PROTOTYPE SYSTEM FOR DOCUMENT MANAGEMENT

by

Jim Mullin

B.S., Missouri Western State College

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

1985

Approved by:

[illegible]

Major Professor

CONTENTS

1. Introduction

1.1 Overview

There is a wealth of information scattered among the employees of large corporations. Creating an information system in which to consolidate this information is imperative for an efficient decision-making process. Such an information system must be economical and user-friendly. The prototype system for document management is an automated method to centrally store information and then make it easily accessible to the people that need it.

This system will be different than current automated methods because it would avoid traditional pitfalls. Andrew E. Wessel has described some of these pitfalls in his book, "Computer-Aided Information Retrieval." He says an information system is "a mix of ill-structured information and well-structured data." "It is now a well-known fact that the same automated systems found relevant in handling well-structured data have been offered to users of ill-structured information."
In regards to simply using data base management systems such as ORACLE or dBASE III, "users have simply been offered additional lists of equally ill or worse structures which no one has been able to use for meaningful searches." The poor

structure of the information presents other problems such as, "When we request everything we want, we get both less and more than we asked."

The well-structured nature of this system will make it beneficial to many people. Specifically, it can meet the needs of marketing and support system engineers who have small multi-user systems. These people must share information in the decision-making process.

A systems engineer relies on information systems to help answer an abundance of questions including, "Has anyone done this before?" A large number of engineers could answer that question, however, individual contact is prohibitive. This prototype system will provide a central depository to help answer these questions. "What if" questions can be stored and retrieved later by engineers with the background to provide an answer.

This system has a wide range of applications beyond systems engineers' needs and could apply to virtually any business.

This outline will cover a review of current services and literature, including how this information applies to the design of this system.

This automated system as described by Steve Lambert in his book "Online", is a method of "storing information electronically and making it accessible to a computer . . . an extension of the library concept." Typically users can perform searches on such an automated system with a "combination of keywords and concepts that identify either a specific piece of information of the general topic you are researching," says Lambert.

The prototype system for document management is a vehicle by which the functions of a manual filing system are automated. This includes storage, retrieval management and user interface.

1.2 Literature Review

The prototype system for document management draws from the many products already dealing in information management, other bulletin board systems and online databases. It is therefore valuable tc survey existing systems for assistance in the design of this system. Steve Lambert describes 4 classifications of information management systems.

- Systems that offer primarily services rather than information.

- Systems that offer access to a wide variety of subjects. Refered to as encyclopedic databases.

- Systems similar to encyclopedic databases but their scope of topics is more limited. These are known as specialized or subject-specific databases.

- Systems which offer both services and access to information. This group is called information utilities.

The systems reviewed fall into these classifications and their design has provided assistance in developing the design of this management system.

1.2.1 Definitions

bulletins --- general messages that are retrieved through functions of the system.

command prompt --- when the system is ready for command input a prompt is displayed on the terminal. This can be a single character such as '$' or a collection of characters 'command:'.

directory --- a directory performs the same function as a file drawer in a filing cabinet, gathering together related files in a common place where they can be found easily.

download/upload --- the transferring of data from the information system to the remote device accessing it.

electronic mail --- mail facilities such as editing, delivery, printing and presentation automated and controlled by computer systems.

keywords --- a word to be used as part of some criteria by which selection of documents are made.

menu --- an informational display containing all available commands that a computer user may enter into the system.

time sharing --- a computer-use technique where by several operators may use a computers resources at the same time.

UNIX --- a comprehensive operating system developed by Bell Laboratories. The system allows efficient sharing of process power and the information storage of a computer system.

1.2.2 RBBS-PC

One of the more popular bulletin board systems is the RBBS- PC offered by the Capital PC user's group. For the most part this system can be classified as a service oriented system. It contains many common bulletin board services such as:

* Electronic Mail
* Upload/Download files
* Use of the Operating System
* Bulletins
* Messages

This system truly represents the typical computer hobbyist bulletin board system. Notes and reminders can be sent to other users through the electronic mail facility. The board serves as a convenient central depository to store and retrieve programs and documentation with the upload/download capabilities. Users with simple, less sophisticated computers can use the bulletin board as a time sharing device through operating system devices available. Information can be obtained through the bulletin facility. The most commonly used feature of any bulletin board system is the message facility.

With the message service a user can enter a message and address it to any other user of the system. This, being a public bulletin board, others may view it and if desired respond to it. All messages are numbered and have pointers associated with them so that anyone interested can follow the chain by which the messages were created. The RBBS-PC system contains 5 commands that deal with the message service. The five commands are:

* Kill message
* Quick scan
* Reading Messages
* Scan messages
* Enter message

When a user scans the message base, four lines are returned for each message in the system. The scan command will return something like the following:

```
Msg# 1557  Dated 08-17-84  04:49:43
From: TOM MACK
   To: ALL
   Re: QUBIE' & RBBS-PC
```

After enough messages have gone by, the user can choose to read any specific messages using the read command and the message number of interest. This bulletin board system, as well as others, automatically keeps track of which messages each user reads so that messages will not be read twice inadvertantly.

Messages and mail differ in the manner that observers can follow associated responses. The message system builds a chain of respones as they are entered into the system. The mail facility only handles the delivery of information and does not allow observers to create respones or follow the response flow.

The complete command menu for the RBBS-PC appears in figure 1.

```
==================    RBBS  MAIN  MENU    ===============

    PERSONAL COMMUNICATIONS     UTILITIES        ELSEWHERE
  B)ulletins       O)perator       H)elp           D)oors
  C)omment         P)ersonal mail  L)ines per page F)iles system
  E)nter message   Q)uick scan     X)pert on/off   G)oodbye
  I)nitial welcome R)ead messages  ?)Functions     U)tilities
  J)oin Conference S)can messages
  K)ill message    V)iew Conference
                   W)ho's on other node

Main Func <B,C,D,E,F,G,H,I,J,K,L,O,P,Q,R,S,U,V,W,X,?>?
```

Figure 1

1.2.3 NewsNet

The information which NewsNet [3] electronically makes available to its subscribers is drawn from newsletters, and therefore can be classified as a specialized information database. NewsNet states the value of its database in terms of what it would cost to subscribe to all the newsletters it represents. That estimate is approximatley $50,000 annually.

Newsnet allows its subscribers to rapidly search through current and back issues for information pertaining to specific subject matter. The savings of this service compared to manual measures are difficult to measure. As described in Online, "If a newsletter were to publish only one article in the entire year that you were interested in, typing a few commands on NewsNet would allow you to read it for about $1. You would save the year's subscription to the newsletter plus about 20 hours of turning pages."

NewsNet, in addition to its basic service offers a valuable feature to its users in its NewsFlash service. You can list up to ten sets of criteria. NewsNet will scan all new information added to its data base and inform you of its location when you log on.

NewsNets database is broken into groups of newsletters. These groups are identified by the industry they serve. There are currently about 40 groups with one to several dozen newsletters in each. To access a particular newsletter a 4 or 5 digit code is used. To retrieve the Daily Petro Futures newsletter the code EY01 would be used. The first two characters, in this case EY, represents the industry code for energy. The second 2 characters represent the periodical.

NewsNet uses "-->" as a command prompt. Refer to figure 2 for the complete command menu.

```
* * * COMMAND MENU * * *

READ.....Read full text          FLASH....Update NewsFlash
SCAN.....Scan Headlines          OAG......Official Airline Guides
SEARCH...Search for keywords     MAIL.....To publishers or NewsNet
HELP.....Detailed instructions   ORDER....Print subscriptions
PAGING...Change page parameters
PROJECT..Track online time
LIBRARY..Low-cost browsing       PASSWORD.Change your password
BACK.....Previous prompt         PRICES...Complete price list
QUIT.....Return to main prompt   INFO.....Information on services
OFF......Sign off                LIST.....Services by subject
```

Figure 2

Interaction with the NewsNet system can be achieved best by looking at the search command and the sequence of events which lead to the access of information.

```
Enter Command on <RETURN>
--> Search
```

After the NewsNet prompt the user enters the command "search".

```
Enter a Service Code, Help, Back, or Quit
--> EV LA RE
```

At the next prompt the user has several options on the material to scan. Service Codes, Industry groups or All are valid choices. In this example three industry groups are entered, they represent: Environment, Law and Real Estate, respectively.

```
Enter LATEST for the latest issue, or other data options
--> 5/1/85
```

The data parameter tells NewsNet to start at the date specified and search to the current date.

```
Enter Keyword(s)
-->
```

Keywords can be combined with Boolean operators to enhance the search capabilities. When NewsNet returns to the user with the number of occurances found to match the criteria, the user may view the data in several ways.

by headlines

by newsletter

by reading each article

1.2.4 OFFICIAL AIRLINES GUIDE

Official Airline Guide or OAG [3] also falls into the specialized database catagory. It is presented in this report because it contains a unique access command interface. The information, limited to airline scheduling, may only be viewed, users can not change data or make reservations.

Because the scope of information is limited to a particular subject group, the command structure is also quite small. See figure 3 for the OAG command structure.

```
         ** OAG COMMAND MENU **
ENTER:/I FOR INFORMATION AND ASSISTANCE
      /F FOR FARES DISPLAYS
      /S FOR SCEDULE DISPLAYS
      /M TO RETURN TO THIS MENU
      /U FOR USER COMMENTS AND
         SUGGESTIONS BOX
      /Q TO EXIT FROM THE OAG EE
```

Figure 3

From the list presented in figure 3, only the /F and /S are used to extract information on airline scheduling.

The OAG system offers several helpful features. The first being that when given a city for departure or destination if no air service is available the closest air service is given. Also available is the ability for OAG to inform on direct flights and connecting flights to the destination city.

Normally OAG returns information in a turse format such as:

```
1 6:05P SEA 750P RDM QX 409 SWN 1
```

However, a user can expand that information with descriptive titles using the "x" command. The same information using the expand feature is presented in figure 4.

```
  EXPANDED DIRECT FLIGHT DISPLAY
LEAVE-  6:05P    ON-27 JUN
FROM-SEATTLE;TACO M A,WA,USA
HORIZON AIR FLIGHT 409
AIRCRAFT-FAIRCHIL D  SWEARINGEN METRO
CLASS-COACH/ECONO M Y
ARRIVE- 7:50P    ON-27 JUN
AT-REDMOND,OR,USA
ELAPSED TRAVEL TIME  1H 45M
```

Figure 4

Another nice feature of OAG is allowing an expert as well as novice path through the system. OAG will lead a novice user through the system with prompts or an experienced user can lump information onto one line.

1.2.5 DIALOG

In contrast to OAG's limited subject scope is DIALOG [3]. DIALOG is an example of an encyclopedic database. This service allows access to over 250 data bases containing over 100 million records. Where OAG's command structure was simple and data dependent, DIALOG is quite complex and may change between each of its different databases. Due to the amount of information DIALOG contains and the complexity of the access, training courses are available. Potential users may select from half-day to day-and-a-half training sessions. Also available is a book "Guide to DIALOG Searching" and a "Bluesheet" for each database. DIALOG also has a monthly newsletter for its users.

It might appear an overkill of education just to search a database for information, but it seems that there is a science to the art of economical information retrieval especially when some databases are charging $300 per hour of access.

DIALOG's database is broken down into files and within these files are fields. In general these fields are identified in figure 5.

To search any database on DIALOG the files the search is restricted to is defined. This is accomplished by the command:

Select Files 213, 247

The next step is to issue the search criteria command. These can be very complex since single-word terms, Boolean expressions and field codes can be lumped into one command.

/AB	(abstract)This is the 200-word summary of the article.
/DE	(Descriptor) These are the single-word or multi-word terms assigned by the abstractor to indicate the primary subjects of the article.
/TI	(Title) This is the title of the article.
AU=	(Author) The person who wrote the article.
CO=	(CODEN) This is a five- or six-character code used by publishing industry to reference magazines journals, and other publications.
CC=	(Classification Code) A four-digit code that defines a topic area.
DT=	(Document Type) The type of document in which the original article appeared.
JC=	(Journal Code) Unique identifier to represent the journal.
JN=	(Journal Name) The name of the journal in which the article appeared.
PY=	(Publication Year) The year an article was published.
SN=	(ISSN: International Standard Serial Number) This is a unique eight-digit code assigned to each journal by the International Serials Data System.
UD=	(Update) Date of entry into the database.

Figure 5

Search Bulletin(w)Board

This command would search files 213 and 247 for matches to the keywords Bulletin and Board. In addition with the (w) modifier the word Board must follow Bulletin directly. This keeps the content of searches to be dependable. Another possible search command could be:

Search HOME/TI and AU= JAMES

Here information with "HOME" in the title and authorized by "JAMES" would be flagged for viewing.

1.2.6 CompuServe

CompuServe [3] is an exapmle of an information utility. Services often range from electronic mail and bulletin boards to electronic conferencing. Information access is strongest in the financial area but other areas include world news, weather and consumer reports.

CompuServe uses menus to allow users to navigate through the system. Figure 6 shows the main menu. From this menu users can respond the the system prompt '!' with single letter commands to scroll forward and backward to menu screens, summon help and even jump past several levels of menu screens.

```
CompuServe                 Page EIS-1

EXECUTIVE INFORMATION SERVICE

1 Communications             6 Shopping
2 Investments & Quotations   7 Weather
3 Decision Support           8 Professional & Technical
4 News                       9 Consumer Information Service
5 Travel                    10 User Information

Enter your selection number,
or H for more information.
```

Figure 6

The system leads users through a series of questions to determine precisely what information is desired. These questions vary depending on the different catagories of information. As an example the systems prompts the user with:

```
Company:
```

in the Investments & Quotations section of the system, whereas, the Weather section needs information about locations such as city and state.

A very powerful service offered by CompuServe is its electronic conferencing. This service allows an unlimited number of users to converse on subjects of common interests. This conference, analogous to a telephone conference, can be both private and public and can be saved so that referencing materials discussed may be printed.

1.2.7 Additional literature

Books also cover some aspects of electronic bulletin boards. "Going On-line With Your Micro" by Lou Haas and "Joys of Computer Networking" by Judy Barrett, give readers a basic education on using bulletin boards, describe the software needed to access remote computers and the type of systems available. However, the books do not discuss how to build a system.

Larry Myers' book entitled "How to Create Your Own Computer Bulletin Board" provides the closest outline of the steps necessary for creating a system. It falls a bit short, however, of the detailed information needed for a system which must handle information in the form of documents.

"<u>Setting up a Technical Information System</u>" by C.J. Schmidt and R.G. Murdick describes two characteristics that would be part of the this system. According to the writers an information system should fulfill the needs of the consumer and be designed to operate as economically as possible. The writers also outline four indexing principles; conventional alphabetic indexing, division of concepts into classes and subclasses, coordinate system in which a document is described by a set of terms, or a combination of the latter two. This document management system will also incorporate the second and third indexing principles.

1.2.8 PSDM

The prototype system for document management - PSDM, is not introducing any new concepts, but tailoring a collection of systems for a specified interest group.

In its simplist form, PSDM is a bulletin board system like the RBBS-PC. The thrust or goal of any bulletin board is for users to make the system valuable, that is, the information comes from the users and the bulletin board is only the transport device. In many ways the bulletin board system should be very simple so that it does not interfere with the business of sharing information.

This system does however differ from RBBS-PC by the type of information stored. The typical bulletin board system information is in the form of conversations. The type of information maintained in this system is in the form of documents and proposals.

Although NewsNet, OAG and DIALOG do not depend on users to add to their information pools, they do more closely relate to PSDM in that they have designed efficient mechanisms to allow users access to information. With the three informations systems surveyed, NewsNet, OAG and DIALOG offer three different file structure strategies. The OAG system falls under a normal predefined key structure database. This is very suited for the type of data it handles. This traditional approach does not work very well on information composed by different individuals and retrieved by matching keywords that attempt to match document content. NewsNet and DIALOG have logically broken up the information database into files or interest areas. Searching time can be reduced by limiting the area of the scan. This system will use a similar concept by which data will be placed in sub-directories for different interest areas. Utilizing the UNIX hierarchial file structure directories and data could be intermixed at great depths making it easy for users to find related information on topics of interest. The structure may appear as presented in figure 7.

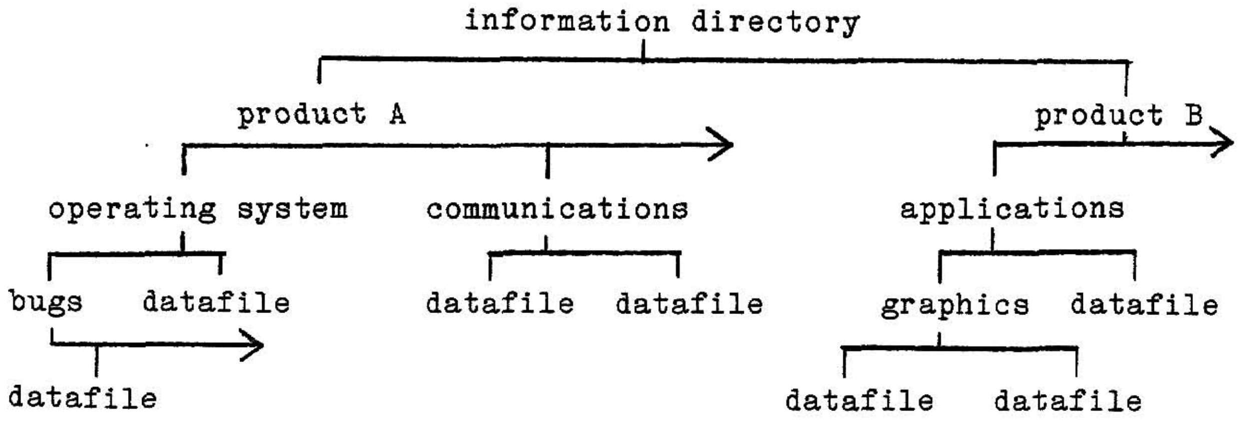

Figure 7

The ability to limit the amount of data a user must search through becomes increasingly important as the amount of data in the system grows. Forcing users to traverse the information directory and perform retrieval operations from specific areas of interest will greatly increase system response.

As stated previously, this system must be easy to use. Two possibilties exist for the main users interface: Menus or commands. With a wide variety of expertise for users a menu system would be the logical choice. However, for PSDM the potential user base will be experienced data processing personel. This will simplify the interface to a set of commands, along with an on-line help facility for user assistance.

Since menus will not be presented, the commands should have some meaning. Therefore, commands of length 1 will not be used as in the OAG and RBBS-PC systems. Rather than use an "a" to read a document, a command such as "view" will be used. As a future enhancement, an expert mode could be implemented. As in RBBS-PC this mode might allow truncated and even concatenated commands.

2.0 System Requirements

For the prototype system for document management to be an effective tool it must meet the following requirements:

* Documents handled in original form

Documents by their very nature must be taken as a whole. By separating a document into small manageable entities, much of its original meaning can be lost or misinterpreted.

* Easy to use

This particular system will undoubtly be used most by what are commonly referred to as casual users. It is therefore imperative that its functions are logical and command structure is designed for easy recall. A good help facility is definitely an important aspect of this requirement.

* Easily tailored

User expectations of the systems functions, as well as strategies for document filing, input and retrieval undoubtedly will not stay constant. The system must be flexible enough to accommodate changes which come from a variety of demand groups.

* Low maintenance

The cost of forcing users to continually bother with maintaince functions can weigh heavily on the benefits offered by the system. If the amount of work required to input and maintain the database becomes burdensome then users will gradually reduce the amount of data stored. This could potentially reduce the value and usefullnes of the system.

* Interactive

The word interactive suggests two modes of operation -input and output. Users of the system must have the capabilities to retrieve information in its original form as well as input thoughts and ideas in the form of documents.

* Selection based on content

This will be the most visible and important facility the system has to offer. The value of the program can be measured on how easliy users locate documents stored in the database. Users must have the capability to select information based on keywords as well as by subject classification.

* Security

The system must have a facility to restrict general public access in-addition to the prevention of unauthorized users from viewing documents containing sensitive material.

3.0 System Design

The prototype system for document management can be functionally separated into 3 phases:

* Initialization
* Command
* Maintenance

3.1 Initialization Phase

During initialization phase users identification is verified, accounting facilities are initiated, and the initial system notice file is displayed.

Users identification is verified by comparing a user entered id number along with password to the bulletin boards user log file. The user will respond to the system prompt:

```
ID:
PASSWORD:
```

for entrance to the system. The system uses log file for each id will contain information such as user name, location and type of access available.

Accounting facilities will be initiated in this phase to record the time and date the user started using the system as well as setting up pointers to buffer areas used during the command phase of the system. The system notice file will be the final action of this phase. This notice will only be presented at initial logon and will contain system news and alerts. This information will be maintained by the system administrator utilizing existing utilities. Therefore, no special design considerations are needed to maintain this part of the system.

3.2 Command Phase

From the command stage 8 command/states are possible:(see figure 8)

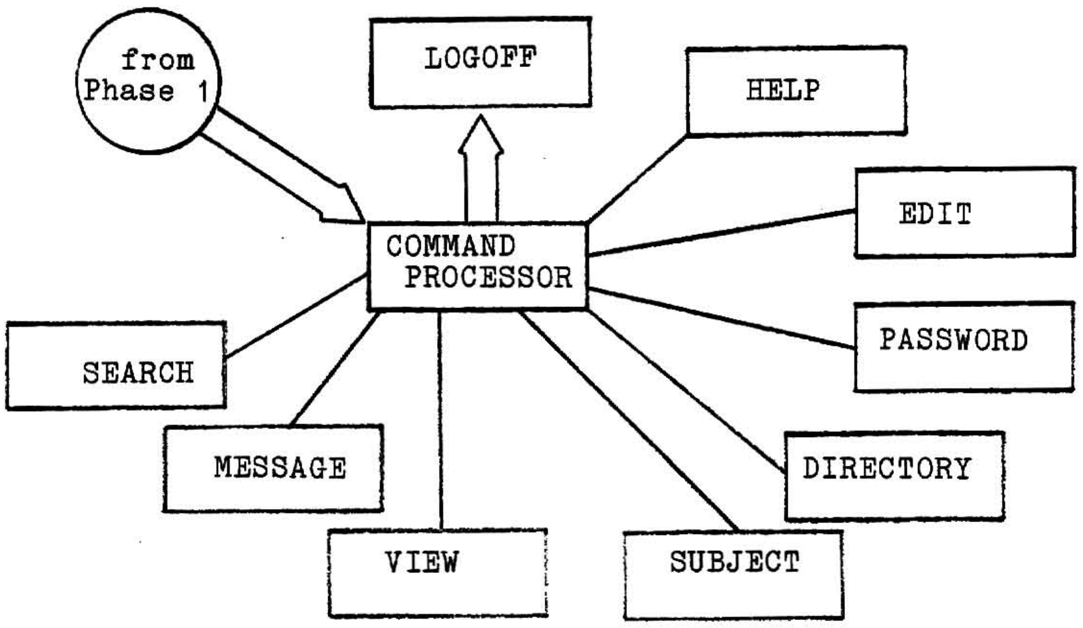

Figure 8

It is at this stage that the user will respond to the system command prompt with valid system command. The command prompt for PSDM will be a single word on a line by itself followed by a colon:

Command:

Commands will be entered by the following syntax:

```
<command> [<parameter> <parameter> ...]
```

where spaces separate each entity. Refer to appendix C for layout of the system commands.

3.2.1 Help

The help command performs exactly what its title describes. Users that have difficulty with the system may use the help feature to gain information on all commands. Both general information on how to use the system and detailed assistance on specific commands will be available. Each of the following would be valid help commands.

```
command:  help
command:  help help
command:  help view
```

3.2.2 Edit

The edit command will handle processing required to input information into the system. As stated previously, editing of input is handled through the unix editor "ed". Upon initiating the editing session, one of two paths must be declared. Either a file is being created or a file is being re-edited. Only the system administrator or the originating user can change an existing document. When editing a new document the document access must be declared. The two types of access are public and restricted. Users with preauthorized access to restricted documents would also have access to public documents. Existing documents can have their access type changed only by the originator or system administrator. It is the responsibility of the user to compose the first line such that it describes the document content. This line will be used by other features that aid in document selection.

```
command: edit {<file> | "new"}  <access type>
```

3.2.3 Password

The password command will give each user the ability to choose their own password. This will be the only user initiated access to the system user-log. Passwords will be restricted in length to nine characters and must be used.

```
command: passwd <password>
```

3.2.4 Directory

The directory command gives the user the ability to direct other retrieval and edit functions to specific areas of the system database. Users will work through the database building paths of interconnecting directories. The collection of paths that is currently being used is called the current working directory. In the initial release users will only be able to traverse down through directories and the associated sub-directories. To change the current working directory a user would have to start at the base or top directory and again build the directory path from that point.

```
command: dir
command: dir <sub-directory>
```

3.2.5 Subject

The first of two mechanisms to give the user some selectivity in viewing documents of interest is found in the subject command. This command will analyze all data documents contained in the current working directory and display documents name, creation date, type of access and the description line. The output produced by this command helps the user to select which document to view.

```
        command: sub
```

3.2.6 Search

The second mechanism of selectively retrieving documents is found in the search command. Here users give keywords which the information system will use to search for in the current working directory and each of the associated sub-directories.

UNIX utility programs will be used to perform the actual search. The system will only build the command string and pass it to the operating system. UNIX has a very powerful file-searching utility, 'awk'.

Using the 'awk' utility a file can be searched for a set of patterns and a specified action will be performed if a match is encountered. The user of PSDM will not have to worry about the syntax required to operate the 'awk' utility as that will be handled by the utility. One to five keywords may be used during each search through the directory.

As with the subject command subscribers may use the information provided by this command to select documents that are useful and should be viewed.

```
command: search <keyword>...
```

3.2.7 View

After the user has selected the documents that contain information of interest the user may issue the view command to display the entire document. Security is maintained at this point by comparing the current users access privileges with the requested documents defined access type. As with any command in the system that can potentially display more lines than a terminal can physically handle, the system will query the user as to when more information is needed. This ability will allow a full screen to be viewed and the user has complete control whether to continue with the remaining data to be displayed or move on to some other function.

```
command: view <file>
```

3.2.8 Message

The final function discussed in the design is the message command. Here messages will be saved in a separate directory for specified user. No special editing features will be implemented. To initiate this function, the message command will be issued along with the recipiants user-id. All data entered from the terminal, up until a single '.' is entered, will be written to the message directory. Users will be notified that mail exists for them during the initialization phase.

```
command: msg [<user-id>] [data...] ["."]
```

3.3 Maintenance Phase

The third and final phase of this design is initiated when the user logs off the information system by using the logoff command. At this point accounting information for the user is updated. System parameters saved in memory are written to the appropriate files. After all final phase maintenance is complete, control is returned to the operating system until the next user initiates access.

3.4 Summary of Design

The primary goal for PSDM is to not get in the way of allowing users to create and access data at a central site. The three phases of the design and the transition from the sections of the command processor should not present any difficulties for the user. A scenario of system commands is located in Appendix A and should serve beneficial in understanding the use of the system.

Certainly this system falls short of the information systems reviewed earlier, however, the requirements the users have for this system should find it easy to build upon the frame work specified.

To simplify the design of the system, existing utilities were used when possible. The UNIX operating system is made up of over 200 utility programs and is designed to allow easy access from with-in application programs.

MISSING PAGE

As an additional feature, research could be done to analyze the feasability of utilizing a relational data base system to make keywords to data documents. This enhancement may be required as the data base grows and sequential searching for keywords dramatically curtails system response.

4.1 Value of System

In general, most data processing applications are proportionally valuable compared to their use. Assuming PSDM contains information of use, accessing it will serve as a tremendously valuable tool. NewsNet states in their example that it's service could save twenty hours of turning pages. But that is when the source of the information is identified. In large corporations the source for answers can be impossible to find. With a central system users have increased the likelihood that questions and answers will be met.

Bibliography

Works Cited

1. Barrett, Judy. Joys of Computer Networking: Byte Book.

2. Haas, Lou. Going On-line With Your Micro: Tab.

3. Lambert, Steve. ONLINE: Microsoft Press, 1985.

4. Myers, Larry. How to Create Your Own Computer Bulletin Board Tab Books Inc, 1983.

5. Schmidt, C.J. and Murdick, R.G.. "Setting Up a Technical Information System", in Machine Design, Vol. 34, No. 11, May 10, 1962, pp. 124-129.

6. Wessel, Andrew. Computer Aided Information Retrievel: Melville Publishing Company, 1975.

Works Consulted

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

8. Mittmen and Bormen. Personalized Data Base Systems: John Wiley & Sons, Inc.

9. Nicols, Elizabeth. Data Communication For Microcomputers: McGraw-Hill Book Company.

10. Series 6000 Operating System Reference Manual: Copyright 1984 by Motorola, Inc.

Appendix A

Structure charts and module descriptions(see fidures A-1, A-2)

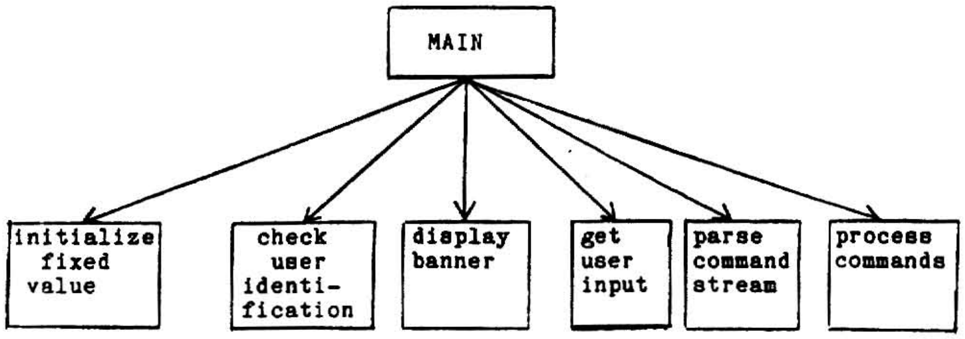

Figure A-1

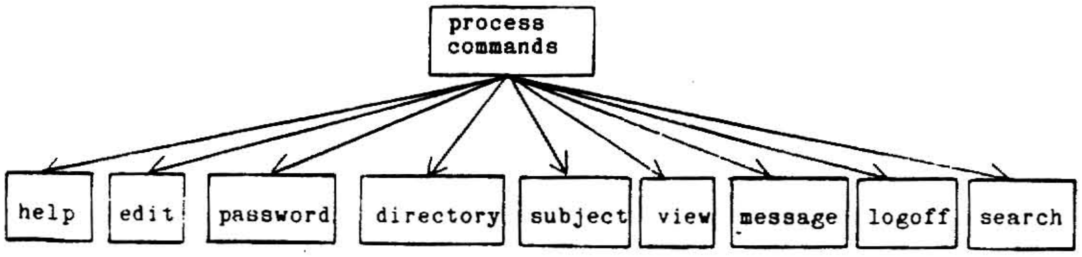

Figure A-2

Initialize fixed values - This module is called only once and performs two functions. First it assigns storage to buffer areas used throughout the system. Secondly, it sets the start time value so that the total system use may be calculated. [init_values()]

Check user identification - In this module, interaction with the user to identify and verify user authorization is performed. A five character user identification number is used to locate the user profile in the system user log. When the profile is located, the user is asked to enter a password which is then verified with the profile password. If either of these two tests fail the user is given an error message and the logon attempt must be tried again. [check_id()]

Display Banner - A banner file, created by the system administrator, is displayed on the terminal. [inline code]

Get user input - This module is used throughout the system when data must be received from the user. The data that is input is placed into a buffer. Cursor-return terminates the input stream. [getline()]

Parse command stream - The buffer used by get-user-input is evaluated so that commands and arguments are separated into isolated entities. Each command/argument string is pointed to by a unique pointer defined in the initialized-fixed- values module. As a system standard each string is terminated with a null character. [parse_cmd()]

Process command - This is the main module for command processing and is made up of modules that perform command tasks. See Figure A-2. [the following command modules can be viewed by refering to Appendix D]

Help - If the 'help' command is entered with no arguments then a file describing general help procedures is presented to the user. If an argument does accompany the command then the appropriate file associated with the argument is displayed on the users terminal.

Edit - In this module the user has the capability to create new documents and edit existing documents. For creating a new document the user will use the UNIX single line editor 'ed'. After editing occurs, the file is modified such that the first line contains the user-id of the creator, the date created and a subject line to describe the document. It is copied from a system defined temporary file to a unique file name in the users current working directory.

In the case of changing an existing document, the system first checks that the user requesting the operation is either the creator or the system administrator. Then the system will re-arrange the first line so that it appears to the user as it was originally entered. As described with creating a new document, 'ed' is used for editing. After editing is completed the first line is again updated and the data is placed in the same file and directory originally specified.

Password - This module allows the user to change their password. Checking is performed to insure that the length of the password is less than nine characters. The new password is copied to the user profile and the user log file is updated.

Directory - This module uses any arguments to create the new current user directory. It then displays file names that are located in the directory. Both data files and directories are displayed.

Subject - This module will read the files contained in the following information:

*the name of the file,

*creation/modification date,

*access type and

*the subject line.

If the file is a directory, the user is informed.

View - The user specifies the file to be viewed by an argument to the view module. The reference material or the first line is displayed with the appropriate headings.

Message - When the user issues this command one of two possible actions may occur. Messages can be retreived for the current user or new messages can be created. The message command with no arguments will cause the system to retrieve all messages in the message directory for the current user. After each message is read, it is deleted. If all messages are deleted than that user's message directory is deleted.

A user that wishes to generate a message includes as an argument the user identification of the recipient. All lines that the user enters, to a terminating line of '.', is sent to the message directory for the specfied user. If needed a directory will be created.

Logoff - This module will update the user profile with the current date and the user log file is re-written. The amount of time the user utilized the system is displayed. All files opened during the session are closed and pointers for work buffers are released. Control is returned to the operating system.

Search - This module uses the current user directory to search data files for user specified keywords. A maximum of five keywords may be used at one time, and are passed to the search module as arguments. The UNIX utility 'awk' performs the actual searching and reporting of matches. The search module builds the call to the shell program and provides file name and keyword arguments.

Appendix B

User Scenario

This scenario of the use of PSDM should help to understand both the concept and its use. Five general area's will be presented:

1. Logon to the system.
2. Create a new document.
3. Retreive documents using the directory, subject, search and view facilities.
4. Create messages.
5. Retrieve messages.

BOLDFACE print signofies user input.

```
$ psdm

id: 12181
password: dana
                ****    ***    ****
                 **     ****    **
                  ***  ******  **
                   *** ****** **     e l c o m e
                    ****    ****


          Four-Phase/Motorola S.E. Document Management
                         System


          system available times:
                                   7:00 - 8:00 pm. cst.
                                   Monday - Saturday


          Good luck!!!!!!

jim mullin          Thank you for accessing PSDM.


command: dir 6300

a01000
a01479
```

```
command: edit new p
0
a
subject line for this document
this area is the body of the document
the UNIX editor 'ed' is being used for
taking data from the user and in placing it
into a file
.
w
173
q
file name :/bbs/info/6300/a01545:


command: dir

6300
2000


command: dir 6300

a01000
a01479
a01545


command: sub

 file      date     access       subject
-------------------------------------------------------
a01000   07-29-85     r    first entry of value
a01479   08-02-85     p    this is the subject line
a01545   08-02-85     p    subject line for this document


command: search data document first
a01000  KEYWORDS MATCHED == 3
a01479  KEYWORDS MATCHED == 2
a01545  KEYWORDS MATCHED == 3
```

```
command: view a01545

Creator: 12181
Date: 08-02-85
Access: PUBLIC
Subject: subject line for this document

this area is the body of the document
the UNIX editor 'ed' is being used for
taking data from the user and placing it
into a file


command: msg 12181
->here is a message for user '12181'
->to be retrieved using the message
->facility
->.


command: msg

from: 12181 jim mullin
date: 08-02-85

here is a message for user '12181'
to be retrieved using the message
facility

 SPACE BAR TO CONTINUE, ANYTHING ELSE TO ABORT:



command: logoff

 Time used in seconds - 365

 END OF PROGRAM
$
```

Appendix C

Command Summary

directory - used to build current working directory

```
dir [<sub-directory>]

dir unix
```

edit - text editing

```
edit {<file> | "new"}  <access type>

edit a0023 r
edit new p
```

help - displays system and command information

```
help [<command>]

help view
help
```

logoff - terminates session

```
logoff
```

message - sends messages to users and reads messages

```
msg [<user-id>] [data...] ["."]

msg
msg 12181
this is sample data
.
```

password - changes user password

```
passwd <new password>

passwd abcdef
```

search - displays files that contain matching keywords

```
search <keyword>...

search system data 1985 ORACLE
```

subject - displays brief description of each file in directory

```
      sub
```

view - displays specified file

```
      view <file>

      view a00123
```

Appendix D

```
/*    PROTOTYPE SYSTEM FOR DOCUMENT MANAGEMENT                    */
/*                                                                */
/*    Author:  Jim Mullin                     Date: 08/01/85      */
/*                                                                */

/*##############################################################*/
/*      Global varaible definitions                             */
/*##############################################################*/
#include  <stdio.h>
#include  <sys/dir.h>              /* directory entry structure */
#include  <sys/stat.h>             /* STAT return values */
#include  <time.h>                 /* time structure */

#define PMODE  0644                /* access permission */

struct userdef {           /* definition for user_log layout*/
 char   u_id[6];
 char   u_pword[10];
 char   u_name[20];
 char   u_dept[4];
 char   u_loc[20];
 char   u_access[1];
 char   u_date[8];
 char   u_total[8];
 char   u_ext[1];
 };

long    tloc, start_time, rec_ptr, work_long;
int     work_int, work_v;
double  dbl_int, dbl_v;

char    buff[80];
char    buff2[80];
char    *argptr[10];
char    *search_ptr, *filcpy_ptr, *filcat_ptr;
char    *date_ptr, *tmp_ptr, *msg_ptr;

int     nargs;             /* number of arguments */
int     file_number;               /* file counter */
int     log_fd;            /* fd for user_log */
```

```
struct userdef userrec;
struct stat stbuf;                /* structure for file status */
struct tm *vtim;        /* structure for time data*/


static char *bbs_files[] = {     /* fixed system file names */
 "/bbs/user_log",
 "/bbs/welcome",
 "/bbs/info",
 "/bbs/help",
        "/bbs/msg",
 };

main()
{

 int c, i ,cmd, exit_flg;

 char *wdir, *p, *t, *file_cat(), *ecvt();

 FILE *fp, *tp, *fopen();

 exit_flg = 0;          /* set to non-zero to exit */

/*##################################################*/
/*     call routine to initialize space for buffers  */
/*##################################################*/
 init_values();

/*##################################################*/
/*     check user_log for valid user identification */
/*##################################################*/
 if ((check_id()) != 0) {
     printf("\n *** Invalid signon attempt ***\n");
     return(-1);
 }

/*##################################################*/
/*     valid user ---> display banner/welcome notice*/
/*##################################################*/
 if ((fp = fopen(bbs_files[1], "r")) == NULL )
      printf("bbs: can't open %s\n", bbs_files[1]);
         else {
      fileprint(fp);
      fclose(fp);
    }
```

```
/*#####################################################*/
/*      acknolwedge that we know who they are          */
/*#####################################################*/
 for (i=0; i<20; i++)
        printf("%c", userrec.u_name[i]);
 printf("  Thank you for accessing our bbs\n");

/*#####################################################*/
/*      main command loop, to exit set exit_flg <> 0*/
/*#####################################################*/
 while (exit_flg == 0) {

    printf("\n\ncommand: ");

           /*#########################################*/
           /*   fill "buff" with input stream        */
           /*#########################################*/
    getline();

           /*#########################################*/
           /*   break input stream into arguments
                for later use                         */
           /*#########################################*/
    parse_cmd();

           /*#########################################*/
           /*   set "cmd" to appropriate comand       */
           /*#########################################*/
    cmd=find_cmd(0);

           /*#########################################*/
           /*   using switch command process input */
           /*#########################################*/
    switch (cmd) {

      case 0:     /* error no command */
        p = argptr[0];
        printf("\n\n '%s' not a valid command\n", p);
        break;

      case 1:     /* help */
   /*#################################################*/
   /*   more than one argument, process as "help cmd"*/
   /*#################################################*/
```

```
     if (nargs > 1) {
        cmd=find_cmd(1);
        if (cmd != 0) {
            p=argptr[1];
            t = file_cat(bbs_files[3], p);
            if ((fp = fopen(t, "r")) == NULL )
                printf("bbs: can't open %s\n", t);
                     else {
                fileprint(fp);
                fclose(fp);
         break;
                }
         free(t);

         }
          else {       /*  no file found, not a command*/
            p = argptr[1];
            printf("\n\n '%s' not a valid command\n", p);
            printf(" try 'help help'.....\n");
            break;
          }

     }
/*################################################*/
/*   "help" alone,  give general information      */
/*################################################*/
     if (( fp = fopen("/bbs/help/help", "r")) == NULL )
         printf("bbs: can't open /bbs/help/help\n");
     else {
           fileprint(fp);
         fclose(fp);
       }

     break;

   case 2:      /* edit */

/*################################################*/
/*   check format, at least one argument          */
/*################################################*/
     if (nargs == 1) {
        printf("bbs: invalid format\n");
        break;
     }
```

```
/*#################################################*/
/*   edit.  first check second parameter.          */
/*      if it is "new" then                        */
/*          1.  insure were in a sub-directory     */
/*          2.  create an empty temp file          */
/*          3.  invoke UNIX "ed" editor            */
/*          4.  copy temp file to new file,        */
/*               including header info.            */
/*      if it is a file name                       */
/*          1. set up path name                    */
/*          2. copy existing file to temp,         */
/*              strip off header info.             */
/*          3. invoke UNIX "ed" editor             */
/*          4. copy temp file to old file,         */
/*              including updated header info      */
/*#################################################*/

    p = argptr[1];
    if (strcmp(p, "new") == 0) {
       if (strcmp(wdir, "bbs/info") == 0) {
          printf("bbs: must be in sub-directory\n");
          break;
        }
        else {
          system("> dune");          /*blank file */
          system("ed dune");
          new_filcpy(wdir);          /* copy it 'new' */
          break;
        }
    }

    else {

      t=file_cat(wdir, argptr[1]);
      old_filprp(t);
      system("ed dune");
      old_filcpy(t);

    }
    break;

  case 3:     /* passwd */

/*#################################################*/
/*   check format, at least one argument           */
/*   second argument, should not be > 9 characters /
/*#################################################*/
```

```
     if (nargs == 1) {
        printf("\n invalid command\n");
        break;
     }
     i = strlen(argptr[1]);
     if (i > 9) {
        printf("\npassword greater thna 9 characters\n");
        break;
     }
/*#################################################*/
/*   passwd.                                       */
/*          1.  copy argument to active user_log   */
/*          2.  put in seperator                   */
/*          3.  find starting location of user     */
/*                in user_log file.                */
/*          4.  write the user entry.              */
/*#################################################*/
     strcpy(argptr[1], &userrec.u_pword[0]);
     userrec.u_pword[i] = ':';
     lseek(log_fd, rec_ptr, 0);
     write(log_fd, (char *)&userrec, sizeof(userrec));
     break;

   case 4:      /* dir */

/*#################################################*/
/*   dir.                                          */
/*      if arg count = 1                           */
/*          1.  set working directory to           */
/*               bbs_files[2].                     */
/*          2.  call routine to display that       */
/*      if arg count > 1                           */
/*          1.  set working directory to current   */
/*               working directory + argument      */
/*          2.  call routine to display that       */
/*               directory                         */
/*#################################################*/
     if (nargs == 1) {
         wdir = bbs_files[2];
         directory(bbs_files[2]);
         break;
     }

     if (nargs > 1) {
         p=argptr[1];
         wdir = file_cat(bbs_files[2], p);
         directory(wdir);
         break;
     }
   case 5:     /* sub */

/*##################################################*/
/*   sub.                                           */
/*      call routine to display subject informatn  */
/*          regarding current working directory    */
/*##################################################*/
     sub_worker(wdir);
     break;

   case 6:     /* view */

/*##################################################*/
/*   view.                                          */
/*      1. set up path name.                        */
/*      2. open file name                           */
/*      3. pretty print header info.                */
/*      4. print document                           */
/*      5. close file.                              */
/*##################################################*/
     p=argptr[1];
     t = file_cat(wdir, p);

     if ((fp = fopen(t, "r")) == NULL)
        printf("bbs: can't open %s\n", p);
     else {
        filehdr(fp);
        fileprint(fp);
        fclose(fp);
     }

     break;
```

```
  case 7:      /* msg */

/*##################################################*/
/*   msg.                                           */
/*      if arg count = 1                            */
/*        read messages for this user               */
/*      if arg count > 1                            */
/*        create message for some other user        */
/*##################################################*/
     if (nargs == 1) {        /* read messages */
        msg_read();
     }
     else {
        msg_creat();
     }

     break;


  case 8:      /* logoff */

/*##################################################*/
/*   logoff.                                        */
/*      1. find offset into user_log for this user*/
/*      2. find current time                        */
/*      3. set user_log to current date/time        */
/*      4. calculate system time used.              */
/*      5. tell the user how long they were on      */
/*      7. write the user_log                       */
/*      8. set exit flag.                           */
/*##################################################*/
     lseek(log_fd, rec_ptr, 0);
     set_date();
     strcpy(date_ptr, &userrec.u_date[0]);
     work_int = tloc - start_time;
     printf("\n Time used in seconds - %d\n", work_int);
```

```
        write(log_fd, (char *)&userrec, sizeof(userrec));
        exit_flg = -1;
        break;

      case 9:     /* wdir */

    /*##################################################*/
    /*   wdir.                                          */
    /*      1. display where we are at                  */
    /*##################################################*/
        printf("\n\ncurrent directory - '%s'\n", wdir);
        break;


      case 10:    /*  search  */

    /*##################################################*/
    /*   search.                                        */
    /*      call routine to search working directory    */
    /*         for specified keywords.                  */
    /*##################################################*/
        search_worker(wdir);
        break;


      default:    /* get the rest of them */

        if ((fp = fopen("/bbs/help/default", "r")) == NULL)
            printf("bbs: can't open /bbs/help/default\n");
          else {
            fileprint(fp);
            fclose(fp);
          }


    }

  }
  printf("\n END OF PROGRAM\n");

}
```

```
/*##########################################################*/
/*check_id.                                                 */
/*     1. ask user for id.   "id:"                          */
/*     2. get user input                                    */
/*     3. search user_log for match on user_id.             */
/*     4. if user id found ask for password.  "password:"*/
/*     5. if password matches terminate user_id             */
/*          and user name with nulls for later processing.*/
/*##########################################################*/
check_id()
{
 int i, len;


 if ((log_fd = open(bbs_files[0], 2)) == -1)
    return;

 printf("\nid: ");

 getline();
 rec_ptr=0;

 while ((len=read(log_fd, (char *)&userrec,
         sizeof(userrec))) >0) {
    if (strcmp1(buff, userrec.u_id, 4) == 0) {
       printf("password: ");
       getline();
       i=0;
       while (userrec.u_pword[i] != ':') {
        if (userrec.u_pword[i] != buff[i]) {
             return(-1);
        }
       i++;
       }
       userrec.u_id[5] = '\0';
       userrec.u_name[19] = '\0';
       return(0);
    }
     else
       rec_ptr = rec_ptr + len;
 }
        return(-1);

}
```

```
/*##############################################################*/
/*old_filprp.   strip off header information for old document*/
/*      1. open old file  (name).                              */
/*      2. open temp file name  ("dune").                      */
/*      3. strip off user_id, date, access.                    */
/*      4. copy remainder of old file to temp file.            */
/*      5. close files.                                        */
/*##############################################################*/
old_filprp(name)
char *name;
{
 int c, tp;
 FILE *fp, *fopen();
 char buf[1];

 if ((fp = fopen(name, "r")) == NULL) {
    printf("bbs: can't open %s\n", name);
    return;
 }

 if ((tp = open("dune", "w")) == NULL) {
    printf("bbs: can't open temp file\n");
    return;
 }

 while ((c=getc(fp)) != ':')                /* bypass user_id */
    continue;

 while ((c=getc(fp)) != ':')                /* bypass date */
    continue;

 while ((c=getc(fp)) != ':')                /* bypass access */
    continue;

 while ((buf[0]=getc(fp)) != EOF)
    write(tp, buf, 1);

 close(fp);
 close(tp);

}

/*##############################################################*/
/*filehdr.   pretty print the file header information          */
/*     1. print title and data for creator, date, access and */
/*         subject.                                            */
/*##############################################################*/
```

```
filehdr(fp)
FILE *fp;
{
 int c;

 printf("\nCreator: ");
 while ((c = getc(fp)) != ':')
    putc(c, stdout);

 printf("\nDate: ");
 while ((c = getc(fp)) != ':')
    putc(c, stdout);

 printf("\nAccess: ");
 if ((c = getc(fp)) == 'p')
    printf("PUBLIC");
 else
    printf("RESTRICTED");

 getc(fp);

 printf("\nSubject: ");
 while ((c = getc(fp)) != '\012')
    putc(c, stdout);
 printf("\n\n");

}

/*##############################################################*/
/*new_filcpy.   creates unique file name and copies temp file*/
/*            header information inserted.                    */
/*       1. set up templet for unique file name.  Path + X's  */
/*       2. let mktemp replace "XXXXXX" with unique file name.*/
/*       3. inform the user as to the name we came up with.   */
/*       4. open both files, new file and temp file.          */
/*       5. copy header info, user_id, date, access.          */
/*       6. copy the rest of temp file to new file name.      */
/*       7. close files.                                      */
/*##############################################################*/
new_filcpy(p)
char *p;
{
 char *j, *t, *mktemp();

 char buf[1];
```

```
int c, z, i;

FILE *fp, *tp, *fopen();

int fd;

t = filcpy_ptr;

sprintf(t, "XXXXXX");
t = file_cat(p, t);
t = mktemp(t);

printf("file name :%s:\n", t);

if ((fd = creat(t, PMODE)) == -1) {
   printf("bbs: can't create file\n");
   return;
}

if ((tp = fopen("dune", "r")) == NULL ){
   printf("bbs: can't open edit temp file\n");
   return;
}
write(fd, (char *)userrec.u_id, 5);

buf[0] = ':';
write(fd, buf, 1);

       set_date();

write(fd, date_ptr, 8);

buf[0] = ':';
write(fd, buf, 1);


if (nargs == 3) {
  j=argptr[2];
  if ( *j == 'p' || *j == 'r' )
    buf[0] = *j;
  else
    buf[0] = 'r';
}
else
   buf[0] = 'r';
```

```
 write(fd, buf, 1);

 buf[0] = ':';
 write(fd, buf, 1);

 while ((buf[0] = getc(tp)) != EOF)
     write(fd, buf, 1);

 close(fp);
 close(tp);

return;
}

/*############################################################*/
/*old_filcpy.   copies temp file to specified old file name */
/*                header information inserted.              */
/*        1. open both files, new file and temp file.       */
/*        2. copy header info, user_id, date, access.       */
/*        3. copy the rest of temp file to new file name.   */
/*        4. close files.                                   */
/*############################################################*/
old_filcpy(p)
char *p;
{
 char *j, *t;

 char buf[1];

 int c, z, i, n;

 FILE *fp, *fopen();

 int tp;

 if ((fp = fopen("dune", "r")) == NULL ){
    printf("bbs: can't open edit temp file\n");
    return;
 }

 if ((tp = open(p, "w")) == NULL) {
    printf("bbs: can't open %s\n", p);
    return;
 }
 write(tp, (char *)userrec.u_id, 5);

 buf[0] = ':';
 write(tp, buf, 1);

        set_date();

 write(tp, date_ptr, 8);

 buf[0] = ':';
 write(tp, buf, 1);


 if (nargs == 3) {
   j=argptr[2];
   if ( *j == 'p' || *j == 'r' )
     buf[0] = *j;
   else
     buf[0] = 'r';
 }
 else
    buf[0] = 'r';

 write(tp, buf, 1);

 buf[0] = ':';
 write(tp, buf, 1);

 while ((buf[0] = getc(fp)) != EOF)
     write(tp, buf, 1);

 close(fp);
 close(tp);

return;
}

/*##############################################################*/
/*file_cat.    concatenates a file name unto the end path.     */
/*      1. get space for temp buffer to put the new path.       */
/*      2. copy all of old path name into buffer.               */
/*      3. add the required "/".                                */
/*      4. copy the file name to be appended.                   */
/*      5. terminate string with a NULL.                        */
/*##############################################################*/
```

```
char *file_cat(s ,p)
char *s;
char *p;
{

 char *v, *t, *malloc();
 int i;

 t = malloc(50);
 v = t;          /* save starting address */

 i=0;

 while (*s != NULL ){
    *t = *s;
    *s++;
    *t++;
 }

 *t++ = '/';

 while (*p != NULL ){
    *t = *p;
    *t++;
    *p++;
 }

 *t = *p;        /* NULL */


 return(v);
}

/*############################################################*/
/*find_cmd.    assigns appropriate command number to        */
/*          argument buffer.                                */
/*       1. search characters for match of commands.        */
/*############################################################*/
find_cmd(j)
int j;
{
char *p;
int i,k;

 p = argptr[j];
```

```
k=0;
i = *p++;
switch (i) {

       case 'h': /* look for help */

           if (strcmp(p, "elp") == 0)
               k=1;

           break;

       case 'e': /* look for edit */

          if (strcmp(p, "dit") == 0)
               k=2;

           break;

       case 'p': /* look for passwd */

           if (strcmp(p, "asswd") == 0)
          k=3;

           break;

       case 'd': /* look for dir */

            if (strcmp(p, "ir") == 0)
           k=4;

           break;

       case 's': /* look for sub */

            if (strcmp(p, "ub") == 0)
         k=5;

            if (strcmp(p, "earch") == 0)
         k=10;

           break;

       case 'v': /* look for view */

           if (strcmp(p, "iew") == 0)
         k=6;

           break;
```

```
        case 'm': /* look for msg */

           if (strcmp(p, "sg") == 0)
               k=7;

            break;

        case 'l': /* look for logoff */

           if (strcmp(p, "ogoff") == 0)
               k=8;

            break;

        case 'w': /* look for wdir */

           if (strcmp(p, "dir") == 0)
               k=9;

            break;

        default:  /* error due help */
            k=0;

 }

 return(k);

}

/*############################################################*/
/*parse_cmd.   break up input stream into arg cells.        */
/*       loop through buffer.                               */
/*         1. if character found copy it to unique arg untl*/
/*               space delimiter is found.                 */
/*         2.  terminate cell with NULL.                   */
/*############################################################*/
parse_cmd()
{

int len;
int offset;
int k, i;
char *p;

 nargs=0;
 offset=len=0;
 k=0;
```

```
  while ((len = cmdlng(buff, &offset)) > 0) {

        p = argptr[k];

        for (i=offset; i<len+offset; i++) {
          *p++ = buff[i];
        }

        *p = '\0';
        offset = len + offset;

        ++k;
        ++nargs;
  }

  for (i=k; i<10; i++) {
       p = argptr[i];
       *p = '\0';
  }
}

/*############################################################*/
/*cmdlng.   return length of characters starting at offset x*/
/*       in buffer s.  commands are terminated by a space.   */
/*############################################################*/
cmdlng(s, x)
char s[];
int  *x;
{

  int i,j,l;

  i = *x;
  j = i;

  while (s[j] != NULL) {
       if (s[j] == ' ')
          j++;
        else {

          *x = j;
                 i = j;
          while (s[j] != NULL)
            if (s[j] == ' ') {
                break;
             }
             else
                j++;
        return(j-i);

      }
 }

 *x = j;
 return(0);

}

/*###########################################################*/
/*getline.  fill buff with input characters until return key*/
/*###########################################################*/
getline()
{
int i, c;

 i=0;

 while (( c=getchar()) != '\012')
    buff[i++] = c;

 buff[i] = NULL;
}

/*###########################################################*/
/*fileprint    print specified file until end-of-file.      */
/*###########################################################*/
fileprint(fp)
FILE *fp;
{
 int c;

 while ((c=getc(fp)) != EOF)
    putc(c, stdout);
}

/*###########################################################*/
/*directory.   displays files found in a sub directory.      */
/*     1.  status UNIX for file information.                 */
/*     2.  insure that it is a legal directory name.         */
/*     3.  open and read the UNIX directory file.            */
/*     4.  check if blank entry. if so read next.            */
/*     5.  don't process the parent or this directory entry. */
/*     6.  print just the name of the file.                  */
/*     7.  and do the next until the end of directory.       */
/*###########################################################*/
```

```
directory(name)
char *name;
{
 struct direct dirbuf;
 struct stat stbuf;

 int i, fd;

 if (stat(name, &stbuf) == -1) {
    printf("bbs: '%s' not found\n", name);
    return;
 }

 if ((stbuf.st_mode & S_IFMT) != S_IFDIR) {
    printf("bbs: '%s' not a directory\n", name);
    return;
 }

 if ((fd = open(name, 0)) == -1)
    return;

 printf("\n");

 while (read(fd, (char *)&dirbuf, sizeof(dirbuf)) >0) {
     if (dirbuf.d_ino == 0)     /* slot empty */
        continue;

     if (strcmp(dirbuf.d_name, ".") == 0
        || strcmp(dirbuf.d_name, "..") == 0)
        continue;

     for (i=0; i < DIRSIZ; i++)
        printf("%c%", dirbuf.d_name[i]);

     printf("\n");

 }
 close(fd);
}

/*##############################################################*/
/*strcmp1.   compares two strings for a specified length l. */
/*##############################################################*/
strcmp1(s, t, l)  /* compare for length l */
char s[], t[];
int l;
{
 int i;
 i=0;

 while (i <= 1) {
    if (s[i] != t[i])
     return(-1);
   i++;
 }
 return(0);
}

/*############################################################*/
/*strcmp.  compares to strings.                               */
/*############################################################*/
strcmp(s, t)      /* return <0 if s<t, 0 s==t, >0 if s>t */
char s[], t[];
{
 int i;

 i = 0;
 while (s[i] == t[i])
    if (s[i++] == '\0')
       return(0);
 return(s[i] - t[i]);
}

/*############################################################*/
/*search_worker.   builds interface to "awk" to perform       */
/*                searches.                                   */
/*     1.  open current working directory as a file.          */
/*     2.  read entries until file name found.                */
/*     3.  print informational message for keywords found.    */
/*     4.  create path name for this file.  current working   */
/*          directory + file name.                            */
/*     5.  build UNIX executable command.                     */
/*     6.  interface to shell script for "awk" utility.       */
/*     7.  do this for all files in working directory.        */
/*############################################################*/
search_worker(name)
char *name;
{

 struct direct dirbuf;
 char *t, *j;
 FILE *tp;
 int i, c, fd;
 if ((fd = open(name, 0)) == -1)
    return;

 while (read(fd, (char*)&dirbuf, sizeof(dirbuf)) >0) {

    if (dirbuf.d_ino == 0)
       continue;

    if (strcmp(dirbuf.d_name, ".") == 0
        || strcmp(dirbuf.d_name, "..") == 0)
       continue;

    for (i=0; i<DIRSIZ; i++)
       if (dirbuf.d_name[i] == ' ')
        break;

    dirbuf.d_name[i] = '\0';

    printf("%s  KEYWORDS MATCHED == ", dirbuf.d_name);

    t = file_cat(name, dirbuf.d_name);

    j = search_ptr;
    sprintf(j, "sh awkin %s %s %s %s %s %s", t, argptr[1],
                 argptr[2], argptr[3], argptr[4], argptr[5]);

    system( j );

 }
}

/*#############################################################*/
/*sub_worker.   display header information for each file in*/
/*                 current working directory.                 */
/*     1.  open current working directory as a file.          */
/*     2.  read entries until file name found.                */
/*     3.  print column title line with seperator line.       */
/*     4.  print the file name.                               */
/*     5.  check if it is a document or sub-directory.        */
/*     6.  open files                                         */
/*     7.  by-pass user_id of creator but print date, access*/
/*         and subject line.                                  */
/*     8.  close file.                                        */
/*     9.  continue for each file in the sub-directory.       */
/*#############################################################*/
sub_worker(name)
char *name;
{
```

```
struct direct dirbuf;

char *t;

FILE *tp;

int i, c, fd;

if ((fd = open(name, 0)) == -1)
   return;

printf("\n");

printf(" file   date    access       subject\n");
printf("---------------------------------------------\n");

while (read(fd, (char*)&dirbuf, sizeof(dirbuf)) >0) {
    if (dirbuf.d_ino == 0)
       continue;

    if(strcmp(dirbuf.d_name, ".") == 0
       || strcmp(dirbuf.d_name, "..") == 0)
       continue;

    for (i=0; i<DIRSIZ; i++)
       if (dirbuf.d_name[i] == ' ')
       break;

    dirbuf.d_name[i] = '\0';

    printf("%s   ", dirbuf.d_name);

    t = file_cat(name, dirbuf.d_name);

    if (stat(t, &stbuf) == -1) {
       printf("    -- file error \n");
       continue;
       }

    if ((stbuf.st_mode & S_IFMT) == S_IFDIR) {
       printf("  -   Directory\n");
       continue;
       }

    if ((tp = fopen(t, "r")) == NULL)
       printf(" can't open");
```

```
     while ((c=getc(tp)) != ':')          /* bypass user_id */
        continue;

     while ((c=getc(tp)) != ':')
        putc(c, stdout);                  /* date */

     printf("    ");

     putc((getc(tp)), stdout);            /* access */
     getc(tp);                    /* by pass : */

     printf("    ");

     while ((c=getc(tp)) != '\n')
        putc(c, stdout);                  /* subject */


     close(tp);

     printf("\n");
 }
 close(fd);
}


/*###########################################################*/
/*msg_creat.   create a message for specified user_id.      */
/*     1.  concatenate user_id with "/bbs/msg".             */
/*     2.  check if directory is available to store message*/
/*         not create one. "MKDIR" one.                     */
/*     3.  create a unique file name "mktemp".              */
/*     4.  open the new file or message.                    */
/*     5.  insert house cleaning, from, date.               */
/*     6.  prompt the user for message "->".  copy input to*/
/*         file until terminating "." found.                */
/*     8.  close file.                                      */
/*###########################################################*/
msg_creat()
{
 char *name, *t, *mktemp();
 int fd, i;

 name = file_cat(bbs_files[4], argptr[1]);

 if (stat(name, &stbuf) == -1) {
    t = msg_ptr;
    sprintf(t, "mkdir %s", name);
    system(t);
 }
```

```
  t = msg_ptr;
  sprintf(t, "XXXXXX");
  t = file_cat(name, t);
  t = mktemp(t);

  if ((fd = creat(t, PMODE)) == -1) {
     printf("bbs: can't create file\n");
     return;
  }

  sprintf(t, "from: %s %s\n", userrec.u_id, userrec.u_name);
  write(fd, t, strlen(t));

  set_date();
  sprintf(t, "date: %s\n", date_ptr);
  write(fd, t, strlen(t));

  write(fd, "\n", 1);

  buff[0] = ' ';
  while (buff[0] != '.') {
    printf("->");
    getline();
    if (buff[0] == '.')
       break;

    write(fd, buff, strlen(buff));
    write(fd, "\n", 1);
  }

  close(fd);
}


/*############################################################*/
/*msg_read.    read all messages for currently active user*/
/*     1.  concatenate user_id with "/bbs/msg".           */
/*     2.  if no directory found, then no messages waiting.*/
/*     3.  if directory found read each file in directory. */
/*     4.  bypass parent and this entry.                  */
/*     5.  create path name, "/bbs/msg/" + user_id + file  */
/*     6.  open file.                                     */
/*     7.  print file using "fileprint".                  */
/*     8.  close file.                                    */
/*     9.  status user as to go on to next message or abort*/
/*    10.  loop through all message for user_id.          */
/*    11.  if all messages deleted, then delete directory. */
/*############################################################*/
```

```
msg_read()
{


 struct direct dirbuf;

 char *j, *t, *name;
 int i, c, fd;

 FILE *tp;

 userrec.u_id[5] = '\0';
 name = file_cat(bbs_files[4], &userrec.u_id[0]);

 if ((fd = open(name, 0)) == -1) {
     printf("\n\n No messages \n");
     return;
 }

 while (read(fd, (char*)&dirbuf, sizeof(dirbuf)) >0) {
     if (dirbuf.d_ino == 0)
        continue;

     if(strcmp(dirbuf.d_name, ".") == 0
        || strcmp(dirbuf.d_name, "..") == 0)
        continue;

     for (i=0; i<DIRSIZ; i++)
        if (dirbuf.d_name[i] == ' ')
        break;

     dirbuf.d_name[i] = '\0';

     t = file_cat(name, dirbuf.d_name);

     if (stat(t, &stbuf) == -1) {
        printf("    -- file error \n");
        continue;
        }


     if ((tp = fopen(t, "r")) == NULL)
        printf(" can't open");

     printf("\n");

     fileprint(tp);
```

```
      close(tp);

      if (next_chk() == -1)
         return;

      j = msg_ptr;
      sprintf(j, "rm %s", t);
      system(j);

      printf("\n");
   }
   close(fd);
   t = msg_ptr;
   sprintf(t, "rm -r %s", name);
   system(t);

  }

/*##########################################################*/
/*next_chk.  ask the user to respond for continued process*/
/*##########################################################*/
next_chk()
{

 printf("\n SPACE BAR TO CONTINUE, ANYTHING ELSE TO ABORT:");
 getline();
 if (buff[0] == ' ')
    return(0);
 else
    return(-1);
}

/*######################################################*/
/*init_values.  initialize space for temporary buffer */
/*######################################################*/
init_values()
{
 char *malloc();

 argptr[0] = malloc(40);
 argptr[1] = malloc(40);
 argptr[2] = malloc(40);
 argptr[3] = malloc(40);
 argptr[4] = malloc(40);
 argptr[5] = malloc(40);
 argptr[6] = malloc(40);
 argptr[7] = malloc(40);
 argptr[8] = malloc(40);
 argptr[9] = malloc(40);
 search_ptr = malloc(70);

 filcpy_ptr = malloc(10);

 filcat_ptr = malloc(50);

 msg_ptr = malloc(60);

 tmp_ptr = malloc(10);
 date_ptr = malloc(10);
 set_date();

 start_time = time( (long *) 0);

}

/*#########################################################*/
/*set_date.    calls time routine and formats date output */
/*#########################################################*/
set_date()          /* date_ptr = mm-dd-yy */
{
 tloc = time( (long *) 0);
 vtim = localtime(&tloc);

 if ( (vtim->tm_mon)+1 < 10)
    sprintf(tmp_ptr, "0%d", (vtim->tm_mon)+1);
   else
    sprintf(tmp_ptr, "%d", (vtim->tm_mon)+1);

 if ( vtim->tm_mday < 10 )
    sprintf(date_ptr, "%s-0%d-%d", tmp_ptr,
             vtim->tm_mday, vtim->tm_year);
 else
    sprintf(date_ptr, "%s-%d-%d", tmp_ptr,
             vtim->tm_mday, vtim->tm_year);

}

/*#########################################################*/
/*strcpy.    copy first specified string to second.       */
/*#########################################################*/
strcpy(s, t)        /* copy s to t */
char *s, *t;
{
 while (*s != '\0')
    *t++ = *s++;
}
```

PROTOTYPE SYSTEM FOR DOCUMENT MANAGEMENT

by

Jim Mullin

B.S., Missouri Western State College

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

1985

"In the U.S. alone, with the power provided by the now indispensable computer, we are creating new documents at a rate of over one million per minute." This quote, found in a recent computer periodical, provides a feeling for the vast amount of information being created. This prototype system is an attempt to develope a mechanism through which, individuals can manage and utilize the plethora of pertinant information available.

Much work has been done in the area of data processing. Information bounded by some criteria, or stated differently data as distinct entities, is easily managed by database management systems. To name a few: Oracle, Unify and System 2000 are very efficient in the handling of this type of data, but fall far short in the management of loosely structured documents.

This paper and its accompaning program will serve as a building block for a system to automate the functions of a manual document filing system. In developing this progect, currently available systems and literature are reviewed. Other areas of the paper are: functions of the system, an overview of the system design and possible future enhancements. The program itself will aid users in the collection, classification, storage and retrieval of documents as stated in the design criteria.