INTERNETWORK SHARING OF LICENSED SOFTWARE

by

LINDA S. NEEL

B.S.,Kansas State University, 1983

A REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computing & Information Sciences

KANSAS STATE UNIVERSITY
Manhattan,Kansas

1987

Approved by:

Major Professor

All terms mentioned in this report that are known to be trademarks or service marks are listed below.

C86 is a trademark of Computer Innovations, Inc.

OPTIMIZING C86 is a trademark of Computer Innovations, Inc.

MS-DOS is a trademark of Microsoft Inc.

IBM a registered trademark of Internation Business Machines.

Zenith is a registered trademark of Zenith Electronics Corporation.

Z-150 & Z-158 are trademarks of Zenith Data Systems Corporation.

PC6300 is a trademark of AT&T.

NATIONAL SEMICONDUCTOR is a registered trademark of National Semicondutor Corporation.

3COM is a registered trademark of 3Com Corporation.

Intel is a registered trademark of Intel Corporation.

TURBO PROLOG is a registered trademark of Borland International Inc.

ACKNOWLEDGEMENTS

I wish to thank Dr. Virgil Wallentine for all the help and support he has provided me throughout my project and report. Special thanks to Dr. Maarten Van Swaay for his unlimited patience, guidance and knowledge of operating systems. Additional thanks go to my fellow graduate students for their constant encouragement and support. Last, but most importantly, thanks to my parents, not only for their love and support, but for helping me realize the importance of an education.

Table of Contents

Figures

CHAPTER 1

INTRODUCTION

1.1 Introduction

Since 1981, microcomputers have become an important tool in the workplace. Businesses, both large and small, recognized that workers who perform routine tasks increase their productivity when aided by microcomputers. In 1985, the U.S. Bureau of Labor estimated that 56 percent of all technical workers, 27 percent of managers, and 29 percent of professional workers had microcomputers. By 1990, the corresponding figures are projected to be 76 percent, 64 percent, and 64 percent, respectively [GOLD87].

One problem arising from the popularity of microcomputers is copy protection of software. The capability to duplicate software easily has led to software piracy, the illegal copying or using of copyrighted or licensed software without the permission of the copyright owner.

The most common method of protecting microcomputer software is the license agreement, which dictates the conditions under which a user or group of users can run a program. The typical license requires that the license holder run the program on only one computer at a time and that any copies made or provided with the software are for backup purposes only and may not be copied or transferred to another party.

With the increased use of microcomputers, ways have been found to increase their efficiency and productivity. One way to make microcomputers more efficient and economical is to connect them to a local area network. A local

area network consists of a group of computers and peripheral devices connected by data communication hardware and software in a way that allows all computers and peripheral devices on the network to share information, files, and peripherals.

Because most software licenses require that the program run on only one computer at a time, a way was needed to control the use of licensed software on local area networks. Because local area networks can share files and programs, most software products could not be legally placed on networks because no guarantee could be made that the product would only be used on one machine at a time.

Several forms of agreement have been developed to define the legal use of licensed software on local area networks. One of these is known as a site license. A site license allows a product to be used legally on a specified number of machines or on an unspecified number of machines located in a specified area.

In the Computing & Information Sciences department, there is a local area network that grants a request for a product if there is a licensed copy available. When individual copies of a software product are purchased, the number of copies of the product purchased is recorded on the network server. The server keeps track of the number of copies currently being used on the network. This ensures that each copy purchased is only being used on one computer at a time.

1.2 Background

For several years, a computer network has been functioning in the Computing & Information Sciences Department at Kansas State University. This network system is a combination of Ethernet's 3COM Local Area Network and in-house code. Brick Verser of the Computing Activities Center and Robert Young

of the Computer Science Department developed the system and wrote the additional code. The 3COM code was disassembled before changes were made and then reassembled. Additional features and functions were coded in Optimizing C86 C.

One feature added to the 3COM code was the facility to use licensed copyrighted programs legally on the file server. The file server on each network has rights to a specified number of copies of licensed software. When a request is made for a licensed program, the file server is responsible for making sure the request is granted only if there are legal copies available. If all copies for a licensed program have been allocated, a message is sent to the requester stating all legal copies are in use and to try later. All legal obligations to software vendors are met by this strategy.

Each network file server contains a file containing names of licensed packages and the number of licenses available for each package. An entry must be made in this file for every licensed product installed on the network.

The system licensing was implemented on each network, therefore each network had a specific number of licensed products to allocate. The strategy has a drawback for packages for which a small number of licenses are available, because the number of licenses on each network cannot be closely matched to the demand on that network.

This report documents the changes that were made to the existing network software to allow the sharing of licensed software between two networks. An RS-232 link allows the file servers to communicate with each other. Whenever a file server receives a request for a licensed software product it cannot grant, that server sends a request to the other file server to borrow a license for the

product. This way all licensed products are available for both networks.

1.3 Overview

The existing network system was designed to share licensed software among user machines on individual networks. Each network had a set of software that could be used. Figure 1 shows the flow when a user machine requests a licensed software product.

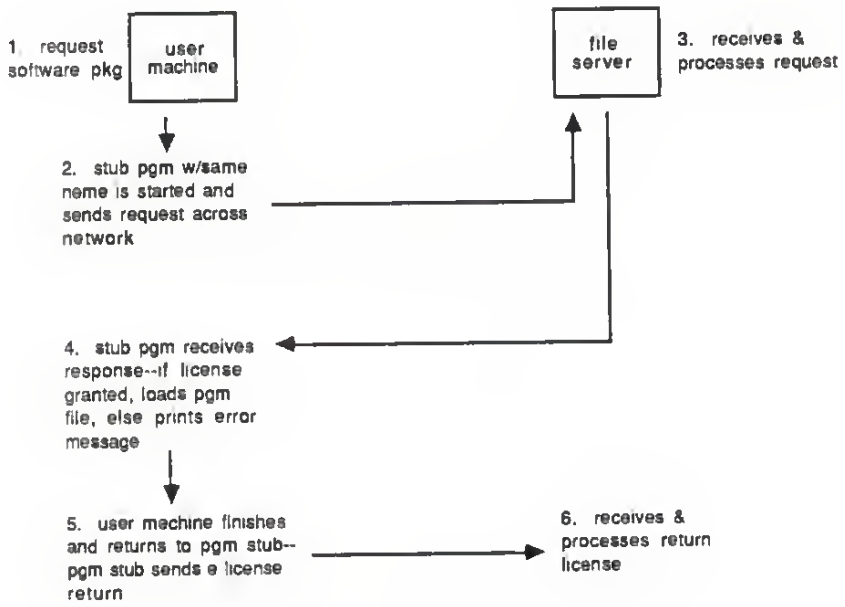

Figure 1

The goal of this project is to expand the sharing of licensed software between two file servers on otherwise independent networks. Flow for the expanded system is shown in Figure 2.

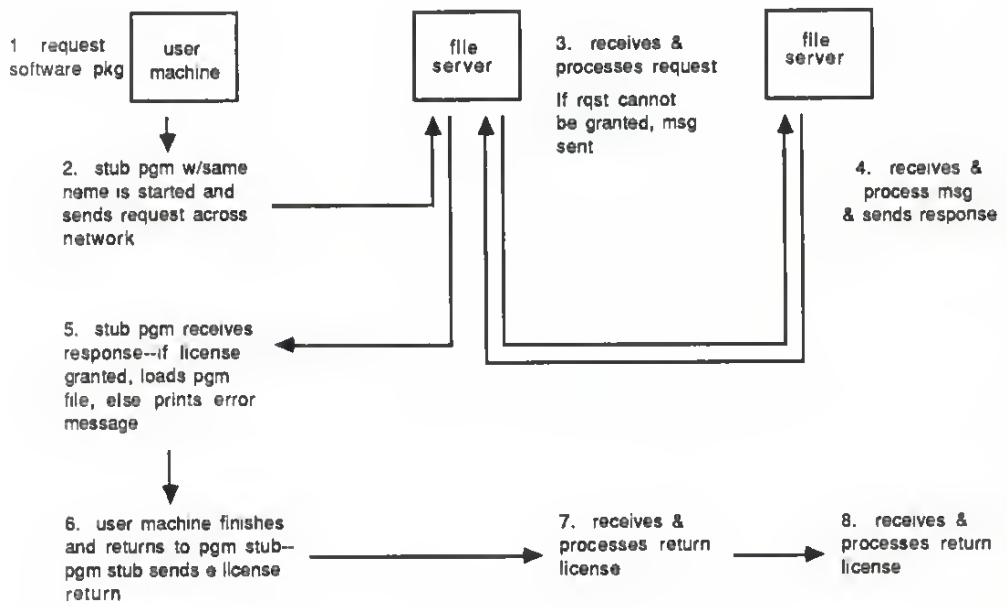

Figure 2

Chapter 2 describes the existing software and hardware and defines the specifications of the expansion. Chapter 3 details the design and implementation of the software. Finally, Chapter 4 describes the performance of the implementation and lists additional features to add at a later date.

CHAPTER 2

EXISTING ENVIRONMENT AND SPECIFICATIONS OF ENHANCEMENTS

2.1 Introduction

The following is a description of the hardware and software used to implement sharing of licensed software between two networks. Specifications for the expanded license sharing are also detailed.

2.2 Existing Hardware

All hardware existed at the beginning of this implementation and changes were not considered. The existing hardware is described in the following sections.

2.2.1 Ethernet

Ethernet network hardware is used to connect the Zenith personal computers. Ethernet is a local area network topology with all machines connected to a shared communication bus consisting of coaxial cable. The Ethernet architecture makes a major division between the physical layer and the data link layer, corresponding to the lowest two levels in the OSI model. The data link layer defines a medium-independent link level communication facility, built on the medium-dependent physical channel provided by the physical layer. Ethernet uses carrier-sense multiple access with collision detection (CSMA/CD) for chan-

nel access.

CSMA/CD gives all stations equal access probability to the network. Any station that wants to transmit a message first listens to the network to see if any other station is transmitting. If not, the station can send a packet. As the packet is sent, the station also listens to see if the packet was garbled by another transmitting station. If so, the station "backs off" a random period of time and then retransmits the message. As the network gets busier, collisions become more frequent and transmission overhead increases.

2.2.2 Personal Computer Hardware

The current network system is composed of Zenith 150 series personal computers for user machines. The file servers are either Zenith 150 series or AT&T 6300 personal computers. The file servers each have a 20 megabyte hard disk where the licensed software and file server programs are located. The following sections describe components of the file servers which are relevant to the implementation.

2.2.2.1 National Semiconductor 8250 UART

The National Semiconductor 8250 UART controls asynchronous serial communication. The 8250 will generate all standard rates up to 19200 baud and will generate four kinds of interrupts. The 8250 appears to the CPU as seven consecutive ports as summarized by the following:

PORT	REGISTER
3F8h	transmit data
3F8h	receive data
3F8h	baud rate divisor L byte
3F9h	baud rate divisor H byte
3F9h	interrupt enable
3FAh	interrupt ID
3FBh	line control
3FCh	modem control
3FDh	line status
3FEh	modem status

The MS-DOS ROM BIOS contains software routines to initialize the port, to receive and transmit data, and to inquire on the serial port status.

2.2.2.2 Intel 8259A Programmable Interrupt Controller

The 8259A PIC (programmable interrupt controller) is the circuit responsible for coordinating interrupt requests made by various hardware devices. Because communication between the file servers is asynchronous, a way is needed to alert the machine that service is needed. An interrupt is generated by a hardware device when service is needed. The 8259 recognizes the interrupt and places a byte on the data bus indentifying the vector. The vector points to the service code. The service code saves the current machine state, services the

device, and then returns to the interrupted process by restoring all flag and register values.

The 8259A PIC coordinates eight independent channels and prioritizes interrupts in order as they happen. IR0 (timer) is the highest and IR7 (parallel printer interface) the lowest. The interrupt mask register determines which devices may generate a request. When the 8259A is initialized during a cold boot, the ROM BIOS disables five of the interrupt lines leaving IR6 (disk), IR1 (keyboard), and IR0 (timer) enabled. Setting its respective bit to zero enables an interrupt. The interrupt mask register can be changed or reconfigured dynamically at any time during program execution. Interrupts can also be disabled by execution of the CLI (clear interrupt flag) instruction and reenabled by execution of the STI (set interrupt flag) instruction.

The Ethernet hardware uses IR3 and IR5 to signal network requests. Interrupts IR3 and IR5 are enabled by the network install program (send60i.asm). IR4 is the hardware interrupt used for serial communications (COM1) and handles communication between file servers.

2.2.3 RS-232

RS-232 is the most common serial communication standard used today. RS-232 is a voltage level convention set by the Electronic Industries Association. It is often used for terminal-modem and terminal-computer connections and in this project as a computer-computer connection. Since 8250 UART serial output cannot be sent reliably over distances of more than a few feet, the TTL signals

generated by the 8250 must be converted to RS-232 signals. The RS-232 standard is specified for distances up to 50 feet but in practice can go at least 100 feet at 9600 baud and farther at slower speeds. The computer-computer connection requires the use of a RS-232 line with two male connectors (DB-25P). Connectors are connected to COM1 of each file server.

2.3 Existing Software

The software used for development and implementation is described in the following sections.

2.3.1 OPTIMIZING C86 C Compiler

The Optimizing C86 C compiler by Computer Innovations, Inc. supports all C language features as defined by Kernighan and Richie. The library includes all standard library functions mentioned in Kernighan and Richie, a selection of UNIX V7 routines, and a set of functions that is specific for its host machine and operating system.

The compiler runs on an 8086 or 8088 processor under DOS version 2.0+ and later versions. 128K of internal memory including an allowance of 16K of memory for the operating system is needed on the machine. At least 256K of disk space is needed to store the compiler and utility programs and to provide working disk space.

The C86 compiler produces object files that are compatible with the regular DOS linker. LINK version 2.20 was used for development and implementation

of this project.

2.3.2 MS-DOS

MS-DOS is the operating system used on the Zenith and ATT 6300 personal computers. MS-DOS is a Microsoft product and is currently an industry standard for the IBM personal computer family and compatibles. MS-DOS is designed as a single-user operating system. It provides an interface between the user and the various devices attached to the computer and supervises the execution of utility and application programs.

2.4 Specification of Enhancements

Sharing licensed software between two networks requires communication between the two file servers as well as changes to the existing file server software. The following describes the requirements of the communications driver and file server software.

2.4.1 Communication driver

MS-DOS has service routines designed to handle some limited functions for the manipulation of the serial port. These routines do not support asynchronous data traffic via the serial port. For that reason, the communication driver must be able to manipulate the serial port registers directly.

The communication driver must be able to place characters into a transmit queue, transmit the characters asynchronously, receive characters asynchronously, and store them in a receive queue. The driver will be associated with a set of supervisor calls that supports user access to the receive and transmit queues. These calls will be responsible for getting characters (strings) from the receive queue, sending characters to the transmit queue, for initialization of the receive and transmit queues, for the initialization of the serial port, and for the control of its hardware interrupt signals.

Because the communication driver will run as a kernel task, it has to be memory resident. MS-DOS provides a way to place a program in memory without it being overwritten with later programs. Using the FIX_IN_MEMORY service at INT 27H makes the program and its data a permanent part of DOS.

2.4.2 File server software

Allocation of licensed software had previously been implemented on a single file server. The main goal of this project is to expand the allocation of licensed software to allow sharing of licensed software between two separate file servers on two networks. Legally, a licensed software product can be simultaneously used on a number of machines that is not larger than the number of licenses owned. To ensure that a product is used only when a license is available, the file server must keep track of the number of licenses owned and the number of copies in use. The file server increments the number of copies in use

whenever a request for a licensed product is made and granted, and decrements the count when the copy is returned. The user machine sends verifications to the file server indicating that its user is using the licensed product. If the file server does not receive a verification within a specified time, it assumes that the user machine has been turned off and the copy is returned.

If one file server cannot grant a request for a software product because all licensed copies are currently in use, or if it doesn't own any copies for the specified product, a request is made to the second file server. The second file server returns a message specifying whether it can grant the request. The requesting machine waits for this response. If the responding file server does not respond in the specified time, the file server does not grant the request. Each file server must now keep track of the number of licenses owned and in use but also of the number of copies it has lent/borrowed.

The number of licenses lent/borrowed becomes important when a file server fails. On reinitialization of the failed file server, the number of licenses that it can allocate must be known to ensure that the file servers do not allocate more copies than the number of licenses owned by both servers.

CHAPTER 3

DESIGN AND IMPLEMENTATION

3.1 Communication Driver

The communication driver used for message passing between file servers is shown in figure 3 and described in the following sections.

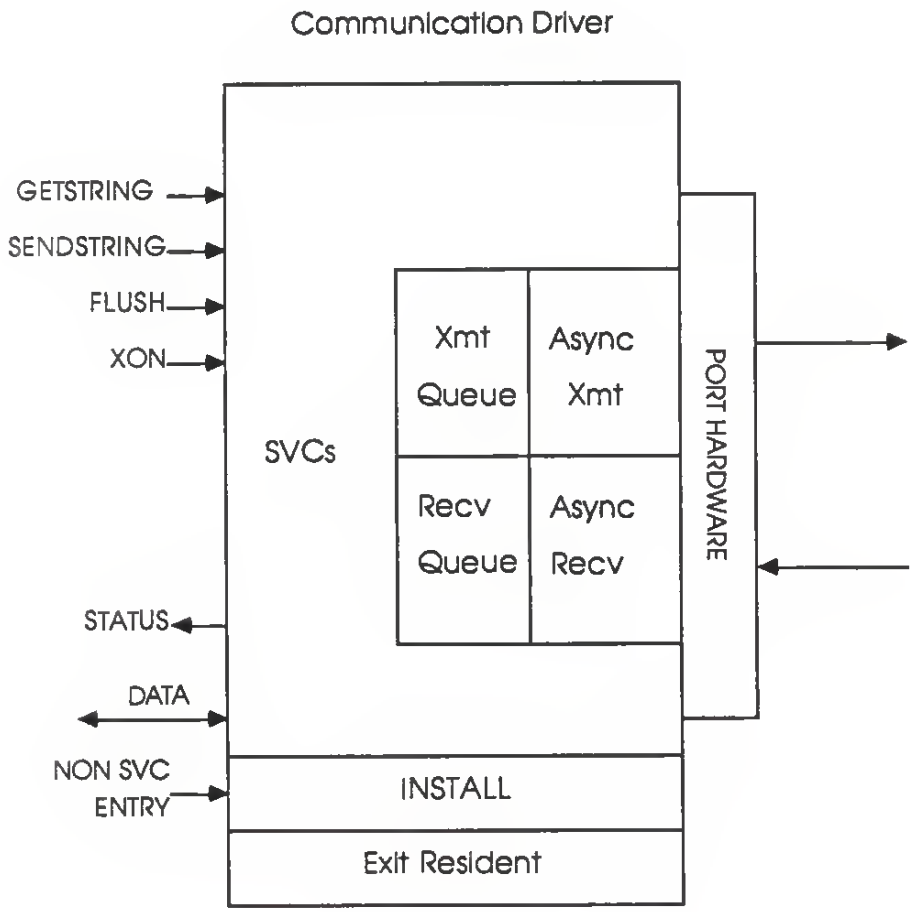

Figure 3

3.1.1 Interrupt Routines

Communication between the file servers is asynchronous. Whenever the serial input port receives a character, it generates an interrupt request that is routed to the interrupt controller. The PIC is controlled by the CPU through a set of I/O ports and, in turn, signals the CPU via the INTR pin and places the interrupt type as an 8-bit pattern on the system bus. The CPU multiplies this number by 4 to find the memory address of the interrupt vector.

The interrupt vector 0CH is used for asynchronous communication port controller 0. Installation of the receive and transmit functions was done by means of MS-DOS facilities. INT21H function 35H retrieves the segment and offset of the current service routine and places them in the ES and BX registers respectively. After loading AL with the interrupt number and DS:DX with the segment:offset of the new interrupt handling routine, INT21H function 25H sets the interrupt vector.

The asynchronous interrupt routine begins by saving register values and reading the interrupt identification register of the serial port. This register contains the interrupt type and reflects whether the interrupt was caused by an input event or an output event. A jump table chooses either the transmit or receive function.

The transmit function verifies that the transmitter holding register is empty. If it is, then processing continues, else the transmit portion of the interrupt handler is exited. The transmit routine checks to see if there are any characters to send. If so, the transmit function sends the current character and updates the queue pointer. If no characters are left to send, the transmit routine

disables the transmitter interrupt and exits.

The receive function first verifies that a character has been received. If so, the receive function reads the character, places it in the queue and updates the queue pointer. If the character was an end-of-message indicator (04H), the function increments the number of complete messages and sets a flag to indicate that at least one complete message is available. The receive function then exits.

The asynchronous interrupt routine exit restores register values, sends an end-of-interrupt (EOI) to the PIC to indicate that interrupt processing is complete, and executes an IRET instruction that restores the original state of the CPU flags, the code segment register, and the instruction pointer.

3.1.2 Supervisor Calls

A set of supervisor calls is associated with the communication driver to support user access to the input and output queues. Specific functions of these calls include retrieving of character strings from a queue, placing character strings in a queue, initializing the queue and queue pointers and controlling hardware interrupt signals. The supervisor calls (also known as software interrupts) are triggered synchronously by a program executing an INT instruction. Interrupt vector 4FH contains the segment and offset of the supervisor calls. The supervisor calls have a common entry point and the function value is passed using standard MS-DOS protocol. The entry code saves register values and jumps to the desired function. The supervisor calls also exit via shared code that restores the register values.

Description and implementation details of the supervisor calls are given in the next sections.

3.1.2.1 GET_STRING

GET_STRING retrieves a string of characters from a queue if a complete message is present. This is the queue in which the receive interrupt places characters. This call will return a status of -1 if there is no message available and a status of 0 for a successful retrieval.

If a message is available, GET_STRING retrieves the character string by reading the first character from the queue. This character is the length of the message. Since the length is in character form, it must be converted to an integer. Clearing the upper four bits of a character converts the character into an integer. After the length has been converted, GET_STRING copies characters from the queue into a character string location. The location of the character string is established before the supervisor call is made and is pointed to by the ES:[DI] register pair. The MOVSB instruction copies the characters from the queue to the string variable. This instruction transfers the memory operand addressed by DS:[SI] into the address pointed to the ES:[DI]. After all characters have been copied, GET_STRING updates the pointer indicating the beginning of the next message, decrements the number of complete messages currently pending and exits.

3.1.2.2 SEND_STRING

SEND_STRING places a character string into the send queue for transmission. The transmit interrupt uses this queue to send characters to the other machine. Like the GET_STRING function, SEND_STRING copies the character string into the queue using MOVSB plus the LODSB instruction. Because the source string is located outside the current data segment, the DS and ES registers must be manipulated to point to the correct source and destination locations. The interface between the file server software and the communication driver passes the address of the character string to SEND_STRING in the ES register. SEND_STRING moves the ES register into the DS register and the ES register is set to point to the queue. SEND_STRING then copies the string into the queue, enables the transmit interrupt, and exits.

3.1.2.3 Buffer Initialization

To initialize the queues used for receiving and transmitting character strings, the pointers are set to the beginning of the queue. The message_count variable is set to zero. This function is used during the initialization of a file server to ensure that the queues do not contain old messages.

3.1.2.4 Interrupt Control

Supervisor call XON and procedure XOFF control the serial hardware interrupt. Whenever the receive function calculates the receive queue is greater than 75% full, it makes a call to XOFF. XOFF sends a control-Q across the line to tell the sending machine to stop transmission. GET_STRING reestablishes transmission when a XOFF has been sent and the receive queue space becomes less than 50% full. A call is made to XON and XON places a control-S in the transmit queue to be sent to the sending machine to restart transmission.

3.1.3 Initialization and Installation

The behavior of the 8250 UART is controlled by the values placed in several housekeeping registers. Because of the length of the RS-232 line, communication parameters of 1200 baud, no parity, one stop bit, and 8 bit characters were chosen. The initialization portion of the communication handler establishes these values by using the MS-DOS service function 14H. The interrupt enable register is set to the value 1 enabling the received data available interrupt. The modem control register is set to the value 0BH. This sets the data terminal ready (DTR), request to send (RTS) and OUT2.

After the 8250 UART is initialized, the interrupt mask register of the 8259A PIC is read and interrupt 4 is enabled. The new mask pattern is written back to the 8259A.

The vector to the software interrupt service routines is installed at inter-

rupt vector 4FH and the vector to the asynchronous service routine is installed at 0CH with the MS-DOS 21H function 25H. At that time, the queues and queue pointers are initialized and the functions are made memory resident by means of the MS-DOS service routine 21H function 31H. This function terminates a process without releasing its memory. The length of the program in paragraphs must be specified in register DX.

3.2 File Server Software

Sharing licensed software between two networks required changes to the existing file server software. Previously, software had been implemented to regulate the use of licensed software on a single network. The terms of the license require that the number of active users of a software product must not exceed the number of licenses owned for the product. To meet this requirement, the original file server software keeps track of the number of licenses currently being used. Requests for a product are not granted if a license is not available.

This same concept was expanded to share licensed software between two networks. Each file server keeps a product table containing the number of licenses owned for a product, the number of licenses in use, and the number of licenses over which the server has control. If the number of licenses controlled by a server is greater than the number of licenses owned, the server has borrowed licenses from the other network. If the number of licenses controlled by a server is less than the number of licenses owned, the server has lent licenses to the other server.

Each file server must have every executable program file that exists in the product table on its hard disk. The product tables for both networks should be identical except for the number of licenses each file server owns. This number is determined by splitting the number of licenses between the file servers appropriately. For example, because Turbo Prolog is used most by graduate students, most or all licenses should be placed on the file server the graduate students use. The system administrator is reponsible for the allocation of licenses to each file server.

To incorporate the concept of sharing licenses between two file servers into the existing software, changes had to be made to existing routines as well as creating new routines. These changes include revisions to the main program loop, license request routines, license return routines, and the creation of product table initialization routines. Figure 4 shows the interaction of the routines and messages.

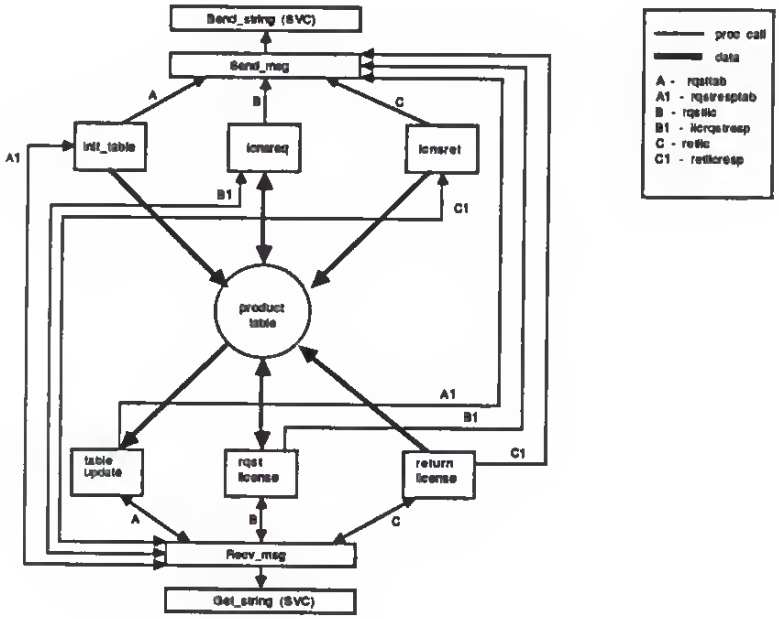

Figure 4

3.2.1 Main Program Loop

The main program consists of two parts: initialization routines and the main program loop.

The initialization routines are responsible for establishing the type of monitoring, initializing the product and user tables, and setting up the boot files.

The main program loop is responsible for processing network requests. It checks to see if there is a network request. If so, checks are made to see if the request is error-free; then a procedure determined by the request type is invoked. If there is no network request, the file server performs some housekeeping routines and again checks for a network request.

The main program loop was modified so that after each network request is processed the file server checks to see if any messages have arrived from the other file server. If there are messages, they are read and served. If no message is present, the loop is started again with the checking for a network request.

An algorithm of the main program follows:

```
Main Program

Initialization
While true
  Get network request
  If network request
    service request
  else
    housekeeping
  endif
  Get server request
  If server request
    service request
  Endif
Endwhile
```

3.2.2 Initialization of Product Table

As stated earlier, it is important that each file server knows the number of licenses it has under its control. The product table is initialized during the initialization process of a file server. The initialization requires two procedures, TABLE_UPDATE and INIT_TABLE.

When the file server software is started, it goes through a series of initialization procedures. One of these procedures is INIT_TABLE. INIT_TABLE clears the receive and send queues of any old messages and sends a request to the second file server asking for information on the number of licenses that the second server has borrowed from the requesting server. The second server responds through TABLE_UPDATE by sending one message for each product for which it currently has borrowed licenses. As the requesting server receives these messages, it updates the product table to reflect the current distribution of licenses. This process is terminated when the second server sends an end-of-list message recognizable by a blank field for the product name.

If both file servers are initialized at the same time, the first server will time out during the product table initialization. This does not cause a problem because if both servers have been down, neither can have active borrowed licenses. If one server has been down, the initialization process will update the product table correctly.

INIT_TABLE and TABLE_UPDATE use two message types: RQSTTAB and RQSTRESPTAB. These messages have the following format:

RQSTTAB	RQSTRESPTAB
length	length
checksum	checksum
message type = 'A'	message type = 'B'
end-of-msg = 04H	product name[8]
	number borrowed[3]
	end-of-msg = 04H

The algorithms for the procedures TABLE_UPDATE and INIT_TABLE are described below:

Table Update

```
While products
  if no_of_prod_owned < no_of_prod_have_use
    sendmsg to requesting server
  endif
  get next product
Endwhile
sendmsg with blank license name
```

End Table Update

```
Init Table

Clear queue
Send request for table update
While not done
 Get message
 If no message
  print error message
 else
  If message type = RQSTRESPTAB
   if product name = blanks
    done
   else
    update table entry
   endif
  endif
 endif
Endwhile

End Init Table
```

3.2.3 License Request

When a user makes a request for a product, a program stub of the same name as the product is executed. This program generates and sends a license request to the file server. The file server receives the request, and checks to verify that it is a known product. After this verification, the server checks to see if the user is already licensed. If the user is not licensed, the procedure LICIDLE is called. LICIDLE looks to see if there are allocated copies of software not being used. If there are, the license permit is returned. To see if a license can be granted to the requester, the number of licenses in use is compared to the number of licenses that it can use. The number of licenses that can be used differs from the number of licenses owned by the number of licenses lent/borrowed. If the server has available licenses, the server grants the request. If the server has no licenses available, it generates and sends a request to the

other file server. That file server checks to see if it can grant the request and sends back a response. If the second server grants the request, it decrements the number of licenses it currently has indicating it has lent a license and the requesting file server increments the number of licenses it currently has indicating it has borrowed a license. If the requesting machine does not receive a response in a specified time, the request is not granted.

LCNSREQ and REQUESTLICENSE use two message types: RQSTLIC and LICRQSTRESP. These messages have the following format:

RQSTLIC	LICRQSTRESP
length	length
checksum	checksum
message type = 'C'	message type = 'D'
product name[8]	product name[8]
end-of-msg = 04H	license grant
	end-of-msg = 04H

The following algorithms are for the license request routines.

```
Lcnsreq

Find product in product table
Unlicense idle users
If file server does not have a license to allocate
  send license request to other file server
  get response
  If no response
    print error message
  else
    if license is granted
      increment number of licenses file server has
    else
      print message
      goto end
    endif
  endif
endif
update user table
update product table

End Lcnsreq


Rqstlicense

Find product in product table
Unlicense idle users
If file server has license to grant
  license granted
else
  license not granted
endif
Send response to requesting file server

End rqstlicense
```

3.2.4 License Return

A user of a licensed product returns a license to the file server in one of three ways:

1) The user exits the licensed program normally; control returns to the program stub. The program stub sends a license return request to the file server.

2) The user exits the licensed program with a Control-C. The stub program again sends a license return request to the file server.

3) The file server expects a license verification every two minutes from the user. This license verification is generated by the stub program. If the file server does not receive a license verification during the specified interval, the file server assumes that the user machine has been shut off and any licenses belonging to the user are returned.

When the file server receives a license return, it looks for the entry in the product table and removes it. The server decrements the number of licenses in use and checks to see if the license was borrowed. If it was a borrowed license, the file server decrements the number of products it can use and sends a message to the other file server to return the license.

When the second file server receives the license return message, it increments the number of licenses it can use and replies with a return license response to the originating file server.

LCNSRET and RETURNLICENSE use two message types: RETLIC and RETLICRESP. These messages have the following format:

RETLIC	RETLICRESP
length	length
checksum	checksum
message type = 'C'	message type = 'D'
product name[8]	product name[8]
end-of-msg = 04H	end-of-msg = 04H

The algorithms for the license return routines are given below:

Lcnsret

```
Find product in product table
Decrement number of license in use
if license was borrowed
  send license return to other file server
endif
update product table
update user table
```

End lcnsret

Returnlicense

```
Find product in product table
Increment number of licenses file server has
Send response to other file server
```

End returnlicense

3.2.5 Message Protocol

The above routines use the SENDMSG and GETMSG procedures to exchange messages with the other file server. SENDMSG receives the information to be

sent from one of the service routines and builds a message. The first field of every message is the length. The length is calculated from the string sent from the service routine and is converted into a character type. The second field of the message is a checksum. The SENDMSG generates a checksum by using the "exclusive or" operator on the message string. The rest of the message is variable depending on the message type. After the message has been assembled, SENDMSG calls an interface procedure to place the message in the transmit queue. The interface procedure is written in assembler language.

GETMSG uses an assembler language interface to retrieve a message from the receive queue. It then verifies that the message arrived correctly using the length and checksum fields. After verification, the message string is passed to the requesting service routine.

The following illustrates how messages are passed between servers.

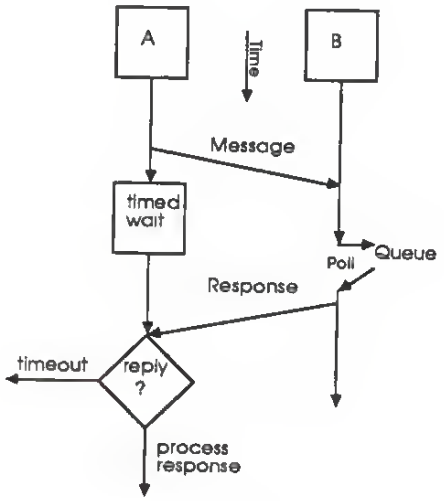

Figure 5

3.2.6 Interfaces

The file server routines are written in C. Supervisor calls are placed from these routines by means of interface procedures written in assembler language.

The SENDBUF procedure initializes the SI register to point to the character string to be copied to the send queue and calls the SEND_STRING supervisor call.

The RECVBUF procedure first calls the settime supervisor call to start the timeout process. It then initializes the DI register to point to the character string to copy the message in the receive queue and calls the SVC GET_STRING. If the message was copied successfully, the timeout process is cancelled through the SVC CANTIME.

The FLUSH procedure calls the software interrupt INIT_QUEUE.

CHAPTER 4

RESULTS AND FUTURE WORK

4.1 Results and Conclusions

Sharing licensed software between two file servers has been implemented and is in operation. Performance testing shows that the new network system operates with a slightly slower response speed. Login time for the old system requires approximately 10 seconds compared to the login time of approximately 12 seconds for the new system.

Request response time is approximately 5 seconds for both the old and new system if the original server can grant the request. However, if the server requests a copy of a license program from the other server, response time nearly triples to approximately 15 seconds. Even with this response time, the new system has advantages over the old system because the old system would have issued a message to "try later" and the user would have to wait for a copy to become available.

With the implementation of shared licensed software between two file servers, users should find using the networks easier. Users of the networks will not have to remember which software is located on what network; all software will be available to both networks.

4.2 Future Work

Additional features that could be added at a later date include adding a scheduler to the network, optimizing software allocation among file servers, and adding additional file servers to share licenses.

4.2.1 Scheduler

The present network file server acts on each network request sequentially and carries the request to completion before executing the next request. By adding a scheduling mechanism to the network, a request that currently uses a busy wait to wait for an event could be blocked and another request started. This would decrease the request response time on the network.

The timeout process used for the communication of messages between file servers is based on the system clock. The clock hardware interrupt signals the waiting process when a specific time has elapsed. This concept would easily fit into a scheduled environment.

4.2.2 Software Allocation

The number of licenses each file server owns is established by the system administrator. A feature which would optimize the number of licenses each file server owned would help the network perform more efficiently. The servers would keep track of the number of licenses borrowed/lent. If trends showed a particular software product was being borrowed/lent often, the file servers would update their product tables with the server who had been borrowing the licenses increasing the number it owned, and the server lending the licenses decreasing the number it owned by the same number. This would decrease license request response time since requests for a copy of a license software product granted by its own server are faster than requests granted from the other server.

4.2.3 Additional File Servers

The sharing of licensed software is currently implemented to share licenses between two file servers. Additional servers could be added with minor changes to the file server software.

One change would be to add an address field to the messages so that they could be distinguished between file servers. A single byte address would be able to address 256 different file servers.

Another change would be to determine the next file server to route a request if the current file server could not grant the request.

BIBLIOGRAPHY

[ANGE86] Angermeyer J., Jaeger K., "MS-DOS Developer's Guide", Howard W. Sams & Co., 1986.

[DUNC86] Duncan R.,"Advanced MSDOS", Microsoft Press,1986.

[GOLD87] Goldstein L.J.,"Microcomputer Applications", Addison-Wesley Publishing Company,Inc., 1987, p. 487-488.

[NATI86] "National Semiconductor Corporation Series 32000 Databook", 1986, p. 4.97-4.101.

[OPTI84] "Optimizing C86 User's Manual", Computer Innovations,Inc., 1984

[ROLL85] Rollins D.,"8088 Macro Assembler Programming", Macmillan Publishing Company, 1985.

[SARG86] Sargent M. III, Shoemaker R.L.,"The IBM PC from the Inside Out", Addison-Wesley Publishing Company,Inc., 1986.

APPENDIX A

Added/Modified Source Code

```
/* LSNSRV5.C - added routines for two servers */

#include <stdio.h>
#include <enet.h>
#include <sysint.h>
#include <bootrec.h>
#include <dosblks.h>
#define EXTERN extern
#include <lsnmdef.h>
#include <bavserv.h>


table_update()

/*****************************************************************************

   Table update is called when the second file server requests information
   on the number of licenses the file server has belonging to the second
   server.  If the current product limit is less than the number of products
   the file server has, then a message is sent to the requesting server that
   name of the product and the number of licenses it is borrowing.  The last
   message sent has a blank product name, which signals the second file
   server that the table update is complete.

*****************************************************************************/


{
 struct prod_tab *p;
 struct info_tab_upd *sm = (struct info_tab_upd *) smbuf;
 int    number,i,len;

 /* pointer to start of prod tab */
 p = prodbase;
 /* message type to be sent */
 sm->ltui_msg_type = RQSTRESPTAB;
 /* search product table */
 while (p != NULL){
  /* are we borrowing a lic? */
  if (p->prod_lim < p->prod_have){
   /* send message to second server */
   len = strlen(p->prod_nam);
   /* copy product name */
   for (i=0; i<len; i++)
    sm->ltui_name[i] = p->prod_nam[i];
```

```
    /* pad with blanks */
    for (i=len; i<8; i++)
      sm->ltui_name[i] = ' ';
    /* calc number borrowed */
    number = p->prod_have - p->prod_lim;
    /* convert to character */
    sprintf(sm->no_have,"%d",number);
    sendmsg(smbuf, sizeof(struct info_tab_upd));
   }
   /* get next product */
   p = p->prod_prd;
 }
 /* termination message */
 for (i=0; i<8; i++)
   sm->ltui_name[i] = ' ';
 sm->no_have[1] = ' ';
 sm->no_have[0] = '0';
 sendmsg(smbuf, sizeof(struct info_tab_upd));
};

init_table()
{

/*****************************************************************************

   Init table is called during the initialization process
   of a file server.  A message is sent to the other file
   server requesting information on the number of licenses
   it second server is currently borrowing.  This procedure
   terminates when the second server sends a message with a
   blank product name.

*****************************************************************************/

 struct info_tab_upd *rm = (struct info_tab_upd *) rmbuf;
 struct rqst_tab_upd *sm = (struct rqst_tab_upd *) smbuf;
 struct prod_tab *pt;
 int done;
 int rmstat,rmlen;
 int number;


 /* flush buffer */
 flush();
 /* message type to be sent */
 sm -> tr_msg_type = RQSTTAB;
 sendmsg(smbuf,sizeof(struct rqst_tab_upd));
 done = 0;
 /* do for each product server has borrowed */
 while (!done){
```

```
    rmstat = getmsg(rmbuf,timeout);
    if (rmstat == -1) {     /* timeout */
      log("TIME*Timeout in Init_table0);
      done = 1;
    } /* if */
    else if (rmstat == 0){  /* process message */
      /* this is the response */
      if (rm -> ltui_msg_type == RQSTRESPTAB){
        /* terminate? */
        if (rm -> ltui_name[1] == ' ')
          done = 1;
        else {
          /* update product table */
          pt = findprod(rm -> ltui_name);
          if (pt == NULL)
            /* don't have product */
            log("PROD*Product not found0);
          else {
            sscanf(rm->no_have,"%d",&number);
            pt->prod_have = pt->prod_have - number;
          } /* else */
        } /* else */
      } /* if */
      else log("UNRT*Unexpected request type0);
    } /* else */
  } /* while */
}; /* init_table */

sendmsg(buf)
UCHAR  *buf;
int    smlen;
{

/*****************************************************************************

   This procedure assembles the message. The checksum and length are
   calculated and added to the message. A call is then made to the
   assembly language routine sendbuf to place the message on the transmit
   queue.

*****************************************************************************/

int i;
char checksum;
char len;


  /* initialize buffer */
  buf[0] = '0';
  buf[1] = '0';
```

```
  /* initialize checksum */
  checksum = ' ';
  /* get length */
  smlen = strlen(buf);
  /* calc checksum */
  for (i=2;i<smlen;i++)
    checksum ^= buf[i];
  buf[1] = checksum;
  /* initialize length */
  len = ' ';
  /* convert to character */
  if (smlen < 16)
    len = smlen + 0x30;
  else
    printf("0nvalid message length");
  buf[0] = len;
  buf[smlen++] = 0x04;
  buf[smlen++] = ' ';
  if (pmflag){
    printf("0acket in sendmsg");
    print_pkt(buf,0x14);
  }
  sendbuf(buf);
}

getmsg(rmbuf,tout)
UCHAR *rmbuf;
int tout;
{

/*****************************************************************************

   This procedure requests a message from the receive buffer.  The length
   and checksum are verified.

*****************************************************************************/

int stat;
int i;
int len,buf_len;
char checksum;
UCHAR b_len;

  /* clear buffer */
  for (i=0;i<20;i++)
    rmbuf[i] = ' ';
  stat = recvbuf(rmbuf,tout);
  if (stat)
    stat = -1;
  else{
```

```
    if (pmflag){
      print_pkt(rmbuf,0x14);
    } /* if */
    b_len = rmbuf[0];
    /* convert to number */
    buf_len = b_len - '0';
    /* check length */
    if ((len = strlen(rmbuf)) != buf_len)
      stat = -1;
    checksum = ' ';
    /* calc checksum */
    for (i=2;i<len;i++)
      checksum ^= rmbuf[i];
    /* check checksum */
    if (checksum != rmbuf[1])
      stat = -1;
  }
  return(stat);
};

rqstlicense(rmbuf)
struct rqst_lic *rmbuf;
{

/*****************************************************************************

    This procedure is called from the main program loop when a rqst license
    message is received.  Rqstlicense sends a response indicating to the
    requesting server whether a license can/cannot be granted.

*****************************************************************************/

  int i;
  struct rqst_lic *rm = (struct rqst_lic *) rmbuf;
  struct rqst_lic_resp *sm = (struct rqst_lic_resp *) smbuf;
  struct prod_tab *pt;

  sprintf(logstr,"LREQ2 %8s0,rm->lrl_name);
  log(logstr);
  pt = findprod(rm->lrl_name);
  /* clear buffer */
  for (i=0;i<20;i++)
    smbuf[i] = ' ';
  /* copy product name */
  for (i=0;i<8;i++)
    sm->lrlr_name[i] = rm->lrl_name[i];
  sm->msg_type = LICRQSTRESP;
  if (pt == NULL){
    sprintf(logstr,"LREQ2*Request for unknown product0);
    log(logstr);
```

```
  sm->lic_grant = '0';
 } /* if */
 else{
  licidle(pt);
  if (pt->prod_use >= pt -> prod_have){
   sm->lic_grant = '0';
   sprintf(logstr,"LREQ2*Request not granted for %s",sm->lrlr_name);
   log(logstr);
  } /* if */
  else{
   pt->prod_have--;
   sm->lic_grant = '1';
   sprintf(logstr,"LREQ2*Request granted for %s",sm->lrlr_name);
   log(logstr);
  } /* else */
 } /* else */
 sendmsg(smbuf);
} /* rqstlicense */

returnlicense(rmbuf)
struct ret_lic *rmbuf;
{

/*****************************************************************************

   This procedure is called from the main program loop when a return license
   message is received.  Returnlicense sends a response indicating to the
   requesting server the message was received.

*****************************************************************************/

 int i;
 struct ret_lic *rm = (struct ret_lic *) rmbuf;
 struct resp_ret_lic *sm = (struct resp_ret_lic *) smbuf;
 struct prod_tab *pt;

 pt = findprod(rm->rl_name);
 if (pt == NULL) {
  sprintf(logstr,"0RET2 %8s Unknown product0,rm->rl_name);
  log(logstr);
 } /* if */
 else {
  pt->prod_have++;
  sprintf(logstr,"0RET2 %8s returned0,rm->rl_name);
  log(logstr);
 }
 /* clear buffer */
 for (i=0;i<20;i++)
  smbuf[i] = ' ';
 /* copy product name */
```

```
  for (i=0;i<8;i++)
    sm->rlr_name[i] = rm->rl_name[i];
  sm->msg_type = RETLICRESP;
  sendmsg(smbuf);
} /* returnlicense */


/* LSNSRV1.C and LSNSRV2.C—modified routines for two servers */

/* (C) Copyright 1985, Brick A Verser and Robert A Young */
/* Network Server */
/* BAVSRV1.C - initialization routines */


main(argc,argv)
int     argc;
char    **argv;
{
        int     rlen,rstat,i,rc;
    int   rmlen,rmstat;
        int     watchflg;
        char    c;
        char    *p;
        struct  user_tab *u;
        struct  ph_hdr *php;
    struct  gen_msg *rm;

        segread(&rv);
        lflag.conlog = 1;               /* normally log to console only */
        logflag = 0;
        statlflg = 0;
        watchflg = 0;                   /* no watchdog timer by default */
    pmflag = 1;
        userbase.user_fre = 1;          /* initial user entry is available */
        userbase.user_ftp = -1;         /* no FTP file open */
        for (i=0; i<6; i++) bcastid[i] = 0xff;  /* ethernet broadcast addr */
        mcastid[0] = 0x80; mcastid[1] = 0; mcastid[2] = 0;
        mcastid[3] = 'K'; mcastid[4] = 'S'; mcastid[5] = 'U';

        argv++; argc--;
        while ((argv) && (*(p = *argv) == '-')) {
           p++;
           while (*p) switch(*p++) {
                case 'p':
                        pflag++;
                        break;
                case 'd':
                        dflag++;
                        break;
```

```
            case 'l':
                  lflag.disklog = !lflag.disklog; /* log to disk */
                  break;
            case 'c':
                  lflag.conlog = !lflag.conlog;   /* log to console */
                  break;
            case 'w':
                  watchflg++;   /* Use software watchdog timer */
                  break;
            default:
                  fprintf(stderr,"invalid option0);
                  break;
      }
      argv++; argc--;
}
printf("KSU 3COM server0);
printf("CS=%04x  DS=%04x0,rv.scs,rv.sds);
printf("TRACSTRT=%04x TRACEND=%04x &TRACCURR=%04x0,
      tracstrt.tracend,&traccurr);
if ((rc=open60(locid))!=0) {
      abort("open60 error0);
};

bootinit();          /* go open boot disk image files */
initcach(100);       /* allocate disk cache */
bpb_init();          /* go read BPB from shared disk */
init1();             /* go read licensed product file */
init_table();        /* see if other file server has products */
loginit();           /* go open log file */
if (watchflg) {
      watchini(WATCHTIM);     /* Start reset timer */
      printf("Watchdog timer started0);
}

printf("Server waiting for work0);

for (;;) {
   bavtrace(0x1001,timestam.hour,timestam.min,timestam.sec);
   rstat=recv60(rbuf,sizeof(rbuf),0x1000,&rlen);
   if (watchflg) watchok();       /* Throw Fido a bone */
      /* qtime(); */              /* Get the time */
      bavtrace(0x1101,timestam.sec,timestam.hnd,rstat);
      if (rstat == -1) {          /* check for timeout */
         logflush();          /* checkpoint log file */
         statlog();           /* get statistics into log */
         c = getcon();        /* check for console command */
         if (c!=EOF) switch(c) {
           case 3:
             goto fini;
             break;
```

```
      case 'S':                  /* print statistics */
        printf("Cache hits = %ld     Disk reads = %ld0,
        stt_chit,stt_drd);
        printf("Packet reads = %ld    Packet writes = %ld0,
              stt_pkrd,stt_pkwr);
        printf("Boot reads = %ld     Coreleft = %xh0,
              stt_brd,coreleft());
        printf("CS=%04x DS=%04x0,rv.scs,rv.sds);
        printf("TSTRT=%04x TEND=%04x &TCURR=%04x0,
              tracstrt,tracend,&traccurr);
        break;
      case 'P':                  /* toggle packet trace */
        pflag = !pflag;
        break;
      case 'D':                  /* toggle disk trace */
        dflag = !dflag;
        break;
      case 'C':                  /* toggle console logging */
        lflag.conlog = !lflag.conlog;
        break:
      case 'W':            /* toggle watchdog timer */
        watchflg = !watchflg;
        if (watchflg) {
          watchini(WATCHTIM);
          printf("Watchdog timer started0);
        } else {
           watchfin();
              printf("Watchdog timer removed0);
         }
        break;
        default:
          printf("I hear you knocking,
                but you can't come in0);
          break;
      }                        /* end switch */
}
else if (rstat == 0) {                  /* receive worked */
  php = (struct ph_hdr *) rbuf;
  if (php->ph_type != ENET_KSU) continue:
    stt_pkrd++;
    if (pflag)
      print_pkt(rbuf,rlen);
    if ((php->ph_msgsz > 0x3c) ?
       (php->ph_msgsz != rlen) : (rlen!=0x3c)) {
          log("PKLE0);  /* packet length error */
          continue;
    }
    /* destaddr isn't ours */
    if (compn(locid,php->ph_dest,6) != 0) {
      /* not broadcast */
```

```
 if (compn(bcastid,php->ph_dest,6)!=0 &&
    compn(mcastid,php->ph_dest,6)!=0) {
        log("INDE0);  /* invalid destination */
        continue;
 }
 if ( (php->ph_req!=REQLOGON) &&
    (php->ph_req!=REQLOGOF)) {
        log("INDE0);  /* invalid destination */
        continue;
 }
}
if (php->ph_req != REQLOGON) {
   u = finduser(rbuf);
   if (u==NULL) {
      if (php->ph_req == REQLOGOF) continue;
      log("UNKU0);  /* unknown user */
      continue;
   }
   /* duplicate pktseq */
   if (u->user_seq == rbuf->ph_seqno) {
     sprintf(logstr,"DUPP %16s %4xh0,
     u->user_uid,u->user_seq);
     log(logstr);
   }
   u->user_seq = rbuf->ph_seqno;
}
switch(php->ph_req) {
   case REQLOGON:
       logon(rbuf);
       break;
   case REQLOGOF:
       logoff(rbuf,u);
       break;
   case REQRDDSK:
   case REQWRDSK:
       rdwrdisk(rbuf,u);
       break;
   case REQLINK:
       linkdisk(rbuf,u);
       break;
   case REQLCNS:
       lcnsreq(rbuf,u);
       break;
   case REQLRET:
       lcnsret(rbuf,u);
       break;
   case REQLVER:
       lcnsver(rbuf,u);
       break;
   case REQOPEN:          /* FTP */
```

```
                    rmtopen(rbuf,u);
                    break;
                case REQCLOSE:
                    rmtclose(rbuf,u);
                    break;
                case REQCOPY:
                    rmtcopy(rbuf,u);
                    break;
                case REQERASE:
                    rmterase(rbuf,u);
                    break;
                case REQBOOT:
                    rmtboot(rbuf,u);
                    break;
                case REQSWAP:
                    rmtswap(rbuf,u);
                    break;
                case REQRMDIR:
                    rmtrmdir(rbuf,u);
                    break;
                case REQMKDIR:
                    rmtmkdir(rbuf,u);
                    break;
                default:
                    log("INRT0);  /* invalid request type */
                    break;
            }
        }
        else {                          /* receive failed */
          sprintf(logstr,"RCVF %x0,rstat);
          log(logstr);    /* receive failed */
        }
rmstat = getmsg(rmbuf,0x01);
if (rmstat == 0){
 rm = (struct gen_msg *) rmbuf;
 switch (rm->msg_type){
  case RQSTTAB:
    table_update();
    break;
  case RQSTLIC:
    rqstlicense(rmbuf);
    break;
  case RETLIC:
    returnlicense(rmbuf);
    break;
  default:
    sprintf(logstr,"0nvalid request");
    log(logstr);
    break;
 }
```

```
        }
        else if (rmstat != -1)
           printf("0nvalid rmstat");
        }
fini:   close60();
        return 0;
}

/* (C) Copyright 1985, Brick A Verser and Robert A Young */
/* Network Server */
/* BAVSRV2.C - main routines */

lcnsreq(rbuf,u)
struct  lic_req *rbuf;
struct  user_tab *u;
{
   int     i,rmstat,len;
   struct  lic_req  *r = (struct lic_req *) rbuf;
   struct  lic_resp *s = (struct lic_resp *) sbuf;
   struct  resp_rqst_lic *rm = (struct resp_rqst_lic *) rmbuf;
   struct  rqst_lic *sm = (struct rqst_lic *) smbuf;
   struct  prod_tab *pt;
   struct  uprd_tab *ut;

   bavtrace(0x2001,0,0,0);
   bldrsp(sbuf,rbuf);
   sprintf(logstr,"LREQ %16s %8s0,u-> user_uid,r-> lic_name);
   log(logstr);
   pt = findprod(r-> lic_name);     /* go look for product table entry */
   if (pt == NULL) {                /* no such thing */
     log("LREQ*Unknown product name0);
     sprintf(s-> lic_msg,"Unknown product name0);
     s-> licr_ph.ph_rcode = 2; /* RC = 2 */
     goto lcr_srsp;          /* go send response */
   }
   ut = finduprd(u,pt);            /* go see if user is already licensed */
   if (ut != NULL) {
     log("LREQ*Duplicate license req granted0);
     goto lcr_dup;
   }
   licidle(pt);                    /* go check for idle users of product */
   if ((pt-> prod_use >= pt-> prod_have) &&
     !(u-> user_flg&USRF_SU)) {   /* already reached limit */
    for (i=0;i<20;i++)
      smbuf[i] = ' ';
      sm-> lrl_msg_type = RQSTLIC;
      len = strlen(r-> lic_name);
      for (i=0;i<len;i++)
        sm-> lrl_name[i] = r-> lic_name[i];
      for (i=len;i<8;i++)
```

```
  sm-> lrl_name[i] = ' ';
sendmsg(smbuf);
rmstat = getmsg(rmbuf,timeout);
if (rmstat == -1){
  sprintf(logstr,"Timeout from getmsg in license request0);
  log(logstr);
  sprintf(logstr,"LREQ*Disallowed due to license limit0);
     log(logstr);
     sprintf(s-> lic_msg,"License denied--all licenses are in use0);
     s-> licr_ph.ph_rcode = 1;        /* RC = 1 */
     goto lcr_srsp;          /* go send response */
}
else if (rmstat == 0){
  if (rm-> lrlr_msg_type == LICRQSTRESP){
    if (rm-> lic_grant == '1'){
      pt-> prod_have++;
      goto lcr_grant;
    }
    else {
      sprintf(logstr,"LREQ2*Disallowed due to license limit0);
        log(logstr);
        sprintf(s-> lic_msg,"License denied--all licenses are in use0);
        s-> licr_ph.ph_rcode = 1;        /* RC = 1 */
        goto lcr_srsp;          /* go send response */
    }
  }
  else switch (rm-> lrlr_msg_type){
    case RQSTLIC:
      rqstlicense(rmbuf);
      break;
    case RETLIC:
      returnlic(rmbuf);
      break;
    case RQSTTAB:
      table_update();
      break;
    default:
      sprintf(logstr,"Invalid message type in rqst license0);
      log(logstr);
  }
  rmstat = getmsg(rmbuf.timeout);
  if (rmstat == -1){
    sprintf(logstr,"Time out in lcnsrqst0);
    log(logstr);
  }
  else if (rm-> lrlr_msg_type == LICRQSTRESP){
    if (rm-> lic_grant == '1'){
      pt-> prod_have++;
      goto lcr_grant;
    }
```

```
        }
          else{
            sprintf(logstr,"LREQ2*Disallowed due to license limit0);
            log(logstr);
            sprintf(s->lic_msg,"License denied--all licenses are in use0);
            s->licr_ph.ph_rcode = 1;        /* RC = 1 */
            goto lcr_srsp;          /* go send response */
          }
        }
      }
      for (i=0; i<UPRDMAX; i++) {     /* need to chain from user_tab */
        if (u->user_prd[i]==NULL) break;
      }
      if (i==UPRDMAX) {               /* user has limit */
        log("LREQ*Disallowed due to user limit0);
        sprintf(s->lic_msg,"License denied--you have %d products in use0,
                UPRDMAX);
        s->licr_ph.ph_rcode = 3;        /* RC = 3 */
        goto lcr_srsp;          /* go send the bad news */
      }
      /* grant the license for a new user */
lcr_grant:
      ut = (struct uprd_tab *) alloc(sizeof (struct uprd_tab));
      ut->uprd_upr = pt->prod_upr;    /* insert new entry at front */
      pt->prod_upr = ut;
      ut->uprd_usr = u;               /* point back to user table entry */
      ut->uprd_prd = pt;              /* and point to prod_tab entry */
      pt->prod_use++;                 /* one more license is in use */
      u->user_prd[i] = pt;            /* point user_tab to prod_tab */
lcr_dup:
      qtime();                        /* query the current time */
      ut->uprd_hg = timestam.hour;    /* remember time we granted license */
      ut->uprd_mg = timestam.min;
      ut->uprd_sg = timestam.sec;
      ut->uprd_hv = timestam.hour;    /* remember time of last verification */
      ut->uprd_mv = timestam.min;
      ut->uprd_sg = timestam.sec;
      ut->uprd_hu = timestam.hour;    /* remember time of last use */
      ut->uprd_mu = timestam.min;
      ut->uprd_su = timestam.sec;
      sprintf(s->lic_msg,"License granted0);
lcr_srsp:
      sendrsp(sbuf,sizeof (struct lic_resp));
      bavtrace(0x2002,0,0,0);
};


lcnsret(rbuf,u)
struct  lrt_req *rbuf;
struct  user_tab *u;
```

```
{
   int     i;
   struct  lrt_req  *r = (struct lrt_req *) rbuf;
   struct  lrt_resp *s = (struct lrt_resp *) sbuf;
   struct  prod_tab *pt;
   struct  uprd_tab *ut;

   bavtrace(0x2101,0,0,0);
   bldrsp(sbuf,rbuf);
   pt = findprod(r-> lrt_name);     /* go look for product table entry */
   if (pt == NULL) {                /* no such thing */
     sprintf(logstr,"LRET %16s %8s Unknown product0,
     u-> user_uid,r-> lrt_name);
     log(logstr);
     sprintf(s-> lrt_msg,"Unknown product0);
     s-> lrt_ph.ph_rcode = 2;        /* RC = 2 */
     goto lcr_srsp;          /* go send response */
   }
   qtime();
   intunlic(u,pt);                 /* go do work of unlicensing */
lcr_sok:
   sprintf(sbuf-> lrt_msg,"License successfully returned0);
lcr_srsp:
   sendrsp(sbuf,sizeof (struct lrt_resp));
   bavtrace(0x2102,0,0,0);
};



intunlic(u,pt)                     /* unlicense product *pt from user *u */
struct  user_tab *u;
struct  prod_tab *pt;
{
   int     i,rmstat,len;
   struct  uprd_tab *ut,*out;
   struct  resp_ret_lic *rm = (struct resp_ret_lic *) rmbuf;
   struct  ret_lic *sm = (struct ret_lic *) smbuf;

   bavtrace(0x2601,0,0,0);
   sprintf(logstr,"LRET %16s %8s0, u-> user_uid, pt-> prod_nam);
   log(logstr);
   ut = pt-> prod_upr;              /* point to first uprd_tab in chain */
   out = NULL;
   while (ut!=NULL) {
     if (ut-> uprd_usr == u) break;  /* got our entry */
       out = ut;              /* remember previous entry */
       ut = ut-> uprd_upr;
     }
     if (ut == NULL) {
```

```
   log("LRET*Not licensed0);
   goto intunxit;          /* Return as if nothing is wrong */
}
/* ungrant the license */
if (out==NULL)                  /* ours was first in chain */
   pt->prod_upr = ut->uprd_upr;    /* unchain our uprd_tab entry */
else                            /* ours wasn't first in chain */
   out->uprd_upr = ut->uprd_upr;   /* rechain without our entry */
free(ut);                       /* give back the storage */
pt->prod_use--;                 /* one less license is in use */
for (i=0; i<UPRDMAX; i++) {     /* need to unchain from user_tab */
    if (u->user_prd[i]==pt) {
      u->user_prd[i]=NULL;
      break;
    }
}
if (pt->prod_have > pt->prod_lim){
pt->prod_have--;
for (i=0;i<20;i++)
  smbuf[i] = ' ';
sm->rl_msg_type = RETLIC;
len = strlen(pt->prod_name);
for (i=0;i<len;i++)
  sm->rl_name[i] = pt->prod_nam[i];
for (i=len;i<8;i++)
  sm->rl_name[i] = ' ';
sendmsg(smbuf);
rmstat = getmsg(rmbuf,timeout);
if (rmstat == -1)
  printf("0ime out from getmsg in intunlic");
else if (rmstat == 0)
   if (rm->rlr_msg_type = RETLICRESP)
     return;
   else
     switch (rm->rlr_msg_type){
       case RQSTLIC:
         rqstlicense(rmbuf);
         break;
       case RETLIC:
         returnlic(rmbuf);
         break;
       case RQSTTAB:
         table_update(rmbuf);
         break;
       default:
         printf("0nvalid message type in intunlic");
    }
else
  printf("0nvalid rmstat in intunlic");
rmstat = getmsg(rmbuf,timeout);
```

```
        if (rmstat == -1)
          printf("0ime out 2 from getmsg in intunlic");
        else if (rmstat == 0)
           if (rm->rlr_msg_type == RETLICRESP)
             goto intunxit;
          printf("0nvalid request or rmstat in intunlic");
    }
intunxit:
    bavtrace(0x2602,0,0,0);
    return;
}
```

```
; Asynchronous Serial Communication Driver
;
; This communication driver is designed to handle asynchronous
; communications between two machines connected by COM 1.
;
;

        page    ,132            ; ask for wide listing format (132 columns)
;
cr      equ     0dh             ;carriage return
lf      equ     0ah             ;line feed
eol     equ     4fh             ;end of line
int_mask equ    10h             ;int mask for irq #4
pic_mask equ    21h             ;programmable interrupt controller
okay    equ     4c00h           ;return code
com_int         equ     0ch             ;serial com #1
ser_br  equ     03f8h           ;serial
ier_ser equ     03f9h           ;interrupt enable register
iir_ser equ     03fah           ;interrupt identification register
ser_lcr equ     03fbh           ;line control register
mod_cntl equ    03fch           ;modem control register
ser_lsr equ     03fdh           ;line status register
mod_stat equ    03feh           ;modem status register
v_num   equ     4fh             ;vector number for SVC
int1c   equ     1Ch             ;timer interrupt
;
;
c_seg   segment
        assume          cs:c_seg
        assume ds:c_seg
main    proc    far
        org     0
seg_org         equ     $
        org     100h            ; for com file
start:  jmp     install
;
asc_int:
        push    ax
        push    bx              ; save registers
        push    cx
        push    dx
        push    si
        push    di
        push    es
        push    ds
        mov     ax,cs           ;set up data segment
        mov     ds,ax
;
        mov     dx,iir_ser
        in      al,dx
```

```
        test    al,01h          ;is there an interrupt?
        jz      int_yes                 ;yes, go service
        jmp     exit

int_yes:sub     bx,bx           ;clear bx
        mov     bl,al           ;al contains interrupt id
        jmp     intvect[bx]     ;jump to correct service code

modstat:jmp     exit            ;no special processing for modem
;
xmt:    mov     dx,ser_lsr
        in      al,dx
        test    al,20h
        jnz     goahead
        jmp     exit
goahead:mov     dx,ser_br       ;ser comm buffer register
        cli
        mov     bx,get_xmt      ;where to start
        cmp     bx,put_xmt      ;is there anything to transmit?
        je      disab           ;no--disable interrupt
        sti
        mov     al,xmt_buf[bx]          ;load byte to transmit
        out     dx,al           ;transmit
        inc     bx              ;point to next byte
        cmp     bx,64           ;are we at the end?
        jb      nowrap3                 ;no--goto nowrap3
        mov     bx,0            ;yes--point to start of buffer
nowrap3:mov     get_xmt,bx      ;update get pointer
        jmp     exit
disab:  sti
        mov     dx,ier_ser      ;int enable register
        mov     al,1            ;disable transmit interrupt
        out     dx,al
        jmp     exit

recv:   mov     dx,ser_lsr      ;line status reg
        in      al,dx
        test    al,01H          ;test to see if char
        jnz     next            ;yes--there is a char
        jmp     exit            ;no--exit
next:   mov     dx,ser_br       ;ser comm buffer reg
        in      al,dx           ;get char
        cmp     al,04h
        jne     nomsgend
        inc     nomsg
        mov     commsg,1
        jmp     exit
nomsgend:
        cli
        mov     bx,p_ptr        ;get put pointer
```

```
        sti
        mov     buffer[bx],al   ;add char to buffer
        inc     bx
        cmp     bx,64           ;are we at the end of the buffer?
        jb      nowrap          ;no--go to nowrap
        sub     bx,bx           ;clear bx
nowrap: cli
        mov     p_ptr,bx        ;update put pointer
        sti
        jmp     exit

recvstat:                       ;not interested in recvstat
exit:
        pop     ds              ;restore registers
        pop     es
        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx
;
        mov     al,20h          ;clear interrupt
        out     20h,al
        pop     ax
        iret
;
my1c_int:
        cmp     cs:counter,0    ;are we timing?
        je      notime          ;no, go on
        dec     cs:counter      ;decrement
        jne     notime          ;has time expired?
        mov     cs:flag,1       ;yes;set flag
notime: jmp     cs:exit_int     ;exit to other clock interrupts

svc_int:
        push    bx              ;save registers
        push    cx
        push    dx
        push    si
        push    di
        push    es
        push    ds
        mov     bx,ax           ;save value passed in cx
        mov     ax,cs           ;set up data segment
        mov     ds,ax
;
        sal     bx,1
        add     bx,offset dsplist
        jmp     cs:[bx]         ;jump to procedure
;
```

```
;GETSTR function (call 0)
getstr:
        mov     ax,0FFFFh       ;init status to -1
tryagain:
        cli                     ;clear interrupts
        cmp     nomsg,0                 ;is there a complete msg?
        jne     gotone
        sti                     ;no;set interrupts
        cmp counter,0           ;are we timing?
        je      timeout
        cmp     flag,0          ;has time expired?
        je      tryagain        ;no; try again
        jmp     timeout         ;yes
gotone:
        sti                     ;set interrupts
        sub     ax,ax           ;set status to 0
        sub     cx,cx           ;clear
        mov     bx,g_ptr        ;get pointer
        mov     cl,[buffer+bx]  ;get number of characters in msg
        and     cx,0Fh          ;turn into an integer
        lea     si,[buffer+bx]  ;load address
more:   movsb                   ;get character
        mov     bx,si
        sub     bx,offset buffer;where are we in buffer?
        cmp     bx,64           ;are we at the end?
        jb      notend          ;yes, no need to worry
        lea     si,buffer
notend:dec      cx              ;decrement number of char left
        jnz     more            ;yes, more chars to get
        mov     bx,si
        sub     bx,offset buffer;calculate pointer
        cli                     ;clear interrupts
        mov     g_ptr,bx        ;update pointer
        sti                     ;set interrupts
        dec     nomsg           ;decrement number of msgs in buffer
timeout:
        jmp     exit2           ;exit
;
;FLUSH function (call 1)
flush:
        cli                     ;clear interrupts
        mov     g_ptr,0         ;clear get pointer
        mov     p_ptr,0                 ;clear put pointer
        sti                     ;set interrupts
        mov     nomsg,0                 ;clear number of messages in buffer
        jmp     exit2           ;exit

;ON function (call 2)
on:
        cli                     ;clear interrupts
```

```
        in      al,pic_mask
        or      al,int_mask
        xor     al,int_mask
        out     pic_mask,al
        sti                     ;set interrupts
        jmp     short   exit2   ;exit

;OFF function (call 3)
off:
        cli                     ;clear interrupts
        in      al,pic_mask
        or      al,int_mask
        out     pic_mask,al
        sti                     ;set interrupts
        jmp     short   exit2   ;exit

;SENDSTR function (call 4)
sendstr:
        sub     ax,ax           ;clear register
        push    ds              ;set up extra register
        pop     es
        pop     ds
        lodsb                   ;get length of message
        push    ds
        push    es
        pop     ds
        cli                     ;clear interrupts
        mov     bx,put_xmt      ;load put pointer
        sti                     ;set interrupts
        mov     xmt_buf[bx],al          ;store char
        inc     bx              ;point to next
        lea     di,xmt_buf[bx]          ;load address
        mov     cx,ax           ;load length into register
        and     cx,0Fh          ;convert into integer
more2:  push    ds              ;set up extra register
        pop     es
        pop     ds
        movsb                   ;move char
        push    ds
        push    es
        pop     ds
        mov     bx,di
        sub     bx,offset xmt_buf;calculate where we are in buffer
        cmp     bx,64           ;are we at end?
        jb      notend2
        lea     di,xmt_buf      ;point to beginning
notend2:
        dec     cx              ;decrement number of char left
        jnz     more2
        mov     bx,di
```

```
        sub     bx,offset xmt_buf;calculate put pointer
        cli                       ;clear interrupts
        mov     put_xmt,bx        ;update put pointer
        sti                       ;set interrupts
        ;enable transmit interrupt
        mov     dx,ier_ser        ;interrupt enable register
        mov     al,3              ;enable transmit register
        out     dx,al
overrun:jmp     short   exit2

;SETTIME (function 5)
settime:
        mov     counter,dx        ;set time
        mov     flag,0            ;set flag
        jmp     short   exit2

;CANCEL TIME (function 6)
cantime:
        mov     counter,0         ;clear time
        mov     flag,0            ;clear flag
        jmp     short   exit2
exit2:
        pop     ds                ;restore registers
        pop     es
        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx
;
        push ax
        mov     al,20h            ;clear interrupt
        out     20h,al
        pop     ax
        iret
main    endp
;
dsplist dw      getstr,getch,flush,test,init,on,off,sendstr,settime,cantime
intvect dw      modstat,xmt,recv,recvstat
p_ptr   dw      0
g_ptr   dw      0
buf_end         dw      64
buffer  db      64 dup(0)
intc_seg dw     0
intc_offs dw    0
flag    dw      0
counterdw       0
put_xmt         dw      0
get_xmt         dw      0
xmt_buf         db      64 dup(0)
```

```
nomsg dw    0
commsg      dw    0
exit_int    dd    ?
rmstat label dword
rmstat_offs dw    ?
rmstat_seg  dw    ?
first:
;
;
install:
        mov   ah,35h          ;get serial communication vector
        mov   al,com_int
        int   21h
        mov   intc_seg,es     ;store
        mov   intc_offs,bx
;
        mov   dx,offset asc_int ;point to our routine
        mov   al,com_int
        mov   ah,25h
        int   21h             ;set serial to our routine
;
        mov   dx,0            ;initialize port to
        mov   al,10000011B    ;1200 baud, no parity
        mov   ah,0            ;1 stop bit and 8 bit chars
        int   14H
;
        mov   dx,mod_cntl     ;modem controller
        mov   al,0bH
        out   dx,al
;
        mov   dx,ier_ser      ;interrupt enable register
        mov   al,1
        out   dx,al
;
        mov   dx,mod_stat     ;reset modem status register
        in    al,dx
        and   al,11110000b
        out   dx,al
;
        in    al,pic_mask     ;8259 int. mask
        and   al,not int_mask       ;enable irq 4
        out   pic_mask,al
;
        mov   dx,offset svc_int ;load SVC address
        mov   al,v_num
        mov   ah,25h          ;set up SVC's
        int   21h

        mov   ah,35h          ;load timer interrupt vector
        mov   al,int1c
```

```
        int     21h
        mov     word ptr exit_int+2,es          ;store
        mov     word ptr exit_int,bx

        mov     dx,offset my1c_int ;add mine to theirs
        mov     al,int1c
        mov     ah,25h
        int     21h

;
;initialize buffers
        mov     p_ptr,0
        mov     g_ptr,0
        mov     buf_end,64
        mov     put_xmt,0
        mov     get_xmt,0

        mov     dx,offset ok ;print message
        mov     ah,09h
        int     21h
;       exit and stay resident
        mov     dx,(offset first - seg_org + 15) shr 4
        mov     ah,31h
        int     21h
;
ok      db      'Serial communication routines have been installed',cr,lf,'$'
;
c_seg   ends
        end     start
```

INTERNETWORK SHARING OF LICENSED SOFTWARE

by

LINDA S. NEEL

B.S.,Kansas State University, 1983

AN ABSTRACT OF A REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Computer Science

KANSAS STATE UNIVERSITY
Manhattan,Kansas

1987

The local area networks at Kansas State University in the Computing & Information Sciences Department shared licensed software among user machines. Licensed software was allocated to user machines on the basis of the availability of licenses for the product. This report details the design and implementation of an expanded network. Network functionality was expanded to share licensed software between two file servers on otherwise independent networks. Sharing licensed software between two file servers is accomplished by establishing asynchronous communication between the file servers with an RS-232 link. By allowing license tokens to be passed between file servers, all licensed software can be used from either network.