBR may be used and thereby attain its goal.

One important element of TYBR is its ability to restrict membership in an interest group. This control over who uses a particular interest group is important because it aids in the enforcement of general rules of conduct for members of the interest group. With some simple rules in place, orderly discussions may be held.

At first, it is expected that topics for interest groups would relate to those subjects common to users of the system on which TYBR is located. These topics may be UNIX or NROFF discussion groups. They would provide members with information relating to the experiences of others and act as a place where questions could be posted with the expectation that others could respond. Students could even use the bulletin board to advertize the sale of their old books to the highest bidder.

Other more recreational interest groups could be added such as camping or skiing clubs. In these interest groups, schedules of up coming activities could be posted. Each activity, its description and date could be entered as a base message, then by reading the responses, the organizers of the activity would

be able to gauge member interest. Using this method the success of an activity could be determined prior to investing much time and effort in its planning.

The literature review section of this paper included two message systems because they had applications that could be applied, at least partly, to this bulletin board system. The Independent Investors Forum concept could be applied to interest group whose members wish to jointly manage a stock portfolio. A proposed investment strategy could be entered as a base message and reaction to that strategy would be in the form of response messages. A study of the messages and responses over time could lead to a winning strategy, or a tax write off.

The Virtual Journal suggested a method of reviewing and distributing scientific papers electronically. This application was envisioned as having a large data base and many reviewers and readers. This application cannot be done on TYBR as described, but it does suggest another application. Using TYBR for the review of documents might be handled in the following manner:

1) A new interest group would be established for the sole purpose of reviewing a single document,

2) Selected reviewers would be listed as the only members of this interest group,

3) Each chapter or page of the document to be reviewed would be divided in to separate base messages posted to the new interest group,

4) Reviewers would leave response messages describing needed changes, additions or deletions for each segment of the paper,

5) Steps three and four could be repeated as many times as necessary until the document was complete.

The completed document could then be posted to an interest group where all interested readers may leave their own remarks in response messages.

These suggestions and examples are provided to encourage others to make use of this bulletin board in a variety of ways. The function of any bulletin board is to share information and help solve problems. When Dorothy was faced with the problem of finding her way back to Kansas, she was told by the Good Witch of the North; "The road to the city of Emeralds is paved with yellow brick, so you cannot miss it. When you get to Oz do not be afraid of him, but tell your story and ask him to help you." [15] This bulletin board may not be the solution to anyone's problems but, it may be The Yellow Brick Road.

Bibliography

1. Uhlig, Farber, Bair, "The Office of The Future", North-Holland Publishing Company.

2. Pearson. M. M. L., Kulp, J. E., "Creating an Adaptive computerized conferencing System on UNIX". Computer Message Systems, Uhlig, R. P. (editor). North-Holland Publishing Company.

3. Lerch, I. A., "The Movable Conference", Byte Publications Inc., May 1984.

4. Roistacher, R. C., "The Virtual Journal", Computer Networks 2(1978) 18-24, North-Holland Publishing Company.

5. Lisanti, S., "The On-Line Search", Byte Publications Inc. December 1984.

6. Panko, R. R., "Standards for Computer-Based Message Services", National Bureau of Standards, NBS GCR 80-220.

7. Bechhoefr, A. S., "Electronic Publishing the New Newsletter", Byte Publications Inc. May, 1984.

8. Small, D., Small, S., "PLATO Rising Online Learning for Atarians", ANTIC, July, 1984 Copyright 1984, Control Data Corporation.

9. Kernighan, B. W., Ritchie, D., M., "The C Programming Language", Prentice-Hall, Inc 1978.

10. Myers, L. L., "How To Creat Your Own Computer Bulletin Board", TAB Books Inc. 1983

11. Rahmlow, H. F., Fratini, R. C., Ghesquiere, J. R., "PLATO", Educational Technology publications

12. Dealy, S. (EDITOR), "UNIX System Users Manual" AT&T Bell Laboratories

13. Horton, M. R., "How To Read The Network News" AT&T Bell Laboratories Document no. 59473-820317.01MF March 17, 1982

14. Tennenbaum, A. S., "Computer Networks" Prentice-Hall, Inc. Englewood Cliffs, New Jersey

15. Baum, L. Frank, "The Wizard of Oz"

Appendix 1

User Scenario

To reinforce the descriptions of TYBR and its functions a limited user scenario is presented. This example of user interaction with the system will demonstrate a few of the basic functions of TYBR. The user commands are demonstrated, in order:

1) Entering the bulletin board
2) Entering an interest group
3) Choosing an interest group to enter
4) Entering the read messages sub system
5) Starting to read with the first message
6) Going on to the next message
7) Responding to a base message
8) Choosing an editor
9) Quitting the read command level
10) Entering a new base message
11) Choosing an editor
12) Exiting the bulletin board

To maintain the continuity of the user scenario, the only annotation will be a number in parenthesis indicating which command from the above list is being exercised.

(1) $ tybr

WELCOME TO
THE YELLOW BRICK ROAD
(TYBR)

This is a new system, and we are open to suggestions. So just mail in your request, along with a broomstick, and I will see what I can do.

Send mail to mcmaniga on this machine.

```
    d,l,a,r,n,e,x,?

(2)  ENTER COMMAND --->e
    1) Geography
    2) Entertainment
    3) History
    4) Art & Literature
    5) Science & Nature
    6) Sports & Leisure
(3) ENTER THE NUMBER OF THE TOPIC YOU WISH TO ENTER --->1

    k,e,s,sr,f,fr,r,rm,m,q,x,?

(4) ENTER COMMAND --->r
    #,f,n,q,x,?
(5) ENTER COMMAND-->f

     message #1 author: mcmaniga   date: 6/20/85  Responses: 0
     title: McCalls patterns
     I need to know the exact location of McCalls patterns
     in Manhattan ka.
     Could anyone help with this local geography question?

     n,b,r,N,R,q,x,?
(6)  ENTER COMMAND-->n
     message #2 author: rich date: 6/21/85  Number of responses: 0
     title: Late summer vacations.
     I will be needing a restful vacation after this summer
     session.  Does anyone have information pertaining to
     John Pennekamp Corai Reef State Park in Florida?
     Is august just to hot?

     n,b,r,N,R,q,x,?
(7)  ENTER COMMAND-->R

     Warning: The first line of your message must be able to stand
     alone as the title of your message.
    e,v,f,?
(8) ENTER COMMAND--->e
    a
    Pennekamp is fine any time.
    August in Florida is hot, but not much worse than Kansas.
    Have a good dive!!
    .
    w
    65
    q
```

```
     n,b,r,N,R,q,x,?
(9)  ENTER COMMAND-->q

    k,e,s,sr,q,qr,r,rm,m,g,x,?

(10) ENTER COMMAND --->e

     Warning: The first line of your message must be able to stand
     alone as the title of your message.

    e,v,f,?
(11) ENTER COMMAND--->e
    a
    Manhattan to Chicago
    What is the quickest route between Manhattan Ka.
    and Chicago?
    I intend to drive there next week.
    .
    w
    82
    q

    k,e,s,sr,q,qr,r,rm,m,g,x,?

(12) ENTER COMMAND --->x
    $
```

For this sample session, outside of editor-related effort, only twelve single letter commands were required to enter an interest group, read two messages, respond to one message, enter a new base message and exit the system. No menus were shown during this example because most of them have been presented in other sections of this paper.

Appendix 2

TYBR SOURCE CODE

```
#include <errno.h>
#include <pwd.h>
#include <ctype.h>
#include <sys/timer.h>
#include <stdio.h>
#define BAD(MSG)\
    {\
    printf("get/put operation failed ret = %d0,bad);\
    printf("   message number was = %d0,MSG);\
    return('q');\
    }
#define LL 256
#define MM 256
#define NMAX 8  /* sizeof(utmp.ut_name)  */
#define TL 26   /* length of time field */
#define NGPS 6   /* Number of interest groups */
#define MSGL 1600
#include "bbs.global"
#include "enter.c"
#include "do_enter.c"
#include "del_msg.c"
#include "readmm.c"
#include "enter_topic.c"
#include "list_topics.c"
#include "read_msg.c"
#include "do_read.c"
```

```
#include "get_msg.c"
#include "put_msg.c"
#include "scan.c"
#include "quit_group.c"
#include "lock_box.c"
#include "unlock_box.c"
main () /* Main entry point for the Electronic Bulletin Board System */
 {
 int i,c;
 char cmd[LL];
 cmd[1] = ' ';
 while (cmd[1] != 'x' && c != EOF ) {
 system ("cat /m34/gfm/bin/bbs.intro"); /* list system entry message */
  cmd[1] = ' ';
  while (cmd[1] != 'd' && cmd[1] != 'x' && c != EOF) {
   printf ("d,l,a,r,");
   printf ("n,e,q,x,?0);
   printf (" ENTER COMMAND --->");
  for (i=1 ; i<LL && (c = getchar()) != '\n' && c != EOF ; ++i)
   cmd[i] = c;
    if (i <= 3 && c != EOF && cmd[1] != 'x' )  {
     switch(cmd[1]) { /* case statement for first command level */
      case 'l':
       list_topics();  /* option l list the topics available */
       break;
      case 'q' :
      cmd[1] = 'x';
      break;
```

```
      case '?':
       menu1(); /* option ? write the menu for level 1 */
       break;
      case 'e':
       cmd[1] = enter_topic(); /* option e enter a */
       break;
      case 'a':
       system("/m34/gfm/cpgm/join.group");
       /* option a request to be added to an interest group */
       break;
      case 'r':
       quit_group(); /* option r request to be
                        removed from an interest group */
       break;
       case 'n':
      system("/m34/gfm/cpgm/new.group");
      /* option n request to add a new group */
        break;
      } /* end case */
     } /* end if */
     } /* end level 1 while */
    } /* end entry while */
} /* end for main */
char topic() /* get info necessary to enter a topic */
  {
   char cmd[LL],uid[9];
   char *getenv(),*name,home[LL],*strcat();
   struct      passwd *pw;
```

```
struct      passwd *getpwuid();
int  i,c;
FILE *fptr,*fopen();
cmd[1] = ' ';
if((fptr = fopen("user.list","r"))!= NULL)
{
/* the user list file exists, so this interest group is closed */
pw = getpwuid(getuid());
fscanf(fptr, "%s", uid);
if (feof(fptr))
 {
 printf("There are no members listed for this interest group.\n");
 return(' ');
 }
 while (strcmp(pw->pw_name,uid,NMAX) != 0)
        {
        fscanf(fptr, "%s", uid);
        if (feof(fptr))
            {
            printf("Your user id was not on the list of
            those who are allowed to enter this interest group.\n");
            printf ("You may request that you be added to
            this interest group.\n");
            fclose(fptr);
            return(' ');
            }
         }
    fclose(fptr);
```

```
    }
else /* no user.list file */
    printf("Welcome to an open interest group.\n");
cmd[1] = ' '; /* initialize command to blank */
/* while (cmd[1] != 'q' && cmd[1] != 'x' && c != EOF)  */
while (c != EOF) {
printf ("\nk,e,s,sr,f,fr,r,rm,m,q,x,?\n"); /* print options 2 */
printf ("ENTER COMMAND --->");
 cmd[2] = ' ';  /* initialize second character to blank */
 for (i=1 ; i<LL && (c = getchar()) != '\n' && c != EOF ; ++i)
  cmd[i] = c;
if (i <= 3 && c != EOF )
    {
     /* if command was one or two characters long then */
  switch (cmd[1]) {
    case 'q':
    return (' ');
    break;
    case 'x':
    return ('x');
    break;
    case 'k':
    cmd[1] = del_msg(); /* kill a message or response */
        if (cmd[1] == 'x')
            return(cmd[1]);
        else
            cmd[1] = ' ';
    break;
```

```
case 'e':
cmd[1] = enter(0,0);
    if (cmd[1] == 'x')
        return(cmd[1]);
    else
        cmd[1] = ' ';
break;
case 's':
 if (cmd[2] == ' ')
     {
    cmd[1] = scan_hdr(1);
    if (cmd[1] == 'x')
        return(cmd[1]);
    else
        cmd[1] = ' ';
    }
 else
    if (cmd[2] == 'r')
      {
      cmd[1] = scan_hdr(2);
        if (cmd[1] == 'x')
            return(cmd[1]);
        else
            cmd[1] = ' ';
        }
break;
case 'f':
 if (cmd[2] == ' ')
```

```
    {
    cmd[1] = scan_hdr(3);
    if (cmd[1] == 'x')
        return(cmd[1]);
    else
        cmd[1] = ' ';
    }
 else
    if (cmd[2] == 'r')
        {
        cmd[1] = scan_hdr(4);
        if (cmd[1] == 'x')
            return(cmd[1]);
        else
            cmd[1] = ' ';
        }
break;
case 'r':
 if (cmd[2] == ' ')
   {
   cmd[1] = read_msg();
    if (cmd[1] == 'x')
        return(cmd[1]);
    else
        cmd[1] = ' ';
   }
 else
    if (cmd[2] == 'm')
```

```
            cmd[1] = readmm();
              name=getenv("HOME");
              strncpy(home,name,LL);
              strcat(home,"/mmf_bbs",LL);
            unlink(home);
              if (cmd[1] == 'x')
                  return(cmd[1]);
              else
                  cmd[1] = ' ';
      break;
      case 'm':
      if (cmd[2] == ' ')
        cmd[1] = member_list();
      break;
      case '?':
       menu2();
       break;
     }  /* end switch */
     }  /* end if */
    } /*end while */
    if (c = EOF)
      cmd[1] = 'x';
    return(cmd[1]);
   }  /* end topic */
/*************************************/
member_list()
{
  FILE *fptr,*fopen();
```

```
    if((fptr = fopen("user.list","r"))!= NULL)
      {
      system ("cat user.list");
      fclose(fptr);
      }
      else
      printf("There is no user list, this is an open interest group.\n");
  }
menu1() /* print the first level menu for the
        *Bulletin Board System */
{
   printf ("Enter the letter corresponding to the option desired\n");
   printf ("\n\n");
   printf ("d = Display system entry messages\n");
   printf ("l = List bulletin board topics\n");
   printf ("a = Request to be added to an interest group\n");
   printf ("r = Request to be removed from an interest group\n");
   printf ("n = Request to establish a new interest group\n");
   printf ("e = Enter an interest group bulletin board\n");
   printf ("x = Exit the bulletin board system\n");
   printf ("? = Display this menu\n");
}
menu2() /* print the options available when attending
        * a meeting/conference */
{
   printf ("Enter the letter corresponding to the option desired\n");
   printf ("\n\n");
   printf ("k  = Kill a message or response (author only)\n");
```

```
    printf ("e  = Enter a message\n");
    printf ("s  = Scan of message header\n");
    printf ("sr = Scan of messages and response headers\n");
    printf ("f  = Fast scan of message headers\n");
    printf ("fr = Fast scan of messages and response headers\n");
    printf ("r  = Read messages and or responses\n");
    printf ("rm = Read marked messages from 's' or 'sr'\n");
    printf ("m  = List members of this interest group\n");
    printf ("q  = Quit-Exit this interest group secession\n");
    printf ("x  = Exit the Bulletin Board System\n");
    printf ("?  = Display this menu\n");
}
```

```
enter_topic() /* enter a user interest group */
{
    int i,c;
    char cmd[LL],topic();
    cmd[1] = ' ';
    i == 1;
    while (cmd[1] != 'x' && c != EOF)  {
        list_topics();
        printf ("ENTER THE NUMBER OF THE TOPIC YOU WISH TO ENTER --->");
        for (i=1 ; i<LL && (c = getchar()) != '\n' && c != EOF ; ++i)
                cmd[i] = c;
        if (c != EOF )  {
                switch(cmd[1])  {
                case '1':
                        chdir("/m34/gfm/bin/deptnews");
                        break;
                case '2':
                        chdir("/m34/gfm/bin/isupbbs");
                        break;
                case '3':
                        chdir("/m34/gfm/bin/ngnbbs");
                        break;
                case '4':
                        chdir("/m34/gfm/bin/tybrbbs");
                        break;
                case '5':
                        chdir("/m34/gfm/bin/testbbs");
                        break;
```

```
            case 'x':
                    return('x');
                    break;
            case 'q':
                    return('q');
                    break;
            } /* end switch */
        if(cmd[1] < '6' && cmd[1] > '0')
        {
            cmd[1] = topic();
            return(cmd[1]);
        }
    } /* end if */
} /* end while */
if (c = EOF)
        cmd[1] = 'x';
return(cmd[1]);
} /* end enter_topics */
```

```
list_topics()
 {
 printf ("1) Dept 0345 / 0346 NEWS\n");
 printf ("2) ISUP Development\n");
 printf ("3) NGN Recovery INFO.\n");
 printf ("4) TYBR discussion group.\n");
}
```

```
del_msg() /*delete message  del_msg */
{
struct  passwd *pw;
struct  passwd *getpwuid();
struct message *mptr,msg,temp_msg;
struct topic_info t_dat;
unsigned int ret,valid,msg_no,min,max,
i,bad,s_msg_no;
int c;
FILE *fptr;
printf(" entering del_msg.c");
if((fptr=fopen("topicinfo","r"))==NULL)
    {
    printf("Could not open BBS topic info. file\n");
    return('q');
    }
fread(&t_dat,sizeof(t_dat),1,fptr);
if(feof(fptr))
    {
    printf(" end of file incountered in BBS topic data file.\n");
    return('q');
    }
fclose(fptr);
i = 0;
valid = 0;
while(!valid && i < 3)
    {
    printf("Enter message number to delete-->");
```

```
    scanf("%d",&msg_no);
    c = getchar();   /* remove the "cr" form the input queue */
    if (t_dat.first_msg_no <= msg_no <= t_dat.last_msg_no)
    valid = 1;
    else i += i;
    }  /* end while  */
mptr = &msg;
bad = get_msg(msg_no,mptr);
if(bad)
    {
    BAD(msg_no);
    }
else /* continue with delete message process */
    {
    pw = getpwuid(getuid());
     if (strcmp(pw->pw_name,msg.auth,NMAX) != 0)
        printf("Only the author may delete a message!\n");
    else /* continue with delete message process */
        {
        lock_box();  /* lock message file during update  */
        mptr = &temp_msg;
        if(msg.pre_msg_lk != 0)
            {
            bad = get_msg(msg.pre_msg_lk,mptr);
            if (bad)
                {
                BAD(msg.pre_msg_lk);
                }
```

```
    else /* update previous message in link list */
        {
        if(temp_msg.rsp_lk == msg_no)
            temp_msg.rsp_lk = msg.nxt_msg_lk;
            else
            temp_msg.nxt_msg_lk = msg.nxt_msg_lk;
        bad = put_msg(temp_msg.msg_no,mptr);
        if (bad)
            {
            BAD(temp_msg.msg_no);
            }
        } /* end of update previous message */
    } /*end if */
if(msg.res_to_lk != 0)
    {
    bad = get_msg(msg.res_to_lk,mptr);
    if (bad)
        {
        BAD(msg.pre_msg_lk);
        }
    else /* update response count in parent message*/
        {
            --temp_msg.q_res;
        bad = put_msg(temp_msg.msg_no,mptr);
        if (bad)
            {
            BAD(temp_msg.msg_no);
            }
```

```
    } /* end of update previous message */
} /*end if */
if (msg.nxt_msg_lk != 0)
    { /* reset link to nxt msg */
    mptr = &temp_msg;
    bad = get_msg(msg.nxt_msg_lk,mptr);
    if (bad)
        {
        BAD(msg.nxt_msg_lk);
        }
    else  /* update next message */
        {
        temp_msg.pre_msg_lk = msg.pre_msg_lk;
        bad = put_msg(temp_msg.msg_no,mptr);
        if (bad)
            {
            BAD(temp_msg.msg_no);
            }
        } /* end for update nxtmsg */
    } /* end if */
    if (msg.rsp_lk)
        {
        bad = get_msg(msg.rsp_lk,mptr);
        if(bad)
            {
            BAD(msg.rsp_lk);
            }
        else /* delete all responses to this
```

```
message  */
    {
    while(temp_msg.nxt_msg_lk)
        {
        s_msg_no = temp_msg.msg_no;
        temp_msg.msg_no = 0;
        temp_msg.txt_lk = 0;
        temp_msg.res_to_lk = 0;
        temp_msg.nxt_msg_lk = 0;
        temp_msg.pre_msg_lk = 0;
        temp_msg.rsp_lk = 0;
        temp_msg.q_reads = 0;
        temp_msg.q_res = 0;
        temp_msg.last_r = 0;
        temp_msg.group_id = 0;
         for (i=0; i< 11;i++)
        temp_msg.auth[i] = '\0';
         for (i=0; i< 26;i++)
        temp_msg.time[i] = '\0';
         for (i=0; i< LL;i++)
        temp_msg.header_ln[i] = '\0';
         for (i=0; i< MSGL;i++)
        temp_msg.msg_txt[i] = '\0';
        bad = put_msg(s_msg_no,mptr);
        if (bad)
            {
            BAD(s_msg_no);
            }
```

```
            bad = get_msg(temp_msg.nxt_msg_lk,mptr);
            if(! bad)
                {
                BAD(temp_msg.nxt_msg_lk);
                }
            } /* end while */
        } /* end else */
    } /* end if */
unlock_box();
if (msg.txt_lk) /* if more text messages */
    {
    bad = get_msg(msg.txt_lk,mptr);
    if (bad)
        {
        BAD(msg.txt_lk);
        }
    else
        {
        while(temp_msg.nxt_msg_lk)
            {
            s_msg_no = temp_msg.msg_no;
            temp_msg.msg_no = 0;
            temp_msg.txt_lk = 0;
            temp_msg.res_to_lk = 0;
            temp_msg.nxt_msg_lk = 0;
            temp_msg.pre_msg_lk = 0;
            temp_msg.rsp_lk = 0;
            temp_msg.q_reads = 0;
```

```
                temp_msg.q_res = 0;
                temp_msg.last_r = 0;
                temp_msg.group_id = 0;
                 for (i=0; i< 11;i++)
                temp_msg.auth[i] = '\0';
                 for (i=0; i< 26;i++)
                temp_msg.time[i] = '\0';
                 for (i=0; i< LL;i++)
                temp_msg.header_ln[i] = '\0';
                 for (i=0; i< MSGL;i++)
                temp_msg.msg_txt[i] = '\0';
                bad = put_msg(s_msg_no,mptr);
                if (bad)
                    {
                    BAD(s_msg_no);
                    }
                bad = get_msg(temp_msg.nxt_msg_lk,mptr);
                    {
                    BAD(temp_msg.nxt_msg_lk);
                    }
                } /* end while */
            } /* end else */
        }  /* end if  */
        s_msg_no = msg.msg_no;
        msg.msg_no = 0;
        msg.txt_lk = 0;
        msg.res_to_lk = 0;
        msg.nxt_msg_lk = 0;
```

```
                msg.pre_msg_lk = 0;
                msg.rsp_lk = 0;
                msg.q_reads = 0;
                msg.q_res = 0;
                msg.last_r = 0;
                msg.group_id = 0;
                for (i=0; i< 11;i++)
                 msg.auth[i] = '\0';
                for (i=0; i< 26;i++)
                 msg.time[i] = '\0';
                for (i=0; i< LL;i++)
                 msg.header_ln[i] = '\0';
                for (i=0; i< MSGL;i++)
                 msg.msg_txt[i] = '\0';
                bad = put_msg(s_msg_no,&msg);
                if (bad)
                    {
                    BAD(s_msg_no);
                    }
        } /*end else ( after uid check) */
} /* end else (after first msg read) */
} /* end del_msg */
```

```
enter(old_msg,rsp_to)
unsigned int old_msg,rsp_to;
{
unsigned int bad,save_msg_no;
struct message msg,*mptr;
int child,c,i;
char ch,cmd[LL],input_file[LL],fname[LL];
char *getenv(),*name,home[LL],*strcat();
FILE *fptr;
save_msg_no = 0;
cmd[1] = ' ';
c = 0;
for (i=0;i < LL ; i++)
    input_file[i] = '\0';
name=getenv("HOME");
strncpy(home,name,LL);
strcat(home,"/EDITF",LL);
  if(old_msg)  /* an old message for revision was input */
    {
    if((fptr = fopen(home,"w"))==NULL)
        {
        printf("Can't open temp edit file in enter_msg\n");
        return('x');
        }
        do /* loop to copy all text into file for edit */
            {
            bad = get_msg(old_msg,mptr);
            if (bad)
```

```
            {
            printf("get_msg failed in enter_msg ret=%d msg=%d\n",
                bad,old_msg);
            return('x');
            }
        for (i = 0; (ch = msg.msg_txt[i]) != '\0' && i < 1600 ;
            i++)
            fputc(ch,fptr);
        save_msg_no = msg.msg_no;
        old_msg = msg.txt_lk;
        }
        while(old_msg);
    fclose(fptr);
  }
if (!save_msg_no)
    {
    printf(" Warning: The first line of your message must");
    printf(" be able to\n stand alone as the title of your
            message\n");
    }
while(c!=EOF )
    {
    printf("\ne,v,f,?\n");
    printf("Enter Command--->");
    for (i=1; i < LL && (c = getchar()) != '\n' && c != EOF ; ++i)
        cmd[1] = c;
    if(i<=2 && c != EOF )
        {
```

```
switch(cmd[1])
    {
    case 'e':
    child = fork();
    if (child == 0)
        {
        setuid(getuid());
        name=getenv("HOME");
        chdir(name);
        system("ed EDITF");
        system("chmod 777 EDITF");
        exit(0);
        }
    wait();
    cmd[1] = do_enter(old_msg,rsp_to);
    return(' ');
    break;
    case 'v':
    child = fork();
    if (child == 0)
        {
        setuid(getuid());
        name=getenv("HOME");
        chdir(name);
        system("vi EDITF");
        system("chmod 777 EDITF");
        exit(0);
        }
```

```
wait();
cmd[1] = do_enter(old_msg,rsp_to);
return(' ');
break;
case 'f':
child = fork();
if (child == 0)
    {
fname[0] = 'c';
fname[1] = 'a';
fname[2] = 't';
fname[3] = ' ';
printf("From $HOME ENTER FILE NAME--->");
for(i = 4; (c=getchar()) != '\n' && c != EOF;
    i++)
    {
    fname[i]=c;
    input_file[i-4]=c;
    }
fname[i++]=' ';
fname[i++]='>';
fname[i++]='E';
fname[i++]='D';
fname[i++]='I';
fname[i++]='T';
fname[i++]='F';
fname[i++]='\0';
setuid(getuid());
```

```
            name=getenv("HOME");
            chdir(name);
            system(fname);
            exit(0);
                }
            wait();
            cmd[1] = do_enter(old_msg,rsp_to);
            return(' ');
            break;
            case 'q':
            return('?');
            break;
            case 'x':
            return('x');
            break;
            case '?':
            printf(" e  = ed editor\n");
            printf(" v  = vi editor\n");
            printf(" f  = Input file name\n");
            printf(" ?  = Display this menu\n");
            printf(" q  = Quit message entry/edit\n");
            printf(" x  = Exit the bulletin board system\n");
            break;
            } /* end switch */
        } /* end if */
    } /* end while */
} /* end enter */
```

```
do_enter(old_msg,res_to)
unsigned int old_msg,res_to;
{ /* enter the message in EDITF into the message file */
int child,c,i,j,crct;
char *getenv(),*name,home[LL],*strcat();
unsigned int bad,save_msg_no;
struct message msg,*mptr,new_msg;
struct topic_info t_dat;
struct passwd *pw;
struct passwd *getpwuid();
char *tdate,*ctime();
long *clock,time();
char ch;
FILE *fptr;
c = 0;
mptr = &msg;
lock_box();  /* lock out other users while entering a message */
if((fptr=fopen("topicinfo","r"))==NULL)
    {
    printf("Could not open BBS topic info. file\n");
    unlock_box();
    return('q');
    }
fread(&t_dat,sizeof(t_dat),1,fptr);
if(feof(fptr))
    {
    printf(" end of file incountered in BBS topic data file.\n");
    unlock_box();
```

```
    return('q');
    }
fclose(fptr);
name=getenv("HOME");
strncpy(home,name,LL);
strcat(home,"/EDITF",LL);
if((fptr=fopen(home,"r"))==NULL)
    {
    printf(" could not open for reading EDITF in enter\n");
    unlock_box();
    return('q');
    }
/* check EDITF for a good message prior to linking it in */
for (i=0;i<=LL && (c=fgetc(fptr)) !='\n' && c != EOF; ++i)
    msg.header_ln[i] = c;
msg.header_ln[i] = '\0';
if (c == EOF)
    {
    printf(" A message must contain more than one line\n");
    printf(" Try again -- first line is the message header\n");
    printf("              the second line to the end is text.\n");
    unlock_box();
    child = fork();
    if (child == 0)
        {
        setuid(getuid());
        name=getenv("HOME");
        chdir(name);
```

```
        system("rm EDITF");
        exit(0);
        }
    wait();
    return('q');
    }
fclose(fptr);
name=getenv("HOME");
strncpy(home,name,LL);
strcat(home,"/EDITF",LL);
if((fptr=fopen(home,"r"))==NULL)
    {
    printf(" could not open for reading EDITF in enter\n");
    unlock_box();
    return('q');
    }
        if (old_msg)  /* if old message was input reuse it */
            {
            bad = get_msg(old_msg,mptr);
            if (bad)
                {
                BAD(old_msg);
                }
            old_msg = 0; /* reset old_msg we don't need it any more */
            }
        else /* not old message */
            {
            if (!res_to)
```

```
{
bad = get_msg(t_dat.first_msg_no,mptr);
if (bad)
    {
    BAD(t_dat.last_msg_no);
    }
    while((msg.nxt_msg_lk) && (!bad))
        bad=get_msg(msg.nxt_msg_lk,mptr);
    if (bad)
        {
        BAD(msg.nxt_msg_lk);
        }
save_msg_no = msg.msg_no;
msg.nxt_msg_lk = t_dat.last_msg_no + 1;
/* get next available and add one */
t_dat.last_msg_no = msg.nxt_msg_lk;
bad = put_msg(msg.msg_no,mptr);
if (bad)
    {
    BAD(msg.msg_no);
    }
msg.msg_no = msg.nxt_msg_lk;
msg.nxt_msg_lk = 0;
msg.pre_msg_lk = save_msg_no;
msg.txt_lk = 0;
msg.res_to_lk = 0;
msg.rsp_lk = 0;
msg.q_reads = 0;
```

```
    msg.q_res = 0;
    time(&clock);
    tdate = ctime(&clock);
    strncpy(msg.time , tdate,26);
    pw = getpwuid(getuid());
    strncpy(msg.auth, pw ? pw->pw_name  : "?",NMAX);
    } /* end not a response */
else /* start is a response */
    {
    save_msg_no = t_dat.last_msg_no+1;
    /* get nxt msg no */
    bad = get_msg(res_to,mptr);
    if (bad)
        {
        BAD(res_to);
        }
    ++msg.q_res;
    if(msg.rsp_lk == 0)
        {
        msg.rsp_lk = save_msg_no;
        save_msg_no = msg.msg_no;
        bad = put_msg(res_to,mptr);
        if (bad)
            {
            BAD(res_to);
            }
        }
    else /* the res_to message has other responses */
```

```
    {
    bad = put_msg(res_to,mptr);
    if (bad)
        {
        BAD(res_to);
        }
    bad = get_msg(msg.rsp_lk,mptr);
    if (bad)
        {
        BAD(msg.rsp_lk);
        }
    while((msg.nxt_msg_lk) && (!bad))
        bad=get_msg(msg.nxt_msg_lk,mptr);
    if (bad)
        {
        BAD(msg.nxt_msg_lk);
        }
    save_msg_no = msg.msg_no;
    msg.nxt_msg_lk = t_dat.last_msg_no+1;
    /* get next available no and add one */
    bad = put_msg(msg.msg_no,mptr);
    if (bad)
        {
        BAD(msg.msg_no);
        }
    }  /* message has other responses   */
++t_dat.last_msg_no;
msg.msg_no = t_dat.last_msg_no;
```

```
            msg.nxt_msg_lk = 0;
            msg.pre_msg_lk = save_msg_no;
            msg.txt_lk = 0;
            msg.res_to_lk = res_to;
            msg.rsp_lk = 0;
            msg.q_reads = 0;
            msg.q_res = 0;
            time(&clock);
            tdate = ctime(&clock);
            strncpy(msg.time , tdate,26);
            pw = getpwuid(getuid());
            strncpy(msg.auth, pw ? pw->pw_name  : "?",NMAX);
                /* at this point we want to send mail to
                   the owner of this message     */
            }   /* end else is a response  */
        }  /* end else is not an old message */
 for (i=0;i<=LL && (c=fgetc(fptr)) !='\n' && c != EOF; ++i)
    msg.header_ln[i] = c;
 msg.header_ln[i] = '\0';
 /* c can not = EOF because that has been checked  */
while (c!= EOF)
    {
    crct = 0;  /* zero out the 'cr' counter */
    for (j=0; j < MSGL ; ++j)
        msg.msg_txt[j] = '\0';
    for (i=0; i<MSGL && crct <= 20 && (c = fgetc(fptr)) != EOF;
        ++i)
        {
```

```
msg.msg_txt[i] = c;
if ( msg.msg_txt[i] == '\n' )
    ++crct;
}
if (c == EOF)
    {
    msg.msg_txt[i] = '\n';
    msg.txt_lk = 0;
    bad=put_msg(msg.msg_no,mptr);
    if (bad)
        {
        BAD(msg.msg_no);
        }
    }
else  /* if c != EOF then more text to output */
    {
    if (msg.txt_lk)
        {
        old_msg = msg.txt_lk;
        bad = put_msg(msg.msg_no,mptr);
        if (bad)
            {
            printf(" put_msg failed in enter");
            printf(" ret=%d msg=%d old_msg done\n",
                bad,msg.msg_no);
            unlock_box();
            child = fork();
            if (child == 0)
```

```
                {
                setuid(getuid());
                name=getenv("HOME");
                chdir(name);
                system("rm EDITF");
                exit(0);
                }
            wait();
            return('q');
            }
        } /* end if txt_lk */
    else  /* no txt_lk but we need one now */
        {
        ++t_dat.last_msg_no;
        msg.txt_lk = t_dat.last_msg_no;
        old_msg = msg.txt_lk;
        bad = put_msg(msg.msg_no,mptr);
        if (bad)
            {
            printf(" put_msg failed in enter");
            printf(" ret=%d msg=%d no txt_lk but need one\n"
                ,bad,msg.msg_no);
            unlock_box();
            child = fork();
            if (child == 0)
                {
                setuid(getuid());
                name=getenv("HOME");
```

```
                    chdir(name);
                    system("rm EDITF");
                    exit(0);
                    }
                wait();
                return('q');
                }

            msg.msg_no = msg.txt_1k;
            msg.txt_1k = 0;
            msg.res_to_1k = 0;
            msg.nxt_msg_1k = 0;
            msg.pre_msg_1k = 0;
            msg.q_reads = 0;
            msg.q_res = 0;
            msg.last_r = 0;
            for (j=0; j<26; ++j)
                msg.time[j] = '\0';
            for (j=0; j<LL; ++j)
                msg.header_1n[j] = '\0';
            for (j=0; j < MSGL ; ++j)
                msg.msg_txt[j] = '\0';
            }  /* end else for no nxt_txt but need one */
        }  /* end else if c != EOF  */
    } /* end while c != EOF  */
fclose(fptr);
if((fptr=fopen("topicinfo","w"))==NULL)
    {
    printf("Could not open BBS topic info. file\n");
```

```
        unlock_box();
        child = fork();
        if (child == 0)
            {
            setuid(getuid());
            name=getenv("HOME");
            chdir(name);
            system("rm EDITF");
            exit(0);
            }
        wait();
        return('q');
        }
    fwrite(&t_dat,sizeof(t_dat),1,fptr);
    fclose(fptr);
        unlock_box();
        child = fork();
        if (child == 0)
            {
            setuid(getuid());
            name=getenv("HOME");
            chdir(name);
            system("rm EDITF");
            exit(0);
            }
        wait();
        return('q');
} /* end do_enter */
```

```
read_msg()
{
unsigned int msg_no,equ;
FILE *fptr;
struct message msg;
struct topic_info t_dat;
struct usr_info u_dat;
char cmd[LL],next_flag;
char *getenv(),*name,home[LL],*strcat();
int c,i,bad,j;
next_flag = 'n';
if((fptr=fopen("topicinfo","r"))==NULL)
    {
    printf("Could not open BBS topic info. file\n");
    return('q');
    }
fread(&t_dat,sizeof(t_dat),1,fptr);
if(feof(fptr))
    {
    printf(" end of file encountered in BBS topic data file.\n");
    return('q');
    }
fclose(fptr);
name=getenv("HOME");
strncpy(home,name,LL);
strcat(home,"/tybr.info",LL);
if((fptr=fopen(home,"r"))==NULL)
    {
```

```
    for( i= 0; i < NGPS; i++)
        u_dat.last_r[i] = 0;
    for( i= 0; i < NGPS; i++)
        for( j= 1; j <= TL; j++)
            u_dat.last_compress[i][j] = ' ';
    strncpy(u_dat.last_compress[t_dat.group_number],
            t_dat.last_compress,TL);
    fptr=fopen(home,"w");
    fwrite(&u_dat,sizeof(u_dat),1,fptr);
    }
else
    {
    fread(&u_dat,sizeof(u_dat),1,fptr);
    if(feof(fptr))
        {
        printf(" end of file encountered in BBS tybr.info file.\n");
        return('q');
        }
    c=strcmp(t_dat.last_compress,
    u_dat.last_compress[t_dat.group_number],TL);
    if(c!=0)
        {
        c=strcmp(t_dat.prev_compress,
        u_dat.last_compress[t_dat.group_number],TL);
        if(c==0)
            {
            /* the u_dat.last_r is not up to date but equ.tab
               file can bring it up to date.  */
```

```
if((fptr=fopen("equ.tab","r"))==NULL)
    {
    u_dat.last_r[t_dat.group_number] = 0;
    next_flag = 'y'; /* flag for warning to user */
    }
else
    {
    c=((sizeof(equ)) *
    (u_dat.last_r[t_dat.group_number] -1));
    c=fseek(fptr,c,0);
    if(c)
        {
        u_dat.last_r[t_dat.group_number] = 0;
        next_flag = 'y'; /* flag for warning to user */
        }
    else
        {
        fread(&equ,sizeof(equ),1,fptr);
        u_dat.last_r[t_dat.group_number]=equ;
        strncpy(u_dat.last_compress[t_dat.group_number],
                t_dat.last_compress,TL);
        if((fptr=fopen(home,"w"))==NULL)
            {
            printf(" Could not open tybr.info.
            file  %s in do_read.c\n",home);
            return('q');
            }
        fwrite(&u_dat,sizeof(u_dat),1,fptr);
```

```
                    fclose(fptr);
                    }
                }
            }
            else  /* if no match on prev_compress  */
                {
                u_dat.last_r[t_dat.group_number] = 0;
                strncpy(u_dat.last_compress[t_dat.group_number],
                        t_dat.last_compress,TL);
                if((fptr=fopen(home,"w"))==NULL)
                    {
                    printf(" Could not open tybr.info.
                    file  %s in do_read.c\n",home);
                    return('q');
                    }
                fwrite(&u_dat,sizeof(u_dat),1,fptr);
                fclose(fptr);
                next_flag = 'y'; /* flag for warning to user */
                }
        }
    }
fclose(fptr);
if( u_dat.last_r[t_dat.group_number] < t_dat.first_msg_no )
    u_dat.last_r[t_dat.group_number] = t_dat.first_msg_no;
if( u_dat.last_r[t_dat.group_number] > t_dat.last_msg_no )
    u_dat.last_r[t_dat.group_number] = t_dat.last_msg_no;
c=' '; /* initilize c */
while((cmd[1] != 'x') && (c!= EOF))
```

```
{
printf("#,f,n,q,x,?\n");
printf("ENTER COMMAND-->");
for (i=1; i<LL &&(c=getchar()) != '\n' &&
    c != EOF; i++)
    cmd[i]=c;
switch(cmd[1])
    {
    case '#':
    printf("Enter message number for start of read. -->");
    scanf("%d",&msg_no);
    c=getchar(); /* pick up the cr */
    if(msg_no < t_dat.first_msg_no ||
        msg_no > t_dat.last_msg_no)
        {
        printf("The message number input was out of range.\n");
        printf(" First = %d  Last = %d\n"
            ,t_dat.first_msg_no,t_dat.last_msg_no);
        return('q');
        }
    else
        {
        bad=get_msg(msg_no,&msg);
        if(bad)
            {
            BAD(msg_no);
            }
        cmd[1] = do_read(msg_no);
```

```
    return(cmd[1]);
    break;
    }
case 'f':
msg_no = t_dat.first_msg_no;
cmd[1] = do_read(msg_no);
return(cmd[1]);
break;
case 'n':
if (next_flag == 'y')
  printf("Because more than one compress of the
  message file occurred since your last read your next
  message was reset to 1.\n");
bad = get_msg(u_dat.last_r[t_dat.group_number],&msg);
if(bad)
    {
    BAD(t_dat.first_msg_no);
    }
if(msg.msg_no == 0)
    {
    bad = get_msg(t_dat.first_msg_no,&msg);
    if(bad)
        {
        BAD(t_dat.first_msg_no);
        }
    while((u_dat.last_r[t_dat.group_number] >
    msg.nxt_msg_lk) && (msg.nxt_msg_lk != 0))
        {
```

```
        bad = get_msg(msg.nxt_msg_lk,&msg);
        if (bad)
            {
            BAD(msg.nxt_msg_lk);
            }
        } /* end while  */
    } /* end if msg.msg_no == 0*/
if(msg.nxt_msg_lk == 0)
    {
    printf("You have already read all the messages.\n");
    return('q');
    }
msg_no = msg.nxt_msg_lk;
cmd[1] = do_read(msg_no);
return(cmd[1]);
break;
case 'q':
return('q');
break;
case 'x':
return('x');
break;
case '?':
printf(" #  = Start reading with the message number input.\n");
printf(" f  = Start reading with the first message for this
topic\n");
printf(" n  = Start reading with the message following the
last message read\n");
```

```
        printf(" q  = Quit the read message function.\n");
        printf(" x  = Exit the bulletin board system.\n");
        printf(" ?  = Display this menu.\n");
        break;
        }  /* end switch */
    } /* end while */
} /* end read_msg */
```

```
do_read(msg_no)
unsigned int msg_no;
{
int child,bad,c,i;
char *getenv(),*name,home[LL],*strcat();
char cmd[LL];
unsigned int last_r,next,prev,this,resp,cont;
struct message msg,*mptr;
struct topic_info t_dat;
struct usr_info u_dat;
FILE *fptr;
mptr = &msg;
name=getenv("HOME"); /* find the home directory for this user */
strncpy(home,name,LL);
strcat(home,"/tybr.info",LL);
if((fptr=fopen(home,"r"))==NULL)
    {
    printf("Could not open tybr.info in do_read\n");
    }
else
    {
    fread(&u_dat,sizeof(u_dat),1,fptr);
    if(feof(fptr))
        {
        printf(" end of file encountered in BBS tybr.info file.\n");
        return('q');
        }
    }
```

```
cmd[1] = ' ';
if((fptr=fopen("topicinfo","r"))==NULL)
    {
    printf("Could not open BBS topic info. file\n");
    return('q');
    }
fread(&t_dat,sizeof(t_dat),1,fptr);
if(feof(fptr))
    {
    printf(" end of file incountered in BBS topic data file.\n");
    return('q');
    }
fclose(fptr);
while((msg_no) && c != EOF)
    {
    cmd[1] = ' ';
    bad = get_msg(msg_no,mptr);
    if (bad)
        {
        BAD(msg_no);
        }
    if (msg.msg_no > u_dat.last_r[t_dat.group_number])
        {
        u_dat.last_r[t_dat.group_number]= msg.msg_no;
        strncpy(u_dat.last_compress[t_dat.group_number],
                t_dat.last_compress,TL);
        }
    next = msg.nxt_msg_lk;
```

```
prev = msg.pre_msg_lk;
this = msg.msg_no;
resp = msg.rsp_lk;
if (resp)
    cont = next;
printf(" message #%d  ",this);
printf(" author: %s   ",msg.auth);
printf(" date: %s",msg.time);
printf(" responses: %d\n",msg.q_res);
printf(" title: %s\n",msg.header_ln);
printf("%s",msg.msg_txt);
while(msg.txt_lk)
    {
    printf("( CONTINUE ) 'cr',q,x,? -->");
    c = getchar();
        if (c=='q')
                {
                child = fork();
                if (child == 0)
                    {
                    c = setuid(getuid());
                    if (c != 0)
                        printf (" setuid failed error number
                        is = %d\n",errno);
                    name=getenv("HOME");
                    strncpy(home,name,LL);
                    strcat(home,"/tybr.info",LL);
                    if((fptr=fopen(home,"w"))==NULL)
```

```
                {
                printf(" 3 Could not open tybr.info.
                file  %s in do_read.c\n",home);
                return('q');
                }
            fwrite(&u_dat,sizeof(u_dat),1,fptr);
            fclose(fptr);
            exit(0);
            }
          wait();
        c = getchar();
        return('q');
        }
else
    if (c=='x')
        {
        child = fork();
        if (child == 0)
            {
            c = setuid(getuid());
            if (c != 0)
                printf (" setuid failed error number
                is = %d\n",errno);
            name=getenv("HOME");
            strncpy(home,name,LL);
            strcat(home,"/tybr.info",LL);
            if((fptr=fopen(home,"w"))==NULL)
                {
```

```
            printf(" 3 Could not open tybr.info.
            file  %s in do_read.c\n",home);
            return('q');
            }
        fwrite(&u_dat,sizeof(u_dat),1,fptr);
        fclose(fptr);
        exit(0);
        }
      wait();
    c = getchar();
    return('x');
    }
else
    if (c=='?')
        {
        printf(" 'cr'  = Carriage return will print");
        printf(" more of the remaining text for this
        message\n");
        printf(" q     = Quit reading this message.\n");
        printf(" x     = Exit the bulletin board system.\n"
        printf(" ?     = Print this menu\n");
        printf(" other = The default for other input
        is is to continue with this message\n");
        }
    else
        {
        bad = get_msg(msg.txt_lk,mptr);
        if(bad)
```

```
                {
                BAD(msg.txt_lk);
                }
            if (msg.msg_no >
            u_dat.last_r[t_dat.group_number])
                {
                u_dat.last_r[t_dat.group_number]=
                msg.msg_no;
                strncpy(u_dat.last_compress
                [t_dat.group_number],
                        t_dat.last_compress,TL);
                }
            printf("%s",msg.msg_txt);
            }
        } /* end while more text */
    while(cmd[1] != 'x' && cmd[1] != '\n' &&cmd[1] != 'n'
    &&cmd[1] != 'b' &&cmd[1] != 'r' && cmd[1] != 'N'
    && c!= EOF)
        {
        printf("\n n,b,r,N,R,q,x,?\n");
        printf(" ENTER COMMAND-->");
        cmd[1] = '\n';
        for(i=1; i<LL && (c=getchar()) !=EOF
        && c != '\n' ; i++)
            cmd[i] = c;
        if(i <= 2 && c!=EOF )
            {
            switch(cmd[1])
```

```
{
case 'n':
case '\n':
msg_no = next;
if ( msg_no == 0)
    {
    if (msg.res_to_1k)
        msg_no = cont;
    }
break;
case 'b':
msg_no = prev;
break;
case 'r':
if(resp == 0)
    {
    printf("No responses for this
    message.");
    printf("continuing with the next
    message.\n");
    msg_no = next;
    }
else
    msg_no = resp;
break;
case 'N':
if(msg.res_to_1k == 0)
    msg_no = next;
```

```
else
    msg_no = cont;
break;
case 'R':
if(msg.res_to_lk != 0)
    {
    printf("You can not respond to a
    message ");
    printf("that is a response.\n");
    printf("Try responding to original
    message;  ");
    printf("That way your response will
    appear after this response.\n");
    msg_no = msg.res_to_lk;
    }
else
    cmd[1] = enter(0,this);
break;
case 'q':
child = fork();
if (child == 0)
    {
    setuid(getuid());
    name=getenv("HOME");
    strncpy(home,name,LL);
    strcat(home,"/tybr.info",LL);
    if((fptr=fopen(home,"w"))==NULL)
        {
```

```
        printf(" Could not open tybr.info.
        file  %s in do_read.c\n",home);
        return('q');
        }
    fwrite(&u_dat,sizeof(u_dat),1,fptr);
    fclose(fptr);
    exit(0);
    }
wait();
return('q');
break;
case 'x':
child = fork();
if (child == 0)
    {
    setuid(getuid());
    name=getenv("HOME");
    strncpy(home,name,LL);
    strcat(home,"/tybr.info",LL);
    if((fptr=fopen(home,"w"))==NULL)
        {
        printf(" 2 Could not open tybr.info.
        file  %s in do_read.c\n",home);
        return('q');
        }
    fwrite(&u_dat,sizeof(u_dat),1,fptr);
    fclose(fptr);
    exit(0);
```

```
                    }
                wait();
                return('x');
                break;
                case '?':
                printf(" n  = Next message or response.\n");
                printf(" b  = Back up one message or
                response.\n");
                printf(" r  = Read the responses to this
                message if any.\n");
                printf(" N  = Next - Quit reading responses
                and go to the next message.\n");
                printf(" R  = Respond to this message.\n");
                printf("      No responses are permitted for
                response messages.\n");
                printf(" q  = Quit reading messages.\n");
                printf(" x  = Exit the BBS.\n");
                printf(" ?  = Print this menu.\n");
                } /*  end switch  */
            } /* end if */
        } /* end while */
    } /* end while */
child = fork();
if (child == 0)
    {
    c = setuid(getuid());
    if (c != 0)
        printf (" setuid failed error number is = %d\n",errno);
```

```
    name=getenv("HOME");
    strncpy(home,name,LL);
    strcat(home,"/tybr.info",LL);
    if((fptr=fopen(home,"w"))==NULL)
        {
        printf(" 3 Could not open tybr.info. file
        %s in do_read.c\n",home);
        return('q');
        }
    fwrite(&u_dat,sizeof(u_dat),1,fptr);
    fclose(fptr);
    exit(0);
    }
  wait();
  return(' ');
} /* end do_read */
```

```
readmm()
{
unsigned int i,bad,msg_no;
struct message *mptr,msg;
int c,text;
char cmd[LL];
char *getenv(),*name,home[LL],*strcat();
FILE *fmm;
text = 0; /* flag for printing more text for long message */
mptr = &msg;
printf("Entering readmm\n");
name=getenv("HOME");
strncpy(home,name,LL);
strcat(home,"/mmf_bbs",LL);
if((fmm=fopen(home,"r"))==NULL)
    {
    printf("No marked messages to read.\n");
    printf("To mark messages first do a scan 's' or ");
    printf("scan messages and responses 'sr'.\n");
    return('q');
    }
fscanf(fmm,"%d ",&msg_no);
while(msg_no)
    {
    bad = get_msg(msg_no,mptr);
    if(bad)
        {
        BAD(msg_no);
```

```
    }
if (!msg.txt_lk)
    text = 0;
if (!text)  /* if continued text only print text */
    {
    printf("message #%d",msg.msg_no);
    printf("  author: %s",msg.auth);
    printf("  date: %s\n",msg.time);
    printf("title: %s",msg.header_ln);
    printf(" number of responses: %d\n",msg.q_res);
    }
printf("%s",msg.msg_txt);
if(msg.txt_lk)
    {
    text = 1; /* set flag to print only text */
    printf("(CONTINUE) 'cr',q,Q,x,? -->");
    c=getchar();
    switch(c)
        {
        case 'q':
        c=getchar();
        msg.txt_lk = 0;
        break;
        case 'Q':
        return('q');
        break;
        case 'x':
        c=getchar();
```

```
            return('x');
            break;
            case '?':
            c=getchar();
            printf("'cr' = Carrage return will print");
            printf(" more of this message.\n");
            printf(" q   = Quit reading this message.\n");
            printf(" Q   = Quit reading marked messages.\n");
            printf(" x   = Exit the bulletin board system.\n");
            printf(" ?   = Print this menu.\n");
            printf(" other = the default for other input is to
            continue.\n");
            break;
            default:
            msg_no= msg.txt_lk;
            } /* end switch */
        }  /* end while */
    else
        {
        if (feof(fmm))
            {
            printf ("No more marked messages.\n");
            return('q');
            }
        fscanf(fmm,"%d ",&msg_no);
        printf("Next message is # %d\n",msg_no);
        printf("'cr',q,x,? -->");
        for (i=1; i< LL && (c = getchar()) != '\n' && c !=EOF ; ++i)
```

```
            cmd[1] = c;
            cmd[1] = c;
            switch(cmd[1])
                {
                case 'q':
                return('q');
                break;
                case 'x':
                return('x');
                break;
                case '?':
                printf("'cr' = Carrage return will print");
                printf(" the next message marked.\n");
                printf(" q   = Quit reading this message.\n");
                printf(" x   = Exit the bulletin board system.\n");
                printf(" ?   = Print this menu.\n");
                printf(" other = the default for other input is to
                } /* end switch */
        } /* end else */
    } /* end while */
    return ('q');
} /* end  readmm  */
```

```
scan_hdr(s_type)
int s_type; /* scan type
                        * 1 = scan message headers
                        * 2 = scan headers messages and responses
                        * 3 = quick scan message headers
                        * 4 = quick scan headers messages
                              and responses
                        */
{
struct message msg,*mptr;
struct topic_info t_dat;
struct usr_info u_dat;
FILE *fptr;
char cmd[LL],next_flag;
int c;
unsigned int last_r,j,i,bad,bad_msg,s_msg_no,equ;
char *getenv(),*name,home[LL],*strcat();
c = ' ';
next_flag = 'n';
cmd[1]=' ';
if((fptr=fopen("topicinfo","r"))==NULL)
    {
    printf("Could not open BBS topic info. file\n");
    return('q');
    }
fread(&t_dat,sizeof(t_dat),1,fptr);
if(feof(fptr))
    {
```

```
    printf(" end of file encountered in BBS topic data file.\n");
    return('q');
    }
fclose(fptr);
name=getenv("HOME");
strncpy(home,name,LL);
strcat(home,"/tybr.info",LL);
if((fptr=fopen(home,"r"))==NULL)
    {
    for( i= 0; i < NGPS; i++)
        u_dat.last_r[i] = 0;
    for( i= 0; i < NGPS; i++)
        for( j= 1; j <= TL; j++)
            u_dat.last_compress[i][j] = ' ';
    strncpy(u_dat.last_compress[t_dat.group_number],
            t_dat.last_compress,TL);
    fptr=fopen(home,"w");
    fwrite(&u_dat,sizeof(u_dat),1,fptr);
    }
else
    {
    fread(&u_dat,sizeof(u_dat),1,fptr);
    if(feof(fptr))
        {
        printf(" end of file encountered in BBS tybr.info file.\n");
        return('q');
        }
    c=strcmp(t_dat.last_compress,
```

```
u_dat.last_compress[t_dat.group_number],TL);
if(c!=0)
    {
    c=strcmp(t_dat.prev_compress,
    u_dat.last_compress[t_dat.group_number],TL);
    if(c==0)
        {
        /* the u_dat.last_r is not up to date but equ.tab
           file can bring it up to date.  */
        if((fptr=fopen("equ.tab","r"))==NULL)
            {
            u_dat.last_r[t_dat.group_number] = 0;
            next_flag = 'y'; /* flag for warning to user */
            }
        else
            {
            c=((sizeof(equ)) *
            (u_dat.last_r[t_dat.group_number] -1));
            c=fseek(fptr,c,0);
            if(c)
                {
                u_dat.last_r[t_dat.group_number] = 0;
                next_flag = 'y'; /* flag for warning to user */
                }
            else
                {
                fread(&equ,sizeof(equ),1,fptr);
                u_dat.last_r[t_dat.group_number]=equ;
```

```
            strncpy(u_dat.last_compress[t_dat.group_number],
                    t_dat.last_compress,TL);
            if((fptr=fopen(home,"w"))==NULL)
                {
                printf(" Could not open tybr.info.
                file  %s in do_read.c\n",home);
                return('q');
                }
            fwrite(&u_dat,sizeof(u_dat),1,fptr);
            fclose(fptr);
            }
        }
    }
    else  /* if no match on prev_compress  */
        {
        u_dat.last_r[t_dat.group_number] = 0;
        strncpy(u_dat.last_compress[t_dat.group_number],
                t_dat.last_compress,TL);
        if((fptr=fopen(home,"w"))==NULL)
            {
            printf(" Could not open tybr.info.
            file  %s in do_read.c\n",home);
            return('q');
            }
        fwrite(&u_dat,sizeof(u_dat),1,fptr);
        fclose(fptr);
        next_flag = 'y'; /* flag for warning to user */
        }
```

```
        }
    }
fclose(fptr);
if( u_dat.last_r[t_dat.group_number] < t_dat.first_msg_no )
    u_dat.last_r[t_dat.group_number] = t_dat.first_msg_no;
if( u_dat.last_r[t_dat.group_number] > t_dat.last_msg_no )
    u_dat.last_r[t_dat.group_number] = t_dat.last_msg_no;
c = ' ';
while((cmd[1] != 'q') && (cmd[1] != 'x') && (c != EOF))
    {
    printf("#,f,n,q,x,?\n");
    printf("ENTER COMMAND-->");
    for(i = 1; i<LL && (c=getchar()) != '\n' && c != EOF; ++i)
    cmd[i] = c;
    if(i <= 3 && c != EOF )
        {
        switch(cmd[1])
            {
            case 'n':
        if (next_flag == 'y')
          printf("Because more than one compress of the message
          file occurred since your last read your next message
          was reset to 1.\n");
            bad = get_msg(u_dat.last_r[t_dat.group_number],&msg);
            if(bad)
                {
                BAD(t_dat.first_msg_no);
                }
```

```
if(msg.msg_no == 0)
    {
    bad = get_msg(t_dat.first_msg_no,&msg);
    if(bad)
        {
        BAD(t_dat.first_msg_no);
        }
    while((u_dat.last_r[t_dat.group_number] >
    msg.nxt_msg_lk) && (msg.nxt_msg_lk != 0))
        {
        bad = get_msg(msg.nxt_msg_lk,&msg);
        if (bad)
            {
            BAD(msg.nxt_msg_lk);
            }
        } /* end while  */
    } /* end if msg.msg_no == 0*/
if(msg.nxt_msg_lk == 0)
    {
    printf("You have already read all the messages.\n");
    return('q');
    }
s_msg_no = msg.nxt_msg_lk ;
cmd[1] = do_s_hdr(s_msg_no,s_type);
return(cmd[1]);
break;
case 'f':
s_msg_no = t_dat.first_msg_no; /* first message in
```

```
group */
cmd[1] = do_s_hdr(s_msg_no,s_type);
return(cmd[1]);
break;
case '#':
bad_msg = 1;
while(bad_msg)
    {
    printf("ENTER MESSAGE NUMBER FOR START OF SCAN-->");
    scanf("%d",&s_msg_no);
    if(!(t_dat.first_msg_no < s_msg_no < t_dat.last_msg_no))
        {
        printf("Error message number out of range\n");
        printf("Valid range is:\n");
        printf("                min message number is %d\n",
            t_dat.first_msg_no);
        printf("              max message number is %d\n",
            t_dat.last_msg_no);
        } /* end if */
    else
    bad_msg = 0;
    } /* end while */
cmd[1] = do_s_hdr(s_msg_no,s_type);
return(cmd[1]);
break;
case 'q':
return('q');
break;
```

```
            case 'x':
            return('x');
            break;
            case '?':
            printf("n  = Next - Continue with the message following
            the last message read\n");
            printf("f  = Start with first message in the interest
            group\n");
            printf("#  = Start with some input message number \n");
            printf("q  = Quit scan and return to previous level \n");
            printf("x  = Exit this bulletin board system \n");
            printf("?  = Display this menu \n");
            } /* end switch */
        } /* end if */
    } /* end while */
} /* end scan hdr */
do_s_hdr(s_msg_no,s_type)
unsigned int s_msg_no,s_type;
{
char *getenv(),*name,home[LL],*strcat();
struct message msg,*mptr;
char cmd[LL];
unsigned int j,i,bad,bad_msg,save_nxt,break_s;
int c,mm;
FILE *fptr;
mptr = &msg;
break_s = 0;
cmd[1] = ' ';
```

```
while(cmd[1] != 'x'
&& c != EOF && cmd[1] != 'q'
&& !break_s) /* while a continued search is called for */
    {
    bad = get_msg(s_msg_no,mptr);
    if (bad)
        {
        BAD(s_msg_no);
        }
    printf("message #%d   ",msg.msg_no);
    printf("author: %s   ",msg.auth);
    printf("date: %s",msg.time);
    printf("title: %s   ",msg.header_ln);
    printf("   responses: %d\n\n",msg.q_res);
    if (s_type <= 2)
    {
    cmd[1] = ' ';
    while(cmd[1] != 'x' && c != EOF && cmd[1] != 'q')
        {
        printf("n,b,#,m,q,x,?\n");
        printf("ENTER COMMAND-->");
        for(i=1; i<LL && (c=getchar()) != '\n' && c != EOF; ++i)
        cmd[i] = c;
        cmd[i] = c;
        if (i <= 3 && c != EOF )
            {
            switch(cmd[1])
                {
```

```
case 'b':
s_msg_no = msg.pre_msg_lk;
if (s_msg_no)
    {
    bad = get_msg(s_msg_no,mptr);
    if (bad)
        {
        BAD(s_msg_no);
        }
    }
printf("message #%d   ",msg.msg_no);
printf("author: %s   ",msg.auth);
printf("date: %s",msg.time);
printf("title: %s   ",msg.header_ln);
printf("  responses: %d\n\n",msg.q_res);
break;
case 'm':
    name=getenv("HOME");
    strncpy(home,name,LL);
    strcat(home,"/mmf_bbs",LL);
    fptr=fopen(home,"a+");
    fprintf(fptr,"%d\n",msg.msg_no);
    fclose(fptr);
case 'n':
case '\n':
s_msg_no = msg.nxt_msg_lk;
if (s_type == 2)
if (msg.rsp_lk)
```

```
    {
    save_nxt = s_msg_no;
    s_msg_no = msg.rsp_lk;
    }
if (s_msg_no == 0)
{
s_msg_no = save_nxt;
save_nxt = 0;
}
if (s_msg_no == 0)
    cmd[1] = 'q';
else
{
bad = get_msg(s_msg_no,mptr);
if (bad)
    {
    BAD(s_msg_no);
    }
printf("message #%d   ",msg.msg_no);
printf("author: %s   ",msg.auth);
printf("date: %s",msg.time);
printf("title: %s   ",msg.header_ln);
printf("  responses: %d\n\n",msg.q_res);
} /* end else */
break;
case 'q':
return('q');
break;
```

```
            case 'x':
            return('x');
            break;
            case '?':
            printf("n  = Print next header\n");
            printf("b  = Print previous message header\n");
            printf("m  = Mark this message to read later\n");
            printf("q  = Quit scan of headers\n");
            printf("x  = Exit the bulletin board system\n");
            printf("?  = Print this menu\n");
            } /*end switch */
        } /* end if */
    } /* end while */
} /* end if */
else /* if s_type is 3 or 4 */
    {
        s_msg_no = msg.nxt_msg_lk;
        if (s_type == 4)
            {
            if (msg.rsp_lk != 0)
                {
                save_nxt = s_msg_no;
                s_msg_no = msg.rsp_lk;
                }
            if (s_msg_no == 0)
                {
                s_msg_no = save_nxt;
                save_nxt = 0;
```

```
                    }
                } /* end if(s_type) */
        if (s_msg_no == 0)
            return(' ');
        }  /* end else */
    }  /* end while  */
}  /* end do_s_msg */
```

```
echo " Because not all groups available to anyone requesting admitance,"
echo " granting your request may not be possible."
echo " Even so, it does not hurt to ask!"
echo " Enter the title of the group you wish to join."
read Title
mail gfm <<!
Join an interest group.
$Title
!
```

```
echo " In order to establish a new group, sufficent interest"
echo " must be shown. By submitting an interest group title"
echo " or description the system operator can determine if "
echo " there is enough interest in the topic to establish a "
echo " new group."
echo " Enter title or short description."
read Title
mail gfm <<!
New interest group suggestion.
$Title
!
```

```
quit_group() /* quit a user interest group */
  {
char cmd[LL],uid[9];
struct  passwd *pw;
struct  passwd *getpwuid();
int  i,c;
FILE *fptr1,*fptr,*fopen();
  cmd[1] = ' ';
  i == 1;
   while (c != EOF)  {
   list_topics();
   printf ("ENTER THE NUMBER OF THE TOPIC YOU WISH TO QUIT--->");
  for (i=1 ; i<LL && (c = getchar()) != '0 && c != EOF ; ++i)
   cmd[i] = c;
   if (c != EOF )  {
    switch(cmd[1])  {
     case '1':
     chdir("/usrb/att/mcmaniga/bin/geobbs");
     break;
     case '2':
     chdir("/usrb/att/mcmaniga/bin/entbbs");
     break;
     case '3':
     chdir("/usrb/att/mcmaniga/bin/hisbbs");
     break;
     case '4':
     chdir("/usrb/att/mcmaniga/bin/artbbs");
     break;
```

```
 case '5':
 chdir("/usrb/att/mcmaniga/bin/scibbs");
 break;
 case '6':
 chdir("/usrb/att/mcmaniga/bin/spobbs");
 break;
 case 'x':
 return('x');
 break;
 case 'q':
 return('q');
 break;
} /* end switch */
 if(cmd[1] < '7' && cmd[1] > '0')
   {
 if((fptr = fopen("user.list","r"))!= NULL)
 {
 /* the user list file exists, so this interest group is closed */
 pw = getpwuid(getuid());
 fscanf(fptr, "%s", uid);
 if (feof(fptr))
  {
  /* There are no members listed for this interest group.  */
  return(' ');
  }
  fptr1 = fopen("temp.user.list","w");
  while (strcmp(pw->pw_name,uid,NMAX) != 0)
         {
```

```
            fprintf(fptr1, "%s0, uid);
            fscanf(fptr, "%s", uid);
            if (feof(fptr))
                {
                printf("Your user id was not on the list of those
                who enter this interest group.0);
                fclose(fptr);
                return(' ');
                }
             }
    while (1)
            {
            fscanf(fptr, "%s", uid);
            if (feof(fptr))
                {
                fclose(fptr);
                fclose(fptr1);
                system("mv temp.user.list user.list");
                return(' ');
                }
            fprintf(fptr1, "%s0, uid);
             }
        } /* end if user.list */
        return(' ');
      } /*end if */
  } /* end if */
 } /* end while */
} /* end quit group*/
```

```
get_msg(msg_no,mptr)
struct message *mptr;
unsigned int msg_no;
{
FILE *msgf;
long good,x;
if (msg_no == 0)
    msg_no = 1; /* trap to avoid a potential problem */
if((msgf=fopen("msg.file","r"))==NULL)
    return(1);
    x=(msg_no - 1)*sizeof(*mptr);
good = fseek(msgf,x,0);
if (good)
    return(2);
good = fread(mptr,sizeof(*mptr),1,msgf);
if (!good)
    return(3);
fclose(msgf);
return(0);
} /* end get_msg */
```

```
put_msg(msg_no,mptr)
struct message *mptr;
unsigned int msg_no;
{
FILE *msgf;
long good,x;
if((msgf=fopen("msg.file","r+"))==NULL)
    return(4);
x=(msg_no - 1)*sizeof(*mptr);
good = fseek(msgf,x,0);
/* if (!good)
   return(5); */
good = fwrite(mptr,sizeof(*mptr),1,msgf);
if (!good)
    return(6);
fclose(msgf);
} /*end  put_msg  */
```

```
lock_box()
{
FILE *fptr;
int i,j;
for (i=0;i<50; i++)
    {
    if ((j=access("lock.box",0)) == -1)
        {
        if((fptr = fopen("lock.box", "a+")) != NULL)
            {
            fclose(fptr);
            return(' ');
            }
            else
                fprintf(stderr,"Cant create lock.box file0);
        }
        sleep(1);
    }
    fprintf(stderr,"TYBR timed out waiting on lock.box0);
    return('q');
}
```

```
unlock_box()
{
unlink("lock.box");
return(' ');
}
```

```
struct message {
        unsigned int msg_no,
                     txt_lk,
                     res_to_lk,
                     nxt_msg_lk,
                     pre_msg_lk,
                     rsp_lk,
                     q_reads,
                     q_res;
        long int     last_r,
                     group_id;
        char         auth[10],
                     time[26],
                     header_ln[255],
                     msg_txt[MSGL];
                     };
struct usr_info {
    unsigned int    last_r[NGPS];
    char            last_compress[NGPS][TL];
    };
struct topic_info {
    unsigned int    first_msg_no,
                    last_msg_no,
                    group_number;
    char
                    last_compress[TL],
                    prev_compress[TL];
    };
```

```
compress()
{
/* Utility program to recover lost space in
   the message file.
   An equivelance file, showing the realtionship between the
   old message numbers and the new message numbers is creates.
   This equivelance file is used to maintain the users last
   message read information.

   OPERATOR CAUTION: Compresses should be done sparingly so
                     users last read information is not lost.
                     If the user does not read after each
                     compress the last message read info
                     will be lost.
                     It is recommented that compresses be run
                     no more than once a month.
 */
int bad,c,i;
char cmd[LL];
unsigned int equ_no;
struct message msg,*mptr;
struct topic_info t_dat;
char *tdate,*ctime();
long *clock,time();
FILE *fptr;
mptr = &msg;
lock_box();  /* lock out other users while entering a message */
fptr = fopen("equ.tab","w");
```

```
bad = 0;
for (i=1;bad == 0;i++)
    {
    bad = get_msg(i,mptr);
    if (!bad)
        {
        if (msg.msg_no != 0)
            ++ equ_no;
        fwrite(&equ_no,sizeof(equ_no),1,fptr);
        }
    } /* end for loop */
fclose(fptr);
if((fptr=fopen("topicinfo","r"))==NULL)
    {
    printf("Could not open BBS topic info. file\n");
    unlock_box();
    return('q');
    }
fread(&t_dat,sizeof(t_dat),1,fptr);
if(feof(fptr))
    {
    printf(" end of file incountered in BBS topic data file.\n");
    unlock_box();
    return('q');
    }
time(&clock);
tdate = ctime(&clock);
strncpy(t_dat.prev_compress , t_dat.last_compress ,TL);
```

```
strncpy(t_dat.last_compress , tdate,TL);
t_dat.last_msg_no = equ_no;
fclose(fptr);
if((fptr=fopen("topicinfo","w"))==NULL)
    {
    printf("Could not open BBS topic info. file\n");
    unlock_box();
    return('q');
    }
fwrite(&t_dat,sizeof(t_dat),1,fptr);
fclose(fptr);
fptr=fopen("equ.tab","r");
bad = 0;
for (i=1;bad == 0;i++)
    {
    printf("Getting message no %d\n",i);
    bad = get_msg(i,mptr);
    if (!bad)
        {
        if (msg.msg_no != 0)
            {
                c = ((sizeof(equ_no)) * (msg.msg_no-1));
                c = fseek(fptr,c,0);
                if (!c)
                    {
                    fread(&equ_no,sizeof(equ_no),1,fptr);
                    }
                msg.msg_no = equ_no;
```

```
if (msg.txt_lk != 0)
    {
    c = ((sizeof(equ_no)) * (msg.txt_lk-1));
    c = fseek(fptr,c,0);
    if (!c)
        {
        fread(&equ_no,sizeof(equ_no),1,fptr);
        }
    msg.txt_lk = equ_no;
    }
if (msg.res_to_lk != 0)
    {
    c = ((sizeof(equ_no)) * (msg.res_to_lk-1));
    c = fseek(fptr,c,0);
    if (!c)
        {
        fread(&equ_no,sizeof(equ_no),1,fptr);
        }
    msg.res_to_lk = equ_no;
    }
if (msg.nxt_msg_lk != 0)
    {
    c = ((sizeof(equ_no)) * (msg.nxt_msg_lk-1));
    c = fseek(fptr,c,0);
    if (!c)
        {
        fread(&equ_no,sizeof(equ_no),1,fptr);
```

```
        }
    msg.nxt_msg_lk = equ_no;
    }
if (msg.pre_msg_lk != 0)
    {
    c = ((sizeof(equ_no)) * (msg.pre_msg_lk-1));
    c = fseek(fptr,c,0);
    if (!c)
        {
        fread(&equ_no,sizeof(equ_no),1,fptr);
        }
    msg.pre_msg_lk = equ_no;
    }
if (msg.rsp_lk != 0)
    {
    c = ((sizeof(equ_no)) * (msg.rsp_lk-1));
    c = fseek(fptr,c,0);
    if (!c)
        {
        fread(&equ_no,sizeof(equ_no),1,fptr);
        }
    msg.rsp_lk = equ_no;
    }
    bad = PUT_MSG(msg.msg_no,mptr);
    if (bad)
        {
        BAD(msg.msg_no);
        }
```

```
            }
    } /* end for loop */
} /* end do_read */
fclose(fptr);
printf (" moving new file to old\n");
system("mv newmsg.file msg.file");
unlock_box();
} /* end compress*/
PUT_MSG(msg_no,mptr)
struct message *mptr;
unsigned int msg_no;
{
FILE *msgf;
long good,x;
printf("entering PUT_MSG msg no = %d\n",msg_no);
if((msgf=fopen("newmsg.file","a+"))==NULL)
    return(4);
x=(msg_no - 1)*sizeof(*mptr);
good = fseek(msgf,x,0);
/* if (!good)
   return(5); */
good = fwrite(mptr,sizeof(*mptr),1,msgf);
if (!good)
    return(6);
fclose(msgf);
} /*end  PUT_MSG   */
```

```
/* utility message file dump program */
/* The program dumps all messages in the message file,
   reguardless of message type.       */
dump_all()
{
int bad,c,i;
char cmd[LL];
unsigned int msg_no,next,prev,this,resp,cont;
struct message msg,*mptr;
mptr = &msg;
printf("entering dump all message file program0);
bad = 0;
i = 1;
bad = get_msg(i,mptr);
for (i=2;bad == 0;i++)
    {
    printf("msg.nxt_msg_lk = %d0,msg.nxt_msg_lk );
    printf("msg.pre_msg_lk = %d0,msg.pre_msg_lk );
    printf("msg.rsp_lk = %d0,msg.rsp_lk );
    printf("msg.txt_lk = %d0,msg.txt_lk );
    printf("msg.res_to_lk = %d0,msg.res_to_lk );
    printf(" message #%d",msg.msg_no);
    printf(" author: %s",msg.auth);
    printf(" date: %s0,msg.time);
    printf(" title: %s",msg.header_ln);
    printf(" number of responses: %d0,msg.q_res);
    printf("%s",msg.msg_txt);
    bad = get_msg(i,mptr);
```

```
    } /* end for loop*/
} /* end dump_all*/
```

```
#include <stdio.h>
#include <pwd.h>
#include <ctype.h>
#define NMAX 8  /* sizeof(utmp.ut_name)  */
#define LL 256
#define NGPS 6
#define MSGL 1600
#define TL 26
#define BAD(MSG)
    {
    printf("get/put operation failed ret = %d0,bad);
    printf("   message number was = %d0,MSG);
    return('q');
    }
#include <sys/timer.h>
#include "bbs.global"
#include "list_topics.c"
main()
{
char *getenv(),*name,fname[LL],home[LL],*strcat();
int child,crct;
char *tdate,*ctime();
long *clock,time();
struct  passwd *pw;
struct  passwd *getpwuid();
unsigned int msg_no,old_msg,bad;
FILE *fptr;
struct message msg,*mptr;
```

```
struct usr_info u_dat;
struct topic_info t_dat;
char cmd[LL];
int c,i,j;
mptr = &msg;
t_dat.first_msg_no = 1;
t_dat.last_msg_no = 1;
time(&clock);
tdate = ctime(&clock);
strncpy(t_dat.last_compress , tdate,TL);
   list_topics();
strncpy(t_dat.prev_compress , tdate,TL);
   list_topics();
   printf ("ENTER THE NUMBER OF THE TOPIC YOU WISH TO START UP--->");
  for (i=1 ; i<LL && (c = getchar()) != '0 && c != EOF ; ++i)
   cmd[i] = c;
   if (c != EOF )  {
    switch(cmd[1])  {
     case '1':
     t_dat.group_number = 1;
     break;
     case '2':
     t_dat.group_number = 2;
     break;
     case '3':
     t_dat.group_number = 3;
     break;
     case '4':
```

```
      t_dat.group_number = 4;
      break;
      case '5':
      t_dat.group_number = 5;
      break;
      case '6':
      t_dat.group_number = 6;
      break;
      case '7':
      t_dat.group_number = 7;
      break;
   } /* end switch */
for (i=1; i <= 6 ; i++)
    u_dat.last_r[i] = 1;
msg.msg_no=0;
if((fptr=fopen("topicinfo","w"))==NULL)
    {
    printf("Could not open BBS topic info. file0);
    return('q');
    }
fwrite(&t_dat,sizeof(t_dat),1,fptr);
fclose(fptr);
     msg.msg_no=1;
     msg.nxt_msg_lk=0;
     msg.res_to_lk=0;
     msg.txt_lk=0;
     msg.pre_msg_lk=0;
     msg.rsp_lk=0;
```

```
 msg.q_reads=0;
 msg.q_res=0;
 msg.last_r=0;
 msg.group_id=0;
for (i=0;i<255;i++)
msg.header_ln[i]=' ';
for (i=0;i<1600;i++)
msg.msg_txt[i]=' ';
    pw = getpwuid(getuid());
    strncpy(msg.auth, pw ? pw->pw_name : "?", NMAX);
 time(&clock);     /*get time  */
 tdate = ctime(&clock);
 strncpy(msg.time, tdate, TL);
 printf(" msg.time = %s",msg.time);
 printf(" tdate  = %s",tdate);
            child = fork();
            if (child == 0)
                {
            fname[0] = 'c';
            fname[1] = 'a';
            fname[2] = 't';
            fname[3] = ' ';
            printf("From $HOME ENTER FILE NAME--->");
            for(i = 4; (c=getchar()) != '0 && c != EOF;
                i++)
                fname[i]=c;
            fname[i++]=' ';
            fname[i++]='>';
```

```
                fname[i++]='E';
                fname[i++]='D';
                fname[i++]='I';
                fname[i++]='T';
                fname[i++]='F';
                name=getenv("HOME");
                chdir(name);
                system("pwd");
                system(fname);
                exit(0);
                    }
                wait();
name=getenv("HOME");
strncpy(home,name,LL);
strcat(home,"/EDITF",LL);
if((fptr=fopen(home,"r"))==NULL)
    {
    printf(" could not open for reading EDITF in enter0);
    }
    for (i=0;i<=LL && (c=fgetc(fptr)) !='0 && c != EOF; ++i)
        msg.header_ln[i] = c;
    if (c == EOF)
        {
        printf(" A message must contain more than one line0);
        printf(" Try again -- first line is the message header0);
        printf("              the second line to the end is text.0);
        unlink(home);  /* rm EDITF */
        }
```

```
while (c!= EOF)
     {
     crct = 0;  /* zero out the 'cr' counter */
     for (j=0; j < MSGL ; ++j)
         msg.msg_txt[j] = ' ';
     printf(" c != EOF0);
     for (i=0; i<MSGL && crct <= 20 && (c = fgetc(fptr)) != EOF;
         ++i)
         {
         msg.msg_txt[i] = c;
         if ( msg.msg_txt[i] == '0 )
             ++crct;
         }
         if (c == EOF)
             {
             msg.msg_txt[i] = '0;
             msg.txt_lk = 0;
             bad=put_msg(msg.msg_no,mptr);
             if (bad)
                 {
                 BAD(msg.msg_no);
                 }
             }
         else  /* if c != EOF then more text to output */
             {
             printf(" in else more text to print0);
             if (msg.txt_lk)
                 {
```

```
        old_msg = msg.txt_lk;
        bad = put_msg(msg.msg_no,mptr);
        if (bad)
            {
            printf(" put_msg failed in enter");
            printf(" ret=%d msg=%d old_msg done0,
                bad,msg.msg_no);
            unlink(home);  /* rm EDITF */
            return('q');
            }
        } /* end if txt_lk */
    else  /* no txt_lk but we need one now */
        {
        ++t_dat.last_msg_no;
        msg.txt_lk = t_dat.last_msg_no;
        old_msg = msg.txt_lk;
        bad = put_msg(msg.msg_no,mptr);
        if (bad)
            {
            printf(" put_msg failed in enter");
            printf(" ret=%d msg=%d no txt_lk but need one0
                ,bad,msg.msg_no);
            unlink(home);  /* rm EDITF */
            return('q');
            }
        msg.msg_no = msg.txt_lk;
        msg.txt_lk = 0;
        msg.res_to_lk = 0;
```

```
                    msg.nxt_msg_lk = 0;
                    msg.pre_msg_lk = 0;
                    msg.q_reads = 0;
                    msg.q_res = 0;
                    msg.last_r = 0;
                    for (j=0; j<TL; ++j)
                        msg.time[j] = ' ';
                    for (j=0; j<LL; ++j)
                        msg.header_ln[j] = ' ';
                    for (j=0; j < MSGL ; ++j)
                        msg.msg_txt[j] = ' ';
                    }  /* end else for no nxt_txt but need one */
                } /* end else if c != EOF  */
            } /* end while c != EOF  */
    }
}
put_msg(msg_no,mptr)
struct message *mptr;
unsigned int msg_no;
{
FILE *msgf;
long good,x;
printf("Entering put_msg message number is %d0,msg_no);
if((msgf=fopen("msg.file","a"))==NULL)
    return(4);
x=(msg_no - 1)*sizeof(*mptr);
good = fseek(msgf,x,0);
/* if (!good)
```

```
   return(5); */
good = fwrite(mptr,sizeof(*mptr),1,msgf);
if (!good)
    return(6);
fclose(msgf);
} /*end  put_msg  */
```

An Electronic Bulletin Board for UNIX Based Systems

by

Gerald F. McManigal

B. S. Northern Illinois University, 1973

An Abstract of A Master's Report

submitted in partial fulfillment of the

requirements for the degree

Master of Science

Department of Computer Science

Kansas State University
Manhattan, Kansas

1986

An Electronic Bulletin Board For UNIX Based Systems

by Gerald F. McManigal

An Abstract of a Master's Report

The term bulletin board brings to mind a vision of a cork backed board, strewn with notes roughly divided into areas of special interest and mounted in a prominent location. In many ways electronic bulletin boards are similar to the board just mentioned. This paper is the basis for the construction and implementation of The Yellow Brick Road (TYBR) an electronic bulletin board. Users of TYBR have the ability to browse through the system reading messages left by others or posting new messages of their own, much the same as their pedestrian counterparts. The goal of TYBR is to provide a basic message handling tool that will fulfill a variety of useful functions.

The paper examines five other bulletin board like systems, concentrating on their user interface and file structure. These very different systems are compared to TYBR. In the later sections of this paper the requirements and design issues are discussed. In the final section some future enhancements are suggested. A user scenario is contained in Appendix 1 and the source code for TYBR is found in Appendix 2.

1430-226
CA 96