/AN ELECTRONIC CALENDAR SYSTEM IN A DISTRIBUTED UNIX® ENVIRONMENT/

by

DOUGLAS M. CLABOUGH

B. S., North Carolina State University, Raleigh, 1980

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:

Richard A. McBride
Major Professor

ACKNOWLEDGMENTS

I am indebted to many people and organizations for making this possible. First, I would like to thank AT&T Technologies for allowing me the time off each summer to come work on this degree. Second, I would like to thank my major professor and committee, Rich McBride, Virg Wallentine and Beth Unger, for the time they took out of their busy schedules to help in the preparation of this paper and project. Finally, there is my wife Peggy for her help in proofing this manuscript.

This is dedicated to Peggy and Brian for their understanding

during the many hours that I was away working on this degree.

TABLE OF CONTENTS

TABLE OF CONTENTS

Chapter One

1.1 Introduction

Current efforts in the area of office automation are concentrating on tools; each of which automates a specific office function. An electronic calendar is one particular tool of this type which aids in the scheduling of personal time, meetings, and common resources. [*Zisman*]

This paper deals with the extensions to an existing electronic calendar. [*James*] The new features provide a conference scheduling capability and a facility for updating calendars on remote machines. These features are important in the modern office environment because they save time in arranging meeting between several people and allows users to maintain a copy of their calendar on any UNIX® machine that they might have an account on. This calendar was originally implemented in the Pascal language and later translated into the "C" programming Language under the UNIX® operating system. In the process of translation, the program lost it's original capability to schedule conferences and it has never had the capability to maintain copies on more than one machine.

The scope of this project was limited to the modification of this program to implement the scheduling capability and the remote system update capability while maintaining calendar integrity and user privacy.

The rest of the report is broken into four chapters and several appendices. The remaining chapters are organized in the following manner.

The second chapter presents a review of the literature dealing with electronic calendars and scheduling; it also provides definitions and vocabulary in the area of office automation. Chapter three gives a detailed look at the work done on this project. The next chapter gives describes the enhancements made to the original calendar program. The last chapter gives a description of the results, some applications, and some possible future extensions to the work in the light of a survey of calendar users.

There are three appendices at the end of the report. Appendix 1 contains a discussion of the items that the UNIX® system administrator needs to be aware of. The second appendix contains a detailed discussion of how each module of the program works. The last appendix is the source code listing of the program.

Chapter Two

2.0 Review of the Literature

This paper deals with the area of networking and office automation. Specifically, it deals with the scheduling of meetings, appointments and the reservation of common office resources using an electronic calendar system in a distributed, multiuser office environment.

Zisman says that the move into office automation is not a revolution, but rather an evolution. He goes on to break this evolution into four stages: initiation, expansion, formalization, and maturity. In the second stage, one of the objectives is the replacement of "paper flows with electronic information flows". To achieve this replacement, the development emphasis will be on the development of tools to automate certain office functions. These tools include such things as form filing systems, document preparation, electronic mail, and electronic calendars. [*Zisman*]

What is meant by "distributed systems"? Bochmann defines a distributed system as consisting of "several interacting components" with the degree of coupling between the systems varying from weak to strong. [*Bochmann*] For the purposes of this paper, a distributed system is two or more computers interconnected using the UNIX® uucp protocol or a compatible protocol. With this definition, the computers are the components, the uucp connections is the coupling mechanism, and the interaction between the systems consists of uucp file transfers and remote command execution by the users on the systems. [*Nowitz*]

In distributed systems, one of the problems that must be addressed is how to share data among users. There are three types of problems associated in this environment. They are lost updates, dirty reads, and unrepeatable reads. Dirty reads are reads that collide with a another user's write, producing invalid data. Unrepeatable reads are reads that occur after another user has modified data and not written it yet. A second read of the same data record will produce different data. Locking of files, records, or devices such that only one user can read/write at a time eliminates these problems. [*Bayer*] In this way, race conditions are eliminated and data integrity is maintained.

The case that Jeffrey Holden puts forth for electronic mail is even more applicable to the scheduling of meetings. [*Holden*] He says that in a study of calls placed by him, only 26 percent of them went through the first time. In other words, to talk with one person, Holden had to attempt to call that person, on average, four times. This is not very surprising in light of the study that Barcomb mentions that shows that managers and professionals spend 40 to 70 percent of their time in meetings. [*Barcomb*] Holden goes on to show how an electronic mail system saves time and trouble. In scheduling meetings, many more people must be contacted, usually more than once. If an electronic calendar system were available that could check the calendars of those who need to attend, then the calendar program could very quickly tell the person scheduling the meeting the common blocks of time that everyone had open in the calendar.

In his book on office automation, David Barcomb describes several characteristics of an electronic calendar/reminder system. These characteristics include the capability to keep track of an individual's schedule, the capability to schedule both meetings and the use of common resources, and an automatic reminder service (ticklers). [*Barcomb*] Also, while the calendar system should be implemented to be easily and conveniently used, it should not sacrifice the privacy of personal calendar information. [*Greif*]

In David James' original implementation of a calendar system, many of these concepts were incorporated. His calendar, implemented in Pascal, was menu driven and allowed for the insertion, deletion and viewing of entries in a person's calendar. Also the automatic scheduling of meetings used the calendar files of the proposed participants to determine an available block of time. [*James*]

Chapter Three

3.1 Project Goals

At present, KSU has an electronic calendar system that keeps track of appointments on an individual basis. This is the same system that was implemented by David James discussed above. This program was recoded in "C" language sometime after David's report and in the process, the capability to schedule meetings was lost. The objective of this project was the modification of this system so that a person could once again tentatively schedule a meeting or appointment with multiple individuals. It would accomplish this by presenting the person scheduling the meeting with a sequence of times for which all of the principals involved in the meeting are free. When a meeting time is picked, each person involved in the meeting would have his calendar marked with a tentative appointment. Also, electronic mail would be sent notifying him that he must confirm or reject that appointment which has been scheduled. The system was also to be extended to a distributed environment, such that a person with calendars on several machines, could automatically update all the calendars with the same information.

3.2 Design Specifications and Implementation Considerations

3.2.1 User requirements

The completed system had the following user requirements and specifications for the new commands:

3.2.1.1 To schedule a conference

1. The user will be prompted for the user identifiers of the proposed meeting participants.

2. The program will then use these identifiers to access the system password file to find the home directory of the identifier. If an identifier is not found in the password file, a message is issued to the user and that identifier is dropped from the list to be scheduled.

3. For each valid login identifier, the home directory is found from the passwd file entry and from this, the path to the calendar directory is created. This directory is then checked to see if a "LOCK" file exists (This "LOCK" file is a special file generated by the program to tell other ecal processes that this particular file is busy). If so, a message stating that this person's calendar is busy is issued and the user is requested to try again later. If no "LOCK" file is found, one is created and an entry is made into a list of active calendar files.

4. Once all calendars are acquired, the user is asked to specify a range of times in which to try and schedule the proposed meeting. This range of times is then searched for times in which every participant is free and common block of free time is then displayed. Based on this information, the user is asked to specify a specific proposed meeting time. This chosen time is then rechecked for conflicts and, if none are found, the meeting entry is made in each calendar. If a conflict is found, the user is asked to try again.

Finally, mail is sent to all proposed participants, requesting confirmation of attendance. Each individual will be able to do this using the confirm command. This command will allow the user to delete or confirm the tentative meeting in their calendar and will automatically send mail to the scheduler, informing them of the action taken.

3.2.1.2 To erase old calendar entries

The program scans all entries in the user's calendar file. It then marks as inactive all entries that are prior to today's date and moves all other entries to the front of the file. When it is done, all active entries are at the front of the file in the same order as they were originally and all inactive entries are at the end of the file. The inactive entries at the end of the file will be used as needed to add new entries. In this way, a lot of inactive entries will not accumulate at the front of the file and the overall file size will tend to be smaller than it otherwise might. The routine does not prompt the user for any input.

3.2.1.3 To update a calendar on a remote machine

The program first calls the erase routine (see above) to ensure that no out of date data is transmitted. The user is then prompted for the name of the machine on which the remote calendar exists and the login identifier of the owner of the calendar on that machine. The calendar file is then scanned and the active data in it is converted to ASCII representation and then sent to the remote machine via the uux(1) command of UNIX®. The owner of the file on the remote machine is also sent mail informing him that someone is about to update is calendar and giving him the name (login identifier) and system from which the update request is sent. uux(1) will send the requester return mail concerning the success or failure of the ecal update on the remote machine.

When the update request arrives on the remote machine, the data file is "catted" and piped to ecal's standard input. The update routine then reads the first two lines of the file and gets the system name and login identifier from the requesting system and the login identifier of the owner of the file to be updated. The system name and login identifier that made the request is put in a temporary file. The rest of the file is then read and the information is converted to the proper formats and a regular calendar insertion is done. Any insertion conflicts are added to the temporary file above. At the end of the update, the temporary file is mailed to the owner of the file being updated. This way, the owner is notified of the update and of any conflicts found so that they can manually resolve these conflicts at a later date.

3.2.2 Design Considerations

The completed system also had the following general requirements and specifications

Since the new system had the potential for several users to try to access the same calendar files, a method was required to prevent simultaneous writes to the same file. This was done through the use of "LOCK" files.

Care must be taken that any "LOCK" files be removed in the event that the program is abnormally terminated. This meant that all signals (see signal(2) in section 2 of the *UNIX User's Manual*) had be caught to avoid having a termination that leaves a "LOCK" file laying around. Unfortunately, a signal "9" cannot be caught, so it is still possible for a "LOCK" file to be left around. This led to the need for the ignoring of any "LOCK" files older than a certain threshold of time.

The use of "LOCK" files meant that the "LOCK" file check/creation must now be done to a user's own calendar before they can be allowed access to it. The "LOCK" files also required that all of a person's calendar information to be kept in a single data file, instead of the many (unlimited number?) originally used [*James*] and an equal number of "LOCK" files.

The program had to ensure that the owner had some privacy of the data in their calendar file while at the same time allowing other users use of the necessary information with which to schedule meetings.

The required capability to update calendars on a remote machine combined with the requirement that the program must be able to open and write to any user's calendar gave rise to the possibility of the unauthorized modification of a calendar file by someone other than the user. In order for the program to open any user's calendar file for the scheduling conferences, it became necessary to run the program with root ownership and the SUID bit set. This raised the requirement that the user not be allowed to escape the program or execute shell commands with the effective UID of root. [*Ritchie, Grampp*]

The program also had to be easily portable to different UNIX® machines. Along this line, all system and library calls used were kept to a very common set of routines that exist on most all machines. The successfulness of this attempt is seen by the fact that the source code, at the time of this writing, has been compiled and run on an AT&T 3B2 , AT&T 3B20, and a VAX 11/780 running 5.0 UNIX®, a VAX 11/780 running 4.2 UNIX® and a VAX 11/780 running 4.2 BSD UNIX®. The BSD version has just one (1) extra line to it than did the version that ran on the other machines. [*Johnson, Ritchie*]

3.3 Order in which work was done

As mentioned before, the program that I started with was a translation of David's original Pascal into "C" language. The work started with converting the format of the source code into a format that was familiar and then studying the code very carefully to see what was done and why. When it was understood how the program worked, work then proceeded to convert the program to work using a single data file per user and made these changes work before continuing.

After the first modifications were working, the routines for the scheduling of conferences were written. This took several iterations of coding and debugging. In the end, these routines included the locking routine, signal trapping and handling routines, and the globalization of several variables and data structures since they needed to be known by the signal handling routine and could not be passed to it. A cleanup routine was also implemented that the calendar system was forced to exit through to ensure that all of the lock files were removed.

The last item implemented was the remote machine update. This feature required two routines. The first routine scanned the calendar data file on the local machine, creating a file in a specific, known format. The calendar program on the remote end uses this file in conjunction with the second routine to update the calendar data on the remote machine. Conflicts found during the update were stored and mailed to the owner of the calendar on the remote machine to help ensure data privacy and security.

Chapter Four

4.0 Enhancements made to the original Program

Besides the previously described additions made to the calendar program just described, the following enhancements were also made to the program in the process.

All of the data is now stored in a single binary data file, owned by "root" and having a mode of 400. This ensures that a user must employ the calendar program (or know the root password) in order to see the contents of the calendar's data. Therefore, the security and privacy of the data is greatly enhanced. This single data file also greatly reduces the number of i-nodes (the i-node is the data structure that UNIX® uses to keep track of files in the file system) used by the system while not using any more of the actual disk space (128 bytes per record).

The use of binary data directly through the buffered binary read/write routines of the standard I/O package also serves to increase the I/O throughput of the system. This is because the program does not have to run "fprintf" every time a write is done or "fgets" and "sscanf" whenever a read is done. Also, since only one file per user needs to be opened, it is now possible to leave that one file open all the time. This saves opening and closing a file every time some form of I/O is done. Also, the six month look ahead limit of the original calendar has been extended to one year.

The disk update routine was rewritten such that it now only writes to disk those entries that have been changed since a day's calendar was read into memory. This cuts the disk write activity to a minimum, since in most cases only one or two entries need to be written in any given session.

While undocumented in the menu selection, a shell escape is available now. This was very helpful in debugging the code and has been left in. It is implemented in such a way that any routines executed by it lose the root effective UID which the rest of the program runs under.

An edit command was also added to the system. This allows a user to edit a description for an already existing appointment. It is implemented using the same routine as the shell escape in the previous paragraph and allows the user the choice of editors to use.

Chapter Five

5.1 Potential Benefits of the System

This project presents one way of implementing a calendar system in a multiuser and/or a distributed environment while still maintaining some data security and privacy. It should be very useful to any group of people having a need to schedule meetings among themselves and to those who find it necessary to maintain a calendar on more than one computer system.

5.2 Possible Extensions to the System

This is a first attempt at moving this electronic calendar into a distributed environment. Originally, the calendar was going to allow the scheduling of meetings with principals with logins on several machines. However, the uucp network proved to be to slow and unreliable for such a function. When a network is available that will allow access to files on remote machines in the same manner as they are available on the local machine (open, read, write system calls work), then the scheduling through remote machines can be added with very little effort.

A third improvement to be made to the program is in the area of the LOCK files. The method used has a race condition inherent in it. For instance, assume that two processes (A and B) want the same file. Process A checks for the existence of the LOCK file and finds it non existent. Before it can create the LOCK file, it is swapped out and process B is swapped in. Process B now checks and finds the file unlocked and proceeds to lock it and go on with its business. Later, if A is swapped back in before B releases the LOCK, then A, having already found the LOCK file nonexistent, will also create a LOCK file and each of the processes will think that it has exclusive access to the file.

The solution to this will require a method of checking for the lock and locking within a single system call. This type of solution would involve modifications to the UNIX® kernel, either in the writing of a device driver that implements a pseudo device to perform this function or the actual addition of a system call to the operating system. Either of these alternatives proved impractical to implement due to the lack of an available system on which to experiment with kernel modifications.

5.3 Recommended Features for Electronic Calendars

C.M. Kincaid [*Kincaid et al*] recently published the results of a survey involving users of electronic calendar systems. This article included a summary of recommendations made by these users as to what features an electronic calendar should have. These recommendations and a comparison of how this calendar system, as currently implemented, meets these recommendations follows.

1. "There should be a daily format in which all events are recorded. The format should be as unstructured as possible, requiring only that optional begin/end times be in some fixed location. There should be no restriction on the amount of text per entry." The present calendar implementation has such a daily format, i.e., begin/end times and a description. However, the format is structured and the begin/end times are required. There is also a 100 character limit on the description. This limit could be changed relatively easily though.

2. "The calendar should allow certain keywords within the daily entry to be marked as "event descriptors" which will appear in a weekly or monthly condensed format." There is no provision made for keywords or condensed format or summarized displays in the current implementation.

3. "A weekly calendar format should summarize the daily events and then present them on a chart showing the event begin/end times." Again, there is no provision for such a feature in the current implementation.

4. "Events should have an optional alarm/reminder time at which the event will appear on the screen." An alarm/reminder function is not implemented either. Such an implementation could be done using a daemon, running in the background, that keep a record of who has requested alarm reminders and sending them at the appropriate time.

5. "The calendar should allow the user to access the next/previous day, week, or month easily. It should also allow a specific date to be accessed." This is implemented in the current calendar system.

6. "The user should be able to switch between daily and weekly or monthly formats while maintaining the current date across all display formats." There are no current weekly or monthly formats.

7. "When an electronic calendar is part of an integrated electronic office system, the calendar should be easily accessible from anywhere in the system." At the moment, this system is not part of an integrated office system.

8. "Events should be allowed to span several days and to be inserted automatically in each day's calendar." This is not currently implemented in such a way that it can be done automatically. Such an implementation is possible but would require modifications to the routine that checks for schedule conflicts and to the routine that requests the begin/end times of events. However, by separately scheduling the event for each day, the user can accomplish this function manually.

9. "The range of the calendar should be unlimited. Events should be scheduled as far in advance as desired. Old entries should be kept as long as the user feels it necessary. An archiving facility should be available." The unlimited scheduling

is implemented. However, past entries are not currently accessible. An archiving facility does not currently exist, but could be added relatively easily.

10. "There should be included a monthly format condensed from the daily calendar and describing a day's events without specifying times." As mentioned above, there are no condensed or summarized formats in the current implementation.

11. "The user should be allowed to block out or reserve the same time period over a range of days, weeks, or months." While this is not specifically implemented, it is possible by using the "day of week" mode for each day and leaving it there for as long as needed.

12. "The system should notify a user, when logging in, of existing calendar events for the day." While not implemented, it could be easily added in the form of a routine that prints the current day's calendar entries and exits. The calendar system would then invoke this routine when it sees some option in the command line. The user would then just have to include a call to the calendar with this new option in their ".profile".

13. "A scheduling facility should be available." The functions recommended for this scheduler are the following:

> a. "Users should be allowed to specify who is to attend (not necessarily themselves)." The current implementation allows the user to specify who is to attend, the only requirement is that all attendees must have an account on the computer system doing the scheduling. However, the calendar does require that the user requesting the meeting be included in the scheduling computations.
>
> b. "The specification of a meeting's time range should be allowed." This is implemented on the current system.

c. "All possible meeting times should be presented, and the user permitted to select the most appropriate time." This is also implemented in the current system.

d. "A warning should be given when conflicts arise, but conflicts should not prevent a meeting request from being sent." The current implementation will not schedule a meeting if there are conflicts.

e. "Each participant should be notified of the tentative meeting, and the system should request a response from each. The participants, however, should be allowed the option of postponing their reply to a later time." Currently, participants are notified by mail of the tentative meeting and are asked to respond by return mail. Thus, the participant can postpone his reply by saving the mail message and coming back to it at a later time.

f. "The person setting up the meeting should be allowed to automatically cancel or confirm the meeting, with a notification sent to all participants." This feature is not currently implemented directly. The user can however include in his description of the meeting that attendance is mandatory.

g. "It should be possible to book resources along with meetings." This is implemented by giving each schedulable resource a "login" on the computer system and thus, its own calendar file.

In addition to the preceding features, I would like to add to the above list that a calendar should be able schedule meetings in a distributed environment. While this calendar does not fully accomplish this last goal, the program has been so structured that it should be relatively easy to add this feature when a network is available that meets the requirements of such a feature.

It should be stressed that this survey deals with features that seem desirable to users. There is no calendar system that currently implements all or even most of these features. Thus, this survey serves as a list of possible extensions not only to this calendar system, but to any calendar system. The calendar system described in this paper meets as many or more of these requirements as any other calendar system, making it an excellent calendar for both single machine environments and distributed environments.

Bibliography

[1] Barcomb, David, "Electronic Calendars", *Office Automation: A Survey of Tools and Technology*, Digital Press 1981, Pp 115-123.

[2] Bayer, R., Graham, R.M., and Seegmuller, G., "Operating Systems: An Advanced Course", *Lecture Notes in Computer Science*, Vol. 60, Springer-Verlag, Berlin, 1978, Pp. 430-438.

[3] Bochmann, Gregor, "Architecture of Distributed Computer Systems", *Lecture Notes in Computer Science*, Vol. 77, Springer-Verlag, Berlin, 1979, Pp. 1-26.

[4] Grampp, F.T. and Morris, R.H., "UNIX Operating System Security", *AT&T Bell Laboratories Technical Journal*, Oct. 1984 Vol. 63, No. 8, Part 2, Pp. 1649-1671.

[5] Greif, Irene, "Computer Support for Cooperative Office Activities" *MIT-LCS*, April 1982.

[6] Holden, Jeffrey B., "Solving Communication Problems - The Case for Electronic Mail", *IEEE 1980 COMPCON FALL*, 1980, Pp65-68.

[7] James, D., *A UNIX Based Electronic Calendar System* A Masters Report from Kansas State University, 1982, Pp 1-42.

[8] Johnson, S.C. and Ritchie, D.M., "Portability of "C" Programs and the UNIX Operating System", *The Bell System Technical Journal*, July-August, 1978, Vol 57, No 6, Part 2, Pp 2021-2048.

[9] Kincaid, Christine M., Dupont, Pierre B., and Kaye, A. Roger, "Electronic Calendars in the Office: An Assessment of User Needs and Current Technology", *ACM Transactions on Office Information Systems*, Vol. 3, No. 1, Jan. 1985, Pp. 89-102

[10] Nowitz, D.A., "UUCP Implementation Description", *UNIX Programmers Manual*, Vol 2, AT&T Bell Laboratories.

[11] Ritchie, D.M., "On the Security of UNIX", *UNIX Programmers Manual*, Vol 2, AT&T Bell Laboratories.

[12] Zisman, Michael D., "Office Automation: Revolution or Evolution?", *Sloan Management Review*, Spring 1978, Pp. 1-16.

APPENDIX 1

This will be a discussion of installation and administration requirements for the program.

The program must be installed in /usr/bin or some other location that is included in the search path of both general users and of incoming *uux* commands.

When compiled and installed, it must have the name *ecal* and be owned by *root* or some other login identifier with a numeric user ID of zero (that of the super user). It should have a mode of 4755 (set effective UID on execution).

The */usr/lib/uucp/L.cmds* file or its equivalent must allow execution of the programs *cat* and *ecal* to allow update of calendars on remote machines. This is the file that tells uucp which programs a remote machine may execute on the local machine.

APPENDIX 2

DETAILED DESCRIPTIONS OF NEW AND CHANGED MODULES

1.0 Changed Modules

Reading the following along with the source code will make it much easier to understand what is happening in the source code. Much of the comments from the source code are included here but the descriptions here go into much greater detail than the source code comments do.

1.1 Header Declarations

The defined variables, global variables, and data structures of interest that are defined at the beginning of the source code are described here.

BSD: This is a conditional compile flag. It should be defined as "1" if the program is being compiled for a 4.X BSD system. It should be undefined for AT&T system 3, 4.0 or 5.0 Unix systems.

DESC_LENGTH: This defines the maximum length of an appointment description.

MAX_APP_NUM: This is the maximum number of appointments a user is allowed to have for any one day. It is essentially limited by the amount of memory a user process is allowed to have.

HOURS: This defines how long a "LCK" file is valid. A running program will update it's LCK files every "HOURS / 2" time period. Any LCK file older than HOURS old is assumed to have come from a program that is no longer running.

MAX_PARTIC: This is the maximum number of participants in a conference scheduling attempt. It is limited by number of file descriptors the particular system allows any one process to have open at any given time. In general, this is:

20:	on most systems, this is the maximum number of file descriptors a process is allowed to have open at one time.
-3:	stdin, stdout, and stderr uses 3 of these descriptors.
-2:	The program uses 2 other descriptors for general purposes.
15:	Thus, 15 are left to be associated with the ecal data files of participants.

sch1: This is the data structure for the modified program. The 4 extra characters in the information field and the extra variable ("a") provides an even 128 bytes (on 32 bit machines) in the data structure that is properly aligned on word boundaries. It consists of short integers that store the: encoded beginning time, encoded ending time, month number, day of week number, day of month number, year number, and three short integer flags, (one used to indicate if the record is for a specific date, one used to indicate if the record has been modified and not yet written to disk, and the third is used to indicate active or non-deleted data). The next two variables are regular integers. The first one stores the record number where the data record is found on the disk. The second one is used by the conference scheduling routines to store the user ID number of the user requesting a tentative conference The last variable in the data structure is a character string of 104 characters in length. The 4 characters over the 100 defined to MAX_APP_NUM is used as filler to bring the size of the structure to 128 bytes. This ensures that the data records on disk will not cross block (either 512 or 1024 byte) boundaries.

partc: This is an array of schedules used to keep track of requested participants in a conference scheduling attempt. The zero index into the array always accesses the current user's schedule.

1.2 **main()**

The routine "setjmp(env1)" is called to set the longjmp return point before the traps set. This way it is known where to return to before the traps are set. This point is moved forward by latter calls to "setjmp()" as things are done that either don't need to be repeated or would cause problems if repeated. Then "set_traps()" is called. This ensures that any "LOCK" files the program creates can be removed before it terminates. The only exception to this is a termination caused by signal 9 (the kill signal), which cannot be caught. However, since this routine runs with an effective user identifier of "0", the only identifier that can send us that signal is root. Thus, it very unlikely that the program will receive this signal.

Next, the routines are run to set the file creation mask, set the alarm clock (to update LCK files), find out the system name, and the current date. Then, the arguments from the command line, if any, are processed. If an argument exists, it is checked. If it is found to be the string "update", then the routines update() and by() are called in order. Any other argument is ignored.

If no valid argument is found, then certain flags are initialized and find_ecal_dir() is called. If find_ecal_dir() returns, then a valid file pointer exists to the user's data file and it is locked to prevent others from writing to it. The jump point is now moved forward. At this point, old data is compressed out of the data file and deleted. Finally, the day's schedule is retrieved and the program enters the Command Interpreter Loop.

In the Command Interpreter Loop, the jump point is set to the top of the loop. This ensures that if a signal is caught from which execution is allowed to continue, the program will return to the top of the loop. Also, if a routine called from within the loop changes the jump point, it is reset to the top of the loop where it belongs. The heading and schedule are next displayed, followed by the command menu and the user is prompted to input a command. The valid commands are:

"c":	set up a conference
"d":	delete an entry for today
"e":	edit an entry for the current day
"i":	insert an entry for current day
"n":	get a new current day
"r":	reply to a requested (tentative) schedule
"s":	get the next current day
"u":	update a calendar on a remote machine
"x":	exit the electronic calendar
"!":	shell escape

All of these commands, except for the shell escape character, are displayed in the menu. The shell escape, from the users point of view, works the same as the "vi" editor shell escape works.

1.3 insert_check()

This routine was part of the original program. It was changed so that it now returns the value of the variable "ok" instead putting the value in the location pointed to by a parameter passed to it when it was called. The error messages were moved to the calling routines. In this way, new routines could use this routine much easier.

1.4 do_insert()

This was also part of the original program. The actual insertion of a schedule entry is performed on array of schedules. Changes were made to reflect the new data structure used. It now works as follows:

First: From the end of the list to position "locn", all entries are moved down one. In this way, the list is kept time ordered (as long as locn was determined by "insert_check()")

Then the new information is put in the now empty slot. For general schedules, the program forgets the month, day of month and year numbers in order to prevent possible later confusion.

1.5 delete()

This was part of the original program. The actual deletion of a schedule entry is performed on an array of schedules. Changes were made to reflect the new data structure used. It now works by checking the length of schedule. If the schedule length is zero, the routine has nothing to delete. Otherwise, the user is prompted for the time of the entry to delete. If the entry exists, it is marked inactive, written to disk, and the entries below it are moved up one in the array. Otherwise, an error message is issued and the routine returns.

1.6 find_ecal_dir()

This was part of the original program. It has been extensively rewritten. Changes were made to reflect the new data structure used. It now works by attempting to move to the "ecal" directory under the user's home directory. The program creates the ecal directory under $HOME if it needs to.

The real user identifier is saved first. This is used in the forked shell in "ex()" to set the effective and real user identifier of the user before exec of command. (since ecal runs with effective user identifier of root, for security). The name field of the participant's data structure is then populated with the login identifier of the user acquired from the password file. Then, open_ecal() is called. If successful, when the routine returns, our data file has been properly opened and locked, and the rest of the participant data structure for us has been populated. If open_ecal() fails, the only thing it can do is exit since is not a valid file pointer to use.

1.7 get_schedule()

This was part of the original program. It has been rewritten to reflect the new data structure that contains the schedule and for binary I/O on the disk. It now reads the entire binary data file and keeps in the list what is needed for the active general or specific day schedule. If the program is reading for a specific day, both data for that specific date and for the general day of the week for that date is kept. Otherwise, only data for the general day of the week is kept. Once all the data needed is in memory, it is sorted by time. This saves having to do the sort every time this schedule is displayed.

1.8 update_disk()

This was part of the original program and also has been rewritten to reflect the new data structure that contains the schedule and for binary I/O on the disk. It writes a schedule (an array of structures) to disk using the wr_dsk() routine.

It first checks for a valid file pointer. This is very useful for debugging the program. Next, it goes through the entire array of schedule structures and for each entry is goes through the following algorithm.

It checks the modification flag. If false, this entry has not been modified and does not need to be written. If True, it checks the record number. If the record number is "-1" then this means that there is no previous entry on disk for this record. The program will try and find a non active entry starting at the record number of the previous record or 0 if this is the first record of the structure. If the end of file is found before an empty entry is found, the data is appended to the end of the file. In this way, data on the disk is kept in a semi-sorted order, reducing the sort time in the get_schedule() routine.

We now are assured of having a data record in need of writing and a known place to write it on disk, so it is written and then the modification is set flag to false.

2.0 New Modules

All of the remaining routines are new to the program or have been do completely rewritten so as to be essentially new (ie, the only thing left in common is the name of the routine).

2.1 compress()

This routine replaced "remove_today()". It searchs the entire data file, deleting all specific entries with dates earlier than todays, and compress entries so that all remaining active entries are at the front of the data file.

2.2 set_traps()

This routine sets the signal traps to call the routine "hndl_sigs()" when any of the possible signals occur.

2.3 hndl_sigs()

This routine specifies what to do when any of the possible signals are caught. The only argument that can be passed to it is the number of the signal that the operating system caught.

It first resets the trap for the caught signal. The Interrupt, and Quit signals call clean_up(), followed by longjmp, generally back to the command loop in main. The Alarm signal touches all active LOCK files so that they will not time out, letting another user ignore them. All other signals break out of the case and cause the program to exit by going through the by() routine.

The Kill signal (#9) cannot be caught, so it is not in the routine. If the program ever receives a 9, the operating system forces us to die immediately and there is nothing it can do about cleaning up. It is for this reason that the LOCK routine will ignore any LOCK file older than the threshold defined in the header information.

2.4 by()

The program will usually exit through here to ensure everything is cleaned up (all "LOCK" files removed, etc.).

2.5 clean_up()

Called by any routine wanting to get rid of all "LOCK" files and close all file pointers, except for those associated with the single data structure the user is using. It also removes any temporary files the program has created and closes those file descriptors.

2.6 rd_dsk()

Reads one binary data structure from the disk. The data it reads is determined by the record number passed to it by the calling routine.

2.7 wr_dsk()

This routine writes an entry to the disk file with an offset of some number of (the record number multiplied by the size of the data structure) bytes from the beginning of the file.

2.8 ex()

This routine is used by the shell escape function and routines needing to run shell commands (such as mail and uux).

It first forks another process. The child process does the work, and the parent waits until the child terminates. A second process is used so that the effective user identifier of the process can be set to the real user identifier of the user for the shell command without losing the "root" effective user identifier of the parent

The child sets effective and real user identifier to that of the real user identifier which was determined when the program started. The command is then execed as the stdin of a shell. If the printf statement is reached, the exec failed, so the child prints a message and exits WITHOUT GOING THROUGH by(); If this process exited through by(), the parent would lose all temporary and "LOCK" files.

The parent checks to ensure the fork was successful. If the fork failed, an error message is issued and the routine returns. Otherwise, the Interrupt and Quit signals are set to ignore and wait is called until the child terminates. Then, the Interrupt and Quit signals are reset to the signal handling routine and the parent returns to the calling process.

2.9 confer()

This is the conference scheduling routine. It prompts for a list of login identifier's to schedule for the conference. This list, in the form of a line of character strings separated by white-spaces (blanks or tabs) is scanned and broken down into login identifier's. The password file is then searched for a matching identifier. If not found, a message to that effect is printed and the next string of characters is processed. If it is found, the path to the ecal directory is constructed and open_ecal is called. If open_ecal fails, confer returns to the calling routine. If open_ecal succeeds, then the data file is now open and the file is locked. The schedule for this participant is then populated and the program goes on to process the next string of characters. The open_ecal() routine also populates the path with the correct path to the ecal "LOCK" file associated with the ecal data file opened

The routine next prompts for a range of times in which to try and schedule the meeting. This range of times is broken down into five minute intervals and run through insert_check() for each active participant. The times that conflict with one or more of the participants are so marked. A list of available times is then displayed and the user is asked to input a specific range of times for the meeting. This range is then double checked for availability, the meeting description is prompted for, and then the insertions are done. All users scheduled are then sent mail concerning the proposed meeting. Before returning, the program runs the clean_up() routine to remove all "LOCK" files, etc.

2.10 up_date()

This routine updates calendars on the remote machine through use of the uux command. The "ecal" and "cat" commands must exist on the remote machine and be executable on that machine.

Mail is sent to the owner of the calendar file on the remote machine concerning the impending update. A data file is then created to be sent to the remote machine. This data file is made up as follows: a line containing the machine and login identifier that initiated the update, a line containing the machine and login identifier to be updated, and groups of 8 lines with an ASCII representation of the:

the beginning time,
the ending time,
the month number,
the day of week number,
the day of month number,
the year number,
the specific flag for entry,
and the text description for the entry.

There is no need to send the active or modify flags, the record number, or the filler variable to the remote machine.

When the creation of this temporary file is complete, it is closed and a shell command line is constructed. This line cats the temporary file and pipes it through ecal on the remote machine with the update argument (i.e., cat tmp | ecal update). This is accomplished using the unix "uux" command and the uucp networking facilities.

A mail message is now sent to the login on the remote machine informing the owner that their calendar has been updated by the login on this machine that originated the request. Finally, the two temporary files are unlinked (removed).

2.11 update()

This routine uses the ASCII data file passed to it from the remote machine to update a user's calendar on this machine. Mail is sent to the owner of the file, notifying him of the update. Conflicts in the calendar are noted and also mailed to the owner of the file on the local machine. A success/failure message is sent to the remote requester.

The routine first reads the path name of the remote identifier requesting the update followed by the machine and identifier for the local machine. A temporary mail file is then created, and the owner of the local file is told who is updating the file. After this, the update is done and any entries that cannot be added to the calendar are noted in this file

In the update, the lines of the file are scanned in groups of 8 to get the next entry to be added, the current date is set to that of the calendar entry to be added, and get_schedule is called to get the day's schedule the user is adding to. Then, insert_check() is called. If insert check returns an error, lines are added to the mail file to identify the error, otherwise, do_insert() is called with the location that insert_check() returned to us, followed by update_disk().

After the entire data file has been read and the last update has been done, the mail file is closed and mailed to the owner of the local file. Finally, the mail file is unlinked and flags indicating their presence are reset.

2.12 open_ecal()

The open_ecal() routine is called with an index into the participant's data structure. This index indicates where the information generated here is to be stored. It is assumed that the structure contains a valid login identifier for the machine and nothing more. The login name is used to acquire the proper line from the password file. This ensures that the program actually does have a valid login identifier and gives us the path to the HOME directory of the login. From this, the program constructs the path to the ecal directory, the "LOCK" file and the actual ecal data file.

The routine then checks for the existence of the ecal directory. If it does not exist, it attempts to create it and checks for its existence again. If it still does not exist, an ERROR condition is returned.

If the index to the structure is 0, then the user is opening his own ecal file and not someone else's. Therefore, the program will change his working directory to the ecal directory. A failure to change directory here will cause an ERROR to be returned to the calling routine.

Next, the existence of the "LOCK" file is checked for. If it exists, someone else has beat the user to the file and the routine will return an ERROR indication. Otherwise, it attempts to create the "LOCK" file. If the routine cannot create it, it will return an ERROR condition.

The routine now attempts to open the actual data file for updating. If that fails, it tries to create the file and then reopen it for updating. If any of these fail, then an ERROR condition is returned. Otherwise, everything is ok and it returns a zero to the calling routine. The participant structure indicated by the index number now contains a valid file pointer to the ecal data file and the full path name to the "LOCK" file.

An error condition is returned to the calling program for any of the following:

1. The login identifier does not exist on system.
2. The ecal directory does not exist and can't be created.
3. Failure to change directory successfully to the ecal directory when index is 0.
4. The "LOCK" file already exists.

5. Cannot create the "LOCK" file.

6. Ecal data file cannot be opened or created.

7. After successful creation, failure to reopen the ecal data file successfully.

2.13 what_sys()

This routine populates the "sys_name" string with the name of the system on which this software is running. It was made necessary because of the fact that Machines running Berkley Unix use the "gethostname(2)" system call while machines running AT&T Unix system 3 and later versions use the uname(2) system call.

2.14 touch_lcks()

The "touch_lcks" routine updates the modification time of the "LCK" files to keep them current. It then resets the alarm clock so that the program knows when another update is needed.

2.15 edit()

This routine allows the user to edit the description of an already existing entry in the calendar. The routine first checks to be sure that there are existing calendar entries for the current day. If there are no entries, the program prints an error message and then returns to the main menu. If there are active entries, the the program prompts for which editor to use. The current choices are "vi" and "ed". The program next checks to see if the command line had the time of the appointment to edit included in it. If not, the user is prompted for the beginning and ending times of the appointment to be edited. If an appointment with this beginning time is not found, an error message is printed and the program returns to the main menu. Now that the program knows which entry to edit, it writes that description into a temporary file and then execs the editor from a forked shell using the "ex()" routine.

When the "ex()" routine returns, the edit session is complete. The program then reopens the temporary file and reads the new description back into the program memory. The new description is truncated to 100 characters in length. The modified appointment record is then written to disk and the temporary file is closed and deleted.

2.16 reply()

The reply() routine allows the user to reply to tentative conference requests. It starts out by searching the current days schedule to find if there are any tentative conferences on the current days schedule. If there are none, a message is printed to the user and the program returns to the main menu. If there are one or more tentative meetings, the program prompts for the starting and ending time of the meeting that is to be confirmed. It then checks to ensure that this is actually a tentative meeting. If it is not, a message is written to the user and the program returns to the main menu. If it is a tentative meeting, the program attempts to retrieve the password file entry for the requester. If one is not found, an error message is printed and the program returns to the main menu. When the password entry is found, the user is prompted as to if he wants to confirm the tentative appointment or to send regrets. An appropriate mail message is generated and mailed to the login id that originally requested the meeting. If confirmation is sent, the calendar entry is changed from a tentative appointment to normal appointment. If regrets are sent, then the tentative appointment is deleted from the calendar files. After the mail is sent, all temporary files are closed and deleted.

APPENDIX 3

"C" LANGUAGE SOURCE CODE

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/dir.h>
#include "time.h"
#include <pwd.h>
#include <errno.h>
#include <signal.h>
#include <setjmp.h>

jmp_buf env1;
extern errno;

#ifdef V5.0
#include <string.h>
#endif

#include <stdio.h>

/* #define      BSD             /* define BSD for 4.X BSD systems    */
                                /* comment out for system V UNIX     */

#define DESC_LENGTH             100
#define MAX_APP_NUM              15     /* maximum number of appointments*/
#define MAX_PARTIC               15     /* maximum number of participants in conf */
                                /* limited by number of file descriptors*/
                                /* allowed to be open- generally:    */
                                /* 20:  max file descrip. for a proc */
                                /* -3:  (stdin, stdout, stderr)      */
                                /* -2:  other discriptors used in prog. */
                                /* 15:  The max number left          */

#define HOURS                     2     /* Number of hours a LCK file is valid     */
#define TRUE            1
#define FALSE           0
#define ME              0
#define ERROR                    -1
```

```
FILE *fopen(), *mail;
char *gets();
char *strcat(), *strcpy(), *strncpy(), *mktemp();


#ifdef BSD
        int sleep(), alarm();
        int longjmp(), exit();
#else
#include <sys/utsname.h>

        struct utsname utsname;
        struct utsname *un = &utsname;

        unsigned sleep(), alarm();
        void longjmp(), exit();
        long time();
#endif


struct utimbuf {
        time_t actime;          /*      access time                     */
        time_t modtime;         /*      modification time               */
};


struct sch1 {                   /*      The data structure for the      */
                                /*      modified program                */
                                /*      The 4 extra characters in the   */
                                /*      info field provide an even      */
                                /*      128 bytes (on 32 bit machines)          */
                                /*      in the data structure           */
        short begin;    /*      encoded begining time                   */
        short end;      /*      encoded ending time                     */
        short month;    /*      month number                            */
        short wday;     /*      day of week number                      */
        short mday;     /*      day of month number                     */
        short year;     /*      year number                             */
        short specific; /*      TRUE if for a specific date             */
        short active;   /*      TRUE if contains active data            */
        short mod;      /*      TRUE if modified and not written to disk*/
        short requestor;/*      login id of person requesting conference*/
        int rec_num;    /*      record number index for data file       */
        char info[104]; /*      description of entry                    */
};
```

```
struct participant {
        FILE *fp;               /* File pointer to ecal file in use        */
        char path[80];          /* Path to directory containing LCK file*/
        int length;             /* Length of participants schedule     */
        int active;             /* This schedule is being used         */
        char name[10];          /* login name of participant           */
        struct sch1 sch[MAX_APP_NUM];   /* Actual schedule structure    */
};

struct participant partc[MAX_PARTIC];

struct tm today;

char *month[12] = {
"January",
"February",
"March",
"April",
"May",
"June",
"July",
"August",
"September",
"October",
"November",
"December"
};

char *weekday[7] = {
"Sunday",
"Monday",
"Tuesday",
"Wednesday",
"Thursday",
"Friday",
"Saturday"
};

int specific;           /*      TRUE if for a specific date             */
                        /*      initialized in main, only modified      */
                        /*      in get_new_day()                        */

int wday_num, mon_num, mday_num, year_num;
int confer_flag, my_uid, lngth, cupid, lck_flg;
char fil_string[3][100], sys_name[100];
```

```
main(argc, argv)
int argc;
char *argv[];
{
char command[100];
char p[100];

        (void) setjmp(env1);  /* Before Traps set so we know where to      */
                              /*       return to.                */
        set_traps();          /* Set Traps early- so we can undoe LCK */
        (void) umask(077);    /*       files created.            */

        (void) alarm((HOURS * 3600 / 2));   /* So any LCK files we*/
                              /* we have can be kept current.              */
        what_sys();
        get_today();
        if (argc > 1)
        {
                if (strcmp(argv[1], "update") == 0)
                {
                /*      for updating from remote machine           */

                        update();
                        by();
                }
                printf("Illegal argument to ecal\n");
                by();
        }

        specific = TRUE;

        confer_flag = FALSE;

        wday_num = today.tm_wday;
        mon_num  = today.tm_mon;
        mday_num = today.tm_mday;
        year_num = today.tm_year;

        find_ecal_dir();
        (void) setjmp(env1);  /* Move jump point forward so what has      */
                              /* already been done will not be repeated*/
        compress();

        get_schedule(partc[ME].fp, partc[ME].sch, &partc[ME].length);
```

```
/*      Command Interperter Loop                                  */

command[0] = NULL;
while (command[0] != 'x')
{
        (void) setjmp(env1);  /* Move jump point forward so             */
                        /* what has already been done will not          */
                        /* be repeated                          */

        disp_heading(weekday[wday_num], month[mon_num]);
        disp_schedule(partc[ME].sch, partc[ME].length);

        printf("\n\tc: confer   d: delete              e: edit\n");
        printf("\ti: insert     n: new                 r: reply\n");
        printf("\ts: successor  u: update (remote)     x: exit\n");
        printf("\nwhich? -> ");

        if (gets(command) == NULL) command[0] = 'x';
        command[99] = NULL;
        (void) strcpy(p, &command[1]);

        switch(command[0])
        {
                case 'c':/* set up a conference                 */
                        confer_flag = TRUE;
                        confer();
                        confer_flag = FALSE;
                        break;

                case 'd':       /* Delete an entry for today    */
                        delete(partc[ME].sch, &partc[ME].length, p);
                        break;

                case 'e': /* edit an entry for the current day  */
                        edit(p);
                        break;

                case 'i': /* insert an entry for current day    */
                        insert(partc[ME].sch, &partc[ME].length, p);
                        break;
                        case 'n':  /* get a new current day                  */
                                get_new_day(p);
                                get_schedule(partc[ME].fp, partc[ME].sch,
                                        &partc[ME].length);
                                break;

                        case 'r':  /* reply to a tentative conference sch*/
                                reply(p);
                                get_schedule(partc[ME].fp, partc[ME].sch,
                                        &partc[ME].length);
                                break;

                        case NULL:
                        case 's':/* get the next current day        */
                                get_successor();
                                get_schedule(partc[ME].fp, partc[ME].sch,
                                        &partc[ME].length);
                                break;

                        case 'u':  /* update a calender on a remote mach*/
                                compress();
                                up_date();
                                break;

                        case 'x':  /* exit calendar                          */
                                printf("\nEnding Electronic Calendar.\n");
                                break;

                        case '!': /* shell escape                    */
                                ex(&command[1]);
                                break;

                        default:
                                printf("Invalid Command\n");
                                break;
                }
        }
        by();
        return(0);
}
```

```
/*                    NO     CHANGES                                    */

/* Sets external "today" to a structure pointer with info about today */
/* Fields used — today.mday : day of month  (1..31);                  */
/*          today.mon  : month of year (0..11);                        */
/*          today.wday : day of week   (0..6);                         */
/*                    THIS IS A LOW LEVEL ROUTINE                      */
/*      (DOESN'T CALL ANY ROUTINES DESCRIBED IN THIS SOURCE CODE)  */

get_today()
{
struct tm *localtime();          /* external, in <sys/time.h>          */
long clock;
        (void) time(&clock);  /* sets clock to seconds since 1/1/70    */

        today = *(localtime(&clock)); /* converts clock to             */
                                       /* struct pointer               */
}



/*                    NO     CHANGES                                    */

/* Common times are converted to mapped values on a 24 hour clock.     */
/* Input parm[num] is assumed to be a common time that is a multiple   */
/* of 5. Mapping starts at midnight : 12:00 PM => 0, 12:05 PM => 1     */
/* ... 11:55 AM => 243, 12:00 AM => 244 ... 11:55 PM => 487.           */
/*                    THIS IS A LOW LEVEL ROUTINE                      */
/*      (DOESN'T CALL ANY ROUTINES DESCRIBED IN THIS SOURCE CODE)  */

time_map(num, a_or_p)              /* ret: scaled value on 24 hour clock */
int num;                         /* in:  Numeric legal time, (100 - 1255)*/
char a_or_p;                     /* in:  AM or PM ('A' or 'P')         */
{
int left, right, value;

        left = num / 100;
        right = num % 100;
        if (left == 12)          left = 0;
        value = (left * 12) + (right / 5);
        if (a_or_p == 'P')       value += 144;
        return(value);
}
```

```
/*                      NO     CHANGES                                  */

/* Mapped values are converted back to conventional times and stored    */
/* (or returned) in a character string. Mapping is reverse of above.  */
/*                      THIS IS A LOW LEVEL ROUTINE                     */
/*      (DOESN'T CALL ANY ROUTINES DESCRIBED IN THIS SOURCE CODE)  */

char *
time_string(i)                  /* ret: string of human readable time */
int i;                          /* in:  scaled value on 24 hour clock  */
{
int left,right;
char a_or_p;
static char string[9];          /* added static decl. for VAX          */

        left = (i % 144) / 12;
        if (left == 0)          left = 12;
        right = (i % 12) * 5;
        a_or_p = (i < 144) ? 'A' : 'P';
        (void) sprintf(string, "%2d:%02d %cM", left, right, a_or_p);
        return(string);
}


/*                      NO     CHANGES                                  */

/* Displays headings for general or specific day's schedules.         */
/*                      THIS IS A LOW LEVEL ROUTINE                     */
/*      (DOESN'T CALL ANY ROUTINES DESCRIBED IN THIS SOURCE CODE)  */

disp_heading(weeekday, mounth)
char *weeekday, *mounth;
{
        if (specific)   printf("\nSchedule for %s, %s %d, 19%d:\n",
                                weeekday, mounth, mday_num, year_num);
        else            printf("\nGeneral %s schedule:\n", weeekday);
}
```

```
/*                      NO    CHANGES                                    */

/* Displays schedule (after headings), with formatted description    */
/* fields that may break between words and take two lines if required.   */
/*                      THIS IS A LOW LEVEL ROUTINE                      */
/*      (DOESN'T CALL ANY ROUTINES DESCRIBED IN THIS SOURCE CODE)  */

disp_schedule(schedule, length)
struct sch1 *schedule;
int length;
{
int i, j, k, l;
char begin_str[9], end_str[9];
char outline[256];

        if (length == 0)
        {
                printf("\n        ***** no appointments scheduled *****\n\n");
        }
        else
        {
                for (i = 0; i < length; i++)
                {
                        if (schedule[i].active == FALSE)        continue;

                        (void) strcpy(begin_str,
                                time_string(schedule[i].begin));
                        (void) strcpy(end_str, time_string(schedule[i].end));
                        printf("\n%8s - %8s ", begin_str, end_str);
                        if (schedule[i].requestor != -1) printf("* ");
                        else                printf("  ");

                        k = 0;
                        for (j = 0; ((schedule[i].info[j] != NULL) &&
                           ((schedule[i].info[j] != ' ') || (j < 45))); j++)
                        {
                                outline[k++] = schedule[i].info[j];
                        }

                        outline[k++] = '\n';
                        if (schedule[i].info[j] != NULL)
                        {
                                /*      if needed, print a second line */
                                for (l=0; l<=20; l++) outline[k++] = ' ';
                                while (schedule[i].info[j] != NULL)
                                {
                                        outline[k++] = schedule[i].info[j++];
                                }
                                outline[k++] = '\n';
                        }
                        outline[k++] = NULL;
                        printf("%s", outline);
                }
        }
        for (i = (length * 2); i < 14; i++)        printf("\n");
}
```

```
/*                      NO    CHANGES                          */

/* Algorith to compute the day-of-the-week, given a numeric date,  */
/* month, and year. Thanx to Bill McDaniel, KSU.                    */
/*                      THIS IS A LOW LEVEL ROUTINE                */
/*      (DOESN'T CALL ANY ROUTINES DESCRIBED IN THIS SOURCE CODE)  */

which_weekd()
{
int funct[12];                  /* Conversion factors for each month */
int i;

        funct[0] = funct[9] = 0;/* The algorith assumes initially    */
        funct[1] = funct[2] =   /* that month doesn't matter, then   */
        funct[10] = 3;          /* corrects itself. (look at a calendar)*/
        funct[3] = funct[6] = 6;
        funct[4] = 1;
        funct[5] = 4;
        funct[7] = 2;
        funct[8] = funct[11] = 5;
        if ((mon_num == 0 || mon_num == 1) && (year_num % 4 == 0))
        {
                i = ((funct[mon_num] + mday_num + year_num
                                + year_num / 4 - 1) % 7);
        }
        else
        {
                i = ((funct[mon_num] + mday_num
                                + year_num + year_num / 4) % 7);
        }
        return(i);
}
```

```
/*                    NO     CHANGES                                   */

/* Gets the next day via modulo arrithmic. For general or specific   */
/* days.                                                              */
/*                    THIS IS A LOW LEVEL ROUTINE                      */
/*      (DOESN'T CALL ANY ROUTINES DESCRIBED IN THIS SOURCE CODE)  */

get_successor()
{
int num;

        wday_num = (wday_num + 1) % 7;
        if (specific)
        {
                if (mon_num==3 || mon_num==5 || mon_num==8 ||
                        mon_num==10 || mon_num == 12)
                {
                        num = 30;
                }
                else if (mon_num == 1)
                {
                        if (year_num % 4 == 0)        num = 29;
                        else     num = 28;
                }
                else     num = 31;

                mday_num = (mday_num % num) + 1;
                if (mday_num == 1)
                {
                        mon_num = (mon_num + 1) % 12;
                        if (mon_num == 0)    year_num = year_num + 1;
                }
        }
}
```

```
/*                    NO     CHANGES                                        */

/* Main routine to perform inserts to schedule arrays.                      */
/* EXTERNAL -        get_time : Parse the input string for beginning &      */
/*                           ending time.                            */
/*              check_insert : Check for schedule conflicts          */
/*                           (overlapping,etc.)                      */
/*              do_insert: Perform actual insertion.                 */

insert(schedule, length, parms)
struct sch1 *schedule; /* Schedule (array of structures) to insert into*/
int *length;     /* Length of array. (Number current appointments    */
char *parms;     /* String exists if user included parms.after 'i'   */
{
char string[20];
char desc[DESC_LENGTH];
char temp_begin[10], temp_end[10], temp2_begin[10], temp2_end[10];
int locn, num1, num2;

        if (*parms != NULL)   (void) strcpy(string, parms);
        else
        {
                printf(
                "Enter beginning and ending time of new appointment. -> ");
                (void) gets(string);
        }
        get_time(2, string, &num1, &num2);

        (void) strcpy(temp_begin, time_string(num1));
        (void) strcpy(temp_end, time_string(num2));

        if (insert_check(schedule, *length, num1, num2, &locn))
        {
                if (((*length) + 1) >= MAX_APP_NUM)
                {
                        printf("No room for a new calendar entry\n");
                        return;
                }

                printf("Enter description of appointment.\n-> ");
                (void) gets(desc);
                do_insert(schedule, length, locn, num1, num2, desc);
                update_disk(partc[ME].fp, schedule, length);
        }
        else
        {
                (void) strcpy(temp2_begin, time_string(schedule[locn].begin));
                (void) strcpy(temp2_end, time_string(schedule[locn].end));
                printf(
                "%s - %s conflicts with previous %s - %s appointment.\n",
                        temp_begin, temp_end, temp2_begin, temp2_end);
                (void) sleep(3);
        }
}
```

```
/*                      NO     CHANGES                                   */

/*                      THIS IS A LOW LEVEL ROUTINE                      */
/*      (DOESN'T CALL ANY ROUTINES DESCRIBED IN THIS SOURCE CODE)  */

insert_check(schedule, length, num1, num2, locn)
struct sch1 *schedule;  /* in: schedule, array of structures.          */
int length;      /* in: length of array, number of appointments.             */
int num1;        /* in: beginning time (scaled) of prospective insert   */
int num2;        /* in: ending         ---------------                  */
int *locn;       /* out: locn to insert (if *ok) or conflict (else) */
{
int ok;          /* TRUE if no conflicts, FALSE otherwise.              */
int i = 0;

        *locn = i;
        ok = TRUE;
        if (length > 0)
        {
                while (i<length && schedule[i].begin < num1)       i++;
                *locn = i;
                if (i == length)
                {
                        if (schedule[i-1].end > num1)
                        {
                                ok = FALSE;
                                (*locn)--;
                        }
                }
                else if (schedule[i].begin == num1)
                {
                        ok = FALSE;
                }
                else
                {
                        if (i > 0)
                        {
                                if (schedule[i-1].end > num1)
                                {
                                        ok = FALSE;
                                        (*locn)--;
                                }
                        }
                        if (schedule[i].begin < num2)
                        {
                                ok = FALSE;
                        }
                }
        }
        return(ok);
}
```

```
/*                      NO     CHANGES                                   */

get_time(count, string, first, second)
int count;
char *string;
int *first;
int *second;
{
int i, num, ct, ok;
char a_or_p;
char temp_begin[10], temp_end[10];

        ok = FALSE;
        while (!ok)
        {
                i = 0;
                ok = TRUE;
                for (ct=0; ct<count; ct++)
                {
                        num = 0;
                        while (string[i] == ' ') i++;
                        while (((string[i] >= '0' && string[i] <= '9') ||
                                string[i] == ':') && string[i] != NULL)
                        {
                                if (string[i] >= '0' && string[i] <= '9')
                                {
                                        num = num*10 + string[i] - '0';
                                }
                                i++;
                        }
                        if (num == 0)
                        {
                                ok = FALSE;
                                printf("Missing time.\n");
                        }
                        else if (num % 5 != 0)
                        {
                                ok = FALSE;
                                printf("%d is not a multiple of 5.\n", num);
                        }
                        else if (num < 100 || num > 1255 || num % 100 > 55)
                        {
                                ok = FALSE;
                                printf("%d:%02d is an invalid time.\n",
                                        num/100, num%100);
                        }
                        if (num < 700 || num >= 1200)           a_or_p = 'P';
                        else            a_or_p = 'A';
/*      get_time() (continued)        NO     CHANGES                                  */



                        while ((string[i] < '0' || string[i] > '9')
                                && string[i] != NULL)
                        {
                                if (string[i] == 'p' || string[i] == 'P')
                                {
                                                a_or_p = 'P';
                                }
                                else if (string[i] == 'a' || string[i] == 'A')
                                {
                                                a_or_p = 'A';
                                }
                                i++;
                        }
                        if (ok)
                        {
                                if (ct == 0)
                                {
                                        *first = time_map(num, a_or_p);
                                }
                                else
                                {
                                        *second = time_map(num, a_or_p);
                                        if (*first > *second)
                                        {
                                                ok = FALSE;
                                                (void) strcpy(temp_begin,
                                                        time_string(*first));
                                                (void) strcpy(temp_end,
                                                        time_string(*second));
                                                printf(
                                        "%s - %s is an illegal time frame.\n",
                                                        temp_begin, temp_end);
                                        }
                                }
                        }
                }
                if (!ok)
                {
                        printf("Reenter time(s). -> ");
                        (void) gets(string);
                }
        }
}
```

```
/*                      NO     CHANGES                                    */

/* The user's command-line input is parsed to determine which day's*/
/* schedule is to be observed. Four possibilities exist:          */
/* (1) A null line entered is assumed to mean the current specific day.   */
/* (2) If a numeric date only is entered, the month is deduced. Specific day.*/
/* (3) If a weekday (eg. "wed") is entered, the general weekday is inferred. */
/* (4) If a month (eg. "sep") is entered, a date is also expected ("sep23"). */
/*                      THIS IS A LOW LEVEL ROUTINE                        */
/*        (DOESN'T CALL ANY ROUTINES DESCRIBED IN THIS SOURCE CODE)  */

get_new_day(parms)
char *parms;
{
extern struct tm today;
extern char *weekday[], *month[];
int i, nwptr, tdate;      /* nwptr - non-white space ptr added for VAX        */
int ok, second;
char temp[4];
char string[10];
int found;                /* added for VAX                              */

        second = FALSE;
        (void) strcpy(string, parms);

        ok = FALSE;
        while (!ok)
        {
                /* skip white space for VAX                              */

                for (i=0; string[i] == ' ' || string[i] == '\t'; i++) ;
                nwptr = i;

                        /* check for parm after the 'n' */
                if (string[nwptr] == NULL || second)
                {
                        printf("Enter date, weekday, or <ret>. -> ");
                        (void) gets(string);
                        /* skip white space for VAX */
                        for (i=0; string[i] == ' ' || string[i] == '\t'; i++);
                        nwptr = i;
                }
                second = TRUE;
                ok = TRUE;
```

```
                        if (string[nwptr] == NULL)
                        {
                                specific = TRUE;
                                wday_num = today.tm_wday;
                                mday_num = today.tm_mday;
                                mon_num  = today.tm_mon;
                                year_num = today.tm_year;
                        }
                        else if (string[nwptr] >= '0' && string[nwptr] <= '9')
                        {
                                /* date based on digit alone */
                                specific = TRUE;
                                tdate = 0;
                                i = nwptr;
                                while ((string[i] >= '0') && (string[i] <= '9'))
                                {
                                        tdate = (tdate * 10) + string[i] - '0';
                                        i++;
                                }
                                if (tdate < today.tm_mday)
                                        mon_num = (today.tm_mon + 1) % 12;
                                else
                                        mon_num = today.tm_mon;
                                if ((tdate>31) ||
                                        (tdate>30 && (mon_num==3 || mon_num==5 ||
                                                mon_num==8 || mon_num==10)) ||
                                        (tdate>29 && mon_num==1))
                                {
                                        printf("%s does not have %d days.\n",
/* get_new_day()*/                              month[mon_num], tdate);
/* (continued) */                       ok = FALSE;
/* NO CHANGES           */      }
                                else
                                {
                                        mday_num = tdate;
                                        if (mon_num < today.tm_mon)
                                        {
                                                year_num = today.tm_year + 1;
                                        }
                                        else
                                        {
                                                year_num = today.tm_year;
                                        }
                                        wday_num = which_weekd();
                                }
                        }
```

```
                else
                {
                        /* date based on month and day or weekday  */

                        (void) strncpy(temp, &string[nwptr], 3);
                        temp[3] = NULL;

#ifdef V5.0
                        temp[0] = toupper(temp[0]);
#else
                        if (temp[0] > 'a') temp[0] = temp[0] - ('a' - 'A');
#endif

                        /* check for weekday ( = general schedule )  */

                        /* added for VAX to exit table search loop      */
                        found = FALSE;
                        for (i=0; i<7 && !found; i++)
                        {
                                if (strncmp(temp,weekday[i],3) == 0)
                                {
                                        found = TRUE;
                                }
                        }
                        if (found)
                        {
                                specific = FALSE;
                                wday_num = i-1; /* -1 kludge added for VAX */
                        }
                        else
                        {
                                /* didn't find general schedule - check */
                                /* for month and day                         */

                                for (i=0; (i<12) &&
                                        (strncmp(temp,month[i],3) != 0);
                                                i++)            ;

                                if (strncmp(temp,month[i],3)==0)
                                {
                                        /* found the month */

                                        specific = TRUE;
                                        mon_num = i;
                                        for ( i=3+nwptr; (string[i] != NULL) &&
                                                ((string[i] < '0') ||
                                                        (string[i] > '9'));
                                                i++)            ;

                                        tdate = 0;
                                        while ((string[i] >= '0') &&
                                                (string[i] <= '9'))
                                        {
                                                tdate = (tdate * 10) +
                                                        string[i] - '0';
                                                i++;
                                        }
                                        if (tdate == 0)
                                        {
                                                printf(
                                "*** Must include date with Month. ***\n");
                                                ok = FALSE;
                                        }
/* get_new_day()*/                      else if ((tdate>31) ||
/* (continued   */                              (tdate>30 && (mon_num==3 ||
/* NO CHANGES        */                                 mon_num==5 || mon_num==8
                                                        || mon_num==10)) ||
                                                (tdate>29 && mon_num==1))
                                        {
                                                printf(
                                                "%s does not have %d days.\n",
                                                        month[mon_num], tdate);
                                                ok = FALSE;
                                        }
                                        else
                                        {
                                                mday_num = tdate;
                                                if ((mon_num < today.tm_mon) ||
                                                  (mon_num == today.tm_mon
                                                  && mday_num <today.tm_mday))
                                                {
                                                        year_num =
                                                        today.tm_year + 1;
                                                }
                                                else year_num = today.tm_year;
                                                wday_num = which_weekd();
                                        }
                                }
                                else
                                {
                                        printf("(%s) is invalid. ",string);
                                        printf("Examples: mon jan23 30.\n");
                                        ok = FALSE;
                                }
                        }
                }
        }
}
```

```
/*      Changes made to reflect the new data structure                    */

/* Actual insertion is performed on array.                         */
/*                  THIS IS A LOW LEVEL ROUTINE                            */
/*      (DOESN'T CALL ANY ROUTINES DESCRIBED IN THIS SOURCE CODE)  */

do_insert(schedule, length, locn, num1, num2, string)
struct sch1 *schedule;  /* out: schedule after insertion.           */
int *length;            /* in/out: length of array, incremented here. */
int locn;               /* in: location in array to insert.         */
int num1, num2;                 /* in: beginning & ending times if insertion.  */
char *string;           /* in: string description of appointment.   */
{
int i;

        /* From the end of the list to position "locn", move all        */
        /* entries down one. This way, the list is kept time ordered  */
        /* (as long as locn was dertermined by "insert_check()")        */

        for (i=(*length); i>locn; i--)
        {
                schedule[i].begin       = schedule[i-1].begin;
                schedule[i].end         = schedule[i-1].end;
                schedule[i].month       = schedule[i-1].month;
                schedule[i].wday        = schedule[i-1].wday;
                schedule[i].mday        = schedule[i-1].mday;
                schedule[i].year        = schedule[i-1].year;
                schedule[i].specific    = schedule[i-1].specific;
                schedule[i].active      = schedule[i-1].active;
                schedule[i].mod                 = schedule[i-1].mod;
                schedule[i].requestor   = schedule[i-1].requestor;
                schedule[i].rec_num     = schedule[i-1].rec_num;
                (void) strcpy(schedule[i].info, schedule[i-1].info);
        }
        /*      Put new information in the now empty slot    */

        schedule[locn].begin    = num1;
        schedule[locn].end      = num2;
        schedule[locn].month    = mon_num;
        schedule[locn].wday     = wday_num;
        schedule[locn].mday     = mday_num;
        schedule[locn].year     = year_num;
        schedule[locn].active   = TRUE;
        schedule[locn].requestor= -1;
        schedule[locn].mod      = TRUE;        /* tell the update routine that */
                                       /* this entry needs writing      */

        schedule[locn].rec_num          = -1;  /* so the update routine will   */
                                       /* not overwrite an active entry*/
                                       /* on the disk                  */
        (void) strcpy(schedule[locn].info, string);
        if (specific)       schedule[locn].specific = TRUE;
        else
        {
                schedule[locn].specific = FALSE;        /* For general  */
                schedule[locn].month = 0;       /* schedules, forget the*/
                schedule[locn].mday = 0;        /* month, day of month          */
                schedule[locn].year = 0;        /* and year- prevents   */
                                                /* confusion later      */
        }
        (*length)++;
}
```

```
/*      Changes made to reflect the new data structure                    */

delete(schedule, length, parms)
struct sch1 *schedule;
int *length;
char *parms;
{
char string[20];
char temp_begin[10];
int i, num1;

        if (*length == 0)
        {
                printf("No appointments to delete.\n");
                (void) sleep(2);
        }
        else
        {
                if (*parms != NULL)   (void) strcpy(string, parms);
                else
                {
                        printf(
        "Enter beginning and ending time of appointment to delete. -> ");
                        (void) gets(string);
                }
                get_time(1, string, &num1, &num1);
                for (i=0; i<*length && schedule[i].begin < num1; i++);
                if (schedule[i].begin == num1)
                {
                        if (schedule[i].requestor != -1)
                        {
                                (void) strcpy(temp_begin, time_string(num1));
                                printf("%s is a tentative appointment",
                                        temp_begin);
                                printf(" : use 'reply' command to decline\n");
                                (void) sleep(1);
                                return;
                        }

                        /* delete by marking active flag false, then      */
                        /* updateing disk.                                */
                        schedule[i].active = FALSE;
                        schedule[i].mod = TRUE;
                        update_disk(partc[ME].fp, schedule, length);
```

```
                    /* Now, compress the unused entry out of the*/
                    /* list- saves steps when displaying          */
                    (*length)--;
                    for ( ; i < (*length); i++)
                    {
                            schedule[i].begin    = schedule[i+1].begin;
                            schedule[i].end      = schedule[i+1].end;
                            schedule[i].month    = schedule[i+1].month;
                            schedule[i].wday     = schedule[i+1].wday;
                            schedule[i].mday     = schedule[i+1].mday;
                            schedule[i].year     = schedule[i+1].year;
                            schedule[i].specific = schedule[i+1].specific;
                            schedule[i].active   = schedule[i+1].active;
                            schedule[i].mod              = schedule[i+1].mod;
                            schedule[i].requestor= schedule[i+1].requestor;
                            schedule[i].rec_num  = schedule[i+1].rec_num;
                            (void) strcpy(schedule[i].info,
                                    schedule[i+1].info);
                    }
            }
            else
            {
                    (void) strcpy(temp_begin, time_string(num1));
                    printf("Cannot find %s appointment to delete.\n",
                            temp_begin);
                    (void) sleep(2);
            }
    }
}
```

```
/*                          REWRITTEN                          */


/* Attempts to move to directory "ecal" under the user's home          */
/* directory. Program creates ecal directory under $HOME if need be.   */

find_ecal_dir()
{
struct passwd *getpwuid(), *my_passwd_rec;

        my_uid = getuid();      /* save the real user ID. used in the    */
                                /* forked shell to set effective UID to Real */
                                /* UID before exec of command. (since ecal  */
                                /* runs set uid to root, for security)      */

        my_passwd_rec = getpwuid(my_uid);
        (void) strcpy(partc[ME].name, my_passwd_rec->pw_name);

        if (open_ecal(ME) != 0)
        {
                printf("can't open or create ecal data file\n");
                by();
        }

        lngth = 1;
}
```

```
/*      REWRITTEN FOR NEW DATA STRUTRURES AND BINARY I/O                   */

/* A specific or general day's schedule is attempted read from disk.   */
/* If a specific date's schedule is desired, the file whose name is    */
/* built by appending the date to the month is attempted opened.       */
/* If that file does not exist (an unsuccessful opening attempt), the  */
/* day-of-the week is determined, and that file is opened, although    */
/* the user doesn't know (or care) where the schedule came from.       */

get_schedule(fp1, schedule, length)
FILE *fp1;
struct sch1 *schedule;  /* out: table of appointments for date/day     */
int *length;            /* out: length of table                        */
{
int i, k, changed;
struct sch1 tmp;

        i = 0;
        k = 0;

        /* Reads the entire binary data file and keeps in the list     */
        /* what is needed for the active general or specific day      */

        while( rd_dsk(fp1, &schedule[i], k++) != NULL)
        {
                if (schedule[i].active == FALSE)        continue;
                if (specific)
                {
                        if (((mday_num == schedule[i].mday) &&
                                (mon_num == schedule[i].month)) ||
                                ((wday_num == schedule[i].wday) &&
                                (schedule[i].specific != TRUE)))
                        {
                                i++;
                        }
                }
                else
                {
                        if(schedule[i].specific) continue;
                        if(schedule[i].wday == wday_num)
                        {
                                i++;
                        }
                }
```

```
		schedule[i].mod = FALSE;
		if ((i + 1) >= MAX_APP_NUM)
		{
			printf("Calendar for this day is full\n");
			break;
		}
	}
	*length = i;


/*	get_schedule() (continued) - ADDED FOR NEW DATA STRUTRURES		*/


	/*	sort the new schedule by time				*/
	/*	saves having to do it every time displayed		*/

	changed = TRUE;
	while(changed)
	{
		changed = FALSE;

		for (k = 0; k < (i - 1); k++)
		{
			if (schedule[k].begin > schedule[k + 1].begin)
			{
				changed = TRUE;

				tmp.begin	= schedule[k].begin;
				tmp.end		= schedule[k].end;
				tmp.month	= schedule[k].month;
				tmp.wday	= schedule[k].wday;
				tmp.mday	= schedule[k].mday;
				tmp.year	= schedule[k].year;
				tmp.specific	= schedule[k].specific;
				tmp.active	= schedule[k].active;
				tmp.mod		= schedule[k].mod;
				tmp.requestor	= schedule[k].requestor;
				tmp.rec_num	= schedule[k].rec_num;
				(void) strcpy(tmp.info, schedule[k].info);
```

```
                                        schedule[k].begin       = schedule[k+1].begin;
                                        schedule[k].end                 = schedule[k+1].end;
                                        schedule[k].month       = schedule[k+1].month;
                                        schedule[k].wday        = schedule[k+1].wday;
                                        schedule[k].mday        = schedule[k+1].mday;
                                        schedule[k].year        = schedule[k+1].year;
                                        schedule[k].specific= schedule[k+1].specific;
                                        schedule[k].active      = schedule[k+1].active;
                                        schedule[k].mod                 = schedule[k+1].mod;
                                        schedule[k].requestor= schedule[k+1].requestor;
                                        schedule[k].rec_num= schedule[k+1].rec_num;
                                        (void) strcpy(schedule[k].info,
                                                schedule[k+1].info);

                                        schedule[k+1].begin     = tmp.begin;
                                        schedule[k+1].end       = tmp.end;
                                        schedule[k+1].month     = tmp.month;
                                        schedule[k+1].wday      = tmp.wday;
                                        schedule[k+1].mday      = tmp.mday;
                                        schedule[k+1].year      = tmp.year;
                                        schedule[k+1].specific  = tmp.specific;
                                        schedule[k+1].active    = tmp.active;
                                        schedule[k+1].mod       = tmp.mod;
                                        schedule[k+1].requestor         = tmp.requestor;
                                        schedule[k+1].rec_num           = tmp.rec_num;
                                        (void) strcpy(schedule[k+1].info, tmp.info);
                                }
                        }
                }
        }
```

```
/*      REWRITTEN FOR NEW DATA STRUTRURES AND BINARY I/O              */

/* Writes a schedule (an array of structures) to disk. The filename is */
/* built by either concatanating the month to the date (for specific   */
/* days), or simply using the weekday string (for general days).       */
/*                    THIS IS A LOW LEVEL ROUTINE                      */
/*      (DOESN'T CALL ANY ROUTINES DESCRIBED IN THIS SOURCE CODE)      */

update_disk(fp1, schedule, length)
FILE *fp1;
struct sch1 *schedule;  /* in:specific or general day's schedule */
int *length;            /* in:length of schedule, (number appointments)      */
{
int i, j;
struct sch1 tmp;

        /* Check for a valid file pointer- very useful for debugging   */
        /* the program                                                 */

        if (fp1 == NULL)
        {
                printf("ERROR: Null file pointer in update\n");
                return;
        }

        for (i=0; i < (*length); i++)
        {
                if (schedule[i].mod == FALSE)continue;
                                        /* No need to write anything   */
                                        /* that has not been modified  */

                /* This means there is no previous entry on disk       */
                /* for this record.  We will try and find a non active */
                /* entry starting at the record number of the previous         */
                /* record or 0 if this is the first record of the  */
                /* structure. If the end of file is found before an    */
                /* empty entry is found, append the data to the end of        */
                /* of the file.  In this way, data on the disk is kept */
                /* in a semi-sorted order, reucing the sort time in the */
                /* read_dsk routine.                                   */
```

```
            if (schedule[i].rec_num == -1)
            {
                    j = 0;
                    if (i > 0)          j = schedule[i - 1].rec_num;

                    while( rd_dsk(fp1, &tmp, j) != NULL)
                    {
                            if (tmp.active == FALSE)        break;
                            j++;
                    }
                    schedule[i].rec_num = j;
            }

            /* The location in the file is stored in recnum, write it */

            wr_dsk(fp1, &schedule[i]);
            schedule[i].mod = FALSE;
        }
}

/*                      END OF ORIGINAL PROGRAM                              */
/*                        ALL NEW ROUTINE                          */
/*                    REPLACES "remove_today()"                    */

/*                   THIS IS A LOW LEVEL ROUTINE                   */
/*     (DOESN'T CALL ANY ROUTINES DESCRIBED IN THIS SOURCE CODE)  */

compress()
{
int wrt, rd;
struct sch1 sch;
int tmp;

        wrt = 0;
        rd = 0;

        /* search the entire data file, deleting all specific entries      */
        /* with dates earlier than todays, and compress entries such   */
        /* all entries are at the front of the disk file.          */

        while( rd_dsk(partc[ME].fp, &sch, rd++) != NULL)
        {
                if (sch.active == FALSE)        continue;

                if ((sch.specific == FALSE)    ||
                  (sch.year > today.tm_year)         ||
                  ((sch.year == today.tm_year) &&
                   (sch.month > today.tm_mon))      ||
                  ((sch.year == today.tm_year) &&
                   (sch.month == today.tm_mon) &&
                   (sch.mday >= today.tm_mday)))
                {
                        if (sch.rec_num == wrt)
                        {
                                wrt++;
                                continue;
                        }
                        else if (wrt > sch.rec_num)
                        {
                                printf("Fatal Error in Compression routine\n");
                                printf("rd: %d, wrt: %d, rec_num: %d\n",
                                        rd, wrt, sch.rec_num);
                                by();
                        }
                        tmp = sch.rec_num;
                        sch.rec_num = wrt;
                        wr_dsk(partc[ME].fp, &sch);
                        sch.rec_num = tmp;
                        sch.active = FALSE;
                        wr_dsk(partc[ME].fp, &sch);
                        wrt++;
                        continue;
                }
                sch.active = FALSE;
                wr_dsk(partc[ME].fp, &sch);
        }
}
```

```
/*                          ALL NEW ROUTINE                              */

/*      This routine populates the sys_name string with the name   */
/*      of the system on which this software is running.  It was made       */
/*      neccessary because of the fact that Machines running Berkley*/
/*      Unix use the "gethostname(2)" system call while machines    */
/*      running AT&T Unix system 3 and later versions use the uname(2)  */
/*      system call.                                               */

what_sys()
{

#ifdef BSD
        (void) gethostname(sys_name, 100);
#else
        (void) uname(un);
        (void) strcpy(sys_name, un->nodename);
#endif
        sys_name[99] = NULL;

}


/*                          ALL NEW ROUTINE                              */


set_traps()
{
int hndl_sigs();


        (void) signal(SIGHUP, hndl_sigs);
        (void) signal(SIGINT, hndl_sigs);
        (void) signal(SIGQUIT, hndl_sigs);
        (void) signal(SIGILL, hndl_sigs);
        (void) signal(SIGTRAP, hndl_sigs);
        (void) signal(SIGIOT, hndl_sigs);
        (void) signal(SIGEMT, hndl_sigs);
        (void) signal(SIGFPE, hndl_sigs);
        (void) signal(SIGBUS, hndl_sigs);
        (void) signal(SIGSEGV, hndl_sigs);
        (void) signal(SIGSYS, hndl_sigs);
        (void) signal(SIGPIPE, hndl_sigs);
        (void) signal(SIGALRM, hndl_sigs);
        (void) signal(SIGTERM, hndl_sigs);
}
```

```
/*                              ALL NEW ROUTINE                              */
/*      What to do when signals are caught                            */

hndl_sigs(sig_num)
int sig_num;
{
        (void) signal(sig_num, hndl_sigs);
        switch(sig_num)
        {
                case SIGHUP:   break;
                case SIGINT:
                case SIGQUIT:
                        printf("ABORT");
                        (void) fflush(stdin);
                        printf("\nDo you wish to return to the main menu");
                        printf(" or exit the program? (r/e):");
                        if (getchar() != 'r')      by();
                        clean_up();
                        (void) longjmp(env1, 1);
                case SIGILL:
                        printf("FATAL ERROR: Illeagal instruction\n");
                        break;
                case SIGTRAP:
                        printf("FATAL ERROR: Trace trap\n");
                        break;
                case SIGIOT:
                        printf("FATAL ERROR: I/O Trap\n");
                        break;
                case SIGEMT:
                        printf("FATAL ERROR: Emulator Trap\n");
                        break;
                case SIGFPE:
                        printf("FATAL ERROR: Floating Point Execption\n");
                        break;
                case SIGBUS:
                        printf("FATAL ERROR: Bus Error\n");
                        break;
                case SIGSEGV:
                        printf("FATAL ERROR: Segmentation Violation\n");
                        break;
                case SIGSYS:
                        printf("FATAL ERROR: System Call Error\n");
                        break;
                case SIGPIPE:
                        printf("FATAL ERROR: Pipe Write Error\n");
                        break;
                case SIGALRM:
                        touch_lcks();
                        return;
                case SIGTERM:
                        printf("FATAL: Software Termination Signal\n");
                        break;
                default:
                        printf("FATAL ERROR: Signal Number %d Caught\n",
                                sig_num);
        }
        by();   /* If we are still here, exit gracefully through by()    */
}
```

```
/*                          ALL NEW ROUTINE                                    */
/*      Update the lock files that we have currently active             */

touch_lcks()
{
int i;
struct utimbuf *buf;

        buf = NULL;
        for (i = 0; i < lngth; i++)
        {
                if (partc[i].active == TRUE) (void) utime(partc[i].path, buf);
        }
        (void) alarm((HOURS * 3600 / 2));
}


/*                          ALL NEW ROUTINE                                    */
/*      Exit thru here always to ensure everything is cleaned up        */

by()
{
        if (lck_flg == 0)          exit(1);
        clean_up();
        (void) fclose(partc[ME].fp);
        (void) unlink(partc[ME].path);
        exit(0);
}
```

```
/*                      ALL NEW ROUTINE                              */
/*      clean-up for all but the data structure that I am using.    */

clean_up()
{
int i;
        for (i = 1; i < lngth; i++)
        {
                if (partc[i].active == TRUE)
                {
                        (void) fclose(partc[i].fp);
                        (void) unlink(partc[i].path);
                        partc[i].active = FALSE;
                }
        }
        lngth = 1;
        if (confer_flag > TRUE)
        {
                while (confer_flag > TRUE)
                {
                        (void) unlink(fil_string[(confer_flag - TRUE - 1)]);
                        confer_flag--;
                }
                (void) fclose(mail);
        }
}
```

```
/*                              ALL NEW ROUTINE                              */
/*      Reads one binary data structure from the disk                        */

rd_dsk(fp1, schl, posit)
FILE *fp1;
struct sch1 *schl;
int posit;              /* The record number of the structure to read*/
{
long bytes;

        bytes = posit * sizeof(*schl);
        (void) fseek(fp1, bytes, 0);
        return( fread((char *)schl, sizeof(*schl), 1, fp1) );
}



/*                              ALL NEW ROUTINE                              */
/*      Writes an entry to the disk file with an offset of the        */
/*      record number multiplied by the size of the data structure    */
/*      bytes from the beginning of the file.                         */

wr_dsk(fp1, schl)
FILE *fp1;
struct sch1 *schl;
{
long bytes;

        bytes = schl->rec_num * sizeof(*schl);
        (void) fseek(fp1, bytes, 0);
        (void) fwrite((char *)schl, sizeof(*schl), 1, fp1);
        (void) fflush(fp1);
        return;
}
```

```
/*                         ALL NEW ROUTINE                              */
/*      Used by the shell escape function and routines needing to      */
/*      run shell commands (send mail, etc)                            */

ex(tmp)
char *tmp;
{
int wait_flg, w;
int (*sig1)(), (*sig2)();

        /*      Forks a shell. The child process does the work, the    */
        /*      parent waits until the child terminates.               */
        /*      A second shell is used so that the effective UID       */
        /*      can be set to the real UID for the shell command       */
        /*      without losing the "root" effective UID of the parent  */

        if((cupid = fork()) == 0)       /*      child process   */
        {
                (void) setuid(my_uid);          /* sets effective and real UID  */
                                        /* to that of the real UID      */
                                        /* determined when the program          */
                                        /* started.                     */
                (void) execl("/bin/sh", "sh", "-c", tmp, 0);
                printf("error in exec\n");      /* If we are still here.*/
                exit(ERROR);            /* the exec failed.             */
                                        /* DO NOT exit through by() or          */
                                        /* we may mess up the parent */
        }

        /*              Parent process... continued                    */

        if (cupid == -1)        /* parent checks to ensure the fork     */
        {                       /* was successful.                      */
                printf("Can't fork process\n");
                return;
        }
        sig1 = signal(SIGINT, SIG_IGN);                 /* Ignore inturupts,    */
        sig2 = signal(SIGQUIT, SIG_IGN);        /* while waiting.       */

        wait_flg = 0;
        while (((w = wait(&wait_flg)) != cupid) && w != -1)
                ;       /*  wait until the child dies   */

        (void) signal(SIGINT, sig1);    /* reset inturupt handleing     */
        (void) signal(SIGQUIT, sig2); /* for things ignored above.     */
}
```

```
/*              ALL NEW ROUTINE                                        */

/*      Conference scheduling routine.                                 */
/* Prompts for a list of login ID's to schedule for the conference.  */
/* This list , in the form of a line of white-space (blanks or tabs)  */
/* seperated character strings is scanned and broken down into login */
/* ID's. The passwd file is then searched for a matching ID. If not   */
/* found, a message to that effect is printed and the next string of  */
/* characters is processed. If it is found, the path to the ecal */
/* directory is constructed and open_ecal is called. If open_ecal fails,*/
/* confer returns to the calling routine. If open_ecal succeds, then  */
/* the data file is now open and the file is locked. The schedule for */
/* this participant is then populated and we go on tho process the   */
/* next string of characters. open_ecal() also popupates the path    */
/* with the correct path to the ecal LCK file for the ecal data opened */
/*                                                                  */
/* The routine next prompts for a range of times in which to try and    */
/* schedule the meeting. This range of times is broken down into    */
/* five minute intervals and run through insert_check() for each active */
/* participant and times conflicting with any one or more participants  */
/* is so marked. A list of available times is then dispayed and the  */
/* is asked to input a specific range of times for the meeting. This  */
/* range is then double checked for availability, the description is  */
/* prompted for, and then the insertions are done. All users scheduled  */
/* are then sent mail concerning the proposed meeting. Before returning*/
/* the program runs the clean_up() routine to remove all LCK files, etc.*/


confer()
{
struct passwd *getpwnam(), *passwd_rec;
char string[20], desc[DESC_LENGTH], temp_begin[10];
char temp_end[10], command[100];
int i, j, k, locn, num1, num2;
short times[290];

        printf("Enter the login ID's of the participants:\n");
        printf("-> %s ", partc[ME].name);
        (void) gets(command);
        i = 0;
        k = 0;
        j = 1;
```

```
/*          confer() (continued)    ALL NEW ROUTINE                        */

        while(command[i] != NULL)
        {
                while ((command[i] == '\n') || (command[i] == ' ')
                        || (command[i] == '\t'))
                {
                        i++;
                        continue;
                }
                if (command[i] == NULL)                 break;
                while ((command[i] != '\n') && (command[i] != ' ')
                        && (command[i] != '\t') && (command[i] != NULL))
                {
                        partc[j].name[k++] = command[i++];
                }
                partc[j].name[k++] = NULL;
                passwd_rec = getpwnam(partc[j].name);

                if (passwd_rec == NULL)
                {
                        printf("user id %s does not exist\n", partc[j].name);
                        partc[j].name[0] = NULL;
                        k = 0;

                        while ((command[i] == '\n') || (command[i] == ' ')
                                || (command[i] == '\t')) i++;

                        continue;
                }
                if (open_ecal(j) != 0)
                {
                        printf("Can't open ecal file for %s\n", partc[j].name);
                        clean_up();
                        return;
                }
                get_schedule(partc[j].fp, partc[j].sch, &partc[j].length);
                partc[j].active = TRUE;
                lngth++;
                j++;
                k = 0;

                if (lngth >= MAX_PARTIC)
                {
                        printf("To many participants entered.  ");
                        printf("Only using the first %d\n",
                                        MAX_PARTIC);
                        break;
                }
        }
```

```
/*          confer() (continued)    ALL NEW ROUTINE                          */

        printf("Enter beginning and ending times of range in which\n");
        printf("to try and schedule the conference. -> ");
        (void) gets(string);

        get_time(2, string, &num1, &num2);
        (void) strcpy(temp_begin, time_string(num1));
        (void) strcpy(temp_end, time_string(num2));
        for (j = 0; j <= 288; j++)        times[j] = TRUE;
        for (i = 0; i < lngth; i++)
        {
                for (j = num1; j <= num2; j++)
                {
                        if (insert_check(partc[i].sch, partc[i].length,
                                j, j+1, &k) == FALSE)
                        {
                                times[j] = FALSE;
                        }
                }
        }

        printf("Available times in this time frame are the following:\n");
        k = 0;
        for (j = num1; j <= num2; j++)
        {
                if (times[j] == TRUE)
                {
                        (void) strcpy(temp_begin, time_string(j));
                        while (times[j] == TRUE)
                        {
                                if (j >= num2)
                                {
                                        (void) strcpy(temp_end,
                                                time_string(j));
                                        break;
                                }
                                j++;
                        }
                        k++;
                        (void) strcpy(temp_end, time_string(j));
                        printf("        %s to %s\n", temp_begin, temp_end);
                }
        }
```

```
/*              confer() (continued)    ALL NEW ROUTINE                         */

        if (k == 0)
        {
                printf("No times available in the time range\n");
                printf("Do you wish to try again (y/n)? ");
                (void) gets(string);
                clean_up();
                if (string[0] == 'y')           confer();
                return;
        }
        printf("enter begining and ending time for the conference: ");
        (void) gets(string);
        get_time(2, string, &i, &j);

        if ((i < num1) || (j > num2))
        {
                printf("time entered is outside of requested range\n");
                clean_up();
                return;
        }
        num1 = i;
        num2 = j;

        for (i = num1; i <= num2; i++)
        {
                if (times[i] != TRUE)
                {
                        printf("The time range entered conatains ");
                        printf("a non-available time.\n");
                        clean_up();
                        return;
                }
        }
        printf("Enter description of conference.\n-> ");
        (void) gets(desc);

        (void) strcpy(command, "mail ");
```

```
for (i = 0; i < lngth; i++)
{
        (void) strcat(command, partc[i].name);
        (void) strcat(command, " ");
        if ((partc[i].length + 1) >= MAX_APP_NUM)
        {
                printf("No room for a new calendar entry ");
                printf("in the calendar file of %s\n", partc[i].name);
        }
        else if (insert_check(partc[i].sch, partc[i].length,
                num1, num2, &locn) == FALSE)
        {
                printf("lost our insertion slot for %s\n",
                                partc[i].name);
        }
        else
        {
                do_insert(partc[i].sch, &partc[i].length,
                        locn, num1, num2, desc);
                partc[i].sch[locn].requestor = my_uid;
                update_disk(partc[i].fp, partc[i].sch,
                                &partc[i].length);
        }
}

/*      Send a mail message to everyone involved        */

(void) sprintf(fil_string[0], "/tmp/ecal.XXXXXX");
(void) mktemp(fil_string[0]);
confer_flag++;
(void) strcat(command, "<");
(void) strcat(command, fil_string[0]);
if ((mail = fopen(fil_string[0], "w")) == NULL)
{
        printf("Unable to send mail to participants\n");
}
```

```
/*              confer() (continued)    ALL NEW ROUTINE                    */
        else
        {
                (void) fprintf(mail, "\n\n\t\tECAL NOTICE:\n\n");
                (void) fprintf(mail,
        "You have tentatively been scheduled for a meeting as follows:\n\n");
                (void) fprintf(mail, "\tDATE: %s, %s %d, 19%d:\n",
                                weekday[wday_num], month[mon_num],
                                mday_num, year_num);
                (void) strcpy(temp_begin, time_string(num1));
                (void) strcpy(temp_end, time_string(num2));
                (void) fprintf(mail, "\tTIME:  %s to %s\n",
                                temp_begin, temp_end);
                (void) fprintf(mail, "\tWHAT:           %s\n\n", desc);
                (void) fprintf(mail, "Please confirm your attendance to");
                (void) fprintf(mail, " %s by return mail\n", partc[0].name);

                (void) fclose(mail);
                (void) chmod(fil_string[0], 0644);
                ex(command);
                (void) unlink(fil_string[0]);
                confer_flag--;
        }

        clean_up();
}
```

```
/*                        ALL NEW ROUTINE                              */

/*      updates calendars on the remote machine thru use of the        */
/*      uux command. "ecal" and "cat" must exist on the remote      */
/*      machine and be executable on the remote machine BY REMOTE      */
/*      MACHINES. Mail is sent to the owner of the calendar file    */
/*      on the remote machine concerning the impending update.         */

up_date()
{
struct sch1 sch;
int i;
char command[512], mac_target[25], log_target[25];

        (void) printf("Enter the machine to update calendar on: ");
        (void) gets(mac_target);
        (void) printf(
                "Enter the login on that machine to update calendar of: ");
        (void) gets(log_target);

        confer_flag = TRUE;

        (void) sprintf(fil_string[0], "/tmp/ecal.XXXXXX");
        (void) mktemp(fil_string[0]);
        if ((mail = fopen(fil_string[0], "w")) == NULL)
        {
                printf("Unable to create temporary files\n");
                return;
        }
        confer_flag++;

        (void) fprintf(mail, "%s!%s\n", mac_target, log_target);

        i = 0;

        (void) fprintf(mail, "%s!%s\n", sys_name, partc[ME].name);

        /* No need to send sch.active, sch.mod, sch.a, or sch.rec_num*/
        /* the remote machine.                                      */
/*              up_date() (continued) ALL NEW ROUTINE                         */

        while( rd_dsk(partc[ME].fp, &sch, i++) != NULL)
        {
                if (sch.active == FALSE)        continue;

                (void) fprintf(mail, "%d\n", sch.begin);
                (void) fprintf(mail, "%d\n", sch.end);
                (void) fprintf(mail, "%d\n", sch.month);
                (void) fprintf(mail, "%d\n", sch.wday);
                (void) fprintf(mail, "%d\n", sch.mday);
                (void) fprintf(mail, "%d\n", sch.year);
                (void) fprintf(mail, "%d\n", sch.specific);
                (void) fprintf(mail, "%s\n", sch.info);
        }
        (void) fclose(mail);

        (void) sprintf(fil_string[1], "/tmp/ecal.XXXXXX");
        (void) mktemp(fil_string[1]);
        if ((mail = fopen(fil_string[1], "w")) == NULL)
        {
                (void) printf("Unable to create temporary files\n");
                return;
        }
        confer_flag++;

        (void) sprintf(command, "mail %s!%s <%s\n", mac_target,
                        log_target, fil_string[1]);
        (void) fprintf(mail, "%s!%s", sys_name, partc[ME].name);
        (void) fprintf(mail, " has updated your calendar\n");
        (void) fclose(mail);

        (void) chmod(fil_string[0], 0644);
        printf("%s", command);
        ex(command);

        (void) sprintf(command, "uux \"%s!cat !%s | ecal update\"\n",
                        mac_target, fil_string[0]);

        (void) chmod(fil_string[0], 0644);
        printf("%s", command);
        ex(command);

        (void) unlink(fil_string[0]);
        confer_flag--;
        (void) unlink(fil_string[1]);
        confer_flag--;
}
```

```
/*                         ALL NEW ROUTINE                              */

/*      This routine take the ascii data file passed to it from the     */
/*      remote machine and uses it to update a users calendar on this*/
/*      machine. Mail is sent to the owner of the file, notifying them */
/*      of the update and what was added.  Conflicts in the calendar */
/*      are noted and sent to the owner of the file on the local        */
/*      machine.  A success/failure message is sent to the remote       */
/*      requestor.                                                      */

update()
{
struct sch1 sch;
struct passwd *getpwnam(), *passwd_rec;
char requestor[25], tmp[512], target[512], *tmp1;
char command[512];
int locn;

        if ((gets(target) == NULL)    ||
           (gets(requestor) == NULL))                 return;

        tmp1 = target;
        while ((*tmp1 != '!') && (*tmp1 != NULL))           tmp1++;
        if (*tmp1 == NULL)   return;
        tmp1++;
        if (*tmp1 == NULL)   return;

        (void) strcpy(partc[ME].name, tmp1);
        passwd_rec = getpwnam(partc[ME].name);
        if (passwd_rec == NULL)              return;
        if (open_ecal(ME) != 0)              return;

        confer_flag = TRUE;
        (void) sprintf(fil_string[0], "/tmp/ecal.XXXXXX");
        (void) mktemp(fil_string[0]);
        if ((mail = fopen(fil_string[0], "w")) == NULL)             return;
        confer_flag++;

        (void) fprintf(mail, "\n%s has updated your ecal calendar\n",
                        requestor);
        (void) fprintf(mail,
                "The following entrys were not added due to conflicts:\n");
```

```
/*              update() (continued)   ALL NEW ROUTINE                    */

        while( gets(tmp) != NULL )
        {
                (void) sscanf(tmp, "%hd", &sch.begin);
                if ( gets(tmp) == NULL )        break;
                (void) sscanf(tmp, "%hd", &sch.end);
                if ( gets(tmp) == NULL )        break;
                (void) sscanf(tmp, "%hd", &sch.month);
                if ( gets(tmp) == NULL )        break;
                (void) sscanf(tmp, "%hd", &sch.wday);
                if ( gets(tmp) == NULL )        break;
                (void) sscanf(tmp, "%hd", &sch.mday);
                if ( gets(tmp) == NULL )        break;
                (void) sscanf(tmp, "%hd", &sch.year);
                if ( gets(tmp) == NULL )        break;
                (void) sscanf(tmp, "%hd", &sch.specific);
                if ( gets(tmp) == NULL )        break;
                (void) sscanf(tmp, "%s", sch.info);

                specific = sch.specific;
                wday_num = sch.wday;
                mday_num = sch.mday;
                mon_num  = sch.month;
                year_num = sch.year;
                get_schedule(partc[ME].fp, partc[ME].sch, &partc[ME].length);
                if (((partc[ME].length + 1) >= MAX_APP_NUM) ||
                    (insert_check(partc[ME].sch, partc[ME].length,
                        sch.begin, sch.end, &locn) == FALSE))
                {
                        if (strcmp( sch.info, partc[ME].sch[locn].info) == 0)
                        {
                                continue;
                        }
                        (void) fprintf(mail, "\n%s, %s %d, %d\n",
                                weekday[sch.wday], month[sch.month],
                                sch.mday, sch.year);
                        (void) fprintf(mail, "%s\n", sch.info);
                        continue;
                }
                else
                {
                        do_insert(partc[ME].sch, &partc[ME].length,
                                locn, sch.begin, sch.end, sch.info);
                        update_disk(partc[ME].fp, partc[ME].sch,
                                        &partc[ME].length);
                }
        }
        (void) fclose(mail);
        (void) sprintf(command, "mail %s <%s\n", partc[ME].name,
                        fil_string[0]);
        (void) chmod(fil_string[0], 0644);
        ex(command);
        (void) unlink(fil_string[0]);
        confer_flag--;
}
```

```
/*                          ALL NEW ROUTINE                              */

open_ecal(i)
int i;
{
struct passwd *getpwnam(), *passwd_rec;
struct stat info;
char sys_string[25], temp[200];
long clock;

        passwd_rec = getpwnam(partc[i].name);
        if (passwd_rec == NULL)             return(ERROR);
        (void) strcpy(partc[i].path, passwd_rec->pw_dir);
        (void) strcat(partc[i].path, "/ecal");
        (void) strcpy(temp, partc[i].path);
        (void) strcat(temp, "/ecal.file");

        if (access(partc[i].path, 00) != 0)
        {
                (void) sprintf(sys_string, "mkdir %s", partc[i].path);
                (void) system(sys_string);
                                        /* only root can use mknod to */
                                        /* create a directory!        */
                if (access(partc[i].path, 00) != 0)       return(ERROR);
        }
        if (i == ME)
        {
                if (chdir(partc[ME].path) != 0)           return(ERROR);
        }
        (void) strcat(partc[i].path, "/LCK.ECAL");
        if (stat(partc[i].path, &info) == 0)
        {
                (void) time(&clock);
                if ((clock - info.st_mtime) < (HOURS * 3600))
                {
                        printf("entry for %s is locked- try later\n",
                                partc[i].name);
                        return(ERROR);
                }
        }
        if ((partc[i].fp = fopen(partc[i].path, "w")) == NULL)
        {
                printf("cannot create lock for %s\n", partc[i].name);
                return(ERROR);
        }
        (void) fclose(partc[i].fp);
        if ((partc[i].fp = fopen(temp, "r+")) == NULL)
        {
                if ((partc[i].fp = fopen(temp, "w")) == NULL)
                {
                        printf("can't open or create ecal data file\n");
                        return(ERROR);
                }
                (void) fclose(partc[i].fp);
                if ((partc[i].fp = fopen(temp, "r+")) == NULL)
                {
                        printf("can't open or create ecal data file\n");
                        return(ERROR);
                }
        }
        lck_flg = 1;
        return(0);
}
```

```
edit(parms)
char *parms;
{
char string[20];
char temp_begin[10], command[100];
int i, num1, ed;

        if (partc[ME].length == 0)
        {
                printf("No appointments to edit.\n");
                (void) sleep(2);
                return;
        }
        printf("which editor would you like to use:\n");
        printf("\t1) vi\t2) ed\nwhich? ");
        (void) gets(string);
        if (string[0] == '2')     ed = 2;
        else if (string[0] == '1')        ed = 1;

        if (*parms != NULL)   (void) strcpy(string, parms);
        else
        {
                printf(
        "Enter beginning and ending time of appointment to edit. -> ");
                (void) gets(string);
        }
        get_time(1, string, &num1, &num1);
        (void) strcpy(temp_begin, time_string(num1));

        for (i=0; i<partc[ME].length && partc[ME].sch[i].begin < num1; i++);
        if (partc[ME].sch[i].begin == num1)
        {
                confer_flag = TRUE;
                (void) sprintf(fil_string[0], "/tmp/ecal.XXXXXX");
                (void) mktemp(fil_string[0]);
                if ((mail = fopen(fil_string[0], "w")) == NULL)                 return;
                confer_flag++;
                (void) fprintf(mail, "%s\n", partc[ME].sch[i].info);
                (void) fclose(mail);
                (void) chown(fil_string[0], my_uid, 0);

                if (ed == 2)
                {
                        (void) sprintf(command, "ed %s", fil_string);
                }
                else    (void) sprintf(command, "vi %s", fil_string);
                ex(command);
                if ((mail = fopen(fil_string[0], "r")) == NULL)
                {
                        printf("can't open edited file\n");
                        (void) sleep(2);
                        return;
                }
                if (fgets(command, 100, mail) == NULL)
                {
                        printf("empty file\n");
                        partc[ME].sch[i].info[0] = NULL;
                }
                else if (command[0] == '\n')  partc[ME].sch[i].info[0] = NULL;
                else (void) strcpy(partc[ME].sch[i].info, command);

                partc[ME].sch[i].mod = TRUE;
                update_disk(partc[ME].fp, partc[ME].sch, &partc[ME].length);
                (void) fclose(mail);
                (void) unlink(fil_string[0]);
                confer_flag = FALSE;
        }
        else
        {
                printf("Cannot find %s appointment to edit.\n",
                        temp_begin);
                (void) sleep(2);
        }
}
```

```
reply(parms)
char *parms;
{
char string[20];
struct passwd *getpwuid(), *passwd_rec;
char temp_begin[10], command[100], temp_end[10];
int i, num1, ok;

        ok = 0;
        for (i = 0; i < partc[ME].length; i++)
        {
                if (partc[ME].sch[i].requestor != -1)
                {
                        ok = 1;
                        break;
                }
        }
        if ( ok == 0)
        {
                printf("No tentative appointments to reply to.\n");
                (void) sleep(2);
                return;
        }

        if (*parms != NULL)   (void) strcpy(string, parms);
        else
        {
                printf(
        "Enter beginning and ending time of appointment to reply to. -> ");
                (void) gets(string);
        }
        get_time(1, string, &num1, &num1);
        (void) strcpy(temp_begin, time_string(num1));

        for (i=0; i<partc[ME].length && partc[ME].sch[i].begin < num1; i++)  ;

        if (partc[ME].sch[i].begin == num1)
        {
                if (partc[ME].sch[i].requestor == -1)
                {
                        printf("%s is not a tentative appointment.\n",
                                temp_begin);
                        (void) sleep(1);
                        return;
                }
                passwd_rec = getpwuid(partc[ME].sch[i].requestor);
        if (passwd_rec == NULL)
        {
                printf("Can't find the requestor in the passwd file\n");
                (void) sleep(1);
                return;
        }

        printf("Do you wish to confirm attendence or send regrets");
        printf(" \(c/r\): ");
        (void) gets(string);

        if ((string[0] != 'c') && (string[0] != 'r'))
        {
                printf("Invalid response to confirmation query\n");
                (void) sleep(1);
                return;
        }


/*      Send a mail message to the requestor                  */

        (void) sprintf(command, "mail %s", passwd_rec->pw_name);
        (void) sprintf(fil_string[1], "/tmp/ecal.XXXXXX");
        (void) mktemp(fil_string[1]);
        confer_flag++;
        (void) strcat(command, "<");
        (void) strcat(command, fil_string[1]);
        printf(command);

        if ((mail = fopen(fil_string[1], "w")) == NULL)
        {
                printf("Unable to send reply mail to requestor\n");
        }
```

```
		else
		{
			(void) fprintf(mail, "\n\n\t\tECAL NOTICE:\n\n");
			(void) fprintf(mail,
			"Regarding the following tentative meeting:\n\n");
			(void) fprintf(mail, "\tDATE: %s, %s %d, 19%d:\n",
					weekday[wday_num], month[mon_num],
					mday_num, year_num);
			(void) strcpy(temp_begin,
				time_string(partc[ME].sch[i].begin));
			(void) strcpy(temp_end,
				time_string(partc[ME].sch[i].end));
			(void) fprintf(mail, "\tTIME:  %s to %s\n",
					temp_begin, temp_end);
			(void) fprintf(mail, "\tWHAT:        %s\n\n",
					partc[ME].sch[i].info);
			(void) fprintf(mail,
				"%s is ", partc[ME].name);

			if (string[0] == 'c')
			{
				(void) fprintf(mail,
				"confirming their intention to attend\n");
				partc[ME].sch[i].requestor = -1;
				partc[ME].sch[i].mod = TRUE;
			}
			else
			{
				(void) fprintf(mail, "unable to attend\n");
				partc[ME].sch[i].mod = TRUE;
				partc[ME].sch[i].active = FALSE;
			}

			update_disk(partc[ME].fp, partc[ME].sch,
					&partc[ME].length);
			(void) fclose(mail);
			(void) chmod(fil_string[1], 0644);
			ex(command);
			(void) unlink(fil_string[1]);
		}
	}
	else
	{
		printf("Cannot find %s tentative appointment to reply to.\n",
			temp_begin);
		(void) sleep(2);
	}
}
```

AN ELECTRONIC CALENDAR SYSTEM IN A DISTRIBUTED UNIX® ENVIRONMENT

by

DOUGLAS M. CLABOUGH

B. S., North Carolina State University, Raleigh, 1980

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

ABSTRACT

Current efforts in the area of office automation are concentrating on tools; each of which automates a specific office function. An electronic calendar is one particular tool of this type which aids in the scheduling of personal time, meetings, and common resources.

This paper deals with the extensions to an existing electronic calendar. The new features provide a conference scheduling capability and a facility for updating calendars on remote machines. These capabilities are important in the modern office environment. The first capability saves much time in arranging meetings between several persons. The second capability allows users who maintain accounts on different machines to have access to an accurate copy of their calendar regardless of the machine on which they happen to be working. This project implemented the scheduling capability and the remote system update capability while maintaining calendar integrity and user privacy.