T125
40

To The Graduate Council:

I am submitting herewith a thesis written by Karen Lauree Gilleland entitled "Distributed Databases Using MUMPS as an Example." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Kevin C. O'Kane, Major Professor

We have read this thesis
and recommend its acceptance:

Accepted for the Council:

Vice Provost
and Dean of The Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirments for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not by allowed without my written permission.

Signature Karen L. Gilleland

Date 6-4-86

DISTRIBUTED DATABASES
USING MUMPS AS AN EXAMPLE

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Karen Lauree Gilleland

August 1986

ACKNOWLEDGMENTS

I would like to thank Dr. Kevin O'Kane for his guidance and support during the course of this project. I am grateful to him for providing his MUMPS system for this project. I would also like to thank Dr. David Straight and Dr. Richard Blake for their support as committee members.

I would like to express thanks to my friends Mr. Leon Binder and Mr. Bob Kroboth for their constant friendship and helpfulness in reading and improving the manuscript.

Finally, I am most appreciative to my family for their encouragement and support during the course of my graduate study.

ABSTRACT

Distributed systems are needed to allow users to access data on remote machines. This project modified the current single computer version of MUMPS and created the network software necessary to implement a distributed MUMPS database system.

The above implementation allows separate MUMPS sites running under the UNIX operating system to be able to communicate and exchange data. The MUMPS distributed database system provides a mechanism to store data on many machines and to reference the data from any machine in the distributed system.

TABLE OF CONTENTS

INTRODUCTION/PROBLEM STATEMENT

This thesis concerns a method to create a distributed database environment for the UT-MUMPS system. The MUMPS, or Massachusetts General Hospital Utility Multi-Programming System, is a database system based on an interpretive language. The UT-MUMPS was developed by Dr. Kevin C. O'Kane. MUMPS was originally written in FORTRAN and developed for the DEC PDP-11 system. It has been translated into the C language and now is able to run on most UNIX machines. It is currently running on a VAX 11/780 and IBM PC-AT's. MUMPS is a database language that was developed mainly for medical applications, but is now used in technical and business applications as well.

The purpose of this thesis is to design amd implement a distributed MUMPS database system. Two separate MUMPS IBM-AT sites are integrated into a distributed system. The MUMPS distributed database system provides a mechanism to store data on many machines and to reference the data from any machine in the distributed system. In order to conform the current MUMPS database single computer system into a distributed system, network communications software must be written. Five of the seven layers of the International Standards Organization model for data communications software are coded and implemented. The MUMPS software is modified in the GLOBAL procedure in order to allow remote database access.

Many applications have database sites that are geographically dispersed and need to communicate with one another. If the separate database sites form a distributed system, then data can be stored only on the machine that uses it most. Other machines could reference the data from this machine.

Storing the data on only one machine would significantly decrease database update time since changes to the data only need to be made in one place. In order to connect separate computers into a distributed system, one must first know exactly what a distributed system is and how it is designed. The different specification and design considerations of a distributed database must be examined. This paper initially gives a general definition of distributed databases and explains why there is a need to have them. In order to have a distributed database, the machines involved must be able to communicate efficiently with one another.

The International Standards Organization provides a general description of data communications and networking design. The International Standards Organization is an organization that develops standards for telecommunications to help standardize worldwide communications. [17] The ISO general description breaks the communications problem into manageable tasks to insure that the overall design is not too complex. The use of separate parts makes maintainablilily and modifiability of the system easier. The model helps to ensure that the machines within a distributed system can communicate with one another efficiently and reliably. Detailed information about each layer needed to created a distributed database is presented.

After considering all of the design considerations, an appropriate distributed method is chosen for the MUMPS application. In order to ensure that one link will not cause the entire system to fail, a fully distributed network is used. In this way the network functions are divided up among the machines in the distributed system and no one machine controls the entire data distribution. Most of the International Standards Organization network layers are used to provide reliable data transfer and communications between the

machines in the distributed system. The original distributed MUMPS system is designed to function on two IBM PC-AT's. A method is given to add additional machines to the distributed system.

CHAPTER I

DISTRIBUTED DATABASE SYSTEMS--A GENERAL OVERVIEW

Introduction

A distributed system is a set of separate sites. Each site has its own memory and there is no shared memory among the sites [16]. A distributed database system allows physical data structures to be referenced by a logical structure so that the user need not be concerned about how the data is physically represented and stored. A user can access the data by name and logical interrelations between the data rather than a physical location [10].

There are five rules that should be observed when designing a distributed system. First, general-purpose processors must be present. Secondly, the individual processors must be able to communicate with one another in order to have distributed control. All communicating processes must cooperate in order to communicate successfully. Thirdly, each processor must have an independent operating system. Fourthly, the different parts of the distributed system must be invisible to the user. The system must appear to be a centralized computer to the user but to perform with additional reliability, adaptability, and modularity of a decentralized system. Lastly, an overall plan that allows the machines to remain separate entities and yet communicate and cooperate with one another should be established [4].

Distributed systems are more modifiable than a conventional centralized computer system. A distributed system does not change when performance requirements change. Since the system is modular and each piece ex-

ists as a separate entity, it is relatively simple to replace some of the general-purpose machines with processors dedicated to a specific purpose, or to increase or reduce the number of components, or to replace certain elements with better ones [4].

Development and maintenance costs are less for distributed systems than conventional computers because the distributed system is a group of smaller elements which can be accessed individually. Likewise, installation is simple since elements may be added or subtracted easily. Since each individual component is a separate entity, debugging is simpler [4].

Design Considerations of a Distributed System

The design of a distributed system can be broken up into four main parts: interprocess communication, system control, network control, and distributed data [4].

1) Interprocess communication (IPC) is composed of an application layer, an operating system layer, and a communications layer. The IPC is the main part of a distributed system. Process to process communication must follow a pre-specified set of rules or protocol. A host to host protocol is necessary due to arbitrary delays and communication errors. Communication should be transparent to the processes using the distributed system. To relay information the source process should only have to know the ID number or name of the destination process. Each processor should have the power to choose when to communicate, to decide the amount of resources to allocate for the communication, and if

it is willing to be blocked on for a specific period of time. Transmission should be efficient. The IPC should not harmfully effect any other processes [4].

2) The second design consideration of a distributed system is system control. A resource manager is used to schedule, allocate, and control global resources. The resource manager handles process control, network control, and data control. Distribution of control mechanisms may vary. For example, network control may be centralized while processes and data may be distributed. Control mechanisms or protocols may be distributed in two different ways. A dedicated system may exist where each processor is assigned a specific task or function for a common problem. Secondly, any processor may be allowed to run in any mode. Each computer has a copy of all processes within the system and works on those assigned to it. This latter approach leads to a system that is better balanced [4].

3) The third design consideration of a distributed system is network control. A network controller must be able to detect local processor overload, global overload, or failure of one or more processors. Network control may be centralized, partially distributed, or fully distributed [4]. For centralized control, one processor becomes the system controller. All processors send a status to this central controller. One main disadvantage to this method is that if the central controller fails, the entire distributed system fails. With partially distributed control, the control can migrate from one processor to another. A processor's status is

sent to all other processors. Several advantages to a decentralized distributed system over a centralized distributed system exist. Bottlenecks and contention between processors are decreased. Since global system functions are broken into tasks that several processors handle, shorter response time and higher throughput result. The overall system controller's primary duty is to detect and resolve possible allocation conflicts [4].

4) The fourth design consideration of a distributed system is the distributed data phase. Data distribution varies from a single file to a multiple file system with files and data replicated. Many systems have a local data-base manager (LDBM) at each processor and a distributed database manager (DDBM) which communicates with each local manager. The distributed database manager must take care of redundant files, concurrency control, and security. The distributed database manager must monitor distributed data, translate queries, and maintain replicated file consistency and coherency [4].

Design and allocation of data is effected by update frequency, file type, consistency requirements, file access concurrency, and design of the underlying network. Data transmission should be efficient. The interprocess controller(IPC) should not disturb any other application processes [4].

Local database managers exist at each processor. A distributed database manager that interfaces with this local database manager is the distributed database manager system. A distributed database manager must take care of redundant files, concurrency control, and data security [4].

The distributed database manager must keep track of distributed data, translate queries, and maintain consistency and coherency among any replicated files. Two approaches exist for choosing where to process the data. The first approach is to move data to the program that needs the data. The second approach is to move the program to the data. Several factors that effect the design and allocation of data are frequency of update, type of file, consistency requirements, file access concurrency, and design of the underlying network.

The data may be distributed among individual systems where each data item exists once in the system. Another approach is to maintain redundant copies on a file or subfile basis. In practice, both methods can exist on the same distributed database system because of different requirements of control and information files. The local database manager must have some method to locate the data. The distributed database manager helps the local database manager locate the needed data. The data distribution should be transparent to the user. This transparency is necessary for portability, reconfigurability, and program development [4].

File Access Problems

Four approaches to solving the file directory or file access problem are:

1) A central control point could grant file access.

2) A shared central directory could be maintained, but access would be left to the requester.

3) Replicated directories could be maintained at all processors.

4) Local directories at each processor could be maintained to respond to requests for access requests made by other processors [4].

The distributed database manager must synchronize changes to any replicated data in order to maintain consistency and coherency between redundant files [4]. Synchronization is the regulation of concurrent processes and observable events as a function of the history of events in the system of processes. In order to resolve synchronization problems, the system must function consistently, prevent deadlock, and treat processes equally or establish priorities [16]. Immediate synchronization can be performed with all other references locked out until the changes have been made. Synchronized messages may be used to synchronize the files. Another approach may be to have immediate synchronization but to allow access to a file while it is being updated. Immediate synchronizing yields good consistency between files but gives slow access time. A different approach is to save changes locally and to submit the changes as a batch transaction later on. A batch approach is only possible when consistency is not critical and good access time is needed. Many systems use a variety of the above data synchronization approaches due to the different requirements of various files. In these systems, different protocols are available and can be selected according to the particular file's characteristics [4].

Distribution of Resources

Distribution of computer resources may vary from a single one-processor computer to a network of multiple general purpose computers each

with identical capability. Data distribution varies from a single file maintained without backups in a central storage database, to a multiple file system with replication of both files and directory distributed across several storage mediums. A distributed computer system is composed of hardware and software, system control, and data to be processed [11].

System control involves distribution of the process and a resource control mechanism which manages the necessary real-time operation of the architecture. Distributed system control can be a single fixed control point from which all decisions are made to a fully distributed set of identical control centers cooperating in the decision making process controlling system operation [11].

Distributed architecture design and implementation involves several design considerations. Some of these considerations are decomposition, highly reliable and fault-tolerant architectures, high performance architectures, distributed performance modeling, and high-growth architectures. Decomposition or partitioning involves the establishment of the concepts, principles, and rules to promote efficient application design. Other considerations are real-time control, distributed database design, and communications [11].

A distributed operating system is composed of cells. A network of these homogeneous cells is distributed across the computer network to control the operation of the message based computer network. These cells cooperate with one another to control the operation of the message based computer network to guarantee no central control point, thus avoiding a central point of failure [11].

Each distributed operating system cell is composed of a local process executive, a local database manager, a distributed database manager, a real-time resource manager, and communications. The local process executive controls the scheduling of applications tasks within the local computer. The local database manager controls the access of the global database by all local application tasks. The local database manager also edits each database access to determine if it is a remote file or a replicated file, in which case the distributed database manager is notified. The distributed database manager controls access of all remotely local files and performs the synchronization of replicated data files once a local update is made. The real-time resource manager controls the network load balancing and load sharing functions to prevent local network overload. The resource manager also controls the dynamic reconfiguration of the network architecture in the event of a non-recoverable failure. Communications controls the transmission and receipt of messages between computers within the network [11].

The distributed operating system (DOS) cell is the interface between the applications software tasks and the physical system. The DOS system provides several network useful features. The physical network architecture is transparent to the application software to enhance software testability, system configurability for fault tolerance, system flexibility, and growth characteristics. Application software tasks can be developed just as though they execute on a centralized computer. It improves the software development schedule and simplifies unit level testing. The network DOS can be integrated and tested with only a subset of the network. Separated DOS and applications provide a controlled research environment for critical algorithm performance evaluation [11].

Distributed Database Design and Management

Database design can effect the design of the distributed system. Database accesses can require networking delays and concurrent accesses by parallel tasks which must be supervised to guarantee data consistency. Real-time databases where the database is distributed and redundant files exist have such concerns as file access, deadlock prevention, and coherency. Several factors must be considered when designing and managing a real-time database. File organization and database distribution across the network must be considered. Network and communications protocols and database security must be taken into account. Unauthorized file accesses should be prevented. File access must be controlled under conditions of a remote data access by a non-local task, redundant file maintenance, and concurrent file access by parallel executing tasks [11].

Several problems could arise if the network is not designed properly. Significant processor overhead could result from I/O handling of message traffic associated with database management and unanticipated network traffic due to database management messages. Remote file access delays could be caused by unpredicted internal processor message software delays. The network could be underutilized due to significant differences in message sizes, unaccounted for header or trailer information, or unaccounted for security header or trailer data [11].

Distributed Database Design

A database management system is a generalized software system designed to provide data organization, access, and control. Due to increased

geographic dispersion of users, data is processed and stored where the data originated or is used most. The major benefits derived by distributing these functions focus on increasing data availibility to the end user and reduced exposure to total system failure due to a hardware or software failure [18]. Most database systems are concerned with minimizing access delay and data management overhead and storage. Reliability and security also effect database design. Three major decision areas that influence database design are data organization, data allocation, and security and recovery. Data organization deals with a definition of the file contents and their structure using both the program view and the network view. Data allocation is the assignment of partitioned files to processors and global data memory across the node. Security ensures no unauthorized access of data and recovery reconfigures the system if a hardware failure occurs. Each of these design decisions effects physical storage size, the management approach used, and the data management software to be developed [11].

Database Allocation

Most allocation models are based on file interconnect models. A file interconnect model examines interdependencies between data and processes and seeks to place the data so that these interdependencies are minimized as much as possible. Another model used is called Foster's approach. This approach divides the problem into a micro-model and a macro-model. The micro model is used to obtain branching probability. The macro model starts with

the optimal branching probabilities in the integer programming model. The macro model uses file access frequencies, file length, and number of storage devices and their capacities. The macro step outputs the assignment of each file to the best device [11].

Another approach to the design of a distributed database is to analyze the types of distributed databases. Three major types of distributed databases that exist are fully redundant, fully partitioned, and partially redundant. A fully redundant database maintains replications of all data used so that the data is moved to the machine that needs it. A fully partitioned database has no replication of data. The data must be referenced on the machine on which it resides. A partially redundant database is a combination of the previous two. The last two require remote access. By weighing the advantages and disadvantages of these three methods in the areas of performance, reliability, and development costs, the best method for a particular implementation can be chosen. Often a combination of the above is used [11].

Distributed Database Management

A distributed operating system has a local data manager and a distributed data manager. The local data manager can create local files and retrieve and update parts of the database that is stored locally. It must acknowledge database requests for remote data or redundant data and must interact with the distributed data manager in these cases. The distributed data manager deals with functions which are unique to the maintenance of distributed data. The distributed database manager must interface with the local database manager and external communications. The basic functions of the distributed database

manager include keeping track of the data distribution, translating user queries into physical data accesses, and maintaining the consistency and coherency of both a partitioned and replicated database under concurrent user requests. The distributed database manager must manage simultaneous data update requests to insure the requests do not interfere with each other in a way which would cause inconsistent results or an erroneous database. The approach selected must be based on an analysis of the file, its reason for distribution, and the data freshness required for a particular application [11].

Two synchronization approaches for files are lockout and merging of tagged updates which can be based on physical or logical clocks. Lockout synchronization gives good consistency but has poor file access. Tagged synchronization has good file access but poor global consistency. Application-file interaction problems such as update frequency, file access rates, and remote accesses determine the file synchronization approach to be used [11].

CHAPTER II

NETWORKING STRUCTURES

Introduction

The network, or data communications software provides the tools that allow computers to communicate. Networking software is needed to provide a means of file transfer and message passing between computers. Data communications software can be very lengthy and complicated. In order to be manageable and modifiable, the network software is divided into manageable layers. By layering, or breaking the networking task into manageable parts, it is easier to modify the software at a later date. The International Standards Organization (ISO) divides the network software into seven layers. The ISO model is called the Open Systems Interconnect (OSI) model. This model has become the standard for network software. The layers near the actual physical communications handling are followed closely. As layers get farther from the actual physical communications layer, the less standardized they tend to be. The OSI model gives a general description of how the networking tasks should be divided up. The seven layers of the OSI model are the physical layer, the data link layer, the network layer, the transport layer, the session layer, the presentation layer, and the application layer [2].

Physical Layer

The physical layer gives a description of how the data is to be transferred over a physical media or physical connection between the two computers. The physical layer tells how the data will be physically represented and encoded [2].

There are two types of data transmission. The first type is called parallel or byte serial. In a parallel transmission, a word, or group of bits is transmitted at the same time. Parallel communications lines are used mainly when high speed data transfer is desired and data travels over short distances. Parallel communication must only be used in short distance communication applications because all bits in a word must arrive at the destination at approximately the same time, otherwise the data will be read incorrectly. Since most communications applications are not restricted to short distances a second type of communications called serial transmission is used. Parallel can only transfer fixed length units [2].

A serial transmission transfers one bit at a time over a single communications line. Only one bit arrives at the destination at a time. The line is polled at fixed intervals in order to detect the bits. Additional control bits must be added to indicate the start and ending of data [2].

There are two types of serial transmissions. The first type is called asynchronous serial data transfer. The asynchronous data is framed with a single start and stop bit. Since only a single bit used to denote the beginning and ending of a frame, only a small number of data bits can be transferred in a frame. Asynchronous transmissions transfer about a word at a time and work at low speeds. After each word is transferred, the line is made idle; that

is, no data is transferred temporarily. An idle state is represented by a one state on the line. The start bit to denote the beginning of data is a single zero bit. A stop bit that signals the end of the data is a single one bit. The least significant bit is usually transferred first. In asynchronous communications, the two machines that communicate must mutually agree upon baud rate, the number of data bits, parity, and the length of time that the line is idle before any data transfer can take place. The idle time between data byte transfers may vary from no time to a significant amount of time [2].

Several types of errors can occur in asynchronous transmissions. A framing error may occur. A stop bit or start bit may be left off. The data may have incorrect parity. The receiver may be overrun by a fast data transfer rate. If eight bits are transferred and there is no idle time, only eighty percent of the data transfer line is utilized. Greater line utilization is possible though. If more data bits are transferred, the chance of a data error increases. Any transition of the line will look like the start of data. If a pattern of start bits were given, then more data could be transferred at a time [2].

Another type of serial transmission is called synchronous transmission. In synchronous transmission, a pattern of several bits is used to denote the start and stop of a data frame. A multi-bit start sequence is much less error prone than a single start bit. When using a single start bit, any noise on the line could cause a state transition and look like the start of a data frame. The noise must occur in a particular pattern in order to be mistaken for a data start pattern when several bits are used in the start sequence. A sequence of bits used to start a data frame is called a SYNC character. Since synchronization characters are used to transfer large amounts of data, it must be ensured that the data is being sampled or polled at the proper time. Two separate clocks for

each processor will not be reliable. A clock pulse must be inherent in the data or transferred over a separate line [2].

Physical Media

Many different types of physical media exist. Twisted pair cable is one type. Twisted pair is used for short distances and low speed transmissions. Twisted pair is only baseband because it is built to transfer one frequency at a time. Baseband means that only one frequency of data can be transferred at a time [2].

Coaxial cable is another type. Coaxial cable transmits broadband signals. Broadband uses a wide spectrum of radio waves. Each wave frequency can be used to transmit or receive a signal. Other types of physical media are optical fiber, microwaves, infrared waves, radio waves, and satellites. These methods are used in various applications. Twisted pair and coaxial cable are the two most common physical media [2].

The physical media can be shared by several different users. Three major types of media sharing exist. The first is a frequency division multiplexer. A frequency division multiplexer can carry several users since it has several frequencies. Each user could have his own frequency. A time division multiplexer is another method. A time division multiplexer uses a single carrier cable or a single line of a frequency division multiplexer [2].

Time division multiplexing is broken up into multiplexing and concentrating. In multiplexing, each user gets a slice of time to use the line. The sum of user input baud rates must equal the bandwidth of the carrier line. This method can only carry a fixed number of users and the user must pay for

the time allotted to him whether he uses it or not. The other category is called concentrating or a statistical multiplexer (stat MUX). A stat MUX can accommodate a large number of users. The theory behind the method is that since there are many users, all users will not want to use the line at the same time. In this method the sum of the bandwidths of the users can exceed the bandwidth of the communications line or trunk line. When a user is ready to transmit, his ID is attached to the data item and the item is placed in a queue until the trunk line is available. This method requires buffering for the queue of requests and flow control to poll the users and determine when a user is ready to send data. Separate lines are used for flow control [2].

Data Link Layer

The data link layer helps to ensure that the physical layer data is accurate and reliable. The data link layer is concerned with data reliability, data framing, data transparency, sequence control, line control, and startup control. The data link layer is only necessary if serial rather than parallel communications are used [2].

Data reliability is used to detect any errors in the data transferred. The errors found may be ignored, corrected, or cause the data to be retransmitted. Several data reliability methods are used. All the methods can detect a single bit error and most can detect multiple bit errors. A few methods can correct one and two bit errors. Parity is one type of error detection routine. Parity can detect single bit errors, thus it is limited in use to a very small number of bits, usually a byte. This method us used in asynchronous data transmission since it can only be used on small amounts of data.

The second method is called a checksum. A checksum is the sum of all the bytes in a frame of data [2].

The longitudinal redundancy check is another method. It is easy to calculate. This method calculates the exclusive-or of all bytes in a frame. The last method is the cyclical redundancy check. This method calculates a polynomial by having each data bit contribute to the polynomial in several different places. This method is easily done in hardware but is very time consuming in software. When an error is detected, it can be retransmitted, ignored, or corrected using the reliability method [2].

Three methods for data retransmission exist. The stop-and-wait protocol is the simplest method. In the stop-and-wait protocol, the sender sends a data frame and then waits for a response before sending any more frames [17]. If a reply is not received within a specified amount of time, a timeout occurs and the message is assumed to be lost and is retransmitted. If the acknowledgment is lost and the data message was received correctly, a duplicate message will be sent. The communications line can be better utilized if the receiver sends back data along with the acknowledgment [2].

A second retransmission method is called retransmit-last-n or the go-back-n strategy. In this method, data frames are sent before waiting for an acknowledgment. In order to identify the data frame, a unique sequence number is assigned to the data frame. As each data frame reaches the receiver, a response is sent back to the transmitter. The response contains the sequence number of the data frame that was received. In this scheme, the sender must keep track of the frame number that is to be sent next and store frames sent but not acknowledged in a buffer in the event the message must be retransmitted. The receiver must keep track of frames received to know

which frame is expected next. If a negative acknowledgment or timeout occurs, the sender retransmits the messages beginning with the last frame for which an acknowledgment was not received. If the receiver gets a frame number that is out of order, all subsequent messages are ignored until the transmitter retransmits the message expected [2].

Selective retransmission is very similar to retransmit-last-n. Messages are sequentially numbered and sent without waiting for an acknowledgment. The receiver keeps track of all message numbers received. The sender retransmits only all frames for which an acknowledgment has not been received until all messages are acknowledged. This method requires a lot of buffers to hold messages until they are acknowledged and is very complex to implement [2].

The retransmission scheme used depends on the amount of memory available on a particular computer, the degree of reliability of the physical media, and the type of physical media used. The go-back-n method uses less buffer space and is less complicated to implement than the selective retransmission strategy. The selective retransmission scheme is more efficient in available transmission use [5].

The data link layer must frame the data that is transferred. Each frame includes the actual data to be transferred as well as needed control information such as the frame beginning, the sequence number of the frame, the host name that is sending the frame, and the frame type. The data transferred should be transparent. The data transferred should not be effected in any way by the characters used in the control information. Some form of sequence control must exist to ensure that all messages are received eventually in the order in which they were sent. A method must be provided to tell which side

of a two way alternate link can transmit and a means of flow control between computers that use the link [2].

Framing

Three types of framing and transparancy mechanisms are byte control, byte count, and bit stuffing. Byte control uses a pattern of bytes to denote the beginning and ending of the data. If a character inside the variable length data field is one of the characters in the stop pattern, the character is doubled in the data. Instead of using a stop bit, the length of the data is given in the control field before the data. The last method is called bit stuffing. Bit stuffing forces all bytes within the control characters to contain at least six consecutive ones. All data bytes are forced to never contain over five consecutive ones. After the fifth one in the data, a zero is automatically inserted by the transmitter and stripped by the receiver [2].

Network Layer

The network layer is used to relay messages through intermediate computer nodes in the network to a specific destination computer node. The network layer must decide the best route through the intermediate network nodes to the destination node. The network layer must control congestion. Only a limited amount of data can travel across the network at a time. Each node in the network is assigned a unique address to uniquely identify it. The network layer may be used to ensure that messages reach the destination in the same order in which they were sent. There are two types of networking models. The virtual circuit model ensures that all data packets are delivered

in order, and without error by the network layer. The datagram model only attempts to deliver messages. Each message is treated as an isolated unit. Some messages may arrive out of order, or not at all [17].

Routing Algorithms

A routing algorithm determines which trunk line a message should be transmitted on. If virtual circuits are used, a route is chosen at the beginning of a session and remains the same throughout the entire communication session. A routing algorithm should be as efficient and fair to each user as possible. For routing, the network must maintain a routing database. The routing database contains a list of all nodes, and a list of the links that connect nodes. Routing algorithms may allow estimates of current traffic to influence which line is used or they may not. The routing information is maintained at each individual routing node or in a centralized routing node [17].

A simple routing algorithm is called flooding. Every packet is sent out on every outgoing line except the one it arrived on. Many duplicate messages are generated. Flooding is not used in most applications. It may be used in distributed database applications when each node in the database needs to be updated simultaneously. Another type of routing called static or directory routing is widely used. Each node maintains a database that contains each possible destination for this node. A problem with this method is that it does not adapt to changes in the network load [17].

A variation of static routing is called centralized routing. Centralized routing maintains a database just like static routing. Centralized routing uses a central node called a routing control center to make all routing decisions.

The routing control center has complete information about the network and can make optimal routing decisions [17]. In centralized routing, routing changes are made at only one node. The failure of the centralized node will lead to the failure of the entire system [2].

If the routing information is contained at each node, the routing is distributed. In distributed routing, routing information is periodically exchanged between neighboring nodes [17]. Distributed routing techniques to update each node if routing information is changed are time consuming. Routing decisions can proceed quickly because the information is at the node. A distributed routing system is less likely to fail since the routing information is not at just a single node. Most networks in use today use distributed routing techniques [2].

Transport Layer

The transport layer ensures that data passed through the network layer arrives at its destination accurately and in order. The transport layer works much the same way as the data link layer. The transport layer must ensure that a delayed data packet does not arrive just late enough to be the correct sequence number for another data packet [2]. Just as in the data link layer all transmitted messages must be buffered until an acknowledgment is received. The transport layer also establishes connections or gateways to other networks [17].

Session Layer

The session layer is used to mediate between the operating system of the machine and the network. An applications program interface to the session layer is similar to an I/O device connection [2]. The session layer is utilized by a user to connect to a process on a remote machine. This connection between two processes on remote machines is called a session. The session layer ensures that the user has sufficient privileges to access a remote process. The session layer decides which transport layer mechanism is to be used if several exist. The session layer must recover from broken transport connections. In a distributed database system, the session layer would ensure that the entire update request reaches the remote node so that the database update could never be aborted in the middle of an update and leave the data inconsistent [17].

Presentation Layer

The presentation layer provides data format conversion and encryption. The format of the data must be changed if different machines with different operating systems are used. Some data format conversion may be required due to differing hardware representations of data. Other types of conversion may be required due to operating system differences. File types may have to be converted. For example, a sequential file may have to be converted to an indexed file. The data from each node must be converted to a network wide data format and then converted from the network wide data format to the remote data format. In this way, the data only has to be converted twice [17].

Each host only has to maintain the means to do a single type of conversion instead of many conversions to each type of data format that a remote host has [2].

The presentation layer is responsible for encrypting data. Since networks use phone lines and satellites, an unauthorized user may gain access to the data. In order to ensure data secrecy, an encryption scheme must be used. The message to be encrypted is known as plaintext. The encrypted message is called ciphertext. Encryption schemes are divided into two main types [17].

The first method is a substitution cipher. A substitution cipher replaces each letter or group of letters by a different letter or group of letters. One type of substitution cipher is called polyalphabetic cipher. This type of cipher uses multiple cipher alphabets in rotation. A polyalphabetic cipher is better than a cipher that uses only a single cipher alphabet or a monoalphabetic cipher. A polyalphabetic cipher can be easily broken if the cryptanalyst has enough ciphertext. The key length is the key to the cipher. The cryptanalyst assumes different key lengths until all the ciphertext letters in each column have been encrypted by the same monoalphabetic cipher. When this occurs, the ciphertext will have the same frequency distribution of characters as regular plaintext. Each column is attacked as if it were a separate monoalphabetic cipher. In order to make the code harder to break, a key that is longer than the message could be used. This key should be a random sequence of characters. Such a method is called a one time key. The key cannot be memorized, thus both the receiver and sender must have a written copy. The total amount of data that can be transmitted is limited by the amount of key available. This problem can be overcome if the text is broken up into pieces and

the same key used to encipher each piece. Since the key is used multiple times, the cipher can be broken [3].

Another type of cipher is called a transposition cipher. A transposition cipher reorders the characters of a plaintext message instead of changing them [17]. A transposition cipher usually rearranges data using some geometric figure [3].

The National Bureau of Standards has defined a Data Encryption Standard (DES) that can be used in unclassified U.S. Government applications. The DES algorithm enciphers sixty-four bit blocks of data using a fifty-six bit key [3]. The DES algorithm is divided into nineteen parts. A transposition of the plaintext is the first stage. The last stage is the inverse of this first transposition. The next to the last stage exchanges the leftmost thirty-two bits with the rightmost thirty-two bits. The remaining sixteen functions are essentially identical but use different functions of the key. A ciphertext generated by this method can be changed back to the original plaintext by applying this method is reverse order [2].

Application Layer

The application layer provides network services to the user. This layer is an interface between applications programs and the network software. This layer initiates connect requests from one process to a remote process. This layer may provide frequently used network access routines. It may provide a login procedure for a particular machine. This layer also provides mail and file transfer services to remote machines [2].

CHAPTER III

THE MUMPS DISTRIBUTED DATABASE SYSTEM

Design Implementation of Distributed Database Theory

The MUMPS database system is based on a command interpretive language. The MUMPS database system is designed for an experienced programmer rather than a casual end user. The MUMPS database is used in a variety of applications in medicine and business. MUMPS can be used in any application that needs an online database and a powerful programming language to manipulate the database [6]. Many applications have a set of separate sites to perform parts of the particular application. For example, a hospital environment has a site for pharmacy records, another site for lab records, and another site for billing records. It would be useful for these separate sites to be able to communicate with one another. Currently, MUMPS is a database language that is designed to access a database that resides on only one computer.

The purpose of this thesis is to design and implement a distributed MUMPS database system. Two separate MUMPS sites on IBM-ATs' are integrated into a distributed system. The distributed system is designed so that a user can easily access data contained on any machine in the distributed system.

This distributed version of MUMPS is needed to allow a user to access and use data on remote machines. Use of this approach will help eliminate redundant copies of data on the various machines by allowing each user to ac-

cess any data they need regardless of the machine that it resides on. The data will be kept on the machine that accesses it most.

This thesis describes a method that allows two or more separate MUMPS sites running under the UNIX operating systems to be able to communicate and exchange data. The design of the distributed MUMPS system is divided into data distribution, the local and distributed database managers, MUMPS standards, and data communications.

The system is designed to be a fully distributed database system as opposed to a centralized database system. For example, in a hospital environment; instead of one large computer that services the lab, pharmacy, nursing stations, radiology, and the business office; a separate computer could be assigned to each separate function. If separate computers are used, the computers must be integrated into a distributed system.

There are several reasons why a distributed system is better than a single large computer. Many smaller computers allow the computer to be at the site where it is needed. If one computer fails, the entire hospital system does not collapse. It is more convenient to have a computer at each site. If the lab computer is not working correctly, it can be repaired without effecting the rest of the hospital. If a distributed system is used, each system can be accessed individually and installation is simple because computers can be added or subtracted easily [11]. For example if a tumor registry is needed, one machine can easily be installed to handle the registry.

A distributed system can be centralized or fully distributed. One computer controls the communications in a centralized system. Each computer controls its own communications in a distributed system. A fully distributed

system is less likely to fail than a centralized system. If one node fails, the entire system will not automatically fail on a fully distributed system [11].

Data Distribution

Currently, MUMPS resides only on one machine. An entire patient record must be stored on this machine; or the patient record must be duplicated and stored on every machine that needs it. This means that the billing office computer must also store the lab data and the pharmacy data in order to bill the patient for any lab tests or prescriptions given while the patient stays in the hospital. This duplication of data is costly to maintain. The database or patient record may be unstable or inconsistent while the update is taking place. The lab computer may have one set of lab results for a patient while the billing computer has an entirely different set of lab results.

One solution to the data duplication problem may be to use just one computer for the entire hospital. If only one computer is used for the entire hospital database, the data redundancy program is eliminated. If this computer malfunctions, the entire hospital is without a computer. As hospital needs expand, a single computer may not be sufficient to adequately handle the entire hospital database.

A better solution would be to have a separate computer for each task. If a separate computer is used for each application in the hospital, the hospital can respond easier to changes in the database. If the lab gets larger and handles a significant amount of additional data, the lab will need to increase the size of the lab portion of the hospital database. If a distributed system is used, one of the computer systems can be replaceed by a larger computer or

an additional system can be added without effecting the rest of the database that resides on different machines.

MUMPS Standards

The distributed remote access within MUMPS must match the accepted syntax standards of the MUMPS system. A local machine data reference to permanently stored data, called a 'global' reference in MUMPS contains the pathname or path to follow to access a particular data item. All global references are in a tree structure and use sparse matrix techniques to make accesses quicker [14]. A local data path would look like

```
^/labtests/blood(ptid)
```

In order to change the contents of this data item, a set command is used. An example of the set command is

```
s ^/labtests/blood(ptid)="type A"
```

To reference any item in this global data structure, one need only give the full path that is taken to reach the item. This path can inherently store data in the path itself as well as at a particular node in the path. In order to expand this global reference idea to a distributed system, it is only necessary to include the remote system name in the pathname. For example if a user on a machine wants to change a data item on remote machine HOST1, the set command would look like

```
s ^/host1/labtests/blood(ptid)="type A"
```

The syntax would essentially remain the same for a remote access as a local access of the global database. When a remote request is generated, a network communications process, or distributed database manager, is created to handle the request and return any results while the requesting machine waits.

The distributed MUMPS system's appearance to the user is the same as the current MUMPS. The procedure to update a patient record is the same for both. The only difference is that the remote location of the patient record must be specified in some manner in a distributed system.

In the current MUMPS, each piece of a patient record is stored using a global variable or array name. Global array names always begin with a circumflex character (^). The entire patient record is viewed as a tree structure. The global array name for a particular piece of a patient record describes a path through the tree to the correct patient record part [15]. For example, to reference the blood test results for a patient whose name is Jones, the global reference would look like:

```
^/patientrecord/Jones/labtests/bloodtests/blood(ptid)
```

Using the distributed MUMPS, in order to reference the same information about Mr. Jones from the pharmacy computer, the appropriate remote

computer name is added to the global array name. For example, if the lab computer is designated lab, then the global reference would look like:

^/lab/patientrecord/Jones/labtests/bloodtests/blood(ptid)

Certain operations can be performed on the data in a distributed system just as in the current MUMPS. In order to change a particular item in the patient record, a SET command (abbreviated S) is used. In order to write out the value of a patient record, the WRITE command (abbreviated W) is used.

In conventional MUMPS, the blood type could be entered into the database using the command:

s ^/patientrecord/Jones/labtests/bloodtests/bloodtype(ptid)="Opos"

The same command would be used to enter the blood type in a distributed system if the user was on the lab computer.

To write out the blood type of Mr. Jones, the following command could be used in the current MUMPS or in a distributed system if the user on the was on the lab computer:

w ^/patientrecord/Jones/labtests/bloodtests/bloodtype(ptid)

In a distributed MUMPS system, if a user on another machine, say the pharmacy computer, wishes to write out the value of the blood type, the following command would be used:

w ^/lab/patientrecord/Jones/labtests/bloodtests/bloodtype(ptid)

The billing computer may need to know each drug administered to a patient in order to bill the patient. There is a large number of pharmacy drugs that could possibly be administered. Instead of having to search through each possible drug, the $NEXT command allows the user on the billing computer to request the location of the next drug given to a particular patient. For example, if drugs is an array, the $NEXT will give the user the next element in the drugs array. The command

j=$N(^/pharmacy/patientrecord/Jones/drugs(-1))

would place into the local variable j, the location of the first drug given to Mr. Jones.

Network Implementation

Each machine in the distributed database must have a means to communicate with remote machines. The network system controls the transmission and receipt of messages between computers within the network. In order to make the network design as manageable and maintainable as possible, the network design is broken up into layers. The International Standards Organization (ISO) network seven layer model is followed. The seven layers within the ISO model are the applications layer, the presentation layer, the session layer, the transport layer, the network layer, the data link layer, and the physical layer. Most of the layers are designed and implemented in this thesis.

Some layer algorithms are taken from various sources and modified and implemented for the MUMPS distributed database. All of the programs are written in the C language since the MUMPS interpreter is written in C.

Application Layer

The application layer is the software needed to initiate the remote communications for the local MUMPS. The application layer acts as an interface between the applications program, or the MUMPS interpreter, and the data communications software needed to send the data to a remote machine. The applications layer is implemented inside the MUMPS code in the procedure global. This layer passes the data request information necessary for the remote access such as the host pathname and any other data that is to be written on the remote database. For example, if a user on the pharmacy computer wishes to write out the value of Mr. Jones's blood type that resides on the lab computer, the write command and the name of the lab computer is passed to the communications software by placing this information in a sequential file. This program forks a new process that does the presentation phase while the original MUMPS program that requested the data, or blood type in this case, waits for the result.

Presentation Layer

The Presentation Layer is responsible for translating the data into a form that is understood by the remote machine. This layer also does any file encryption that may be needed. This thesis is implemented using the assumption that all machines in the distributed database system are using the UNIX

operating system. Data translation is unnecessary because all data on all machines should be the same since they are all running UNIX. If at some point another machine with an operating system other than UNIX is desired, it could be added, but this would require extensive additions to the Presentation Layer.

Data encryption is desired because the data is traveling over direct communications lines or phone lines and these lines can easily be tapped by an unauthorized user. Since MUMPS applications in the medical field deal with sensitive data, it should be encrypted so that data transmissions cannot be intercepted and easily read. Many different encryption schemes are available. These encryption schemes range from techniques very easy to break to methods that are difficult to break. Unfortunately the more difficult the scheme, the slower it is in performance. Rather than choose one difficult scheme, two simpler schemes are implemented. It is usually harder to decode data that has been encrypted several times as opposed to data that has only been encrypted once [3].

There are two main types of encryption algorithms. A transposition cipher moves the letters of the text around in some in order to encode it. A substitution cipher substitutes one character for another character. One method from each type is chosen [3].

The first encryption scheme is called a Vigenere cipher. It is a polyalphabetic substitution method bases on shifted alphabets. The key K is a sequence of letters where each letter in the key K indicates which alphabet should be used to encrypt a particular letter in the plaintext message [3]. The Vigenere Cipher has a square matrix containing the alphabet characters and a key. The key is written over the data message. The key letter over the data

letter is the row into the encrypting matrix and the data letter is the column. To encrypt each letter in the original data or plaintext , the ciphertext letter is the letter in the column pointed to by the plaintext letter and the row pointed to by the key letter. To decrypt the ciphertext, the plaintext letter is the column that contains the encrypted letter in the row pointed to by the key letter [3].

The second encryption scheme used is a transposition cipher. A transposition cipher rearranges characters in the data. In the "rail-fence" cipher, the letters of a plaintext unencrypted message are written down in a pattern that looks much like a rail fence, and then removed by rows. The key to breaking this cipher is determining the depth of the fence. The rail-fence writes the letters of plaintext in a pattern similar to a rail fence and then removes the data by rows [3]. For example,

thisisamessage

becomes

t i e g
h s s m s a e
i a s

OR:

tieghssmsaeias

The data is doubly encrypted to make unauthorized reading more difficult. A skilled cryptographer could decipher the code with little trouble. Other encryption methods were not used because the encrypted text must contain all printable characters or must in some way be converted to all printable characters or the physical layer will not work correctly since the physical method used requires that blocks of data must be transferred at a time, instead of a character at a time.

Session, Transport, and Network Layers

The session layer is not needed because there is only one transport method that can be chosen. A small transport layer is used to unsure that the request is executed at the remote machine. A timer is set up that will time out within a specified amount of time, in this case five minutes. If the timer goes off before a reply reaches the original user, the message is assumed to be lost and the remote host is assumed to be unavailable.

The network layer is not currently needed because all routing is the responsibility of the user due to the way that a request path is specified in a command. Few computers will be connected together, thus the paths will be very short. A large system with many computers would work better if scheduled automatically. Overhead of these layers is eliminated altogether.

Data Link Layer

The next layer needed in communications is the data link layer. The data link layer detects and corrects essentially all errors generated by the physical layer. The data link layer ensures that the data transferred over a physical

media is error free. To increase performance this layer could be omitted. If errors occur however, the database could be corrupted and erroneous results could occur. It is necessary to add this layer to keep the database uncorrupted. Several components that effect the structure of the data link layer are desired line utilization, services provided and required by the physical layer, the quality of service, reliability of the physical layer, data rate of the physical layer, and end-to-end delay of the physical layer [2].

The data must be framed in some way to know when it starts and stops. A checksum is used to ensure data reliability. The UNIX operating system has a utility called SUM that calculates a 16-bit checksum that is the sum of all eight bit bytes transferred. The checksum is placed onto the front of the data [2]. When the data frame is transferred, the receiver sends back an acknowledgment if the transmission is successful. Line control is not really necessary because the side requesting the remote database access will always begin the data transfer.

If the physical line error rates are adequate for the application, no error checking is required [2]. Since the database should be protected against corruption as much as possible, additional error checking routines must be used. One solution may be to ignore the frames in error and let higher level layers detect the error and retransmit the frame. This approach is used in Ethernet. Another solution may be to retransmit the message. Lastly, the error may be corrected from the check sequence used [2].

The first option is discarded because it is used only in a connectionless service. In a connectionless service a network connection exists all of the time and the data is just passed to it. A connectionless service provides no way to check physical errors in the data link layer. A connected service is

used instead. If the message is to be corrected from the check sequence used, the check sequence must be detailed and more information must be transferred across the communications lines. Since the communications lines are across small distances and have few errors, a retransmission scheme is used.

There are three basic retransmission schemes. The first scheme is the stop and wait protocol. In this method each message is followed by a response to the message. Messages are retransmitted until an acknowledgment is returned. If no response is received within a certain time period, the message is assumed to be lost and is retransmitted. This method only allows communications in one direction at a time. If the communications line is full duplex, the line is greatly underutilized.

The next strategy is called selective retransmission. Messages are sent. After a set of messages are received, a list of messages to be retransmitted is sent to the transmitter from the receiver. This method required a lot of buffer space. It is very complex to implement. Delay in acknowledging messages held in buffers can be very long [2]. The last method is called retransmit last n. Messages are accepted as they come in. If a message comes in out of sequence, the receiver tells the transmitter to resend all messages since the last one received correctly and in sequence.

Since direct connect lines are used, the number of retransmitted frames should be minimal. The retransmit last n method requires some buffering, but not near as much as selective retransmission. Since the database is distributed over personal computers with a limited amount of memory and direct communications lines are used to connect them, the retransmit last n method is used.

The retransmit last n or go back n strategy works in the following way. The local machine, or the machine that initiates communications, sends a data frame across the communications line to the remote machine. Each frame contains the actual data to be sent, the frame number, the checksum of the data, the ID of the machine sending the frame, and the type of frame. The frame to write out the blood type of Mr. Jones would look like:

P0D118 0 0 0 0/lab/patientrecord/Jones/labtests/bloodtests/bloodtype

where:

P -- denotes that this message is coming from the machine that initiated the communications--the primary machine

0 -- is the number of this particular message

118 -- is the checksum of the message

0000-- contains the information about the command used-in this case the write command

/lab/etc. -- is the description of the path taken to reach the blood type of Mr. Jones

When the remote machine receives the frame it checks to make sure that the frame has the correct number and that the checksum is correct to ensure that the data is correct. If everything is found to be correct, the remote machine sends an acknowledgment frame to the local machine to indicate that the data was received. The acknowledgment does not wait indefinitely for data

to go over. There is a counter set up to wait for a short while for a message to be sent and then send just an acknowledgment message if no data message is sent within the required time. If the remote machine receives corrupted data or does not receive a particular frame, it will not send back anything. If other frames arrive containing the wrong frame number, they are discarded.

After a brief period of time, the local machine realizes by a time-out that the frame has not arrived. It then goes back to the last frame correctly received and retransmit the frames of the message until it has received acknowledgments for all of the frames. Once a frame is sent, a copy of the frame is placed in a holding queue until an acknowledgment has been received for that particular frame.

The go back n strategy was chosen over the selective retransmission strategy because the go back n strategy requires fewer buffers to hold frames and because it is important that the frames arrive in order. Even though the go back n strategy is less efficient in its use of the transmission capacity than a selective retransmission scheme, it was chosen because direct communication lines have relatively few errors and the data access requests are not usually very big and can be represented by a small number of frames [5]. This distributed system may be used on small computers as well as large computers. Small computers have a limited amount of memory that should not be wasted with large buffer spaces.

Physical Layer

The lowest layer is the physical layer. The two IBM PC-AT's are connected by a DB9 cable. The physical layer uses the IOCTL routine in UNIX to actually establish the physical communications link between machines. It is not clear whether this method uses a synchronous or asynchronous data transfer method. Since this routine sets up the actual physical data transfer, it is not necessary to know which method is used. No error checking routines are mentioned in the documentation, thus it is assumed that none are used.

The IOCTL function establishes the communications link and allows access to the link just as any regular file would be accessed. The IOCTL allows baud rate, parity, data transmission protocols, and echo to be set. The C program opens this line much like any other file and reads for it and writes to it just as it would a file.

Two channels are opened to the device. One channel is opened for read only and one channel is write only. The transmitter uses these two channels to communicate with the remote machine. The remote machine just writes to the screen to communicate with the transmitting machine and just reads from the screen since this connection is essentially just like a virtual login to the remote machine.

Since no error checking routines are mentioned, a checksum is computed for the data before it is sent. This checksum is checked for comparison when the data is received. If these two sums match the data is assumed to be correct. If they do not match, this data frame is ignored.

When the remote machine has received the request and reassembled it, the presentation layer on the remote machine decrypts the message and an ap-

plications layer executes the request using the local MUMPS program. The output from the access is sent to the presentation layer where it is encrypted and passed to the data link layer. The frame is created just as it is in the local machine and passed to the physical layer that transports it back across the communication line to the local machine.

For example, the write command needed to write out Mr. Jones' blood type is reassembled from the information passed across the communication line and executed on the lab computer. The output of the write command, or the blood type, is passed to the presentation layer where it is encrypted. The encrypted blood type is passed to the data link layer and sent back to the user in the pharmacy department.

Implementation Notes

When a user issues a remote request, a timer is started. If within a specified amount of time the result has not arrived, the network will be assumed to not be functioning and a message will be printed. Using the blood type example, if the pharmacy user requests the blood type of Mr. Jones, a remote request is generated just as described above. If for some reason, the blood type request does not reach the lab computer; or if the lab computer is unable to successfully pass back the blood type to the pharmacy computer, an error message will be printed on the pharmacy computer to indicate that the lab computer is availble at this time.

Only one user may use the communications line at a time. A lock file is created to allow only one user to access at a time. For example, if a second pharmacy user wishes to access the nurses station for the patient drug ad-

ministration records, he must wait until the blood type has been given to the first pharmacy user.

The distributed database is implemented using multiple concurrent processes. Low-level pipes and temporary files are used to communicate between processes. Signals are used to synchronize and signal the completion of an event. The kill command is used to send the appropriate signal to the desired process.

Adding a Node to the Distributed System

The process of adding a node to the system should be rather straightforward. In order to be able to use the existing software, the machine must have UNIX, MUMPS, and a port to connect to. A directory on both machines wishing to communicate must be created called /usr/mumps/mps . This directory must contain the files OK.C, HOST.C, and MPSEX.C. The OK.C has a variable that must be set to tell whether the program is to initiate communications between machines (set PRIM=1) or to receive communications at first and be the secondary communications protocol (set PRIM=0). The actual device name used must replace tty00 in the login routine open statement inside OK.C in order for the correct device to be opened. this communications line must be enabled on the primary machine and disabled on the secondary machine by the system administrator. Once these variable is modified, the programs should be compiled using the following commands for the primary machine:

```
cc -o oka.e ok.c
cc -o host.e host.c
```

The next set of commands should be used on the secondary machine.

```
cc -o okb.e ok.c
cc -o mpsex.e mpsex.c
```

Once these programs are compiled a MUMPS remote request should work fine. The user must know how the machines are connected so that he can give the correct path to get from one machine to another one. If the software seems to not give a response, the login routine inside ok.c may have to be modified on the primary machine because the echo or parity or other such parameters may be set wrong for the machine that is being attached to.

Conclusions

The purpose of this thesis was to design and implement a distributed MUMPS system using existing MUMPS software. The distributed system allows the user to access data located on any machine in the network. The user treats the remote request much like he or she would a local request. The thesis proves that MUMPS can operate effectively as a distributed system. Remote databases can be accessed with no difficulty. A remote access tends to be slow compared to a local access. A typical remote access takes approximately two minutes. A remote request can be approximately as long as any local request. A remote request can transfer a maximum of four hundred characters at a time. Since remote accesses are infrequent, this method is more efficient and cost effective than having to maintain the some data on several machines though.

The method implemented works for one user at a time. Each user request must be processed totally before another request is accepted. The program waits for completion of each transaction. A test variable of some sort could be implemented that would allow the user to keep executing while this remote transaction proceeds.

In order to increase performance and efficiency, a job command could be implemented in MUMPS that would move over an entire MUMPS program piece, execute it on the remote machine, and pass the entire results back to the requesting machine. The MUMPS code job section would need a lot of modification, but the communications layers already implemented could be used for this purpose as well. The single command implementation would still be needed for single requests, but large requests should execute more quickly because the entire program is executed at one time instead of having the additional overhead of transferring each command and result separately.

If the data link and physical layers were executed at a level closer to the hardware of the actual machine, the response time should be faster. The use of UNIX and MUMPS made the remote request proceed more slowly. If UNIX is removed, the presentation layer would be much more complicated and may keep the possible response time increase minimal.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Ahuja, Vijay. *On Congestion Problems in Communication Networks.* Gaithersburg, Maryland: National Bureau of Standards, 1978.

[2] Cantrell, Mark. *Class Notes* University of Tennessee, Data Communications Class, 1986.

[3] Denning. *Cryptography and Data Security.* Reading, Mass: Addison Wesley, 1982.

[4] Fox, Mary and Wilson, Melissa. *Design Considerations Within a Distributed Data Processing System,* 1984.

[5] Halshall, Fred. *Data Communications and Computer Newtorks* Reading, Mass: Addison Wesley, 1985.

[6] Hirst, Norman. *The Logic of DDP and Its Impact on Programming Languages.* IEEE, 1980 Compcon Fall Proceddings--Distributed Computing Sept. 23-25 1980 Fall. New York: IEEE Publishing Service, 1980.

[7] Hoare, C.A.R. *Communicating Sequential Processes,* Communications of the ACM, Vol 21, Num 8, Aug. 1978.

[8] Karge, P. A. *IEEE 1978 Trends and Applications in Distributed Processing.* Gaithersburg, Maryland: National Bureau of Standards, 1978.

[9] Karpman, Ralph and Pard, Ted. *The Functionally Distributed Database--ANS MUMPS as an Implementation Tool* IEEE, 1980 Compcon Fall Proceddings--Distributed Computing Sept. 23-25 1980 Fall. New York: IEEE Publishing Service, 1980.

[10] Lampson, B. and Sigert, H. *Distributed Systems--Architecture and Implementation--An Advanced Course.* New York: Springer--Verlag, 1981.

[11] Mariani,M.R. *Distributed Data Processing--Technology and Critical Issues.* New York: TRW Systems and Energy, 1984.

[12] Martin, James. *Computer Networks and Distributed Processing.* Englewood Cliffs, New Jersey: Prentice--Hall Inc., 1981.

[13] McNamara, John. *Technical Aspects of Data Communication.* Bedford, Mass: Digital Publishers, 1977.

[14] O'Kane, Kevin. *C-Based MUMPS Interpreter User's Guide,* 1983.

[15] O'Kane, Kevin. *Introduction to ANSI Standard MUMPS,* 1984.

[16] Parker Y. and Verjus,J.P. *Distributed Computing Systems--Synchronization, Control and Communication.* New York: Academic Press,Inc., 1983.

[17] Tenenbaum, Andrew S. *Computer Networks.* Englewood Cliffs, New Jersey: Prentice--Hall, 1981.

[18] Weitzman, Cay. *Distributed Micro/Minicomputer Systems Structure, Implementation, and Application.* Englewood Cliffs, New Jersey: Prentice--Hall, Inc., 1980.

APPENDIXES

APPENDIX A.

MUMPS CODE MODIFICATIONS

APPENDIX A. MUMPS CODE MODIFICATIONS

```
/*****************************************************************
*
*       mumps c-based interpreter
*
*       copyright (c) 1984, 1985 by Kevin C. O'Kane
*
*       All rights reserved - Duplication prohibited
*
*****************************************************************/
/*****************************************************************
* This GLOBAL procedure is contained in the file SYS2.C in the
* Dr. O'Kane's current version of MUMPS. This file contains
* the necessary routines and calls nesessary to convert the
* conventional MUMPS to a distributed database system. The
* two remote procedure calls and variables added are clearly
* marked. The remote communicatons requires the addition of
* three procedures to the current MUMPS--REMOTE,LCK,UNLCK
* The REMOTE procedure is essentially the application layer
* of the data communications layers.
*****************************************************************/

#include <signal.h>
#include <stdio.h>
#define DEBUG1 0
        short g5recs;
        unsigned char direc[255];
        unsigned char direc1[255];
        int gblfd;
        int gpad;


/*******
        vars added
**********/
        int kk;
        int rtn;
global(g,v1d,bd)

        unsigned char v1d[],bd[];
        short g[];
{

        union cvt { long int cvtflt; char cvtchr[4]; } r1;
        short kkk,allocblk();
        long int rec;
        union { long int lftflt; char lftchr[4]; } left;
        long trace[50],newentry,trx;
        long int ii,i,k,l,m,n,t1,root;
```

```
short j,is,js,ks,ms,ls,ns,tracep[50];
static long int one=1,zero=0;
static unsigned char naked[255]={ 0 };
unsigned char byte,block[1024],key1[256],dat1[256];


/***** vars added
*****
*******/
int stat,remote();
if (g[3]==5) { /* count empties */ empties(bd); return(1); }

if (v1d[1]=='^'&&v1d[2]==206) {
        strcpy(key1,naked); strcat(key1,&v1d[3]);
        strcpy(&v1d[1],key1); strcpy(naked,key1); }
   else strcpy(naked,&v1d[1]);
is=strlen(naked); js=is;
for (is-=2; is>1; is--) if (naked[is]>128) { naked[is+1]=0; goto n1; }
naked[js++]=206; naked[js]=0;

n1:  if (v1d[1]=='^') strcpy(&v1d[1],&v1d[2]);


/****** NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE
*******
******* code for distributed remote machine accesses
*******
*******/
rtn=0;
signal(SIGTERM,SIG_IGN);
stat=remote(v1d,g,bd);
if (rtn==1)
{
        signal(SIGTERM,SIG_IGN);
        rtn=0;
        return(stat);
}
/*******
*******
*******  end remote call
*******/


if (v1d[1]!=2) {
        if (v1d[1]!='/') {
                strcpy(key1,direc); strcat(key1,"/");
                for (is=0; key1[is]!=0; is++)
                        if (key1[is]=='/') key1[is]=2;
                strcat(key1,&v1d[1]); strcpy(&v1d[1],key1); }
           else {
                for (is=1; v1d[is]!=0&&v1d[is]!=1; is++)
```

```
                        if (v1d[is]=='/') v1d[is]=2; }
                }

        for (is=1; v1d[is]!=0; is++) { if (v1d[is]>127) v1d[is]=1; }
        if (g[3]==3) { if (v1d[is-1]!=1) { v1d[is]=1; v1d[is+1]=0; } }

        /* pad numeric subscripts for collating sequence compatibility */

        if (gpad) {
        ls=is;
        for (is=1; v1d[is]!=0; is++) {
next_arg:       if (v1d[is]!=1) continue;
                is++; if (v1d[is]==0) break; ks=0;
                for (js=is; v1d[js]!=0; js++) {
                        if (v1d[js]==1) break; ks++;
                        if (ks>8) goto next_arg;
                        if (v1d[js]>='0'&&v1d[js]<='9') continue;
                                else goto next_arg; }
                for (ms=ls+(8-ks); ms>is; ms--) v1d[ms]=v1d[ms-(8-ks)];
                ls=ls+(8-ks);
                for (ns=0; ns<=(7-ks); ns++) v1d[is+ns]=' ';
                is=is+(8-ks); } }

        else for (is=1; v1d[is]!=0; is++);

        if (g[3]==2||g[3]==4||g[3]==6) { /* $n prep */

                for (js=is-2; v1d[js]!=1&&js>1; js--); /* find strt lst indx */
                if (js>1) substr(&v1d[1],key1,1,js);
                        else strcpy(key1,&v1d[1]);
                if (g[3]==6) { key1[0]=0; g[3]=2; is++; }
                if (js>1) js++;
                if (v1d[js]=='-'&&v1d[js+1]=='1'&&v1d[js+2]==1) {
                        v1d[js]=' '; v1d[js+1]=1; v1d[js+2]=0; }
                else { v1d[is-1]=' '; v1d[is]=1; v1d[is+1]=0; } }

        /****** NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE
        *******
        ******* code for distributed remote machine accesses
        *******
        *******/
        rtn=0;
        signal(SIGTERM,SIG_IGN);
        stat=remote(v1d,g,bd);
        if (rtn==1)
        {
                signal(SIGTERM,SIG_IGN);
                rtn=0;
                return(stat);
        }
```

```
        /*******
        *******
        ******* end remote call
        *******/
srch:   lseek(gblfd,0L,0); read(gblfd,&root,4);

        if (root==0) { /*no tree*/

                if (g[3]==0) {bd[1]=0; return(0); }

                if (g[3]!=1) {strcpy(&bd[1],"-1"); return(0); }

                if (g[3]==1) { /*insert case*/

                        k=allocblk();
                        lseek(gblfd,0L,0); read(gblfd,block,1024);
                        move(block,&k);
                        lseek(gblfd,0L,0); write(gblfd,block,1024);
                        for (i=0; i<=1023; i++) block[i]=0;
                        j=4; strcpy(&block[j],&v1d[1]); /*key*/
                        j=strlen(&v1d[1])+j+1;
                        strcpy(&block[j],&bd[1]); /*data*/
                        j=j+strlen(&bd[1])+1;
                        move(&block[j],&zero);
                        block[j+4]='~'; block[j+5]=0;
                        lseek(gblfd,k*1024L,0); write(gblfd,block,1024);
                        return(1); }
                }

        /*search for key*/

        i=root; trx=0;

nxtblk:         lseek(gblfd,i<<10,0); read(gblfd,block,1024); trace[trx++]=i; j=0;

nxtkey:         if (trx>0) tracep[trx-1]=j; js=j; j+=4;

        if (g[3]==3) {
                is=1; k=j;
                while (v1d[is]==block[k]&&v1d[is]!=0) { is++; k++; }
                if (v1d[is--]==0&&v1d[is]==1&&block[k-1]==1) is=0;
                else is=strcmp(&v1d[1],&block[j]); }

        else is=strcmp(&v1d[1],&block[j]);

        if (is<0) {
                move(&left.lftflt,&block[js]);
                if (left.lftflt<=0) goto fail;
                i=left.lftflt; goto nxtblk; }

        if (is!=0) {
```

```
                j=j+ strlen(&block[j])+ 1; /*key*/
                j=j+ strlen(&block[j])+ 1; /*data*/
                goto nxtkey; }

        /*found*/

                if (g[3]==0) { /*search operation*/

                        j=strlen(&block[j])+ j+ 1;
                        strcpy(&bd[1],&block[j]); return(1); }

next:           if (g[3]==2||g[3]==4)
                        return(next1(g,v1d,bd,j,key1,block,trace,trx,tracep));

                if (g[3]==1) /* store */ return(store1(g,v1d,bd,j,block,i));

                if (g[3]==3) { /* delete */
                        delete1(g,v1d,bd,block,i,j,trace,tracep,trx,js);
                        goto srch; }

                /*end found*/

fail:   if (g[3]==0||g[3]==3) {
                bd[1]=0; return(0); }
        if (g[3]==2||g[3]==4) goto next;

        trx--;

        /*store*/

        chop(g,v1d,bd,j,i,block,trace,tracep,trx);
        return(1);

        }

/******* routine added for remote machine access
********
********/
#include <signal.h>
#include <fcntl.h>
remote(v1d,g,bd)
unsigned char bd[];
short g[];
unsigned char v1d[];
{
        FILE *h2ptr,*h1ptr,*ffptr,*fptr;
        int ii,hh,pid;      /** loop variables ***/
        int begin,end;
        int par;
        int fdes;
```

```
int stat;
int err;
int count,lck();
int cnt,cntend;
int num;
char c,test,ans[512],str[512]; /**strings for request**/
char gstr[80];
unsigned char host[5];

/*** variable initializations
****
**** set all strings to end with a null char
***/
c=' ';
cnt=0;
ans[500]=0;
str[500]=0;
gstr[79]=0;
host[4]=0;
bd[200]=0;
v1d[200]=0;

/*****pass request to presentation layer in HOST.C
*****
****/
ffptr=fopen("/usr/mumps/mps/fil2","w+ ");
fclose(ffptr);
fdes=open("/usr/mumps/mps/fil2",O_RDWR|O_CREAT|O_TRUNC);

/**** pull off possible host name
****
****/
for (ii=1;v1d[ii]!=0;ii++)
{
        if (v1d[ii]=='/')
                break;
}
begin=ii+1;
for (ii=begin;v1d[ii]!=0;ii++)
{
        if (v1d[ii]=='/')
                break;
}
end=ii-1;
hh=0;
for (ii=begin;ii<=end;ii++)
{
        host[hh++]=v1d[ii];
}
host[hh]=0;
```

```
/**** see if request is on a remote host
*****
***** NOTE NOTE to add other hosts just add to the if
***** statement below
****/
if (( strcmp(host,"host1")==0))
{
        rtn=1;
        for (ii=0;ii<5;ii++)
        {
                write(fdes,&host[ii],1);
        }

/**** rebuild request from information gotten from global routine
****/
for (ii=1;ii<=276;ii++)
{
        if (v1d[ii]==2 )
                v1d[ii]='/';
        if (v1d[ii]==206 )
                v1d[ii]='(';
        if (v1d[ii]==207 )
                v1d[ii]=')';
        if (v1d[ii]==208 )
                v1d[ii]=',';
}

/**** pull off pathname part of request
*****/
for (ii=end+1;ii<=276;ii++)
{
        if (v1d[ii]==0)
        {
                write(fdes,&v1d[ii],1);
                break;
        }
        else
        {
                write(fdes,&v1d[ii],1);
        }
        if (ii==275)
        {
                v1d[ii]=0;
                write(fdes,&v1d[ii],1);
                break;
        }

}

/**** pull off type of request desired
****/
```

```
sprintf(gstr,"%d%c%d%c%d%c%d%c",g[0],c,g[1],c,g[2],c,g[3],c);
for (ii=0;ii<80;ii++)
{
        if (gstr[ii]==0)
        {
                write(fdes,&gstr[ii],1);
                break;
        }
        else
        {
                write(fdes,&gstr[ii],1);
        }
        if (ii==79)
        {
                gstr[ii]=0;
                write(fdes,&gstr[ii],1);
                break;
        }
}

/****pull off answer part of request
*****
****/
for (ii=1;ii<=276;ii++)
{
        if (bd[ii]==0)
        {
                write(fdes,&bd[ii],1);
                break;
        }
        else
        {
                write(fdes,&bd[ii],1);
        }
        if (ii==275)
        {
                bd[ii]=0;
                write(fdes,&bd[ii],1);
                break;
        }
}
bd[0]=10;
write(fdes,&bd[0],1);
close(fdes);

/*** check to see if another user in using line
**** line busy- count>150
****/
count=lck();
if ( count <150)
{
```

```
        /**** generate process to do remote communications
        *****
        ****/
        pid=fork();
        if (pid == 0)
        {
                err=execl("/usr/mumps/mps/host.e","/usr/mumps/mps/host.e0
                printf("execl failed0);
                count=10000;
        }
}
else
{
        printf("another user is accessing database- try later0);
}
/****
****** read in results and value of $test from remote access
*****/
signal(SIGTERM,SIG_IGN);
signal(SIGQUIT,SIG_IGN);
ffptr=fopen("/usr/mumps/mps/outMMM","w");
fclose(ffptr);
ffptr=fopen("/usr/mumps/mps/outMMM","r+ ");
c=-1;
printf("processing remote request0);

/**** wait for results to come back
***** if no results return within approx. 5min.
***** then print error message and assume link
***** is down
****/
while ((c==-1) && (count<=250))
{
        sleep(2);
        c=getc(ffptr);
        count++;
}
printf("still processing remote request0);

/**** process results
*****
****/

/**** blank out answer array before storing new
***** answer.
*****/
for (ii=0;ii<=20;ii++)
        ans[ii]=' ';
if (count<250)
{
        ans[0]=c;
```

```
        /****
        ***** separate parts of answer information
        ****/
        ii=1;
        while ((c=getc(ffptr)) != '&' && (ii<=20))
        {
                ans[ii++]=c;
        }
        test=getc(ffptr);
        for ( ii=0;ii<=20;ii++)
        {
                if (( ans[ii]>='A') && (ans[ii]<='Z'))
                        ans[ii]=ans[ii]+32;
        }
        ans[21]=0;
        printf(" 0);

        /***** release communications line
        *****/
        unlck();

        /**** blank out answer array before storing new
        ***** answer.
        *****/
        for (ii=0;ii<=200;ii++)
                bd[ii]=' ';

        /***** if command if $next or write put result in bd
        *****
        ****/
        if (g[3]==0 |g[3]==2 |g[3]==4 |g[3]==6)
        {
                strcpy(&bd[1],ans);
        }
        fclose(ffptr);

        /****return value of $test to tell if
        ***** operation was successful
        ****/
        if ( test == '1')
                return(1);
        else
                return(0);
}

/****** result did not come back within 5 min.
*****/
else
{
        printf("WARNING remote host temporarily down, try later0);
}
```

```
   }
}


/******** end remote access routine
*********
*********/


/***** routine to ensure that only one user has acces to the
****** communications line at a time- it creates a file
****** called LCK..tty00 when the line is in use and checks for
****** the existance of this file to ensure exclusive access of
****** line.
******/
int lck()
{
        FILE *fp;
        FILE *cp,*popen(),*pclose();
        char fname[26],lckname[26];
        char c;
        long int flag;
        long ii;
        lckname[0]='l';
        lckname[1]='o';
        lckname[2]='c';
        lckname[3]='k';
        /**** see if file exists-if line is locked *****/
        strcpy(fname,"                    ");
        cp=popen("grep -s lock /usr/mumps/mps/LCK..tty00","r");
        if (((c=getc(cp))!=0) && (c!=EOF))
        {
        fscanf(cp,"%s",fname);
        }
        pclose(cp);

        /***** see if file exists
        *****/
        for (ii=0;ii<=3;ii++)
        {
           if ( fname[ii] == lckname[ii])
              flag=1;
           else
              flag=0;
        }

        /*** if file does not exist, create it
        ****
        ****/
```

```
        if ( flag == 0 )
        {
                fp=fopen("/usr/mumps/mps/LCK..tty00","w+ ");
                fprintf(fp,"lock file0);
                fclose(fp);
                return(0);
        }
        else
        {
                return(10000);
        }
}

/***** routine to ensure that only one user has acces to the
****** communications line at a time- it removes lock file after
****** user has finished with it
******/
unlck()
{
        FILE *cp,*popen(),*pclose();
        /**** remove lock file *****/
        cp=popen("rm /usr/mumps/mps/LCK..tty00","r");
        pclose(cp);
}
```

APPENDIX B.

PRESENTATION LAYER ON LOCAL MACHINE

APPENDIX B. PRESENTATION LAYER ON LOCAL MACHINE

```
#include <stdio.h>
#include <fcntl.h>
#include <signal.h>
/*******************************************************/
*
* Host.c - program to receive data request from
* local database and encrypt the data request and
* pass the request to the newtork software. This
* routine is the presentation layer in the network
* communications software. This
* routine initiates the network communications
* software, in particular the datalink and physical
* layers in the routine OK.C.
*
*******************************************************/

FILE *fptr,*hptr,*f1ptr;
int err;
int fd;
char mat[56][56]= {
"#$%&'()*+ ,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ",
"$%&'()*+ ,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#",
"%&'()*+ ,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$",
"&'()*+ ,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%",
"'()*+ ,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&",
"()*+ ,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'",
")*+ ,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'(",
"*+ ,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()",
"+ ,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*",
",-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ",
"-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,",
"./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-",
"/0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-.",
"0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./",
"123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0",
"23456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./01",
"3456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./012",
"456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123",
"56789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./01234",
"6789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./012345",
"789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456",
"89:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./01234567",
"9:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./012345678",
":;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456789",
";<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456789:",
"<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;",
"=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<",
">?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=",
```

```
"? @ ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>",
"@ ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>? ",
"ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>? @ ",
"BCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>? @ A",
"CDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>? @ AB",
"DEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>? @ ABC",
"EFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>? @ ABCD",
"FGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>? @ ABCDE",
"GHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>? @ ABCDEF",
"HIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>? @ ABCDEFG",
"IJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>? @ ABCDEFGH",
"JKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>? @ ABCDEFGHI",
"KLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>? @ ABCDEFGHIJ",
"LMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>? @ ABCDEFGHIJK",
"MNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>? @ ABCDEFGHIJKL",
"NOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>? @ ABCDEFGHIJKLM",
"OPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>? @ ABCDEFGHIJKLMN",
"PQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>? @ ABCDEFGHIJKLMNO",
"QRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>? @ ABCDEFGHIJKLMNOP",
"RSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>? @ ABCDEFGHIJKLMNOPQ",
"STUVWXYZ#$%&'()*+ ,-./0123456780:;<=>? @ ABCDEFGHIJKLMNOPQR",
"TUVWXYZ#$%&'()*+ ,-./0123456780:;<=>? @ ABCDEFGHIJKLMNOPQRS",
"UVWXYZ#$%&'()*+ ,-./0123456780:;<=>? @ ABCDEFGHIJKLMNOPQRST",
"VWXYZ#$%&'()*+ ,-./0123456780:;<=>? @ ABCDEFGHIJKLMNOPQRSTU",
"WXYZ#$%&'()*+ ,-./0123456780:;<=>? @ ABCDEFGHIJKLMNOPQRSTUV",
"XYZ#$%&'()*+ ,-./0123456780:;<=>? @ ABCDEFGHIJKLMNOPQRSTUVW",
"YZ#$%&'()*+ ,-./0123456780:;<=>? @ ABCDEFGHIJKLMNOPQRSTUVWX",
"Z#$%&'()*+ ,-./0123456780:;<=>? @ ABCDEFGHIJKLMNOPQRSTUVWXY",
};
main()
{
        char key[13];       /* key for Vigenere Cipher*/
        char tmsg[71],crptmsg[71]; /* strings encrypted*/
        char mmsg[71],ccmsg[71]; /*messages          */
        int mlen,klen;     /*mlen,klen-length of key and*/
                           /* encrypted message      */
        int pid,ii;         /* process id and loop var  */
        int err;
        char ss[80];        /* strings to hold data    */
        char host[5];        /* request               */
        char bd[257];
        short g[4];
        int dd,tpid;
        char c,v1d[257];
        char c1,c2;
        int cnt;
        char ans[512];
        FILE *pptr,*popen();
        int fdes;
        FILE *ptr;
        char msg[400];
```

```
/**** initialize all strings to have null
***** character at the end and blank out
***** strings to be encrypted
****/
tmsg[70]=0;
crptmsg[70]=0;
fmsg[70]=0;
mmsg[70]=0;
ccmsg[70]=0;
ss[79]=0;
host[4]=0;
ss[79]=0;
msg[399]=0;
bd[256]=0;
v1d[256]=0;
strcpy(mmsg,"                                                    ");
strcpy(crptmsg,"                                                 ");
strcpy(tmsg,"                                                    ");
strcpy(ccmsg,"                                                   ");
hptr=fopen("/usr/mumps/mps/fil2","r+ ");
dd=0;

/***** read in data base request
******
*****/
for (ii=0;ii<5;ii++)
{
        host[ii]=getc(hptr);
        msg[dd++]=host[ii];
}
c=' ';
for (ii=1;c!=0;ii++)
{
        c=getc(hptr);
        v1d[ii]=c;
        msg[dd++]=c;
}
c=' ';
for (ii=0;c!=0;ii++)
{
        c=getc(hptr);
        ss[ii]=c;
        msg[dd++]=c;
}
sscanf(ss,"%d%c%d%c%d%c%d%c",&g[0],&c,&g[1],&c1,&g[2],&c2,&g[3],&c);
c=' ';
for (ii=1;c!=0;ii++)
{
        c=getc(hptr);
        bd[ii]=c;
```

```
        msg[dd++]=c;
}
msg[dd]=0;

/**** code to break up request into manageable pieces to
***** send across network.
****/
f1ptr=fopen("/usr/mumps/mps/fil1","w");
sleep(5);
pieces(mmsg,msg,dd);
fclose(f1ptr);

/***** initiate datalink and physical layers for data
****** transfer.
*****/
tpid=fork();
if ( tpid !=0)
{
        err= execl("/usr/mumps/mps/oka.e","/usr/mumps/mps/oka.e",0);
        printf("err= %d0);
}

/**** wait on answer to return across communications lines
*****
*****/
signal(SIGQUIT,SIG_IGN);
hptr=fopen("/usr/mumps/mps/fil3","w");
fclose(hptr);
hptr=fopen("/usr/mumps/mps/fil3","r");
c=-1;
while ( c <1 )
{
        sleep(2);
        c=getc(hptr);
}
mmsg[0]=c;
for (ii=1;ii<72;ii++)
{
        c=getc(hptr);
        mmsg[ii]=c;
}
mmsg[72]=0;
fclose(hptr);

/***** decrypt message
******
*****/
mlen=51;
klen=13;
strcpy(key,"COOKIEMONSTER");
defence(ccmsg,mmsg);
```

```
        decrpt(tmsg,key,ccmsg);

        for (ii=0;ii<dd;ii++)
        {
                if ( tmsg[ii]==0 )
                        tmsg[ii]='+';
                if ( tmsg[ii]==' ')
                        tmsg[ii]='&';
        }

        /**** pass answer to applications program  data base
        *****
        *****/
        fptr=fopen("/usr/mumps/mps/outMMM","w");
        sleep(5);
        fprintf(fptr,"%s0,tmsg);
        fclose(fptr);

        /**** signal ok.e to terminate
        *****
        *****/
        signal(SIGQUIT,SIG_IGN);
        kill(0,SIGQUIT);
        sleep(5);
}

/***** break up message into pieces
******
*****/
pieces(mmsg,msg,dd)
char msg[],mmsg[];
int dd;
{
        int msgcnt,ii,cnt,num;
        num=0;
        cnt=0;
        if ((dd > 350 ) && ( dd<400))
        {
                strcpy(mmsg,"headerblock8blocks                                    ");
                preparehd(mmsg,-1);
                for (ii=0;ii<51;ii++)
                        mmsg[ii]=msg[cnt++];
                prepare(mmsg,0);
                for (ii=0;ii<51;ii++)
                        mmsg[ii]=msg[cnt++];
                prepare(mmsg,1);
                for (ii=0;ii<51;ii++)
                        mmsg[ii]=msg[cnt++];
                prepare(mmsg,2);
                for (ii=0;ii<51;ii++)
                        mmsg[ii]=msg[cnt++];
```

```
        prepare(mmsg,3);
        for (ii=0;ii<51;ii++)
                mmsg[ii]=msg[cnt++];
        prepare(mmsg,4);
        for (ii=0;ii<51;ii++)
                mmsg[ii]=msg[cnt++];
        prepare(mmsg,5);
        for (ii=0;ii<51;ii++)
                mmsg[ii]=msg[cnt++];
        prepare(mmsg,6);
        for (ii=0;ii<51;ii++)
                mmsg[ii]=msg[cnt++];
        prepare(mmsg,7);
}
else if ( dd>300)
{
        strcpy(mmsg,"headerblock7blocks                                ");
        preparehd(mmsg,-1);
        for (ii=0;ii<51;ii++)
                mmsg[ii]=msg[cnt++];
        prepare(mmsg,0);
        for (ii=0;ii<51;ii++)
                mmsg[ii]=msg[cnt++];
        prepare(mmsg,1);
        for (ii=0;ii<51;ii++)
                mmsg[ii]=msg[cnt++];
        prepare(mmsg,2);
        for (ii=0;ii<51;ii++)
                mmsg[ii]=msg[cnt++];
        prepare(mmsg,3);
        for (ii=0;ii<51;ii++)
                mmsg[ii]=msg[cnt++];
        prepare(mmsg,4);
        for (ii=0;ii<51;ii++)
                mmsg[ii]=msg[cnt++];
        prepare(mmsg,5);
        for (ii=0;ii<51;ii++)
                mmsg[ii]=msg[cnt++];
        prepare(mmsg,6);
}
else if ( dd > 250)
{
        strcpy(mmsg,"headerblock6blocks                                ");
        preparehd(mmsg,-1);
        for (ii=0;ii<51;ii++)
                mmsg[ii]=msg[cnt++];
        prepare(mmsg,0);
        for (ii=0;ii<51;ii++)
                mmsg[ii]=msg[cnt++];
        prepare(mmsg,1);
        for (ii=0;ii<51;ii++)
```

```
                mmsg[ii]=msg[cnt++];
        prepare(mmsg,2);
        for (ii=0;ii<51;ii++)
                mmsg[ii]=msg[cnt++];
        prepare(mmsg,3);
        for (ii=0;ii<51;ii++)
                mmsg[ii]=msg[cnt++];
        prepare(mmsg,4);
        for (ii=0;ii<51;ii++)
                mmsg[ii]=msg[cnt++];
        prepare(mmsg,5);
}
else if ( dd > 200)
{
        strcpy(mmsg,"headerblock5blocks                                    ");
        preparehd(mmsg,-1);
        for (ii=0;ii<51;ii++)
                mmsg[ii]=msg[cnt++];
        prepare(mmsg,0);
        for (ii=0;ii<51;ii++)
                mmsg[ii]=msg[cnt++];
        prepare(mmsg,1);
        for (ii=0;ii<51;ii++)
                mmsg[ii]=msg[cnt++];
        prepare(mmsg,2);
        for (ii=0;ii<51;ii++)
                mmsg[ii]=msg[cnt++];
        prepare(mmsg,3);
        for (ii=0;ii<51;ii++)
                mmsg[ii]=msg[cnt++];
        prepare(mmsg,4);
}
else if ( dd > 150)
{
        strcpy(mmsg,"headerblock4blocks                                    ");
        preparehd(mmsg,-1);
        for (ii=0;ii<51;ii++)
                mmsg[ii]=msg[cnt++];
        prepare(mmsg,0);
        for (ii=0;ii<51;ii++)
                mmsg[ii]=msg[cnt++];
        prepare(mmsg,1);
        for (ii=0;ii<51;ii++)
                mmsg[ii]=msg[cnt++];
        prepare(mmsg,2);
        for (ii=0;ii<51;ii++)
                mmsg[ii]=msg[cnt++];
        prepare(mmsg,3);
}
else if ( dd > 100)
{
```

```
                strcpy(mmsg,"headerblock3blocks                                 ");
                preparehd(mmsg,-1);
                for (ii=0;ii<51;ii++)
                        mmsg[ii]=msg[cnt++];
                prepare(mmsg,0);
                for (ii=0;ii<51;ii++)
                        mmsg[ii]=msg[cnt++];
                prepare(mmsg,1);
                for (ii=0;ii<51;ii++)
                        mmsg[ii]=msg[cnt++];
                prepare(mmsg,2);
        }
        else if ( dd > 50)
        {
                strcpy(mmsg,"headerblock2blocks                                 ");
                preparehd(mmsg,-1);
                for (ii=0;ii<51;ii++)
                        mmsg[ii]=msg[cnt++];
                prepare(mmsg,0);
                for (ii=0;ii<51;ii++)
                        mmsg[ii]=msg[cnt++];
                prepare(mmsg,1);
        }
        else
        {
                strcpy(mmsg,"headerblock1blocks                                 ");
                preparehd(mmsg,-1);
                for (ii=0;ii<51;ii++)
                        mmsg[ii]=msg[cnt++];
                mmsg[ii]=0;
                prepare(mmsg,0);
        }

}


/***** encrypt message and pass it to data link layer
******
*****/
prepare(mmsg,num)
char mmsg[];
int num;
{
        int num,mlen,klen;
        int ii;
        char crptmsg[71],tmsg[71],ccmsg[71];
        char key[13];
        crptmsg[70]=0;
        fmsg[70]=0;
        tmsg[70]=0;
```

```
        ccmsg[70]=0;
        strcpy(crptmsg,"                                                                    ");
        strcpy(tmsg,"                                                                    ");
        strcpy(ccmsg,"                                                                    ");
        mlen=51;
        klen=13;
        strcpy(key,"COOKIEMONSTER");
        /*** code to build entire message into one string to be encrypted
        **** is included in input loops- eg. mmsg[dd++]=c
        *****/
        for (ii=0;ii<51;ii++)
        {
                if ( mmsg[ii]==0 )
                        mmsg[ii]='+';
                if ( mmsg[ii]==' ')
                        mmsg[ii]='&';
                /****  convert to capital letters
                *****
                *****/
                if ( mmsg[ii]>=97 )
                        mmsg[ii]=mmsg[ii]-32;
        }

        mlen=51;
        klen=13;
        mmsg[0]='R';
        crptmsg[50]=num+60;
        crpt(mmsg,key,ccmsg);
        fence(ccmsg,crptmsg);
        crptmsg[60]=0;
        fprintf(f1ptr,"%s0,crptmsg);
}

/**** prepare and send over headerblock that tells
***** length of this message
****/
preparehd(mmsg,num)
char mmsg[];
int num;
{
        int ii;
        for (ii=50;ii<61;ii++)
                mmsg[ii]=0;
        for (ii=0;ii<61;ii++)
        {
                if ( mmsg[ii]==0 )
                        mmsg[ii]='+';
                if ( mmsg[ii]==' ')
                        mmsg[ii]='&';
                /****  convert to capital letters
                *****
```

```
            *****/
            if ( mmsg[ii]>=97 )
                  mmsg[ii]=mmsg[ii]-32;
      }
      mmsg[60]=0;
      fprintf(f1ptr,"%s0,mmsg);

}



/*****
****** encrypts a message passed to it by the rail-fence cipher.
****** this cipher is a transposition cipher where characters
****** in the data are rearranged.
****** The letters of a plaintext message are written down in a
****** pattern resembling a rail fence, and then removed by rows.
****** The key to the cipher is given by the depth of the fence.
****** The depth for this program is 3.
******
******/
fence(msg,cmsg)
char msg[70],cmsg[70];
{
      char ary1[20],ary2[36],ary3[18];
      int ii,jj,kk,indx;
      indx=0;
      ii=0;
      jj=0;
      for ( kk=0;kk<=12;kk++ )
      {

          if ( ii<=50)
          {
             ary1[indx]=msg[ii];
          }
          if ( ii+1<=50)
          {
             ary2[jj++]=msg[ii+1];
          }
          if ( ii+2<=50)
          {
             ary3[indx]=msg[ii+2];
          }
          if ( ii+3<=50)
          {
             ary2[jj++]=msg[ii+3];
          }
             ii=ii+4;
             ++indx;
```

```
        }
        ary1[13]=0;
        ary2[26]=0;
        ary3[13]=0;
        ii=0;
        indx=0;
        while (ary1[indx]!=0)
                cmsg[ii++]=ary1[indx++];
        indx=0;
        while (ary2[indx]!=0)
                cmsg[ii++]=ary2[indx++];
        indx=0;
        while (ary3[indx]!=0)
                cmsg[ii++]=ary3[indx++];
}


/***** decrypts a message encrypted by the rail-fence routine
****** with a fence width of 3.
******/
defence(msg,cmsg)
char msg[71],cmsg[71];
{
        char ary1[20],ary2[40],ary3[20];
        int ii,jj,kk,indx;
        indx=0;
        for ( ii=0;ii<=12;ii++)
                ary1[ii]=cmsg[indx++];
        for ( ii=0;ii<=25;ii++)
                ary2[ii]=cmsg[indx++];
        for ( ii=0;ii<=12;ii++)
                ary3[ii]=cmsg[indx++];

        ary1[13]=0;
        ary2[26]=0;
        ary3[13]=0;
        indx=0;
        ii=0;
        jj=0;
        for ( kk=0;kk<=17;kk++)
        {
            if (( ii<=50) && ( ary1[indx]!=0))
                msg[ii]=ary1[indx];
            if (( ii+1<=50) && ( ary2[jj]!=0))
                msg[ii+1]=ary2[jj++];
            if (( ii+2<=50) && ( ary3[indx]!=0))
                msg[ii+2]=ary3[indx];
            if (( ii+3<=50) && ( ary2[jj]!=0))
                msg[ii+3]=ary2[jj++];
                ii=ii+4;
```

```
            ++indx;
        }
        msg[71]=0;
}


/***** routine to encrypt messages using the Vigenere Cipher
****** to encrypt- plaintext a and key letter k, the ciphertext c
****** is the letter in col a of row k
****** Note: Ideas for this algorithm gotten from CRYPTOGRAPHY AND
****** DATA SECURITY by Denning -Ref 3
*****/
crpt(msg,key,crptmsg)
char msg[71],key[13],crptmsg[71];
{
        int kk,ii,row,col;
        char tmp;
        int mlen,klen;
        mlen=51;
        klen=13;
        kk=(-1);
        for ( ii=0;ii<=mlen-1;ii++)
        {
          if ( kk==klen-1)
               kk=0;
          else
               kk++;
          row=key[kk]-35;
          col=msg[ii]-35;
          tmp=mat[row][col];
          crptmsg[ii]=tmp;
        }
        msg[65]=0;
        crptmsg[65]=0;
}


/***** routine to decrypt a message that has been encoded using a
****** vegenere Cipher.
****** to decrypt- for ciphertext c, the plaintext a is the col
****** containing c in row k.
******/
decrpt(msg,key,crptmsg)
char msg[],key[],crptmsg[];
{
        int ii,kk,jj,row,col;
        int mlen,klen;
        char tmp;
```

```
        kk=(-1);
        mlen=51;
        klen=13;
        for ( ii=0;ii<=mlen-1;ii++)
        {
          if ( kk==klen-1)
               kk=0;
          else
               kk++;
          row=key[kk]-35;
          for (jj=0;jj<=56;jj++)
               if (mat[row][jj]==(crptmsg[ii]))
               {
                    msg[ii]=jj+35;
               }
        }
}
```

APPENDIX C.

DATA LINK AND PHYSICAL LAYERS

APPENDIX C. DATA LINK AND PHYSICAL LAYERS

```
/*****************************************************
* OK.C-This routine handles the physical and data link layers
* It reads in an encrypted request broken up into blocks
* of 50 bytes long and sends the request over to a remote
* machine. This program initiates communication with
* the remote machine, sends the data request over, waits
* for a response and then breaks the connection when it
* receives a signal from the application layer
* NOTE Physical layer in login() was derived from algorithm
* given by the UNIX Software Development Manual in the
* IBM PC-AT UNIX Manuals.
* Prot() go back n strategy algorithm was derived from
* algorithm in ref 17 by Tenenbaum.
*****************************************************/

/**** NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE
**** NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE
**** NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE
**** PRIM is 1 if this machine initiates communication
**** PRIM = 0 if this machine do not inititate communication
****/
#define PRIM 1
#include <fcntl.h>
#include <stdio.h>
#include <sys/ioctl.h>
#include <termio.h>
#include <sys/ulimit.h>
#include <unlstd.h>
#include <signal.h>
#include <errno.h>
#define HOSTREADY 2
#define FRAMEARRIVAL 3
#define CKSUMERR 4
#define TIMEOUT 5
#define DATA 6
#define ACK 7
#define NONE 8
int msgcount,sstart,startmps,signalstart,blknum,err;
int ptr,hhptr;
int sum,cksum(),cksum1(),arrive,time,timecnt;
FILE *fdopen(),*fp,*fpr,*fpw;
char frbuff[80];
int nxtframesend,ackexpected,frameexpected,nbuffered;
int enablehost;
char hbuff[70],buff[8][70],tohbuff[70],trbuff[80],twbuff[70],hd[80];
int notdone,ev,maxseq;
struct termio *argp,tt;   /** new unix terminal interface ***/
char rbuff1[80];
```

```
int term_]flag;
unsigned char term_cc4,term_cc5;
int open(),close(),fd,hhptr;
#define RD 0
#define WR 1
static int pid;
int chan[2];
int parent;
int pipptr,pipptw,piptrw;
FILE *pppptr,*dumptr;
main()
{
        int die();
        char ppid[5];
        char parid[5];
        int verifysum();
        signal(SIGQUIT,die);
#if PRIM
        login();
#else
        printf("readytoreceivedata                    ");
#endif
         /*** initialize all strings to have a NULL on the end
         ****/
        ppid[4]=0;
        parid[4]=0;
        hbuff[69]=0;
        tohbuff[69]=0;
        trbuff[79]=0;
        twbuff[69]=0;
        hd[79]=0;
        frbuff[79]=0;
        hd[79]=0;
         /*** oen a piipe for process communication
         ***/
        if (pipe(chan)<0)
                printf("   error in opening pipe0);

         /****open files for process communication
         *****/
        pppptr=fopen("/usr/mumps/mps/pipesMMM","w");
        fclose(pppptr);
        pipptr=open("/usr/mumps/mps/pipesMMM",O_RD ONLY);
#if PRIM

        pipptw=open("/usr/mumps/mps/pipesMMM",O_WRONLY);
        hhptr=open("/usr/mumps/mps/fil1",O_RD ONLY);
#else
        piptrw=open("/usr/mumps/mps/pipesMMM",O_WRONLY);
        hhptr=open("/usr/mumps/mps/fil1",O_RD ONLY);
#endif
```

```
        /**** spawn a process a read data from the remote system
        ***** and have the other process operate on incoming
        ****  data from the other system and send data to
        ****  another system
        ****/
       pid = fork();
       if (pid!=0)
       {
             read(chan[RD],parid,5);
             sscanf(parid,"%d",&parent);

#if PRIM
             rda();
#else
             rdb();
#endif
       }
       else
       {
             /*****
             ****** pass pid of parent to child to allow child
             ****** to send a signal to parent
             *****/
             sprintf(ppid,"%d",pid);
             write(chan[WR],ppid,5);
#if PRIM
             prot();
#else
             ptr=open("/usr/mumps/mps/fil3",O_WRONLY);
             signal(SIGALRM,SIG_IGN);
             signal(SIGTERM,SIG_IGN);
             prot();
#endif

       }
}

/*****
******function to terminate program when given a delete char
******/
die()
{
       signal(SIGTERM,SIG_IGN);
       signal(SIGTERM,die);
       exit(1);
}


/*****
```

```
****** reader for secondary process in network communications
****** read in data from the remote system
*****/
rdb()
{
      int ii,nchars;
        for (ii=1;ii<=100;ii++)
        {
            gets(frbuff,80);
            if (( frbuff[1] != 0) && (frbuff[1] != ' ')&&(frbuff[0]!='S'))
            {
                  sleep(7);
                  if (verifysum()!=1 )
                        frbuff[0]='C';
                  nchars=write(piptrw,frbuff,80);
            }
        }
}


/*****
****** compares checksum passed across lines with checksum just
****** just generated
****** sends over CKSUMERR if error, else sends over block
*****/
int verifysum()
{
      int aframenu,aacknu,oldsum,sm;
      char temp,atype,abuff[70],tmp;
      frbuff[69]=0;
      abuff[69]=0;
      sscanf(frbuff,"%c%d%c%d%c%d%c%s",&temp,&aframenu,
                  &atype,&aacknu,&tmp,&oldsum,&tmp,abuff);
      if (atype != 'A')
      {
        sm=cksum(abuff);
        if (sm==oldsum)
            return(1);
        else
            return(0);
      }
      else
        return(1);
}


/*****
******reader for primary process in network communications
*****/
rda()
{
```

```
        int nchars;
        int ii;
          for (ii=1;ii<=10000;ii++)
          {
              fgets(frbuff,80,fpr);
              if (( frbuff[1] != 0) && (frbuff[1] != ' ')&&(frbuff[0]!='P'))
              {
                    sleep(7);
                    if (verifysum()!=1 )
                          frbuff[0]='C';
                    nchars=write(pipptw,frbuff,80);
              }
          }
}


/*****
****** initial process for primary network routine to establish
****** communications between 2 machines
*****/
int login()
{
        int ii;
        /* to set parameters on terminal *****/
        /*** get address terminal interface. **************/
        argp=( &tt);

        /*** open tty00 for communication ***/
        fd=open("/dev/tty00",O_RDWR|O_NDELAY);
        if ( fd == -1 )
             printf("       error in opening file fd0);

        /**** set up terminal parameters  *****/
        /**** store current values to restore later
        *****/
        ioctl(fd,TCGETA,argp);
        term_lflag=tt.c_lflag;
        term_cc4=tt.c_cc[4];
        term_cc5=tt.c_cc[5];

        /**** set terminal input to map CR to NL on input
        ***** strip a character
        ***** signal interrupt on break
        ****/
        tt.c_iflag |= ICRNL|BRKINT|ISTRIP;
        tt.c_iflag &= ~INPCK;

        /**** set output buffer to map NL to CR-NL on output
        ***** select CR,TAB,BS,VT,FF,NL delays
        ****/
        tt.c_oflag |= ONLCR|CR0|TAB0|BS0|VT0|FF0|NL0;
```

```
        /**** set hardware control of terminal to
        ***** to 1200 baud,for a direct connect local line
        ***** to hang up on last close and eight bit character size
        ****/
        tt.c_cflag &= ~CBAUD;
        tt.c_cflag |= CLOCAL|HUPCL|B1200|CS8;

        /**** set line discipline to control terminal functions
        ***** disable echo,enable erase char and NL after killchar
        ****/
        tt.c_lflag &= ~ECHO;
        tt.c_lflag |= ECHOE|ECHOK;


        /**** define special control characters
        ****/
        /****** VMIN = 4 VTIME = 5 ****************/
        tt.c_cc[VMIN]=1;
        tt.c_cc[VTIME]=0;
        ioctl(fd,TCSETA,argp);

        /**** open device for both reading and writing *******/
        fcntl(fd,F_SETFL,fcntl(fd,F_GETFL,0)&~O_NDELAY);
        fpr=fopen("/dev/tty00","r");
        fpw=fopen("/dev/tty00","w");

        /*** want unbuffered I/O ************/
        setbuf(fpw,0);

        /**** send over username and password to login *****/
        sleep(2);
        fprintf(fpw,"mumps                ");
        sleep(5);
        fgets(rbuff1,80,fpr);
        fprintf(fpw,"mumps                ");
        for (ii=1;ii<=50;ii++)
        {
              if ( fgets(rbuff1,80,fpr) != NULL )
                  if(rbuff1[1]!=0)
                     if (strncmp(rbuff1,"readytoreceivedata",18)==0)
                         goto here;
        }
here:   return(fd);

}


/**** procedure to set up protocol between at least 2 machines
```

```
***** in a network
****/

prot()
{
        int ii,cnt,sendack;
        int nchar;
        char temp,atype;
        int aframenu,aacknu;
        int timeout();
        char ppid[5];
        char tmp;
        int oldsum;
#if PRIM
        FILE *ptr,*fopen();
#else
        sleep(25);
#endif
        /**** variable initializations
        ****/
        maxseq=15;
        signal(SIGALRM,timeout);
        nxtframesend = 0;
        ackexpected = 0;
        frameexpected = 0;
        nbuffered = 0;
        notdone=1;
        cnt=0;
        sendack=0;

#if PRIM
        timecnt=0;
#else
        sstart=1;
        strcpy(twbuff,"                                                    ");
        strcpy(hd,"                                                    ");

#endif

        for(;;)
        {
#if PRIM
          wait();
#else
          wait();

           /***** see if request is ready to be passed to the
           ***** presentation layer for decryption and reassembling
           *****/
          if ( startmps==1)
          {
```

```
            startmps=0;
            signalstart=0;
            fflush(ptr);
            close(ptr);
            pid=fork();
            if (pid ^=0)
            {
              signal(SIGALRM,SIG_IGN);
              signal(SIGTERM,SIG_IGN);
              sleep(10);
              err=execl("/usr/mumps/mps/mpsex.e",
                        "/usr/mumps/mps/mpsex.e",0);
              printf("  execl failed0);
            }
        }
#endif

        /***** check to see whether to send a message or to
        ****** process a received message
        *****/
       switch (ev)
       {

          /******* ready to send a message
          ********/
          case HOSTREADY:
            ev=NONE;
            hbuff[69]=0;
            sum=cksum1(hbuff);
            strcpy(buff[nxtframesend],hbuff);
            nbuffered++;
            senddata('D',nxtframesend);
            nxtframesend++;
            sendack=0;
            sleep(4);
            break;

          /***** ready to receive a message
          ******/
          case FRAMEARRIVAL:

            tohbuff[56]=0;
            trbuff[79]=0;
            ev=NONE;
            sscanf(trbuff,"%c%d%c%d%c%d%c%s",&temp,&aframenu,
                    &atype,&aacknu,&tmp,&oldsum,&tmp,tohbuff);
            tohbuff[56]=0;
#if PRIM
            ptr=fopen("/usr/mumps/mps/out2MMM","w");
            fprintf(ptr," tr=%s0,trbuff);
            fclose(ptr);
```

```
#endif
                /**** see if the frame contains data or is just an
                ***** acknowledgment frame
                *****/
                if ( atype== 'D')
                {
                  if ( aframenu == frameexpected )
                  {
                    tohost();
                    frameexpected++;
                    sendack=1;
                  }
                }
                printf("      0);

                 /**** see if the frame received is the one expected
                 *****/
                while ((between(ackexpected,aacknu,nxtframesend))==1)
                {
                    nbuffered--;
                    timecnt=0;
                    ackexpected++;
                }
                break;

              /**** process error in checksum for block
              *****
              ****/
              case CKSUMERR:
              /**** ignore bad frames
              ****/
                ev=NONE;
                break;


              /*****
              ****** resend messages not acknowledged within
                ****** a specified time period
              *****/
              case TIMEOUT:
                ev=NONE;
                nxtframesend=ackexpected;
                for (ii=1;ii<=nbuffered;ii++)
                {
                    senddata('D',nxtframesend);
                    nxtframesend++;
                }
                break;

               /**** nothing to process this time through loop
```

```
            *****
            *****/
          case NONE:
                ev=NONE;
                break;

            /***** default to nothing if above are not specified
            ******
            *****/
          default:
            break;
        } /*** end switch ****/

        /***** send acknow for frame if no data needs to be
        *****  sent over along with the ack
        ******/
        cnt++;
        if ( cnt>25)
        {
            cnt=0;
            if (sendack==1)
            {
              sendack=0;
              senddata('A',nxtframesend);
            }
        }

        /*****see if any frame have not been acknowledged
        ****** recently
        *****/
        timecnt++;
        if ( timecnt>100000)
        {
            ev=TIMEOUT;
            timecnt=0;
        }

        /*****allow host to send messages as long as buffers with
        ****** outstanding messages are not full
        *****/
        if ( nbuffered < maxseq)
            enablehost=1;
        else
            enablehost=0;

    } /*** end endless loop ***/
}

/*****
****** routine to see if frame received is the one expected
*****/
```

```
between(a,b,c)
{
      int a,b,c,btwn;
      if (((a<=b)&&(b<c))||((c<a)&&(a<=b))||((b<c)&&(c<a)))
            btwn=1;
      else
            btwn=0;
      return(btwn);
}

/*****
******  waits for event to be set by outside action
*****/
wait()
{
      int flag,nchar;
      char tmp[80];
            sleep(1);

              /**** check to see if a frame has arrived
              ***** from the remote host
              *****/
            nchar=read(pipptr,trbuff,80);
            if (nchar >0)
            {
                  trbuff[79]=0;
                  sscanf(trbuff,"%c%s",&flag,tmp);
                  if ( flag=='C')
                        ev=CKSUMERR;
                  else
                        ev=FRAMEARRIVAL;
            }
            else
            {

              /***** see if the host is ready to send over more
              ****** data
              *****/
                  nchar=read(hhptr,hbuff,61);
                  if (nchar>0)
                  {
                        hbuff[61]=0;
                        ev=HOSTREADY;

                  }
            }

}

/****
```

```
***** pass data received up to presentation layer
****/
tohost()
{
#if PRIM
        int ii,pid,err;
        FILE *ptr;
        ptr=fopen("/usr/mumps/mps/fil3","w");
        for ( ii=0;ii<70;ii++)
        {
                fputc(tohbuff[ii],ptr);
        }
        fclose(ptr);
#else
        int ii,err;
        tohbuff[60]=10;
        tohbuff[61]=0;
        for ( ii=0;ii<61;ii++)
                err=write(ptr,&tohbuff[ii],1);
        printf(" th=%s ",err);
        if (sstart==1)
        {
                sstart=0;
                signalstart=0;
                msgcount=tohbuff[11]-48;
        }
        if (signalstart>=msgcount)
        {
                startmps=1;
                signalstart=1;
        }
        signalstart++;
#endif

}


/*****
****** stop timeout timer when ack is received
*****/
stoptimer()
{
        timecnt=0;

}

/*****
******send data over to remote machine
*****/
senddata(type,framenu)
int framenu;
```

```
char type;
{
   char prim;
   char secnd,tmp;
   int acknu;
   prim='P';
   secnd='S';
   tmp=' ';
   hd[0]=' ';
   hd[1]=' ';
   hd[2]=' ';

   /**** prepare data frame
   *****/
   if ( type == 'D')
   {
       strcpy(twbuff,buff[framenu]);
       starttimer(framenu);
   }
   else
       type='A';
   twbuff[55]=0;
   acknu=(frameexpected+ maxseq)%(maxseq+ 1);
#if PRIM
   sprintf(hd,"%c%d%c%d%c%d%c%s",prim,framenu,type,acknu,tmp,sum,tmp,
            twbuff); ",hd);
       fprintf(fpw,"%s
#else
   sprintf(hd,"%c%d%c%d%c%d%c%s",secnd,framenu,type,acknu,tmp,sum,tmp,
            twbuff);
       printf("%s
#endif

   /**** blank out array after printing
   *****
   ******/
    strcpy(twbuff,"                                                    ");
}

/*****
****** start timer to wait for ack for frame
*****/
starttimer(frnu)
int frnu;
{
timecnt=0;
}

/*****
******retransmit frame that was not received correctly
*****/
```

```
timeout()
{
        signal(SIGALRM,SIG_IGN);
        time=1;
        signal(SIGALRM,timeout);
}

/**** checksum routine
**** compute checksum of block to be transferred
****/
int cksum(buffer)
char buffer[70];
{
        FILE *popen(),*pp,*fpck,*fopen();
        char cmd[24];
        int checksum;
        int blkcnt;
        buffer[69]=0;
        fpck=fopen("/usr/mumps/mps/file1","w+ ");
        fprintf(fpck,"%s",buffer);
        fclose(fpck);
        strcpy(cmd,"sum /usr/mumps/mps/file1");
        pp=popen(cmd,"r");
        fscanf(pp,"%d%d",&checksum,&blkcnt);
        fclose(pp);
        strcpy(cmd,"rm /usr/mumps/mps/file1");
        pp=popen(cmd,"w");
        fclose(pp);
        return(checksum);
}

/**** checksum routine
**** compute checksum of block to be transferred
****/
int cksum1(buffer)
char buffer[70];
{
        FILE *popen(),*ppp,*fpck1,*fopen();
        char cmd[24];
        int checksum;
        int blkcnt;
        fpck1=fopen("/usr/mumps/mps/file1M","w+ ");
        fprintf(fpck1,"%s",buffer);
        fclose(fpck1);
        strcpy(cmd,"sum /usr/mumps/mps/file1M");
        ppp=popen(cmd,"r");
        fscanf(ppp,"%d%d",&checksum,&blkcnt);
        fclose(ppp);
        strcpy(cmd,"rm /usr/mumps/mps/file1M");
        ppp=popen(cmd,"w");
        fclose(ppp);
```

```
        return(checksum);
}
```

APPENDIX D.

PRESENTATION LAYER ON REMOTE

APPENDIX D. PRESENTATION LAYER ON REMOTE

```
/*****************************************************
* MPSEX.C - program to decrypt and reassemble data
* base request and execute request and pass result
* base to communications software datalink layer in
* ok.c to send back across network to requestor
*****************************************************/

#include <stdio.h>
#include <signal.h>
#include <fcntl.h>
FILE *fptr;
int err;
/***** matrix for Vigenere scheme
******
******/
char mat[56][56]= {
"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ",
"$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#",
"%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$",
"&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%",
"'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&",
"()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'",
")*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'(",
"*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()",
"+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*",
",-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+",
"-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+,",
"./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+,-",
"/0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+,-.",
"0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+,-./",
"123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+,-./0",
"23456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+,-./01",
"3456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+,-./012",
"456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+,-./0123",
"56789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+,-./01234",
"6789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+,-./012345",
"789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+,-./0123456",
"89:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+,-./01234567",
"9:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+,-./012345678",
":;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+,-./0123456789",
";<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+,-./0123456789:",
"<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+,-./0123456780:;",
"=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+,-./0123456780:;<",
">?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+,-./0123456780:;<=",
"?@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+,-./0123456780:;<=>",
"@ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+,-./0123456780:;<=>?",
"ABCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+,-./0123456780:;<=>?@",
"BCDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+,-./0123456780:;<=>?@A",
```

```
"CDEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>?@AB",
"DEFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>?@ABC",
"EFGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>?@ABCD",
"FGHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>?@ABCDE",
"GHIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>?@ABCDEF",
"HIJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>?@ABCDEFG",
"IJKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>?@ABCDEFGH",
"JKLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>?@ABCDEFGHI",
"KLMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>?@ABCDEFGHIJ",
"LMNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>?@ABCDEFGHIJK",
"MNOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>?@ABCDEFGHIJKL",
"NOPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>?@ABCDEFGHIJKLM",
"OPQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>?@ABCDEFGHIJKLMN",
"PQRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>?@ABCDEFGHIJKLMNO",
"QRSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>?@ABCDEFGHIJKLMNOP",
"RSTUVWXYZ#$%&'()*+ ,-./0123456780:;<=>?@ABCDEFGHIJKLMNOPQ",
"STUVWXYZ#$%&'()*+ ,-./0123456780:;<=>?@ABCDEFGHIJKLMNOPQR",
"TUVWXYZ#$%&'()*+ ,-./0123456780:;<=>?@ABCDEFGHIJKLMNOPQRS",
"UVWXYZ#$%&'()*+ ,-./0123456780:;<=>?@ABCDEFGHIJKLMNOPQRST",
"VWXYZ#$%&'()*+ ,-./0123456780:;<=>?@ABCDEFGHIJKLMNOPQRSTU",
"WXYZ#$%&'()*+ ,-./0123456780:;<=>?@ABCDEFGHIJKLMNOPQRSTUV",
"XYZ#$%&'()*+ ,-./0123456780:;<=>?@ABCDEFGHIJKLMNOPQRSTUVW",
"YZ#$%&'()*+ ,-./0123456780:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWX",
"Z#$%&'()*+ ,-./0123456780:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXY",
};
int mcnt,indx;     /** counters in indexes into request*/
int h5ptr;         /** parts                          */

main()
{
        /***** vars used to hold encrypted data and key
        *****/
        char key[14];
        char tmsg[71],crptmsg[71],fmsg[71];
        char mmsg[71],ccmsg[71];
        char msg[400];
        int mlen,klen;
        int ii;
        char ss[80];
        char host[5];
        char cc,bb,def;
        char bd[257];
        short g[4];
        int dd,tpid;
        char c,v1d[257];
        char c1,c2;
        char ans[512];
        int test;
        int cnt;
        int jj;
```

```
/**** used to open files for answer and
***** to execute mumps request
****/
FILE *fptr,*pptr,*popen();
int fdes;
FILE *ptr;

/**** initialize strings to blanks
***** and ending with NULL
****/
sleep(5);
tmsg[70]=0;
crptmsg[70]=0;
fmsg[70]=0;
mmsg[70]=0;
ccmsg[70]=0;
ss[79]=0;
host[4]=0;
bd[256]=0;
v1d[256]=0;
ans[76]=0;
strcpy(tmsg,"                                                                ");
strcpy(crptmsg,"                                                                ");
strcpy(mmsg,"                                                                ");
strcpy(fmsg,"                                                                ");
strcpy(ccmsg,"                                                                ");

/**** initialize encryption key and message length
***** of encryption
****/
mlen=51;
klen=13;
strcpy(key,"COOKIEMONSTER");

/***** read  in headerblock to tell length of
****** data request
*****/
fflush();
h5ptr=open("/usr/mumps/mps/fil3",O_RDONLY);
c=-1;
ii=0;
err=read(h5ptr,mmsg,60);
err=read(h5ptr,&c,1);
sleep(1);

/**** process header block and get number of messages
*****
*****/
mcnt=mmsg[11]-48;

/*****
```

```
****** process parts of incoming request
*****/
indx=0;
for ( ii=0;ii<mcnt;ii++ )
{
        err=read(h5ptr,tmsg,60);
        err=read(h5ptr,&c,1);
        defence(ccmsg,tmsg);
        decrpt(mmsg,key,ccmsg);
        for ( jj=0;jj<50;jj++ )
        {
                msg[indx++]=mmsg[jj];
        }
}

/***** convert data request back to its original form
****** small letters instead of caps.
*****/
for (ii=0;ii<indx;ii++ )
{
        if ( msg[ii]=='+' )
                msg[ii]=0;
        if ( msg[ii]==' ')
                msg[ii]='&';
        if  ((msg[ii]>='A') && (msg[ii]<='Z') )
                msg[ii]=msg[ii]+32;
}

/*****pull off this host name
*****/
dd=0;
for (ii=0;ii<5;ii++ )
{
        host[ii]=msg[dd++];
}

/****get data request
****/
v1d[1]=' ';
for (ii=1;msg[dd]!=0;ii++ )
{
        v1d[ii]=msg[dd++];
}
ss[0]=' ';
dd++;    /*** skip over 0 planted in string for separator ***/

for (ii=0;msg[dd]!=0;ii++ )
{
        ss[ii]=msg[dd++];
}
sscanf(ss,"%d%c%d%c%d%c%d%c",&g[0],&c,&g[1],&c1,&g[2],&c2,&g[3],&c);
```

```
dd++;
for (ii=1;msg[dd]!=0;ii++)
{
        bd[ii]=msg[dd++];
}



/*****
****** build mpscmd file tst.mps and execute it
*****/
ptr=fopen("/usr/mumps/mps/tst.mps","w");
fputc('o',ptr);
fputc('p',ptr);
fputc('e',ptr);
fputc('n',ptr);
fputc(' ',ptr);
fputc('1',ptr);
fputc(':',ptr);
fputc('"',ptr);
fputc('o',ptr);
fputc('u',ptr);
fputc('t',ptr);
fputc('.',ptr);
fputc('d',ptr);
fputc('a',ptr);
fputc('t',ptr);
fputc('/',ptr);
fputc('n',ptr);
fputc('e',ptr);
fputc('w',ptr);
fputc('"',ptr);
fputc(10,ptr);
fputc('u',ptr);
fputc('s',ptr);
fputc('e',ptr);
fputc(' ',ptr);
fputc('1',ptr);
fputc(10,ptr);

/**** first find out which command to execute
*****
*****/
switch (g[3])
{
        /****set command
        *****/
        case 1:
        {
          fputc('s',ptr);
```

```
   fputc(' ',ptr);
   fputc('^',ptr);
for (ii=1;v1d[ii]!=0;ii++)
{
if(v1d[ii]!=0)
        fputc(v1d[ii],ptr);
}
fputc('=',ptr);
fputc('"',ptr);
for (ii=1;bd[ii]!=0;ii++)
{
if(bd[ii]!=0)
        fputc(bd[ii],ptr);
}
fputc('"',ptr);
fputc(' ',ptr);
fputc(10,ptr);
   break;
}

/***** write command
******/
case 0:
{
   fputc('w',ptr);
   fputc(' ',ptr);
   fputc('^',ptr);
for (ii=1;v1d[ii]!=0;ii++)
{
        if(v1d[ii]!=0)
                fputc(v1d[ii],ptr);
}
fputc(' ',ptr);
fputc(10,ptr);
   break;
}

/****** $next command
*******/
case 2:
case 4:
case 6:
{
   fputc('s',ptr);
   fputc(' ',ptr);
   fputc('j',ptr);
   fputc('=',ptr);
   fputc('$',ptr);
   fputc('n',ptr);
   fputc('(',ptr);
   fputc('^',ptr);
```

```
            for (ii=1;v1d[ii]!=0;ii++)
            {
              if(v1d[ii]!=0)
                  fputc(v1d[ii],ptr);
            }
            fputc(')',ptr);
            fputc(' ',ptr);
            fputc(10,ptr);
            fputc('w',ptr);
            fputc(' ',ptr);
            fputc('j',ptr);
            fputc(' ',ptr);
            fputc(10,ptr);
            break;
          }

}

/***** write out variable set or written to
******
*****/
fputc('w',ptr);
fputc(' ',ptr);
fputc('^',ptr);

for (ii=1;v1d[ii]!=0;ii++)
{
        if(v1d[ii]!=0)
                fputc(v1d[ii],ptr);
}
fputc(' ',ptr);
fputc(10,ptr);

/*** write out value of $test
**** to tell if command was successful
****/
fputc('w',ptr);
fputc(' ',ptr);
fputc('$',ptr);
fputc('t',ptr);
fputc('e',ptr);
fputc('s',ptr);
fputc('t',ptr);

fputc(10,ptr);

/****  close output file
******
*****/
fputc('c',ptr);
fputc(' ',ptr);
```

```
fputc('1',ptr);
fputc(10,ptr);
fputc('z',ptr);
fputc('e',ptr);
fputc(10,ptr);
fclose(ptr);


/***** build file that executes the data request
*****/
fptr=fopen("/usr/mumps/mps/mpscmd","w");

/****  use tail to strip file of unwanted front parts
*****
*****/
fprintf(fptr,
"cat /usr/mumps/mps/tst.mps |/usr/mumps/mps/mumps |tail +8 >outMMM;
fclose (fptr);
pptr=popen("chmod +xrw /usr/mumps/mps/mpscmd","w");
pclose(pptr);
sleep(3);
pptr=popen("/usr/mumps/mps/mpscmd","r");
pclose(pptr);

/*****
****** open answer file for reading and get answer
******/
pptr=fopen("outMMM","r");


/**** prepare to send encrypted answer to data link layer in network
*****
*****/
cnt=0;
dd=0;
for (ii=0;ii<78;ii++)
{
        mmsg[ii]=' ';
}
c=' ';
while (( c=getc(pptr)) != '>');
dd=0;
c=getc(pptr);
while ((c=getc(pptr)) !=10)
{
        ans[dd++]=c;
}
c=getc(pptr);
c=getc(pptr);
ctest=getc(pptr);
```

```
fclose(pptr);

/**** combine ans and $test values into mmsg
*****
*****/
dd--;
for ( ii=0;ii<=dd;ii++ )
{
        mmsg[ii]=ans[ii];
}
mmsg[ii]='&';
mmsg[++ii]=ctest;
mmsg[++ii]=0;

/*** code to build entire message into one string to be encrypted
**** is included in input loops- eg. mmsg[dd++]=c
*****/
for (ii=0;ii<=dd+2;ii++)
{
        if ( mmsg[ii]==0 )
                mmsg[ii]='+';
        if ( mmsg[ii]==' ')
                mmsg[ii]='&';
        /**printf("    mmsg[ %d]=%c0,ii,mmsg[ii]);  **/
}
for (ii=dd+3;ii<53;ii++)
        mmsg[ii]='+';
mmsg[69]=0;

/****  convert to capital letters
*****
*****/
for (ii=0;ii<=50;ii++)
        if ( mmsg[ii]>=91 )
                mmsg[ii]=mmsg[ii]-32;

mlen=51;
klen=13;
mmsg[55]=0;
key[13]=0;
ccmsg[55]=0;
crptmsg[55]=0;
strcpy(key,"COOKIEMONSTER");

crpt(mmsg,key,ccmsg);
fence(ccmsg,crptmsg);
crptmsg[52]=0;
fptr=fopen("/usr/mumps/mps/fil1","w");
sleep(5);
fprintf(fptr,"%s0,crptmsg);
```

```
        fclose(fptr);
}

/***** routine to encrypt messages using the Vigenere Cipher
****** to encrypt- plaintext a and key letter k, the ciphertext c
****** is the letter in col a of row k
****** Ideas for this algorithm gotten from ref. 3
*****/
crpt(msg,key,crptmsg)
char msg[71],key[14],crptmsg[71];
{
        int kk,ii,row,col;
        char tmp;
        int mlen,klen;
        mlen=51;
        klen=13;
        kk=(-1);
        for ( ii=0;ii<=mlen-1;ii++ )
        {
          if ( kk==klen-1)
               kk=0;
          else
               kk++;
          row=key[kk]-35;
          col=msg[ii]-35;
          tmp=mat[row][col];
          crptmsg[ii]=tmp;
        }
        msg[65]=0;
        crptmsg[65]=0;
}

/***** routine to decrypt a message that has been encoded using a
****** vegenere Cipher.
****** to decrypt- for ciphertext c, the plaintext a is the col
****** containing c in row k.
******/
decrpt(msg,key,crptmsg)
char msg[],key[],crptmsg[];
{
        int ii,kk,jj,row,col;
        int mlen,klen;
        char tmp;
        kk=(-1);
        mlen=51;
        klen=13;
        for ( ii=0;ii<=mlen-1;ii++ )
        {
          if ( kk==klen-1)
               kk=0;
          else
```

```
            kk++;
        row=key[kk]-35;
        for (jj=0;jj<=56;jj++)
            if (mat[row][jj]==(crptmsg[ii]))
            {
                msg[ii]=jj+35;
            }
    }
}


/*****
****** encrypts a message passed to it by the rail-fence cipher.
****** this cipher is a transposition cipher where characters
****** in the data are rearranged.
****** The letters of a plaintext message are written down in a
****** pattern resembling a rail fence, and then removed by rows.
****** The key to the cipher is given by the depth of the fence.
****** The depth for this program is 3.
******/
fence(msg,cmsg)
char msg[70],cmsg[70];
{
    char ary1[20],ary2[36],ary3[18];
    int ii,jj,kk,indx;
    indx=0;
    ii=0;
    jj=0;
    for ( kk=0;kk<=12;kk++)
    {

        if ( ii<=50)
            ary1[indx]=msg[ii];
        if ( ii+1<=50)
            ary2[jj++]=msg[ii+1];
        if ( ii+2<=50)
            ary3[indx]=msg[ii+2];
        if ( ii+3<=50)
            ary2[jj++]=msg[ii+3];
            ii=ii+4;
            ++indx;
    }
    ary1[14]=0;
    ary2[26]=0;
    ary3[13]=0;
    ii=0;
    indx=0;
    while (ary1[indx]!=0)
        cmsg[ii++]=ary1[indx++];
    indx=0;
    while (ary2[indx]!=0)
```

```
            cmsg[ii++]=ary2[indx++];
        indx=0;
        while (ary3[indx]!=0)
            cmsg[ii++]=ary3[indx++];
}


/***** decrypts a message encrypted by the rail-fence routine
****** with a fence width of 3.
******/
defence(msg,cmsg)
char msg[71],cmsg[71];
{
        char ary1[20],ary2[40],ary3[20];
        int ii,jj,kk,indx;
        indx=0;
        for ( ii=0;ii<=12;ii++)
            ary1[ii]=cmsg[indx++];
        for ( ii=0;ii<=24;ii++)
            ary2[ii]=cmsg[indx++];
        for ( ii=0;ii<=12;ii++)
            ary3[ii]=cmsg[indx++];

        ary1[13]=0;
        ary2[25]=0;
        ary3[13]=0;
        indx=0;
        ii=0;
        jj=0;
        for ( kk=0;kk<=17;kk++)
        {
            if (( ii<=50) && ( ary1[indx]!=0))
                msg[ii]=ary1[indx];
            if (( ii+1<=50) && ( ary2[jj]!=0))
                msg[ii+1]=ary2[jj++];
            if (( ii+2<=50) && ( ary3[indx]!=0))
                msg[ii+2]=ary3[indx];
            if (( ii+3<=50) && ( ary2[jj]!=0))
                msg[ii+3]=ary2[jj++];
                ii=ii+4;
                ++indx;
        }
        msg[71]=0;
}
```

APPENDIX E.

MUMPS DISTRIBUTED PROGRAM EXAMPLE

APPENDIX E. MUMPS DISTRIBUTED PROGRAM EXAMPLE

IBM PC/AT MUMPS

```
> s ^/host1/patient="jones"
processing remote request
still processing remote request
> w ^/host1/patient
processing remote request
still processing remote request
jones
> s ^/host1/patient/blood(1)="op"
processing remote request
still processing remote request
> w ^/host1/patient/blood(1)
processing remote request
still processing remote request
op
> j=$n(^/host1/patient/blood(-1))
processing remote request
still processing remote request
1
```

VITA

Karen Lauree Gilleland was born in Macon, Georgia on November 18, 1962. She graduated from Gilead Christian Academy in June 1980. The following September she entered Mercer University in Macon, Georgia; and in June 1984 she received a Bachelor of Science degree in Computer Science.

In September of 1984 she entered the Graduate School of the University of Tennessee, Knoxville. She began a teaching assistantship in January of 1985. She received the Master of Science degree with a major in Computer Science in August 1986.

She is a member of the Association for Computing Machinery and the Computer Society of the Institute of Electronic and Electrical Engineers.