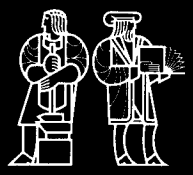
LABORATORY FOR
COMPUTER SCIENCE



MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TR-316

LOOSE CONSISTENCY IN A PERSONAL COMPUTER MAIL SYSTEM

Michael H. Comer

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

Loose Consistency in a Personal Computer Mail System

by

Michael H. Comer

This paper describes research performed at
Hewlett-Packard Personal Office Computer Division

Massachusetts Institute of Technology
Laboratory for Computer Science
Cambridge, Massachusetts 02139

Loose Consistency in a Personal Computer Mail System

by
Michael H. Comer

Submitted to the
Department of Electrical Engineering and Computer Science
on May 11, 1984 in partial fulfillment of the requirements
for the Degrees of Bachelor of Science and Master of Science

Abstract

This thesis discusses a prototype mail system designed with the personal computer in mind. The nature of the personal computer presents a number of problems for the traditional networked mail system. The primary problem arises from the fact that personal computers are powered down more often then their mainframe counterparts. This problem is compounded by the fact that a personal computer that is powered up might not be listening to the network (online). Because of this, it is potentially difficult to find a time when both the source and destination personal computers are online so that a message can be exchanged. This problem is addressed by the mail system by using a mail serving machine and the concept of localization of control, which states that the initiative for all exchanges of information resides with the local system (as opposed to with the server).

A secondary problem presented by the personal computer results from the fact that personal computers are inexpensive. Because of this, a given user might have a number of machines. This user would likely want a single mail identity across all of his/her machines. In addition, he/she would like to have equal access from these machines to his/her messages, with a minimum of repetition. For example, a message that is read and saved on one machine should not appear as a new message on any of the other machines; but rather, it should be available for examination as a saved message at any of these other machines. Such consistency, however, cannot always be strictly maintained. Thus, the mail system implements the concept of loose consistency. The idea of a machine is generalized to the concept of a mailbox, which contains the mail functionality and files and which can be moved from one machine to another. A protocol is developed to maintain loose consistency of a user's mailboxes. A prototype mail system using this protocol was built and tested, but not placed in service.

Finally, integration of the protocol into a mail system where loose consistency is maintained across heterogeneous machines, is discussed. There is a discussion of how the prototype mail system might be integrated with XEROX's Grapevine mail server. There is also a discussion of how the loose consistency algorithm can be used in other systems that maintain multiple copies of some object.

Key words: personal computer, electronic mail, loose consistency, mail server, weighted voting.

Acknowledgements

I would like to thank Professor Jerome Saltzer for his comments and suggestions that prompted me to examine the subject in a much broader light, and for the time he invested in reading the drafts of this thesis.

This topic was investigated and the mail system developed at Hewlett-Packard Personal Office Computer Division. I would like to thank Greg Sorknes for his supervision of the thesis at Hewlett-Packard.

Table of Contents

Table of Figures

Chapter One

Introduction

Electronic mail is one of the primary services of a networked environment. Traditionally, this mail system was restricted to multi-user computers where the high cost and complexity of providing the network link could be distributed over a large number of users. The advent of the personal computer, with its increasingly powerful processors, and the development of LSI and VLSI networking technology, however, threaten to break this tradition.

Currently, the thrust of the networking market for personal computers tends to be directed toward sharing expensive peripheral devices such as printers and large disks. Mainframe systems may be supported on the network but they will function primarily as intelligent servers. Neither the personal computer nor the mainframe will be a full partner in the network of the other.

Although electronic mail is not yet a big issue in the personal computer network market, it seems as if it is just around the corner. As soon as personal computer networks are in place, users (and vendors) will be looking for ways to take advantage of their network. Mail is a logical candidate.

The next step seems to be the integration of the personal computer mail system into the mainframe mail system. In this way, mail can be sent longer distances over the existing mainframe mail system. This integration would probably occur in one of two ways.

One way to integrate the two systems is to simply build an interface between the two. For example, there might be a mainframe server on the personal computer network that is connected to another network. This server could act as a mail gateway, shuttling, and possibly translating, mail between the two networks. This type of integration would probably be used in systems where the mail protocols for the two networks differ.

The second way involves making the personal computer a full partner in the mainframe network. This is a much more complex undertaking than the idea of the personal computer network presented above. The interface to the mainframes is potentially much different than the notion of a server. This type of integration would probably be used in systems that are designed from the start for this

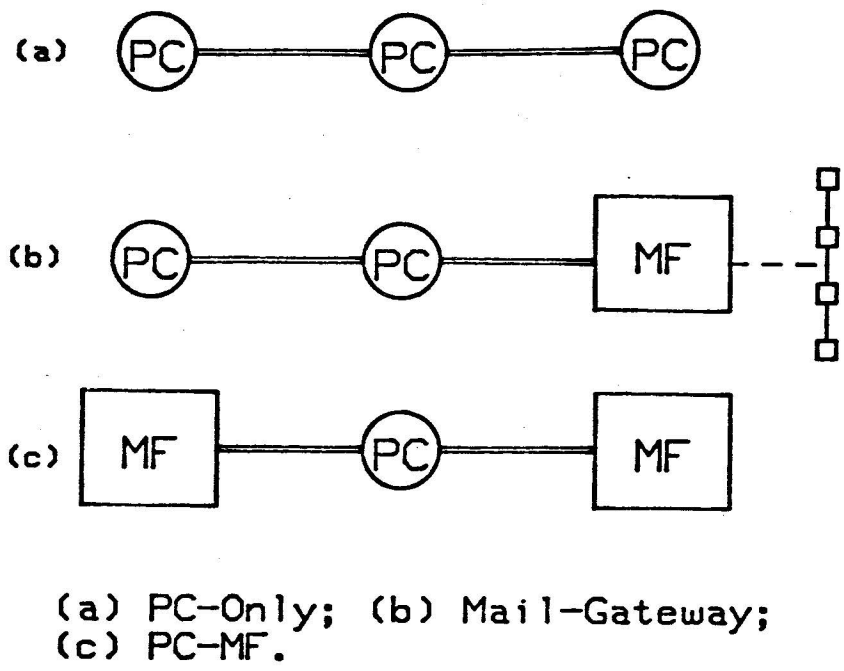


Figure 1-1:Types of Mail Networks

purpose.

Three kinds of personal computer mail networks have been presented (see figure 1-1). They are: 1) Personal computers only (PC-Only); 2) Personal computers with mail gateway (Mail-Gateway); and 3) Mixture of personal computers and mainframes (PC-MF). There are problems associated with each of these schemes which result from the nature of the personal computer.

The personal computer tends to be offline more often then its mainframe counterpart. Because it *is* personal, it will tend to be powered down by its owner when it is not in use. In addition, it might be powered up but disconnected from the network, as might be the case if it is portable and/or its connection to the network is via telephone line. As a result, the sending computer must take responsibility for a potentially large requeuing effort, and, therefore, must be prepared to allocate the appropriate resources. As the amount of resources across the network dedicated to this function becomes large, it becomes reasonable to share these resources by (functionally) centralizing them within the network.

This centralization of computing resources is the first step toward a mail server. This server would probably be a more powerful machine that could handle the routing of all the mail within its network

and the requeuing of mail to nodes which are not online. (The serving machine could also perform other functions, such as managing shared resources.) Thus, any message from one personal computer to another first goes to the mail server, which then queues that message for transmission. The server transmits the message only when the destination machine asks for it. In this way, it can be sure that the destination is ready to receive the message, minimizing requeuing efforts.

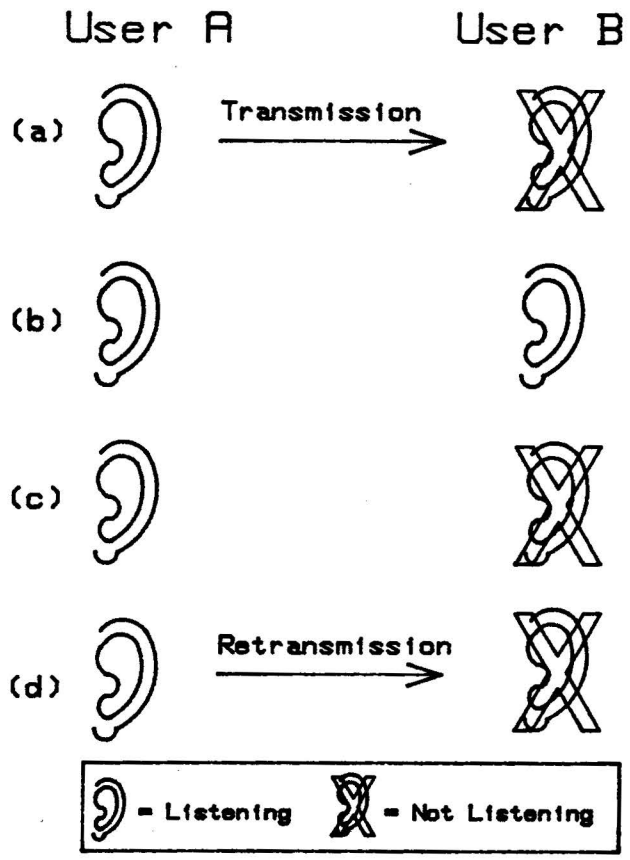


Figure 1-2:A Tries to Send Message to B

There is another problem associated with the fact that a personal computer tends to be offline more than its mainframe counterpart. Suppose there are two personal computers: A, online, and B, offline. A transmits a message to B (figure 1-2a), and, receiving no acknowledgement, requeues the message for later transmission. B now comes online, works for a while, and powers down again (figure 1-2b,c). A retransmits the message, which fails again (figure 1-2d). In this manner, B might never receive the message from A. The presence of a mail server would preclude this.

The argument is the same for the Mail-Gateway scheme. In this scheme, it would make sense for the machine that is acting as the mail gateway also to act as the local net mail server (although this is not necessary).

The situation for the PC-MF scheme is different. It is certainly desirable to have a single mail facility that spans the entire network. This network-wide mail facility is, in fact, the goal of XEROX PARC's Grapevine system [BIRRELL82]. Such a system would be even more desirable if it allowed the integration of personal computers while preventing the problems discussed above. If such a network-wide server is not available, it is necessary to have some kind of server to orchestrate mail flow between the personal computers and the rest of the network.

It seems that each of these proposed personal computer mail systems requires the presence of a mail server, the primary function of which is to queue messages from one machine for another. The maintenance of message integrity is an important part of this queuing function. Thus, when a personal computer sends a message, the server is responsible for ensuring that the message is queued for delivery.

If a moderately powerful machine is to be on the network, what is the use of the mail server? It seems as if the mail system, at least on the local level, could be implemented using files on the larger machine and remote file access to read and write to different mail files. This type of system would work fine for a network consisting of diskless computers and a shared resource manager (SRM).

As the central machine becomes more loaded down, however, it becomes increasingly unreliable, prone to crashing. A user will want to have access to his/her mail files regardless of the state of the central machine. This problem can be solved by keeping a copy of all a user's messages on a local disk, and providing a means by which the local machine can queue up messages to be sent when the connection is reestablished. This set of local files and functionality is called a *mailbox*.

A user's mailbox resides within a particular memory. The form this memory takes can vary across the spectrum from floppy disks to the hard disks typically found with very large systems. The larger the memory the greater the potential for storing mailboxes. A single machine could, therefore, support a large number of users if each user had his/her own floppy disk mailbox or if there was a single disk containing all the mailboxes.

The first situation, a floppy disk for each user, introduces the idea of a portable mailbox, a mailbox

that exists outside any single machine. Such a mailbox cannot be accessed until it is connected to some machine and cannot access the mail server until it is connected to some machine on the network. This connection of a mailbox to a machine on a network is called *activating* that mailbox. Thus, the user has the ability to carry around his/her mailbox, activating it from time to time to receive new messages. In this way, a single mailbox can be used on many machines. It is, therefore, the memory that is important for consideration, rather than the physical machine.

The second situation, a single disk, shows a situation where the memory is not portable. In this situation, in order for the user to access his/her mail on another machine, that user would require a mailbox on that other machine. Problems arise, however, when a given user wants to have more than one mailbox. Suppose user A has two mailboxes, one of which is currently on a machine at work and the other on a machine at home. Suppose further that user B sends A a message that is placed in the mailbox at work. A does not read the message at work, but rather goes home and logs on to receive his/her new messages. Of course, the message from B will not be listed as a new message. A is forced either to wait until he/she goes to work again or to examine all his/her messages in the server's copy of his/her mail log, looking for messages not yet seen. Neither of these alternatives is attractive.

On the other hand, suppose that A has two activated mailboxes. B now sends a message to A that is placed is both of these mailboxes. A later reads the message from one of the mailboxes and deletes it. A certainly does not want that same message to appear as new in the other mailbox.

These problems introduce the concept of *loose consistency*. Ideally, the mail logs in each of a given user's mailboxes should be identical. Unfortunately, this is not always possible since one of the user's mailboxes might not be activated for a long period of time. Loose consistency guarantees that the mail logs will *eventually* become consistent. If a message is deleted from one mailbox, it will eventually be deleted from all mailboxes. If a message is kept in one mailbox, it will eventually be kept in all of them. Following from this, once a new message has been *read*, it must be either kept or deleted, (loosely) ensuring that it will not be displayed as new again.

The primary goal of this project is to develop a mail system that implements the concept of loose consistency. A further goal is make the system insensitive to the state of the server, providing the user with the ability to read and create messages at any time, through the implementation of the mailbox. The final goal is to build a mail gateway allowing communications between mail networks. A prototype mail system satisfying these goals was built and tested, but not placed in service.

Chapter Two

The Prototype Environment

The prototyping effort for the mail facility was performed using Hewlett-Packard's Shared Resource Management System. This system consists of a number of workstations (see figure 2-1) and the shared resource manager (SRM) connected together through a central multiplexer (see figure 2-2). The purpose of the multiplexer is to arbitrate requests for SRM resources (printers, files, etc.) from each of the workstations. The network is, therefore, in a star configuration. Direct communications between workstations is not supported by the network. The execution of user programs on the SRM is also not allowed.

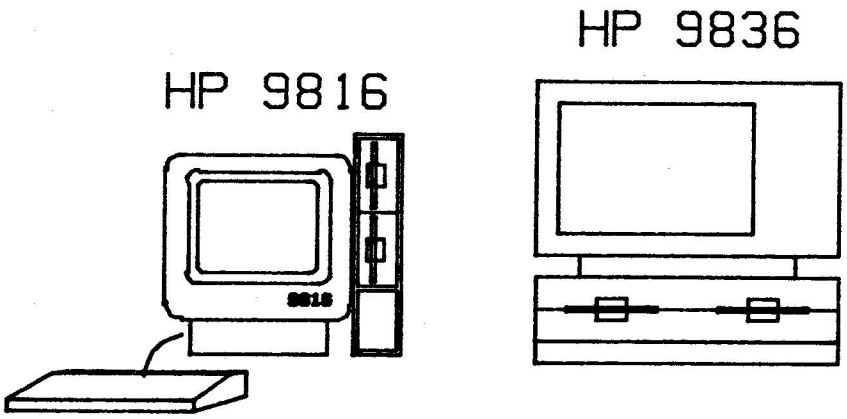


Figure 2-1:Workstations

The workstations supported on this network are the Hewlett-Packard (HP) 9816, 9826, and 9836 desktop computers. Each of these workstations can provide access to the SRM through either a PASCAL or BASIC operating system. Because the PASCAL system runs faster than the BASIC system, PASCAL was chosen as the development environment. The PASCAL operating system also provides a mechanism through which a user-written routine can be linked in with a timer interrupt

service routine (ISR -- See Glossary). In this way, the user can create a routine that will periodically awaken and perform some task.

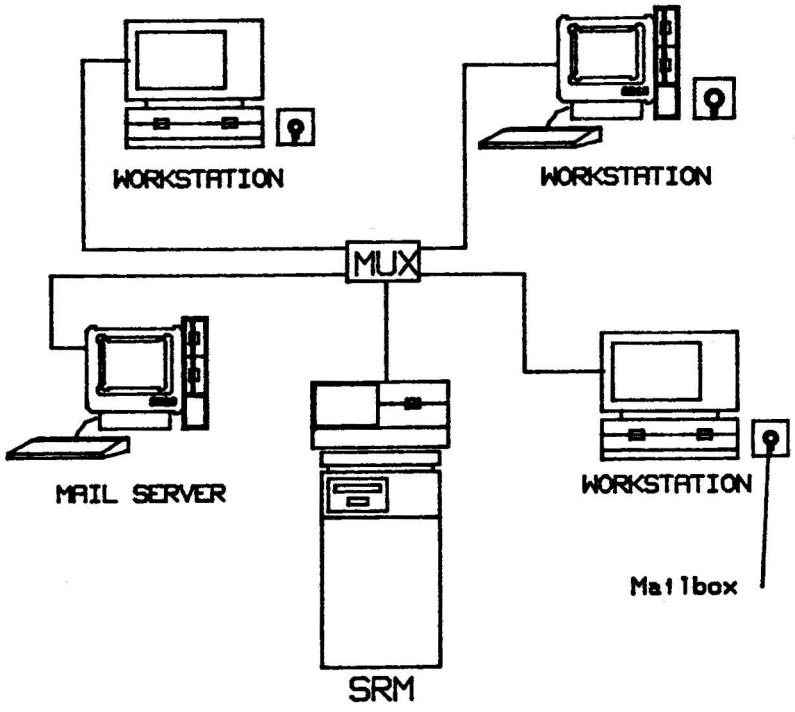


Figure 2-2:Prototype Target Network

The prototype mail facility is based upon an existing mail facility on the SRM network. This existing mail facility uses a shared disk to store the mailboxes of each user. For example, if user A wants to send a message to user B, A merely writes the message into the message file belonging to B. B has a routine that (by using the timer ISR) periodically wakes up and examines this file to see if there is any mail. If there is, a message appears on B's screen.

Chapter Three

Mail System Fundamentals

3.1 The Mail Protocol

The mail protocol describes the sequence of messages required to maintain loose consistency in the mail system. Under normal circumstances, all of these messages originate at a workstation. The server merely routes the messages, either from one user to another, or from one mailbox to another belonging to the same user. The only time the server creates a message is when a user must be informed of an error, such as an illegal destination name. There are three classes of messages that are used in the protocol: new, keep, and delete.

A new message is a message that is to be presented to the user as not having previously been read in any of the user's mailboxes. When a message, M, is created for a user, that message is marked as new and queued for the destination user, D. All of D's mailboxes that are activated will eventually query the server and obtain M and all other new messages. These new messages will be queued in the mailbox until the user makes a decision concerning their dispositions, whether each is to be kept or deleted.

Suppose that D now accesses the mail system through one of his/her activated mailboxes. He/she will then be presented with each of the new messages. After each is read, D decides whether the message is to be kept or deleted. If the message is kept in the mailbox, another message is sent to the server reflecting that fact. The server then queues the keep message for the other mailboxes belonging to the user. The server maintains a copy of all the messages that are kept by the user. The primary reason for this is to provide security against loss of information in the event of the destruction of the mailbox. If the message is deleted, a message to that effect is sent to the server, which deletes the orginal message and queues the delete message for the user's other mailboxes.

When a mailbox receives a keep message, it stores the specified message and then removes that message from the queue of new messages. When a mailbox receives a delete message, it removes the specified message from the queue of new messages unless that message has previously been kept. In this case, that message is removed from the set of kept messages. Keep and delete messages are

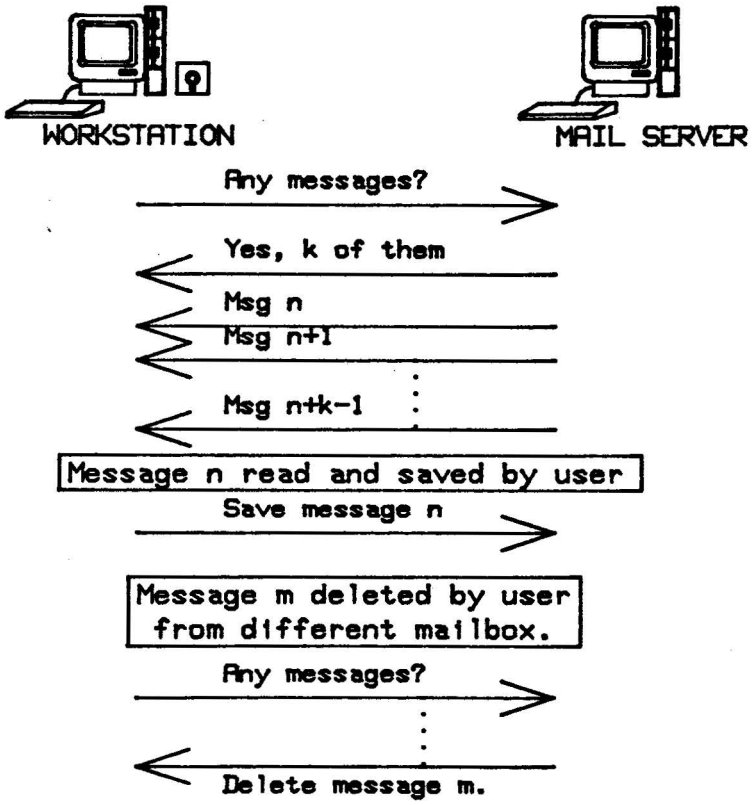


Figure 3-1:Mail System Protocol Example

transmitted by the server to the mailbox in the same way that new messages are. Therefore, when a user requests messages from the server, the messages he/she will receive may be of the types new, keep, or delete.

Figure 3-1 displays a sample situation. The mailbox in the workstation requests any new messages from the mail server. The mail server indicates that there are k of them and proceeds to transfer them to the mailbox. The owner of the mailbox then accesses the mailbox to read his/her mail. He/she decides that message n should be kept. A message to that effect is therefore sent to the mail server to be reflected to the user's other mailboxes. The disposition of the next k-1 messages is not displayed.

Some time later, the user accesses the mail system through a different mailbox and decides that message m should be deleted. A message to this effect is sent to the server where it is queued for

each of the user's other mailboxes. The *delete message m* message is placed in the first mailbox when that mailbox reads in its messages, as shown in the last line of the figure. Message m is then deleted from that mailbox.

3.2 The Mail System Architecture

The mail system architecture is composed of two fundamental functional units: the mailbox and the server. The mailbox represents the user end of the communications path. It provides the user with a facility for reading messages transmitted to him/her and for creating and queuing messages to be sent to other users. There is theoretically no limit to the number of mailboxes a given user can have. An arbitrary, artificial limit of 8 mailboxes, however, has been imposed in the prototype design.

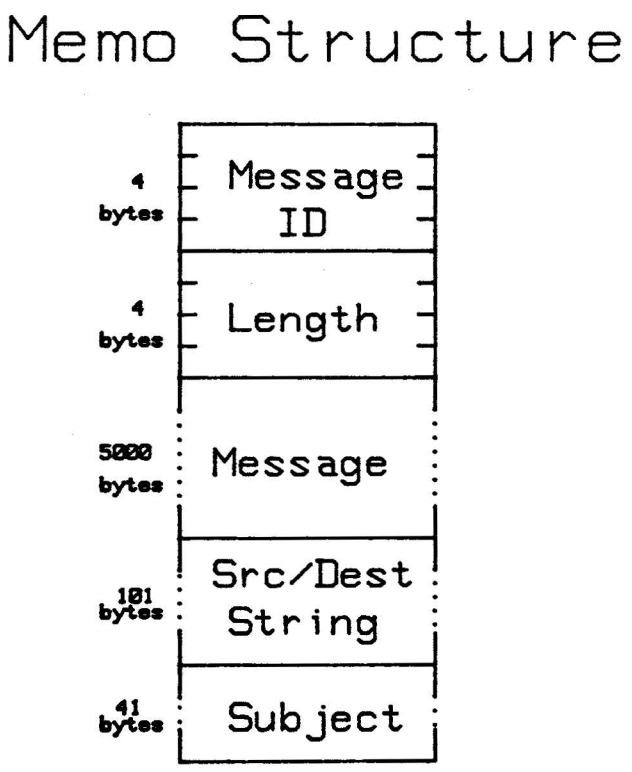


Figure 3-2:Memo Structure

The server represents the communications path itself. It is responsible for ensuring the transmission

of a message from one user to another. It is further responsible for ensuring that each of a given user's mailboxes is loosely consistent with each of the user's other mailboxes. The server is designed to handle incoming messages from all the users and outgoing messages to each of the users, while maintaining the aforementioned responsibilities.

In order to handle incoming and outgoing messages, the server is divided into two subunits. The first subunit is called the *server in-queue*. The server in-queue is responsible for accepting messages from all the users. The number of in-queues for the local mail network is small. The prototype was built with only one. One would probably be sufficient to support a moderately-sized network.

The second subunit is the *server out-queue*. The server out-queue is responsible for holding messages bound for a particular user. There is, therefore, one out-queue for each user. The out-queue is the subunit that is responsible for keeping the information necessary to maintain loose consistency.

The fundamental structure around which both the mailbox and the server are built is the *memo*. The memo structure is the basic unit of communications between a mailbox and the server, containing both the text of the message to be sent and administrative information specific to that message. The structure, shown in figure 3-2, is made up of five fields. The first two fields pertain to administrative details of the message. The message id field is the unique identifier by which a particular memo can by identified to the server or to a mailbox. The length field specifies the length of the body of the message.

The next three fields pertain to the actual text of the memo. The first is the message field. This field contains the textual body of the memo. The length of the message cannot exceed 5000 characters. The next field is the source/destination string. This string contains information concerning the source of the memo, the destination list of the memo, and the time that the memo was created. This string is used by the mail facility to determine the source of a memo and by the user interface to display with the message. The third field is the subject field. This field contains a string that specifies the subject of the memo. This string is used solely by the user interface when displaying a message.

Both the server and the mailbox contain a number of macrostructures designed to store memos and to allow efficient access to these stored memos. These macrostructures will be examined in detail in the next sections.

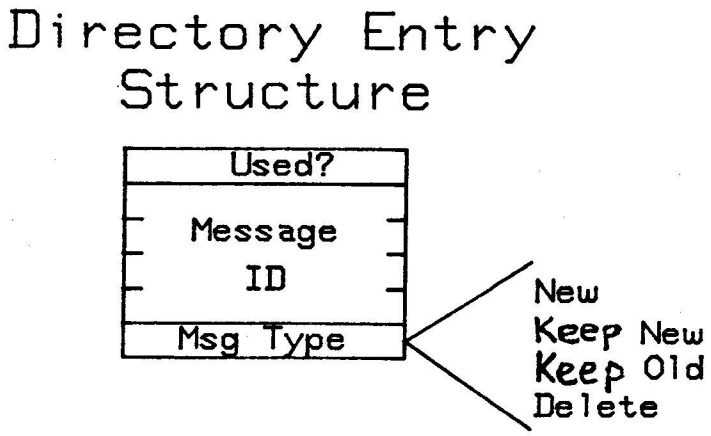


Figure 3-3:Directory Entry Structure

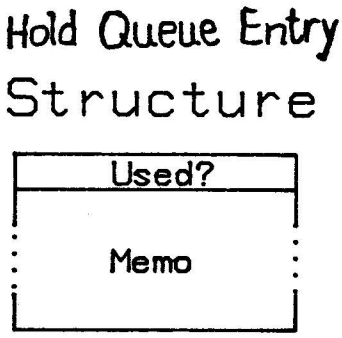


Figure 3-4:Hold Queue Entry Structure

3.2.1 The Mailbox

The mailbox is composed of three macrostructures: the memo queue, the hold queue, and the memo out-queue. The memo queue is the macrostructure charged with holding memos that have not yet been seen by the owner of the mailbox. This macrostructure is basically a file of memo structures.

In order to efficiently access the memos in the memo queue, a directory macrostructure is implemented. This macrostructure is basically a file of directory entry structures, a structure for each memo structure in the memo queue. The directory entry structure, as shown in figure 3-3, contains three fields. The first field indicates whether the directory entry is in use. If the entry is in use, the corresponding memo is valid. The second field specifies the unique identifier of the corresponding memo.

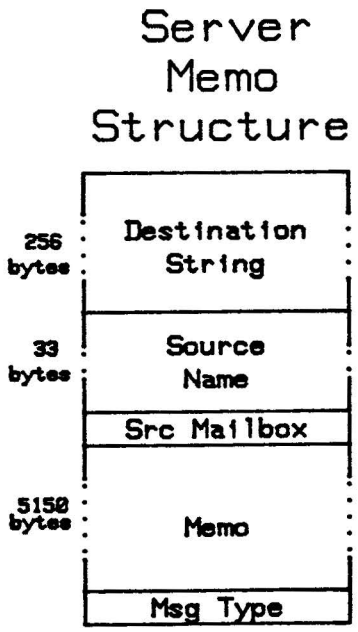


Figure 3-5:Server Memo Structure

The third field gives the message type of the corresponding memo. Memos come in four varieties: new, keep new, keep old, and delete. A new type memo indicates that the memo has (probably) not been read by the user at any of his/her mailboxes. A keep new type memo indicates that the user has already read the memo at another mailbox and that the memo should be kept by the mailbox. A keep old type memo indicates that the user has already read memo M at another mailbox and that M, which has been placed in this mailbox's memo queue at some earlier time, should be kept. The delete type memo indicates that the user has deleted memo M from another mailbox so M should also be deleted from this mailbox.

In the mailbox, the directory and the memo queue macrostructures each have the same number of entries and are of fixed size. In this way, the presence of a specific pointer is avoided. For instance, suppose one wants to find the subject of the first valid message. The first directory macrostructure entry might be unused so the second entry would be examined. If this entry is used, the second entry of the memo queue would be examined for the desired subject. The number of entries in the macrostructures depends on trade-offs in memo delivery time versus disc space. In the prototype, a value of ten was chosen.

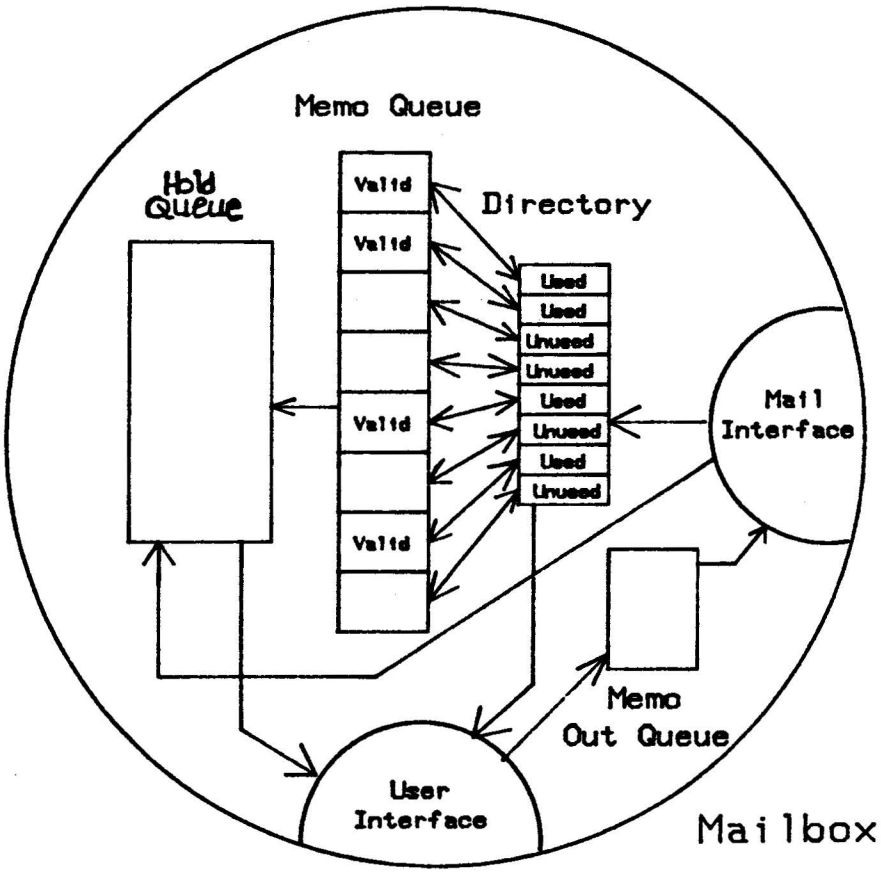


Figure 3-6: Mailbox Architecture

When a new type memo in the memo queue has been examined by the user, that memo must be kept

or deleted. If the memo is deleted, it is removed from the memo queue. If it is kept, it is transferred into the hold queue macrostructure. This macrostructure is a file composed of hold queue entry structures. The hold queue entry structure, as shown in figure 3-4, is simply made of two fields: the used field and the memo field. The used field specifies whether the memo to follow is valid. This field allows the deletion of a memo from within the hold queue without requiring the whole hold queue file to be rewritten.

The mail facility of the mailbox is completed by the memo out queue. The out queue is responsible for queuing messages to be sent to the server. For this reason, an entry in the out queue macrostructure is identical to that of the server in queue macrostructure. This entry has the form of the server memo structure.

This structure, called the server memo structure, consists of five fields, as shown in figure 3-5. The first field is the destination string. This string contains the names of all users to which this memo is to be sent. The second and third fields, source name and source mailbox, pertain to the source of the message. Source name is the user name of the creator of the memo and source mailbox is the number of the mailbox where the memo was created. The last two fields contain the actual memo and its message type.

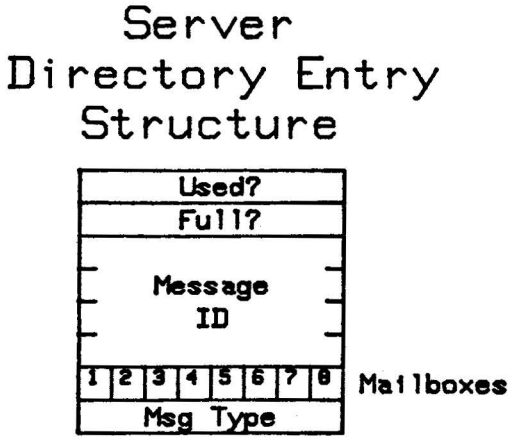


Figure 3-7: Server Directory Entry Structure

The complete mailbox appears in figure 3-6. The mail interface is the interface between the mailbox

and the server, and the user interface is that between the mailbox and the user.

3.2.2 The Server In-Queue

The server in-queue macrostructure is essentially a file whose entries are server memo structures (figure 3-5). The purpose of this macrostructure is to queue all messages from mailboxes until the server can process them. Once the server has examined a memo from the in-queue and determined its destination, that memo is deleted from the in-queue and placed in the out-queues of the destination users.

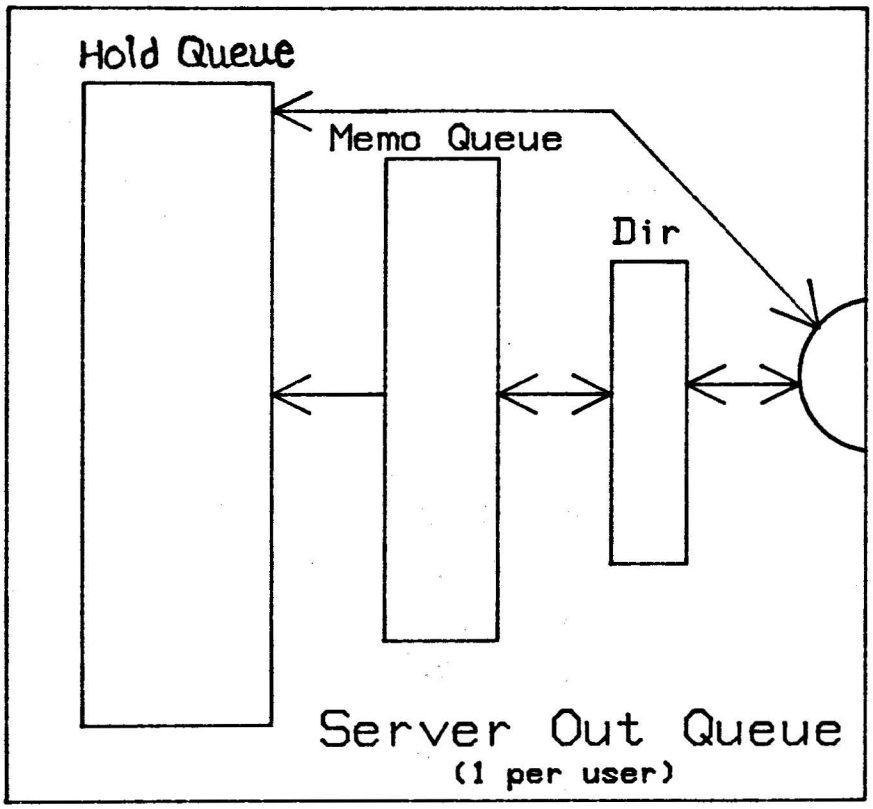


Figure 3-8:Server Out-Queue Architecture

3.2.3 The Server Out-Queue

The server out-queue is divided into two macrostructures: the memo queue and the hold queue. The server memo queue macrostructure is a file whose entries are memos (figure 3-2). The primary purpose of this macrostructure is to queue messages for transmission to a particular user. The memo queue also plays a key role in the maintenance of loose consistency. This role is facilitated through the server directory macrostructure, very similar to the mailbox's directory macrostrure (figure 3-3).

The server directory macrostructure is a file composed of server directory entry structures. The server directory entry structure, as shown in figure 3-7, is a superset of the memo directory entry structure (figure 3-3). In this manner, the server directory and the server memo queue perform exactly the same as their mailbox counterparts. It is the extra two fields that provide the additional functionality.

The two extra fields are Full and Mailboxes. The mailboxes field is used to keep track of which mailboxes have read the corresponding message. When a new message is sent to a user, the entries (in the mailboxes field) for each of the user's mailboxes are set to zero, all unused entries are set to one. When a mailbox receives a message, the corresponding entry in the mailboxes field is set to one. When all the entries of the mailboxes field are set to one, the full field is set to true. (The full field is used primarily for reclamation purposes.)

The server hold queue macrostructure functions identically to the mailbox's hold queue macrostructure (figure 3-4). The purpose of this macrostructure is to maintain a copy of each user's hold queue that is identical to each of the user's copies, as loose consistency also pertains to memos placed in the hold queue. The functional diagram of the server out queue, incorporating the hold queue and the memo queue, is shown in figure 3-8.

A completed picture of the mail system architecture appears in figure 3-9. The dashed circle represents the sphere of influence of the server. The circles represent mailboxes and the squares, server out queues. Note the presence of one server in-queue, one server out-queue per user, and a multiple number of mailboxes allowed per user.

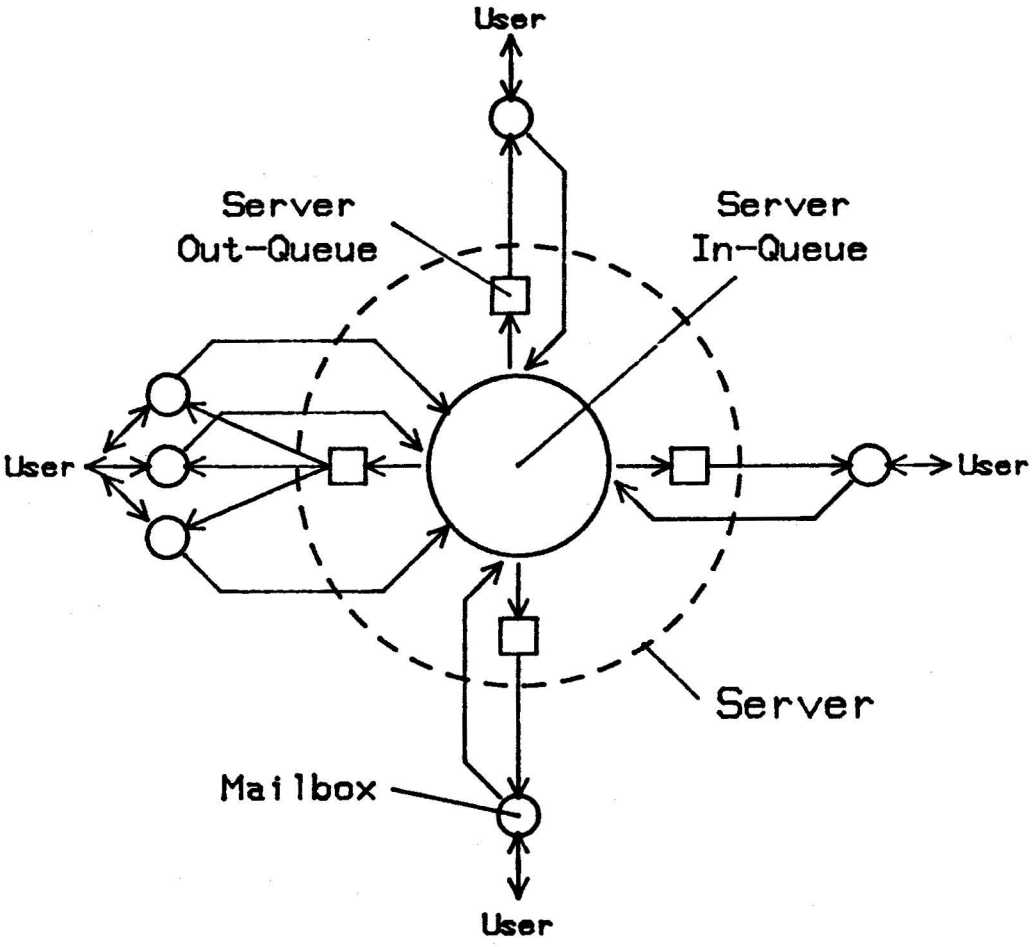


Figure 3-9:Mail System Architecture

Chapter Four

Mail System Operation

The mail system operation is specific to the type of network for which the prototype is targeted. Since the prototyping environment was based around a shared disk, much of the server functionality, as defined by the protocol, is placed in the mailbox. In this way, messages to and from the server are simulated by accesses to shared files. Any operation performed on the server's files constitutes a server operation in the protocol. This section, therefore, explains a specific implementation of the protocol. The governing concepts, however, may be generalized to other forms of networks.

The operation of the mail system can be separated into the mailbox and the server components in the same way that the mail system architecture is. The operation of each of these components is independent of the other in the sense that the failure of one of the components will not cause a failure of the other.

4.1 Mail System Operation: The Mailbox

The mailbox operation is characterized by a set of functions that execute as a result of being called by a timer interrupt service routine and a set of functions that execute as a result of user interaction. In general, the automatic functions (those that execute from the timer ISR) are responsible for moving memos between the mailbox and the server. The user-initiated functions deal with the memos once they have been placed in the mailbox.

4.1.1 Automatic Operation

Periodically a timer ISR is activated at the workstation which sets in motion the functions in the mailbox designed to process incoming memos. The first step in processing is to make sure a connection to the shared resource manager (SRM) has been established, as this is where the server keeps its files. If a connection has been established, the mailbox attempts to access the server out-queue belonging to its owner. If this is not possible, the mailbox assumes that the connection is bad.

Once the correct server out-queue is accessed, the mailbox begins to examine each entry in the directory looking for memos that have not already been placed in the mailbox. This is done by checking the mailboxes field to obtain the value in the subfield corresponding to the number of the mailbox. For example, mailbox 3 would check the subfield marked 3. If the value of the correct subfield is one, the memo has already been placed in the mailbox so the next directory entry will be searched.

When an entry is found with the correct subfield set to zero, the mailbox prepares to transfer that message. The type of transfer that happens depends on the message type of the memo. Recall that the four message types are: new, keep new, keep old, and delete.

If the message type is new, the memo has probably not been read by the user. It must, therefore, be placed in the memo queue until the user has read it and decided its disposition. In order to prepare to place the memo in the memo queue, the mailbox examines the memo queue directory, looking for an entry that is not used. If no such entry is found, the memo is left just as it is in the server out-queue.

If a free entry is found, the memo is transferred from the server out-queue into the free entry in the memo queue. The correct subfield of the server directory entry is then set to one, indicating that the memo has been transferred. Finally, the memo directory entry is updated to indicate the presence of the transferred memo. This entry is updated last in case the connection is broken after the transfer but before the server directory entry is updated.

If the message type is keep new, the memo has been read and kept at another mailbox so it must be kept in the hold queue of this mailbox also. In order to minimize the number of files that are accessed during the ISR's execution, this memo is kept in the memo queue for further processing. The transfer from the server out-queue is handled exactly the same as if the memo were of the new message type.

If the message type is keep old, a memo that has already been placed in the mailbox as new has been kept by the user at another mailbox. All that this message type requires is that the existing memo (of message type new) with the same message identifier as the keep old type memo be modified in such a way that it will be stored in the hold queue without being shown to the user as new. This modification can be accomplished by modifying the message type field of the memo directory entry corresponding to the original memo, from new to keep new.

If the message type is delete, a memo that has already been placed in the mailbox has been deleted by the user from another mailbox. The delete message must be handled differently from the other messages. The reason for this is that the memo could have been deleted from the hold queue instead of from the memo queue of the mailbox that originated that delete request. The mailbox must, therefore, be able to handle three situations: the memo to be deleted appears as new in the memo queue, it appears as keep new in the memo queue (in order to be in the hold queue at all, it must have been kept at some point), or it appears in the hold queue.

The first two situations are handled identically. Since the message to be deleted has not been processed, the corresponding directory entry can merely be marked as unused, effectively deleting the message. The third situation is just as easily handled. A given memo cannot appear both in the memo queue and in the hold queue of a single mailbox. For this reason, if the memo is not in the memo queue, it is probably (because the consistency is loose) in the hold queue. If it is in the hold queue, it is deleted by marking the hold queue entry as unused. If, for some reason, the memo does not appear in the hold queue, it must have already been deleted (loose consistency again) so the mailbox does not care.

The delete type memo, like the other types, is removed for that user from the server out-queue when it has been processed. For each type of message, this removal is accomplished by setting the correct subfield of the mailboxes field of the server directory entry for that memo: In the process of performing this removal, the mailbox also performs another function. After the field has been set, the mailbox checks to see if each of the subfields are set for that server directory entry. If they are all set, the mailbox then sets the full field. This field indicates that the corresponding memo has been sent to all the user's mailboxes, making reclaiming the server directory entry easier. The primary purpose for the full field is so that the server does not have to examine each of the subfields when it is looking for an empty directory slot. Since the mailboxes field is accessed anyway, it is easier to test the subfields from the mailbox side.

In order not to place too large a processing burden on the ISR, only a small number of memos will be processed per wakeup. In the prototype, this number was set at two, but was tunable. If there are no memos to be processed or if there is no server connection, the memo queue is examined for any needed processing. Any memos that are of the keep new message type are automatically stored in the hold queue. If any new memos are encountered, a message to that effect is placed on the screen for the user.

The final function performed by the ISR is transferring memos from the memo out-queue to the server in-queue. Under normal operations, there will be no such memos. There will be some, however, if the user created a number of memos while not connected to the server.

4.1.2 User-Initiated Operation

User-initiated operation results from explicit interaction between the user and one of his/her mailboxes. This interaction comes in the form of either creating a memo or reading a memo. The creation of a memo directly involves the creation of a server memo structure (figure 3-5) to be queued in the memo out-queue. Creation of a memo can occur in two different ways, either by entering the message at the console or by forwarding a previously received memo.

Reading a memo indirectly involves the creation of a server memo structure, since reading a memo implies that a memo concerning that memo's disposition must be sent to the server. Reading a memo, therefore, implies the creation of a keep[1] or a delete type memo.

These two forms of interaction have one essential difference. Creating a memo involves sending that memo to other users while reading a memo causes the creation of a memo to be send to other mailboxes of the same user. For this reason, the fields of the server memo structure take on varying importance depending on the form of interaction.

The destination string field of the server memo structure is only important for the creation form of interaction. This information is used to determine the users to whom the created memo will be sent. This information cannot safely be determined from the source / destination string subfield of the memo field due to the differences in length of the two strings (the destination string is of maximum string length; the source / destination string may be truncated, resulting in a message header that does not contain all the destination users).

The source name is only important to the form of interaction involving the reading of a memo (the update form, henceforth). The source of a created message is also important to the creation form of interaction, but this information is already stored in the source / destination string of the memo. The source name field of the server memo structure is used by the server to determine which user's

---

[1]The notion of a keep new or keep old memo originating from the mailbox has no meaning. The mailbox uses a memo of message type keep old to indicate a keep request to the server. The expression *keep memo* will, therefore, be used for such a memo in order to avoid confusion with memos that are specified as keep old by the server.

mailboxes to update. This information may (but need not necessarily) appear in the source / destination string. The source mailbox, also a field only used in the update form of interaction, further specifies the source of the update.

The memo field is important almost solely to the creation form of interaction. This field contains the memo that will be sent to each of the users in the destination string. The update form of interaction requires only the message id subfield. This subfield is used to indicate which memo has been updated.

The message type field is new for a new or forwarding memo, keep for a keep update, and delete for a delete update.

The creation form of interaction is very straight-forward in operation. The text of the message is obtained, either from the keyboard or from another message and, along with the other information obtained from the keyboard (subject, destination, etc.), entered into the correct fields of the server memo structure. This structure is then placed in the memo out-queue for later transmission. The operations of the keep and delete functions are more complex.

The delete function is more complex because it can originate from two different places: the memo queue and the hold queue. A memo can be explicitly deleted from the memo queue only as a result of it being displayed to the user as new. When such a situation occurs, the actual deletion is performed by setting the used field of the corresponding memo directory entry (figure 3-3) to false. A similar operation is performed when the deletion is requested for a memo in the hold queue. Here, the corresponding used field in the hold queue entry (figure 3-4) is set to false. In either case, the update memo is created and queued in the memo out-queue.

The keep function can only be performed on a memo in the memo queue. When the keep function is requested, the memo is appended to the hold queue and deleted from the memo queue by setting the correct used field to false. The update message is then created and queued.

When the user is ready to exit the mail system environment, he/she executes a quit command. Before the system is exited, however, the mailbox attempts to flush the memo out-queue to the server. In order to perform the flush, the mailbox again attempts to establish a connection with the server. If the connection is made, the memo out-queue is transferred to the server in-queue, then purged. If no connection is made, nothing happens to the memo out-queue until the automatic

process discovers that a connection has been reestablished.

4.2 Mail System Operation: The Server

The primary responsibility of the mail server is to take the incoming memos from the in-queue and direct them to their final destinations. The server also has the secondary responsibility of checking the validity of each request and reflecting any errors to the creator of the errant memo. Once the server starts processing memos from the in-queue, it continues to process them until they have all been processed. For the period of time that the memos in the in-queue are being processed, all mailboxes are barred from access to it. The mailboxes are, therefore, required to wait before they can transmit any memos. To ensure that the time that any mailbox is required to wait is small, the in-queue is processed often.

Memos that are processed by the server come in two types: new memos and update memos (those with a message type of keep or delete).

4.2.1 Processing New Memos

When a new type memo is found in the server in-queue, the server parses the destination list to obtain the list of users to which the memo is to be sent. The server then checks the existence of each of the destination users. For each non-existent user, the server creates an error memo of the form *User <user name> not found.* These memos are then treated as if they are new memos and are sent to the user who originated the memo. For each of the valid destination users, a copy of the message is made and the server attempts to place that copy in the correct server out-queue.

In order to place the memo in a user's out-queue, the server must find an unused entry in that user's server directory. An unused entry is found by examining each entry until one is found that either has the used field set to false, or has the full field set to true and has a message type that is not new. The reason the message type cannot be new is that the server does not erase a memo before its disposition has been decided. If a new type memo with the full field set to true is reclaimed, a subsequent keep of this memo cannot be reflected in the server's copy of the user's hold queue.

Once an unused entry has been found, the new memo is then written into this unused entry in the out-queue and the directory entry updated. The server then correctly sets the mailboxes field of the directory entry. This involves setting the subfields for each of the user's mailboxes to zero. Since the

mailboxes are required to be numbered consecutively, the server only needs to know the number of mailboxes the destination user has. This information is obtained from the same source that validated the existence of the user.

Currently, a problem exists if an unused directory entry cannot be found. If such a situation arises, the creator of the memo will receive a memo indication that the message was undeliverable. In a mail system where each of a user's mailboxes are activated somewhat regularly, this situation is not likely to happen. A short-term solution to this problem, if it arises, is to increase the size of the out-queue. A long-term solution would be to add a requeue buffer to the server's in-queue. The server would then have to ensure that requeued memos do not seriously slow down the process of directing memos.

4.2.2 Processing Update Memos

Due to loose consistency, the server cannot be guaranteed that the memo for which an update (keep or delete) request is made has not already been updated. For this reason, update situations will be divided into two catagories: normal and race. Normal situations are those where the update request operates on a new memo.

Figure 4-1 shows an example of a normal situation where a keep is requested. In this example, the user has four mailboxes. Over some period of time, the new memo is placed in mailbox 2 then mailbox 1. At some later point, the user accesses the mail system through mailbox 1 and decides to keep the memo. The server must now modify the state of the out-queue to reflect the fact that mailbox 2 should keep the memo already placed within it and the fact that the remaining two mailboxes should keep the memo as soon as it is placed within them.

In order to do this, the server requires a free slot in the out-queue directory. This slot is obtained in the same way as it would for a new memo. The obtained slot is then modified to make a memo of the keep old message type. This modification entails setting used to true, message id to the same message id as the original memo, and message type to keep old. The mailboxes field is then modified by setting each of the subfields corresponding to valid mailboxes to the inverse (zeroes become ones etc.) of that setting in the memo to be updated. The subfield of the updating mailbox is set to one. The full field is then set if necessary. The server then changes the message type of the memo to be updated to keep new. The final frame of figure 4-1 shows the update.

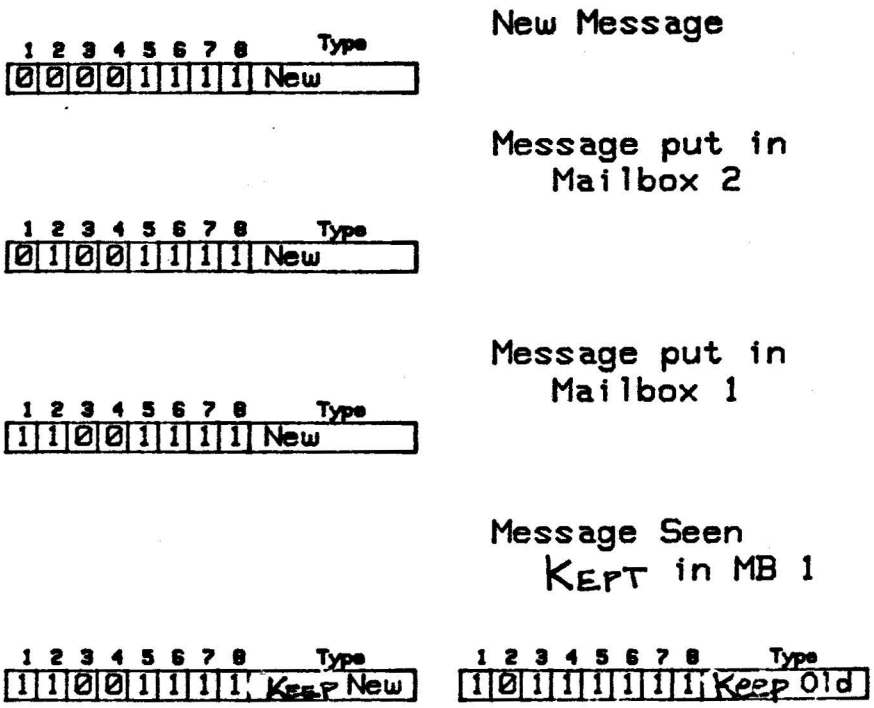


Figure 4-1:Normal Keep Update

Figure 4-2 shows an example of the normal situation where a delete is requested. As before, the user in this example has four mailboxes. Again, the memo is placed in mailbox 2, then mailbox 1. At some later point, the user accesses the mail system through mailbox 1 and decides to delete the memo. The server must now modify the state of the out-queue to reflect the fact that mailbox 2 should delete the memo from his/her in-queue.

To reflect this fact to mailbox 2, the server changes the message type of the memo to delete and modifies the mailboxes field of the memo's directory entry to indicate that only mailbox 2 should read the delete message (the memo is *addressed* to mailbox 2 only). In general, the updating of the mailboxes field is done by inverting each subfield that corresponds to an actual mailbox. The subfield corresponding to the updating mailbox is then set to one. In the example, subfields 1 and 2 become zero and subfields 3 and 4 become one. Subfield 1, the source mailbox, is then explicitly set, resulting in what appears in the final frame of figure 4-2.

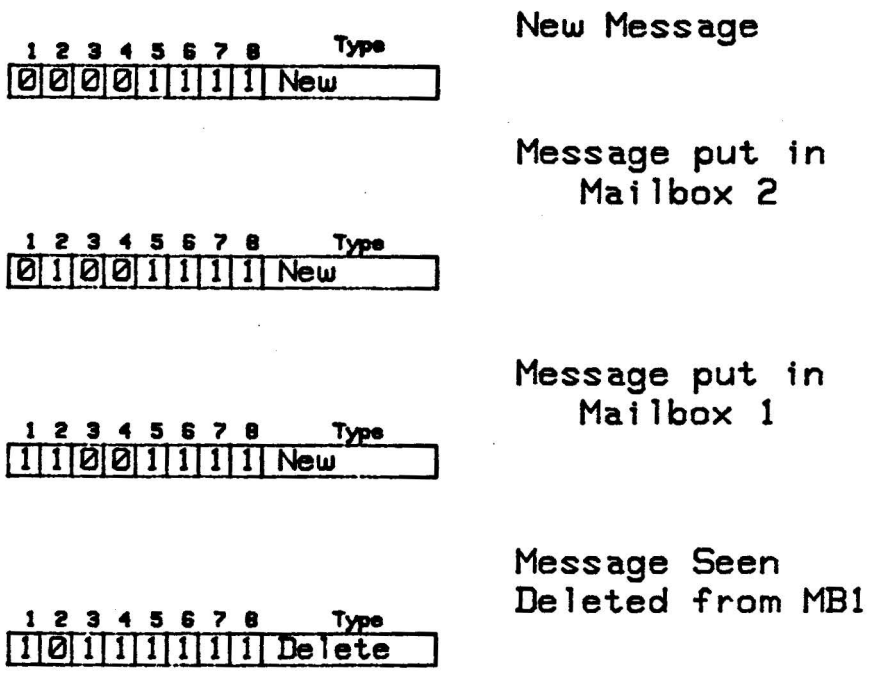


Figure 4-2:Normal Delete Update

Race situations occur when a mailbox attempts to update a memo that has already been updated. For example, suppose a user has two activated mailboxes into both of which a memo is placed. The user now accesses the mail system through one of these mailboxes and keeps the message. He/she then immediately accesses the mail system through the other mailbox and, for whatever reason, deletes the same memo before the keep can be reflected to the deleting mailbox. Because it is much easier for the server to delete after keeping than to keep after deleting, the server allows deleting to have precedence over keeping. That is, if a memo is both kept and deleted, the deletion will prevail.

There are basically four different race situations. These are 1) keep after keep, 2) delete after delete, 3) delete after keep, and 4) keep after delete. The first two are the trivial cases and do not merit discussion as their operations will proceed as if only one update has been made. The second two are more difficult.

Figures 4-3 and 4-4 show these two race situations. Figure 4-3 displays the delete after keep race situation. Its first three frames show the normal operation of the keep type update. The memo is

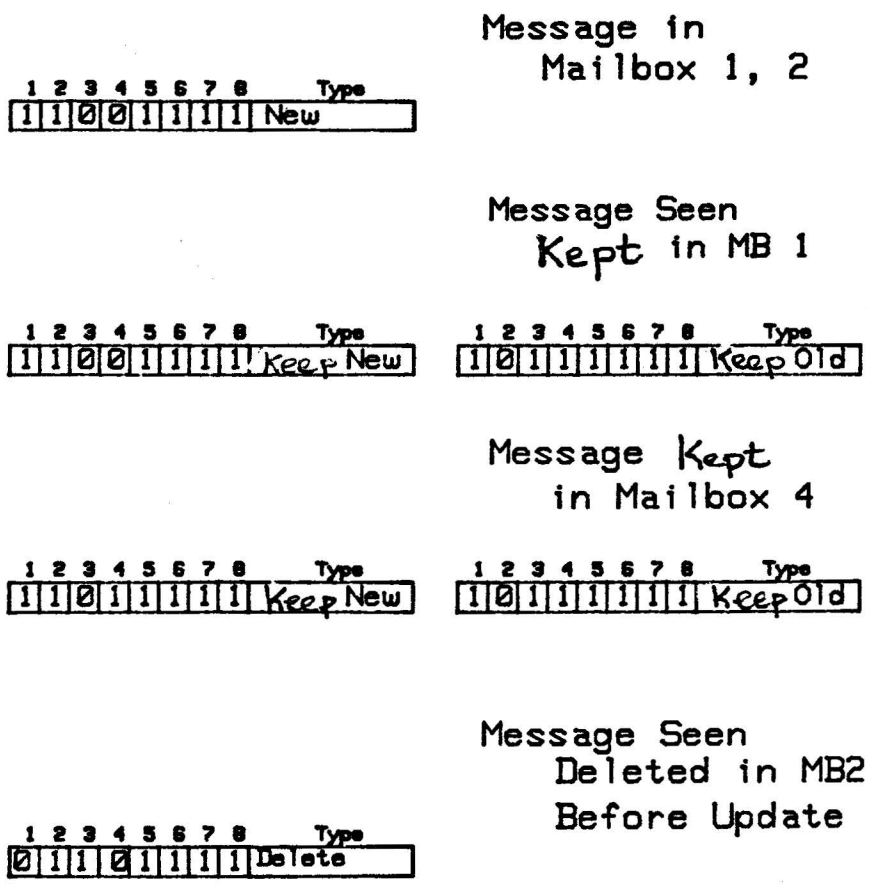


Figure 4-3: Delete After Keep Race

placed as new in mailboxes 1 and 2, and, after the user decides to keep the memo from mailbox 1, as keep new in mailbox 4. The user then decides, while accessing the mail system through mailbox 2, that the memo should be deleted. The server is then faced with the responsibility of removing the two keep memos and creating a delete memo to be sent to mailboxes 1 and 4. This created memo is shown in the final frame of figure 4-3.

Figure 4-4 displays the keep after delete race situation. The first two frames show the normal delete operation. The memo is placed in mailboxes 1 and 2, and then deleted through mailbox 2. The user then decides to keep the memo through mailbox 1. Since the memo has already been deleted, the

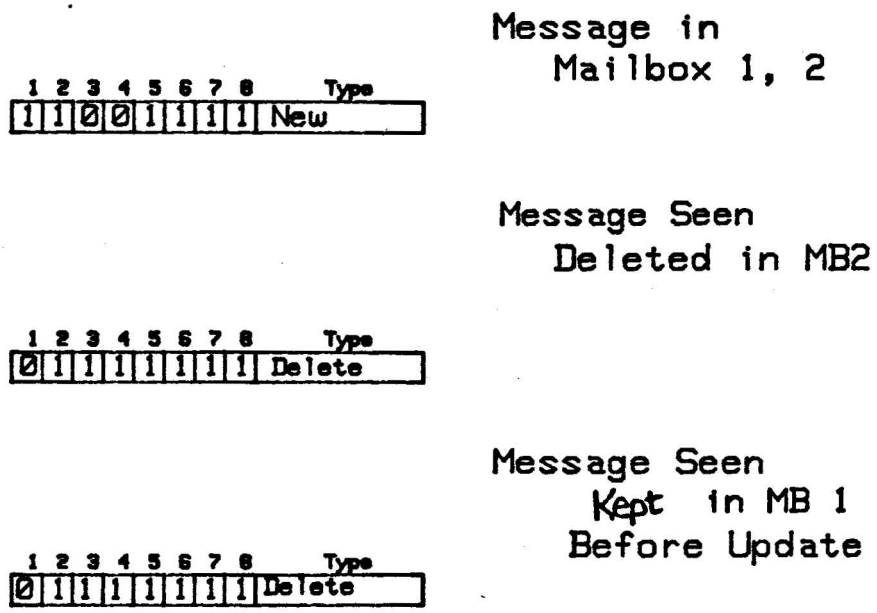


Figure 4-4:Keep After Delete Race

server disallows this request. The delete memo stands, as shown in the final frame of figure 4-4.

In order to work in the above manner, the server uses specific criteria to tell when a given update is allowed and how a given update is to be reflected in the out-queue directory entry. The operation of the keep update is designed to reduce to the normal (nonrace) situation. When a keep is requested, the server checks to see if the memo to be kept has a message type of new. If this is the case, the update is allowed to continue as described above. If the message type is not new or the entry not found, the update is disallowed. Note that this strategy also takes care of the keep after keep race situation.

What is gained in terms of simplicity of operation for the keep update is paid for with the complexity of the delete update. The delete update is further complicated by the fact that a delete request can originate from the user's hold queue. For this reason, when the server is looking for a memo whose message id is the same as the delete memo, it might find no memos, one memo, or two memos.

No memos would be found if the memo had already been deleted or if it had already been kept in the

hold queue. Because the outcome of the memo is not known, the server must assume that the message has already been kept in the user's hold queue. It, therefore, creates a delete message that is addressed to all mailboxes except the one making the delete request. If a mailbox receiving this memo cannot find the specified memo in its in-queue or in its copy of the hold queue, it simply ignores the delete request.

One memo would be found if either the memo has message type new, the memo has message type delete, or the memo has been kept and one of the keep memos has been reclaimed. The case of the new type memo is simply the case described above as the normal operation. In the case of the delete type memo, the update is ignored, taking care of the second trivial case. The case for a kept message is more complex.

If the message type is keep old, the server knows that the accompanying keep new type memo has been reclaimed and, by extension, that all the user's mailboxes must have received the memo. The server must, therefore, convert the keep old memo into a delete memo that is addressed to all the user's mailboxes except for the originator of the delete request. In other words, the subfields of the mailboxes field corresponding to actual mailboxes will be set to zero. The subfield corresponding to the originator of the delete request will be set to one.

If the message type is keep new, the server knows that each mailbox for which the keep new memo is destined (addressed) has not received a copy of the memo in any form. The server then converts the keep new memo into a delete memo that is addressed to all mailboxes except the mailbox that originated the delete request and those mailboxes that never received the keep new memo. In other words, each subfield of the mailboxes field is inverted. The subfield corresponding to the originator of the request is then set to one.

If the server finds two memos with the same message id as the delete request, they had better be of message types keep new and keep old. One of these memos is removed from the out-queue and the other is converted into a delete memo. This memo is then addressed to any mailbox that has had the original new memo or the later keep new memo placed within it. This can be determined by examining the keep new memo in the same fashion as described above. The subfields that are set to one in the keep new memo's mailboxes field correspond to those mailboxes that have had the memo placed in them in some form. These mailboxes are, therefore, the ones to which the delete memo should be addressed. In order to do this, the server inverts the values of the subfields of the keep new

memo and uses them for the delete memo. The subfield corresponding to the mailbox that originated the delete request is then set to one. In frame 3 of figure 4-3, subfields 1, 2, and 4 become zero and subfield 3 becomes one. After, subfield 2 is set to one, as mailbox 2 originated the delete request.

4.3 Remote Operation

The mail system has a built-in facility for communicating with other similar mail systems. This inter-mailnet communications facility uses an existing mainframe mail system, HPMAIL, to transport the memo from one mail system to another. The facility brings up the concept of message formats, to be signified by personal computer message (PCM) and mainframe message (MFM).

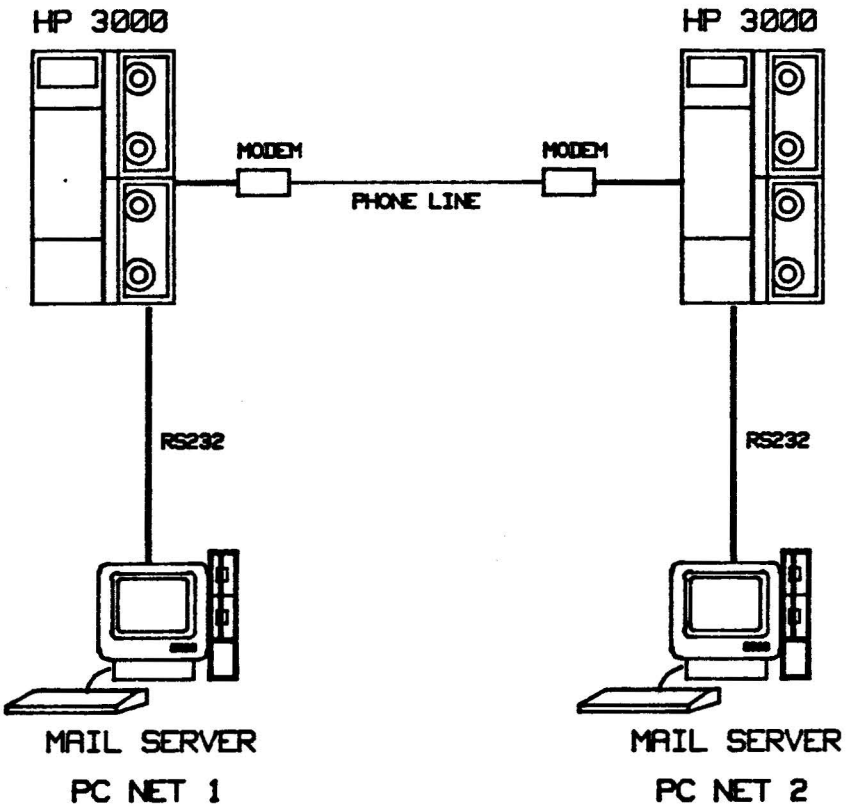


Figure 4-5:Remote Operation

The goal of the server's remote operation is to take a PCM and convert it into a MFM. This MFM is shipped off via the mainframe mail system to a place where the destination server can retrieve it and convert it back into a PCM. In the mail system, this process is accomplished by establishing an RS232 link between the mail server and the HP 3000, which is the host mainframe. The server then logs into HPMAIL like any user would. The PCM is converted to the MFM by HPMAIL since the server simulates an HPMAIL user typing in a message from the keyboard. The MFM is then routed over phone lines to the destination host, where the message is retrieved by simulating a user reading the message. The overall layout is shown in figure 4-5.

In order for the destination server to receive the message correctly, the MFM must have a specific format. The first line of the MFM must be the source of the message as specified by the creator's mail system user name. The second line must be the destination within the receiving mail system, again a local mail system user name. The rest of the message contains the text of the original memo and can be read as such if sent to a normal HPMAIL user. In other words, the interface can also be used to send messages to regular HPMAIL users and HPMAIL users can return messages to the mail system by following the format expected by the server (specifying the first two lines correctly).

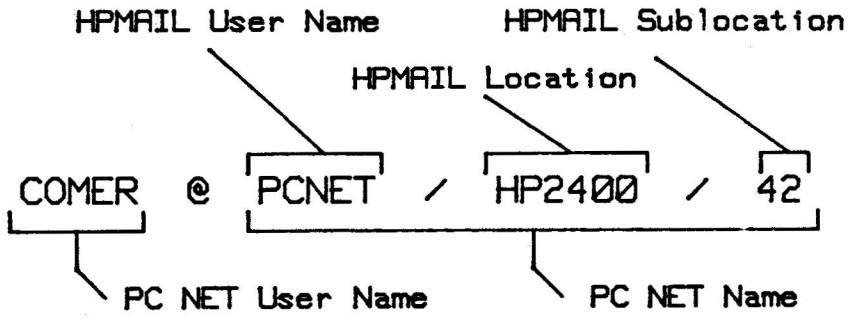


Figure 4-6:Remote Name Breakdown

A memo destined for another network begins as any other memo within the mail system. The only difference is that when the destination is specified, it must be of the form *<user name> @ <mail network name>*. The mail network name will be used by HPMAIL in routing the memo to the correct mail network. Figure 4-6 shows the breakdown of how HPMAIL uses this name.

Once the memo has been created and transmitted to the server, the server places it in the remote memo queue for transmission to HPMAIL. Note that if the remote mail network specified is the same as the local one, the destination is modified and the memo becomes local, and is then treated as such. After some period of time, the server will begin to process all the messages in the remote memo queue. In order to do this, the server first attempts to log into HPMAIL. If this fails, the operation will be aborted and retried later.

If the server succeeds in logging into HPMAIL, it attempts to send the first memo. This involves answering questions from the HPMAIL user interface. The server specifies the remote mail network name, a valid HPMAIL address itself, as destination. If the destination name is invalid, i.e. the location specification is incorrect, the server sends a *cannot send* error message to the originator of the memo.

In any case, the server continues with the transmission of all the memos in the remote memo queue until all have been sent. It then checks for any messages that might have come in from remote networks. If there are any, they are put into the server in-queue for later processing. The server obtains the return path by taking the source user name and concatenating it with the source mail network's name.

Chapter Five

The User Interface

5.1 The Prototype System

The user interface of the mail system is divided among three separate programs: makeuser, mail, and mailserver. Makeuser is the program whose purpose it is to create new user names for the mail system. Makeuser first asks the new user for his/her desired user name. A user name is the name by which the user is identified in all correspondence. Once the name has been entered by the new user, it is checked against all known user names to ensure that it is unique. If the name is not unique, the user must try again with another name.

When the new user has entered an acceptable name, the program then prompts for the user's password. This password will be encrypted using a one-way function. The encrypted version of the password will then be used to protect the server out-queue and the server's copy of the user hold queue. The reason the password is encrypted is to ensure that the user can have access to his/her server out-queue only through the server.

The final piece of information required by the program is the number of mailboxes that the user has. The number of mailboxes along with the user name and the password are combined and kept in a file called the user file. This user file, therefore, contains information on all the users of the local mail system. When a prospective user name (for a new user) is to be validated, the user file is searched for a match. If no match is found, the name must be unique.

In addition to the creation capability, the makeuser program also has the ability to update information about a given user. Updating can involve changing the user's password, changing the number of mailboxes the user has, or deleting the user. At the present, any change in the user's status results in his/her corresponding server out-queue being reinitialized. This reinitialization can cause inconsistencies and loss of memos, so it should only be done shortly after all the user's mailboxes have accessed the out-queue.

Mail provides the actual user interface to the mail system. When the program is first executed, it asks

the user for the physical disk drive on which the mailbox can be found. The program then checks this disk to see if it can determine the startup information about the user. This will be discussed later. If it cannot, as would be the case the first time a particular mailbox is used to access the mail system, it asks the user to enter the information.

The first piece of information requested is the user name. This name will be used by the ISR mail routine, in order to access the proper server out-queue, and by the mail program when specifying the source of a created message or of an update request. The mail program then requests the user's password to further validate the user. The user is required to enter his/her password twice to ensure correct entry. If the two do not agree, the user must start over. The password is then encrypted to derive the form used by the server. The password is not checked here since nothing is guaranteed to be present to check it against. Instead, the password will be implicitly checked when the server out-queue for the user is accessed.

The mailbox number is required next. This information is used by the program to specify the source of an update and to determine if a memo in the server out-queue has been placed in this mailbox.

The final data required is the frequency of the timer wakeup, in minutes. The ISR mail routine will wake up on the interval specified and access the server out-queue.

Once all this information has been entered, the program creates a configuration file that resides in the mailbox. Since the operating system does not allow password protection for local disks, the mail system password is stored doubly-encrypted in the configuration file. Henceforth, whenever the user accesses the mail system through this mailbox, the mail program will only ask for the disk the mailbox is on and the user's password. The password is validated this time by doubly encrypting it and checking this against the one in the configuration file. The singly encrypted version of the password is kept by the program to facilitate access to the proper server out-queue.

The user now has access to the mail system. He/she can now perform the expected functions such as creating memos and reading new memos. Memos that are read must be either kept or deleted but they can be forwarded before their disposition is decided. The user also has functions which access the hold queue. He/she can move from memo to memo within the hold queue, examining the headers of the memos. When a desired memo is found, it can be examined (and/or forwarded) or deleted. Since deleting a memo leaves an unusable spot in the hold queue, the user is also provided with the ability to crunch the hold queue to remove these dead spaces.

The user has a limited ability to list the names of all the users in the mail system. This facility works by attempting to make a connection to the user file, which is stored in the same place as the server files. If this connection is made, the names are listed to the user. The password and the number of mailboxes for each user do not, of course, appear in the listing. If no connection can be made, the user is informed of this condition.

Finally, the user has the ability to modify the wake up parameter specified when the mail program was loaded. The user also has the ability to enable or disable an alarm bell that sounds when new mail is present.

The server program completes the user interface of the mail system. The server console (i.e. the console of the workstation executing the server program) is designed to allow supervisory access to the mail system. It is assumed that the server workstation is physically secure.

The primary access that the server console allows is access to the directory of the server out-queue and the server's copy of the hold queue for a particular user. This access is provided to allow a supervisor to monitor loose consistency. Access to the hold queues is provided to ensure that loose consistency is being maintained even within the hold queue.

The server console also provides the ability to reinitialize a user's out-queue and optionally delete the server's copy of that user's hold queue. This ability is provided to allow the supervisor to clean up the server files in the event of a catastrophic server error.

In addition to supervision of loose consistency, the server console provides facilities for overall system supervision. It allows the supervisor to monitor all error memos created by the server. By examining these, he/she can determine possible problem areas in the mail system. He/she can even specify that all output be logged to a print file for later examination.

Finally, the server console provides some commands to ensure its own smooth operation. It provides a crunch command that will remove the dead space from the server's copy of every hold queue in the mail system. This command is used to reclaim wasted disk space. Due to the potentially long execution time, this command should not be executed often.

The console also provides a command to flush the remote memo buffer to HPMAIL. This command would be used to prevent a large backup of remote memos from accumulating in the remote memo

queue, as might occur if HPMAIL were going down for a long period of time.

The rest of the server program is dedicated to the direction of memos between users. The writing of a memo into a server out-queue of a particular user requires the knowledge of that user's password. The updating of a user's memo requires the knowledge of the number of mailboxes owned by that user. The validation of a user in a destination list requires the knowledge of all the users in the mail system. All of these requirements are fulfilled by providing the server with access to the user file.

5.2 Improvements

There are several improvements to the user interface that would make the mail system more powerful and more convenient to use. Some of these improvements would concern only fine points of the user interface, such as allowing the use of distribution lists in the destination list. This type of improvement will not be discussed in this paper. The type that will be discussed concerns those that require changes in the fundamental mail system architecture.

An example of such an improvement is the creation of a facility whereby the user can examine the headers of the new memos before reading them. The implications of this are that the user does not have to decide the disposition of each memo when he/she accesses a mailbox containing that memo. These memos must, therefore, remain in the memo queue until the user decides to read them. This situation results in two new problems.

The first problem is the limited size of the memo queue. If there are many memos that the user does not read, the memo queue will become tied up, making the transport of memos from the server inefficient, at first, and impossible, later. A solution to this problem is to warn the user of impending inefficiency when the memo queue becomes full to a certain point with deferred memos. This problem requires a change in the programs but does not require a change in the architecture.

The second problem requires the change in the architecture. The user certainly does not want a deferred memo to keep generating a *new mail* message on the workstation console. In order to prohibit this, a *deferred* state must be created. Now, when a message's header is read, a memo must be sent to the server indicating that the memo has been deferred. The server then has the responsibility of reflecting this fact to each of the user's other mailboxes. A deferred memo would be treated as a new memo by everything except the facility that determines if there is any new mail. This

facility was not implemented since it did not appear in the mail system on which the prototype system was built.

A second improvement involving a change to the mail system architecture is to allow the user to dynamically reconfigure the number of mailboxes he/she has. This would allow the user to remotely delete a mailbox that is no longer needed. This would also allow the user to create a new mailbox and optionally load a copy of his/her existing hold queue.

This additional facility requires a number of different changes in the mail system architecture. The server must be responsible for telling the mailbox what its number is. In addition, the server must be able to send the mailbox a memo indicating that the mailbox should renumber itself. This is necessary since the user's mailboxes must be numbered consecutively. If renumbering were not possible, a deleted mailbox might leave a hole in the order of mailboxes. This ability to renumber requires the creation of a new memo type.

There must also be two new types of memos requesting the server to create and to delete a mailbox. In both cases, the user's password must be included in the memo for validation purposes. If the request is to delete, the user must obtain the mailbox number from the mailbox to be deleted. This number must appear in the delete mailbox request. Once the delete request is validated, the server renumbers (at most) one mailbox to fill in the order. Each directory entry in that user's server out-queue is modified to reflect the renumbering.

If the request is to create, the server will return the mailbox's number or an indication that no more space remains for mailboxes. If the mailbox is created, the server will then modify each entry whose message type is new or deferred to indicate that the new mailbox should read in these memos. The keep memos are not modified since the user would have the ability to read in a copy of the hold queue. The delete memos are not modified since the mailbox contains nothing to be deleted.

This function was not included in the prototype design because of the complexity it would add to the server program. Since the goal of this project is to investigate loose consistency, attempts to create this facility were dropped.

Chapter Six

Extensions

6.1 The Protocol

The protocol described can be used on any type of network to implement a network obeying loose consistency. In particular, the protocol can be used to maintain loose consistency across heterogeneous machines, as long a common message format exists among them. For example, a user could have an account on a VAX and a personal computer, both on some network, but have a single mail identity across both machines. The next step is to allow a user to have a single mail identity across the internetwork.

In order to facilitate this global naming strategy, it is necessary to develop a powerful, distributed mail server. The XEROX Grapevine system [BIRRELL82] is an example of a server that strives toward this goal. In addition, Grapevine attempts to maintain the mail facility even when one or more servers go down.

In order to do this, Grapevine introduces the concept of the *inbox*. An inbox is a place where a mail user can receive mail. Any given user can have as many inboxes (each on a different serving machine) as he/she wants. The internet addresses of these mailboxes are placed in a list and ordered by preference. Thus, if a user receives a message, it is stored in the first inbox on the list, if that server is available. If that server is not available, the next one is checked, etc.

In order to read his/her mail, the user invokes the local mail system program which determines the list of possible inboxes. Each of these inboxes, in turn, is polled for new mail. Any new mail is transferred to the local mail system and disposed of there. This *localization of control* is a key concept in Grapevine. No messages are sent to the user without his/her first asking for them.

The importance of localization of control is that it does not require the user to be tied to any one machine. The user has a single mail name across the entire network so he/she should be able to receive mail anywhere within that network. Someone sending a user a message does not necessarily know at which machine that user is currently accepting mail, but, with Grapevine, it does not matter.

The user is, therefore, free to move from machine to machine.

This concept is particularly important when it comes to personal computers. Because the mailbox is so portable, a single network address can have many different users in a space of several minutes. The server, by itself, has very little chance of determining where a given user is at a given instant, short of imposing some kind of artificial restraint such as requiring the user to be registered with the server while his/her mailbox is activated. Localization of control is, therefore, key to the design of the system.

Because both systems rely on localization of control, they are conceptually easy to merge. If we have Grapevine, however, what is the need for the prototype mail system? Obviously, a new message read from one machine will never be read as new on another. The prototype mail system, however, offers something in addition to the localization of control. It offers a (loosely) consistent mail environment independent of the state of the server or, more specifically, the state of the connection to the server.

For a mainframe computer, the chance of having a bad connection to the distributed server is very small. For a personal computer, however, it is much greater, particularly in light of portable personal computers. The personal computer may be connected quite infrequently but it should still have copies of stored messages and should be able to queue messages locally even if the connection is not present. Presence of personal computers on the network, therefore, requires the loose consistency protocol.

Modifying Grapevine to incorporate this protocol primarily involves the addition of the update message types. The server must assume the responsibility of orchestrating the updates by maintaining a mechanism similar to the mailboxes field of the directory structure. The total system would have be to modified to allow the mailbox to obtain the internet address of the server so that the disposition of a message can be returned to the server (as a member of the distributed server) that contains the information about the original message.

The operation of the personal computers would be essentially the same. A process would periodically awaken and poll for new mail. The operation of any mainframe in the system would be like the personal computer except that automatically polling for messages for a user and queueing them locally until that user can see them is probably not necessary.

In the absence of an internet-wide mail server, a local server can be used to coordinate mail between

its mail network and the rest of the network, whether it is organized as individual machines or as mail networks. The server would, in any case, act in an analogous manner to the server described in the prototype system, employing localization of control.

6.2 The Algorithm

The algorithm for maintaining loose consistency is essentially a means by which a number of copies of some entity (the memo in the case of the mail system) are kept consistent. The candidates for this entity, however, are quite constrained due to the limited nature of the algorithm. A more general solution is implemented by David Gifford in the Violet calendar system [GIFFORD79a].

Violet is a system that provides access to a distributed database. This database allows multiple copies of information to be stored throughout the network in order to improve reliability and performance. The database records are the entities to be kept consistent. Because of the multitude of different states that a typical database record can enter, the version number abstraction is used to simplify the state change diagram. Every time the database entry is changed, the version number of that entry is incremented. The state change diagram is, therefore, just a series of increasing version numbers.

Consistency among copies of a particular entry is maintained by a weighted voting scheme [GIFFORD79b]. It this scheme, each copy of an entry is assigned a certain number of votes. The entry, itself, contains two numbers: the number of votes required for a read access and the number required for a write access. These two numbers are known as the read and write quorum, and are constrained such that their sum is greater than the total number of votes assigned to the copies of the entry. A quorum is said to be reached when the sum of the votes of all the copies that have been examined in an operation exceeds the quorum for that operation.

When a read operation is requested, copies are examined until the read quorum has been reached. Once the quorum has been reached, the copy with the largest version number is assumed to be the current version of the entry. When a write operation is requested, copies are again examined until a read quorum has been reached, in order to obtain the current version number. The copies are then examined until a write quorum has been reached. If no write quorum is ever reached, all copies are updated to the current version, but no write operation is allowed. If the write quorum is reached, the copies are updated to the new version and the current version number is incremented.

The algorithm guarantees that a read quorum will always contain a current version. Let w be the write quorum and r be the read quorum. Every write operation updates at least w votes to the current version. Every read examines at least r votes. In order for this set of r votes not to account for any current versions, the two sets of votes would have to be disjoint. This is impossible since it would require r+w to be less than or equal to the total number of votes, which violates a previous constraint on r+w.

Gifford further refines the algorithm with the definition of a *weak copy*. A weak copy is a one that has no votes. The reason for its existence is to allow local, temporary copies to be made. These local copies may improve the performance of the lookup operation. Weak copies have the advantage of not requiring the sophisticated underlying memory system required by the non-weak copies.

In some ways, the loose consistency algorithm can be thought of as a special case of this weighted voting algorithm. In the context of a voting system, the copy of a memo that is resident on the server can be said to have all the votes. The local mailboxes can, therefore, be said to have only weak copies. Because the state change diagram of a memo is so simple and constrained, the use of the version number abstraction is not necessary.

The mail server as the ultimate authority on the state of any memo is key to the loose consistency algorithm. Because there is no doubt as to the official state of any memo, there is no contention, and therefore, no need for the specific voting algorithms. In addition, the simplicity and structure of the state change diagram of a memo make it possible for the mail system to allow access to the local copies (in the mailboxes) without access to the mail server. Damage will not result because the final state is independent of the order of the updates. For example, a keep followed by a delete results in a delete; a delete followed by a keep also results in a delete.

The loose consistency algorithm is applicable in situations where there is a central authority on the state of each entity, and the state change diagram is such that the resultant state of any set of updates is independent of the order of the updates. Note that with such a state change diagram, it is easy to add other states in the same way that the *deferred* state was added in section 5. As long as the new state adds no cycles to the diagram, the order-invariant property will remain.

Glossary

Activated A mailbox is said to be activated when it has been placed in a machine on the network and is ready to accept or transmit mail.

Addressed A memo is addressed to a particular mailbox if the subfield of the mailboxes field of that memo that has the same number as the mailbox, is set to zero. A memo that is addressed to a mailbox will eventually be placed in that mailbox.

Crunching The process by which unusable space is consolidated and reclaimed for reuse. In this paper, crunching is applied to an hold queue in order to reclaim the dead space left when a memo is deleted from that hold queue.

Hold Queue The hold queue is the place where kept messages are placed. There is one hold queue in each mailbox and logically one kept by the server for each user. The consistency of these hold queues is maintained by the loose consistency protocol.

Inbox An inbox is a term used in the Grapevine mail system to indicate a serving machine which the user will poll for new messages.

ISR ISR stands for interrupt service routine. An ISR is a routine that executes as a background process as a result of an external event. For instance, a routine that refreshes a clock display on the screen seemingly without interrupting normal operation of the machine, is a timer ISR.

Localization of Control
Localization of control is a concept that arises from the idea that a mail user should be an entity on a network rather than an entity on a machine. The serving machine, therefore, need not know where a user is at any time. Localization of control means that all transactions between user and server originate with the user.

Mailbox The mailbox is the local center of functionality in the mail system. It contains the memo queue, the memo out-queue, and the hold queue, as well as the access routines necessary to administer these structures.

Mailboxes Field The mailboxes field contains information concerning which of a user's mailboxes have received a particular memo. This field exists in the directory structure of the server out-queue for the user.

Memo A memo is the unit of information exchange in the mail system. It contains the message as well as other bookkeeping information.

Memo Out-Queue The memo out-queue is the place where memos originating with the mailbox are queued until a connection can be made to the server to transmit them.

Memo Queue The memo queue is the place in the mailbox where incoming memos are kept until they can be processed by the mailbox or by the user.

Message A message is the textual portion of the memo. It contains the human readable information.

Message Type The message type specifies the variety of a particular memo. The possibilities are: new, keep new, keep old, and delete. New specifies that the memo has probably not been read. Keep new indicates that the text of the memo should be kept in the mailbox's hold queue. Keep old indicates that the text of a memo already in the memo queue should be kept. Delete indicates that the specified memo should be deleted from the memo queue or from the hold queue.

One-way Function A one-way function is a function which is computationally easy to compute but whose inverse is difficult to compute. One-way functions are used to provide security in the password scheme.

Reclamation Reclamation is the process whereby directory entries in the mailbox and the server out-queue that contain information that is no longer useful are discovered and made available for new information. The used field plays an important part in this process for both the mailbox and the server. The full field plays a particularly important part in this process on the server side.

Remote Memo Queue
The remote memo queue is the place where memos destined for mail users on other mail networks are kept until the remote connection is made and the memos transferred.

Server In-Queue The server in-queue is the place in the server where incoming memos are kept until they can be processed by the mail server. There is logically one of these for the mail network.

Server Memo The server memo is an expanded structure which contains a memo as well as other information required specifically by the server.

Server Out-Queue The server out-queue is the place in the server where memos to a particular user are kept until that user asks for them. There is logically one server out-queue for each user.

SRM SRM stands for shared resource manager. It describes the file server of a network based around peripherals that are shared among the users of the network.

Update Memo An update memo is a memo sent to the server indicating that the status of a specified memo should be changed. An update memo can specify that the target memo is kept or deleted. The server uses this information to update the memo status so that loose consistency can be maintained.

References

[BIRRELL82] Andrew D. Birrell, Roy Levin, Roger M. Needham, and Michael D. Schroeder; *Grapevine: An Exercise in Distributed Computing*, Communications of the ACM April 1982, Volume 25, Number 4; p. 260.

[CLARK82] David D. Clark; *Names, Addresses, Ports, and Routes*, Arpanet RFC 814, SRI International, Menlo Park, CA, July 1982.

[CROCKER82] David H. Crocker; *Standard for the Format of ARPA Internet Text Messages*, Arpanet RFC 822, SRI International, Menlo Park, CA, August 13, 1982.

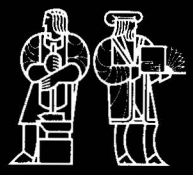
[GIFFORD79a] David K. Gifford; *Violet, an Experimental Decentralized System*, Computer Networks Journal, December 1981, Volume 5, Number 6; p. 423.

[GIFFORD79b] David K. Gifford; *Weighted Voting for Replicated Data*, Proceedings of the Seventh Symposium on Operating System Principles, December 1979, Pacific Grove, California; p. 150.

[POSTEL82] Jon Postel; *Simple Mail Transfer Protocol*, Arpanet RFC 821, SRI International, Menlo Park, CA, August 1982.

[REDELL83] David D. Redell and James E. White; *Interconnecting Electronic Mail Systems*, Computer Magazine, September 1983, Volume 16, Number 9; p. 55.

LABORATORY FOR
COMPUTER SCIENCE



MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TR-316

LOOSE CONSISTENCY IN A PERSONAL COMPUTER MAIL SYSTEM

Michael H. Comer

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139