NAVAL POSTGRADUATE SCHOOL
Monterey, California

DEPARTMENT OF THE NAVY

THESIS

JANUS/ADA IMPLEMENTATION OF A STAR CLUSTER NETWORK OF PERSONAL COMPUTERS WITH INTERFACE TO AN ETHERNET LAN ALLOWING ACCESS TO DDN RESOURCES

by
Robert L. Hartman
and
Alec F. Yasinsac

June 1986

Thesis Advisor: Uno R. Kodres

Approved for public release; distribution is unlimited
JANUS/ADA IMPLEMENTATION OF A STAR CLUSTER NETWORK OF PERSONAL COMPUTERS WITH INTERFACE TO AN ETHERNET LAN ALLOWING ACCESS TO DDN RESOURCES

Hartman, Robert L. and Yasinsac, Alec

Master's Thesis

1986, June

1986, June

341

LAN; Concentrator; Cluster; DON Access; Janus/Ada

This thesis demonstrates the viability of implementing a local area network connecting a star cluster of Z-100 personal computers to an ETHERNET local area network and allowing access to a wide area network, ARPANET, through a host on ETHERNET, the VAX 11-780 minicomputer operating under UNIX. The system allows local file and message transfer in port-to-port and broadcast mode between Z-100's on the star network and remote login and file transfer to computers that are hosts on ETHERNET or are accessible through ARPANET. The microcomputers in the cluster can share expensive resources such as laser printers, the Gemini multi-level secure system, the ETHERNET medium, and the network control processor.

Components of the system are programmed in the Janus/Ada programming language for both the Z-100 microcomputers and the Intel 86/12A single board computer.
Janus/Ada Implementation of a Star Cluster Network of Personal Computers With Interface to an ETHERNET LAN Allowing Access to DDN Resources

by

Robert L. Hartman
Lieutenant Commander, United States Navy
B.S., United States Naval Academy, 1974

and

Alec F. Yasinsac
Captain, United States Marine Corps
B.S., Appalachian State University, 1979

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1986

Authors:  

Robert L. Hartman

Alec F. Yasinsac

Approved by:

Uno R. Kodres, Thesis Advisor

Bruce J. MacLennan, Second Reader

Vincent Y. Lum, Chairman, Department of Computer Science

Kneale T. Marshall, Dean of Information and Policy Sciences
ABSTRACT

This thesis demonstrates the viability of implementing a local area network connecting a star cluster of Z-100 personal computers to an ETHERNET local area network and allowing access to a wide area network, ARPANET, through a host on ETHERNET, the VAX 11-780 minicomputer operating under UNIX. The system allows local file and message transfer in port-to-port and broadcast mode between Z-100's on the star network and remote login and file transfer to computers that are hosts on ETHERNET or are accessible through ARPANET. The microcomputers in the cluster can share expensive resources such as laser printers, the Gemini multi-level secure system, the ETHERNET medium, and the network control processor.

Components of the system are programmed in the Janus/Ada programming language for both the Z-100 microcomputers and the Intel 86/12A single board computer.
DISCLAIMER

Many terms used in this thesis are registered trademarks of commercial products. Rather than attempt to cite each individual occurrence of a trademark, all registered trademarks appearing in this thesis are listed below the firm holding the trademark:

Gemini Computers Incorporated, Carmel, California
Gemini Computers

Zenith Data Systems Corporation, St. Joseph, Michigan
Z-DOS Operating System
Z-100 Microcomputer

Microsoft Corporation, Belview, Washington
MS-DOS Operating System

Digital Research Incorporated, Pacific Grove, California
CP/M-86 Operating System
PL/I-86 Programming Language

Intel Corporation, Santa Clara, California
86/12A Single Board Computer
MULTIBUS Architecture

Digital Equipment Corporation, Maynard, Massachusetts
VAX 11/780 Minicomputer
VMS Operating System

Interlan Corporation, Chelmsford, Massachusetts
NI3010 ETHERNET Controller Board

Xerox Corporation, Stamford, Connecticut
ETHERNET Local Area Network

Bell Laboratories, Murray Hill, New Jersey
UNIX Operating System

RR Software, Inc.
Janus/Ada Programming Language

United States Government
Ada Programming Language
# TABLE OF CONTENTS

I. INTRODUCTION ................................................. 9
   A. BACKGROUND .............................................. 9
   B. PROJECT DESCRIPTION .................................... 11
      1. Proposed Capabilities ............................... 11
         a. Local File Transfer ........................... 11
         b. Operations via ETHERNET ................. 11
      2. Telecommunication Layers ......................... 12
      3. Target Hardware ................................. 13
   C. STRUCTURE OF THE THESIS ............................... 13

II. NETWORK AND HARDWARE CHARACTERISTICS .................. 15
   A. GENERAL DISCUSSION .................................... 15
   B. CONCEPTS ................................................ 17
   C. PROTOCOLS ............................................... 20
   D. NETWORK HARDWARE .................................... 24

III. REMOTE LOG IN ............................................. 36

IV. FILE TRANSFER PROTOCOL .................................. 40
   A. INTRODUCTION ........................................... 40
      1. FTP Purpose ....................................... 40
      2. FTP Description .................................. 41
   B. SYSTEM DESCRIPTION ................................... 42
      1. The Concentrator .................................. 42
      2. The Z-100 ......................................... 44
      3. The Connection .................................... 46
V. LOCAL FILE TRANSFER AND PRINTING

VI. IMPLEMENTATION SUMMARY

A. THE HARDWARE CONFIGURATION

B. THE SOFTWARE CONFIGURATION

1. The Operating System

2. The Ada Programming Language
   a. Why The Ida Language?
   b. Useful Features
   c. Problems

C. SYSTEM PROGRAMMING ON THE CONCENTRATOR

1. Resource Sharing

2. Managing Memory

D. THE SYSTEM DESIGN

1. The Structure of the Problem

2. Principles
   a. Remain Standard
   b. Modularize

3. Methodology
   a. Prototype
   b. Top Down

4. The Modules
   a. Global
   b. Poller
   c. TELNET
   d. FTP
   e. Local
LIST OF FIGURES

1.1 System Network Configuration -------------------- 10
2.1 DDN ARCHITECTURE -------------------------- 19
2.2 Sample Host Name -------------------------- 20
2.3 User Datagram Header Format ----------------- 24
2.4 RS-232 Pin Connections --------------------- 25
2.5 MULTIBUS/ETHERNET Connection --------------- 26
2.6 ETHERNET Report/Reset Format --------------- 27
2.7 Receive Data Block in MULTIBUS Memory ------ 28
2.8 ETHERNET Interrupt Handler Code ----------- 30
2.9 Transmit Data to ETHERNET Code ------------- 31
2.10 Receive Status Block Code ------------------- 32
3.1 TCP/IP Protocol Headers --------------------- 38
4.1 FTP COMMAND/REPLY SEQUENCE ----------------- 42
5.1 Local Protocol Datagram Format -------------- 48
6.1 Serial Port Cable Pin Connections ----------- 60
6.2 Handshaking State Diagram ------------------ 61
6.3 Protocol/System Layers ---------------------- 67
6.4 Concentrator Package Structure Chart -------- 68
6.5 System Level 0 Diagram --------------------- 71
I. INTRODUCTION

A. BACKGROUND

The AEGIS weapon system is critically dependent on electronic communication between computer systems. Microprocessors are clustered in a star configuration and connected to other clusters with ETHERNET. These networks are interconnected to other networks to form large communication and processing systems. Software development and resource availability are areas of research within the AEGIS development project.

A testbed for research in this area is a MULTIBUS computer configuration comprising multiple single board computers connected to a minicomputer over an ETHERNET Local Area Network. This testbed proved suitable to develop a prototype local area network connecting a cluster of microcomputers to a minicomputer across ETHERNET using a single board computer as a concentrator. Figure 1.1 graphically depicts the configuration. Implementation of this LAN allows sharing of expensive resources by clustered processors and allows software development to be distributed across the cluster.

A large volume of previous research applies directly to this thesis. The research conducted to allow programming and testing on the single board computer within
Figure 1.1   System Network Configuration
the multiuser system and the working programs that allow interface with ETHERNET provided the foundation for our work. This thesis is a direct follow on to the thesis done by Lt. Col. Don Reeke, USMC, [Ref. 1]. His research provided some background in TCP/IP protocols and included a program written in PL/I which provided a capability to monitor communications on ETHERNET. Another program was able to mimic TCP/IP protocols and navigate the layers of protocol to initiate communication with a foreign site over ETHERNET. This thesis extends that navigation to allow login, logout, and file transfer with a remote site over ETHERNET.

B. PROJECT DESCRIPTION

1. Proposed Capabilities
   a. Local File Transfer

   The 'star' network configuration allows efficient single or multiple file transfer. Any two micro-computers in the cluster should be able to transfer files in either direction asynchronously. Additionally, any micro-computer should be able to transfer files to multiple computers at the same time. This feature should prove particularly useful to instructors and system maintenance personnel.

   b. Operations via ETHERNET

   The concept employed is to allow a user of a microcomputer on the cluster to act as a remote terminal to
any of the computers on ETHERNET. A user may enter a process that allows him or her to transfer files to or from any computer on ETHERNET. The user may also login to a remote host and perform any functions available to a terminal directly connected to that system. A user desiring to retrieve files from a system on ARPANET may use the remote login capabilities to connect to the remote system, trigger the file transfer system on the remote system to retrieve a file from ARPANET, then transfer the file to the microcomputer using the ETHERNET transfer process.

2. Telecommunication Layers

Though a detailed presentation of network layers is presented in [Ref. 2], some general comments are appropriate here. This thesis required attention to six of the seven ISO standard layers. These six are the physical, data link, network, transport, presentation, and application layers. The physical layer is the ETHERNET interface board, the data link, network, and transport layers are handled by TCP/IP, the presentation layer is FTP/TELNET, and the application layer is programmed on the microcomputers. TCP/IP is the protocol accepted by the target mainframe computer and is also an ARPANET standard. A more detailed summary of the network layers is contained in Chapter III.

12
3. Target Hardware

The proposed local area network consists of up to twenty microcomputers, the ETHERNET cable and its interface processor, a mainframe computer, and a single board computer with multiple RS-232 ports accessed via MULTIBUS.

Much of this thesis is dedicated to writing software for the computers involved. The single board computer will operate in total on software created for this thesis. All of the application level software for the microcomputers was written by the authors. The primary task has been producing the software to match the protocols presented by ETHERNET, TCP/IP, and FTP.

C. STRUCTURE OF THIS THESIS

At the heart of this thesis is the code to allow implementation of the network. The text provided is intended to convey the purpose behind design decisions, explain problems encountered, facilitate maintenance programming, and explain operating procedures. Chapter II contains descriptions of specific network characteristics and hardware/software that apply to the system. Chapters III, IV, and V are descriptions of the major subsystems of the project: remote log in, remote file transfer, and local file transfer. Chapter VI is a summary of the network implementation strategies and procedures. Our appendices include a users manual for the Z-100 software, a
program maintenance manual for all original software, a glossary of acronyms and terms, a bibliography, and helpful figures and charts.
II. NETWORK CHARACTERISTICS

A. GENERAL DISCUSSION

Networking has evolved over the years to include large, worldwide, real-time systems that share resources under many services. The Defense Data Network is one such system that is central to our discussion.

Defense Data Network (DDN) is a powerful operational military network composed of several large subnetworks including MILNET and ARPANET. Originally ARPANET was one large subnetwork which has split into the present two subnetworks. ARPANET is primarily for experimental research and development while MILNET has become more of a semi-fixed, operational network utilized by many activities. These networks allow easy and quick communication between users hundreds and thousands of miles apart, round table discussions with several users, information sharing, passing programs and tools to enhance local capabilities, remote login to host computers and electronic mail. The three major services of the network are electronic mail, file transfer and remote login [Ref. 3].

The most used service on the DDN is electronic mail service [Ref. 3]. A system has been implemented which allows users to send messages electronically to one another. The system stores the messages that come in for a
user until he or she has time to read and act on them. Mail can be printed, read, deleted and replied to with little effort. To send mail to another user, one simply specifies their network mailbox, usually of the form: USERNAME@HOSTNAME. Most hosts implement some form of mail handling capability.

File Transfer Protocol (FTP) is another service on DDN which allows moving a file from one computer to another. The enhanced features of FTP allow conversion from one file storage format to another.

TELNET is a protocol used to log in to a remote host from a local host. Once logged in, users are able to use a remote host as if they are using a terminal directly connected to that remote host. Files can be accessed, data entered and programs run from a remote location. TELNET maintains three basic principles:

1. Each terminal is made to appear as a virtual terminal (ie. all terminals appear to be the same to the hosts).

2. Options must allow more sophisticated terminals to use their built-in functions.

3. Rules are implemented to prevent infinite loops of acknowledgements sent back and forth.

The Network Information Center provides services to users of the network. Among the services are:

1. A program named WHOIS/NICNAME, that looks up information in an electronic listing of network users. This service is much the same as "white pages" in a phone directory. A local host program queries the NIC database for information on users of the network. Searches are made by name, partial name,
handle (in case of multiple "hits"), hostname, TAC, and Node name;

2. NIC/QUERY is a browsing system to access the general information stored by the DDN.

3. TACNEWS offers help to TAC users.

B. CONCEPTS

The DDN uses Packet Switch Node (PSN) computers which pass information in packets to a destination. The packet contains information such as destination node, source node, and other information that is explained in more detail in Chapters III, IV and Appendices A and B. The packets are sent out to the destination without a predetermined, dedicated path. Circuit switched networks, on the other hand, create a dedicated path to the destination which is used from the first packet to the last or end of connection. In packet switched systems a packet that was sent may reach the destination before an earlier packet. Information must be contained in the packet to put the packets back together in the correct order. Packets are also broken up into smaller packets if necessary for transmission to hosts with smaller size packet capability.

What previously was termed an Interface Message Processor (IMP) has been replaced with PSN (name only) as discussed above. The PSN's are the backbone of DDN providing the hosts connected to them the necessary network interface. Packets are assembled in a host and sent to a PSN which passes it on through the network. Since networks
do not universally guarantee that all packets will actually arrive at the destination, a reply packet is used to acknowledge receipt of packets. Timeouts are used to retransmit packets not acknowledged.

Figure 2.1 depicts a typical network structure. A terminal may be connected to a host directly, through a telephone connection (using a modem), through a local area network (LAN) or via TAC. The hosts, in turn, are connected to a PSN which are in the wide area network. A gateway can then connect one network to another. In order to ensure that connections can be made across networks, a coordinating agency must oversee the use of destination addresses and host names. The Defense Communications Agency coordinates network usage much like the FCC oversees the use of the airways by broadcast stations. An address of a host would include the network number, PSN number, and host port number on PSN. The network number for MILNET is 26 and for ARPANET is 10. A sample host name is shown in Figure 2.2.

Personal computers (PCs) can be used to access the network. At the present PCs are used only as terminals to a host connected via the various ways mentioned previously. The DDN Project Management Office is studying various means of connecting PCs to the network, including allowing them host status. Eventually, as the capability of PCs
FIGURE 2.1  DDN ARCHITECTURE
increases, they will be able to implement the network protocols and will attach directly to a PSN.

C. PROTOCOLS

To implement a network system such as DDN, standard protocols have been adopted. The Transmission Control Protocol (TCP) and INTERNET Protocol (IP) are standard protocols initially implemented for the Defense Data Network in the early 1970's. The TCP is designed to be a highly reliable host-to-host protocol in a packet switched communications network. An in-depth description of these protocols is contained in the SRI handbook [Ref. 2]. The IP is designed to allow packet transfers across different networks through "gateways," with fragmentation and reassembly occurring as needed. These protocols have been implemented on many different systems using various languages and under several operating systems. Much of the background research for this thesis was to understand these two protocols. Therefore, a brief discussion is now
provided to help explain the system requirements that have been implemented.

Transmission Control Protocol is designed to provide robustness in the presence of communication unreliability for military computer networks. TCP is a standard interprocess communication protocol which can support a broad range of applications. It is declared to be the basis for all DoD-wide inter-process communications. TCP is a connection-oriented, end-to-end protocol to fit into a layered hierarchy of protocols which support multi-network applications. It assumes that the layers below it are potentially unreliable datagram protocols. TCP can be used in hard-wired connections, packet-switched or circuit-switched implementations.

TCP interfaces with the upper layer user or application processes and lower level protocols (eg. INTERNET Protocol). TCP has the ability to transfer a continuous stream of octets in each direction. To ensure that data is not stored at an intermediate location awaiting more data, a "push" control is employed to send the data through to the destination. The network protocol underneath TCP is assumed unreliable, therefore, data objects that are damaged, lost, duplicated, or delivered out of order must be corrected. Each byte of information is assigned a sequence number, requiring a positive "acknowledgment" sent from the receiver. Damaged packets are identified by two
checksum fields. A means of controlling how much data is sent by a sender in any one packet is available to the receivers. The maximum amount to be sent is governed by a "window" field describing the maximum a potential receiver is willing to accept. Since only one copy of TCP is normally stored and many users may need its service, a method of multiplexing many processes in a single host is achieved by use of addresses or ports within each host. Concatenation of port addresses and the host addresses enable identification of the destination by this so-called "socket." Since local sockets may be used by several foreign processes at the same time, a pair of "sockets" identify a connection. Consider, for example, a remote login from a foreign host. A "well-known" socket for remote logins is 0017 hex. If two different foreign users desired to do a remote login at the same time, the local socket for both would be the same (local host address and the well-known TCP socket address). The distinction between the two connections is made by inspection of the foreign socket. Since the connections are by two different hosts, the host addresses will be different. If two users from the same foreign host wanted to do a remote login, the separate connections can also be distinguished by the distinct port number assigned to them by the foreign host. A host cannot assign the same port number to two different
processes. TCP also allows users to indicate the security level and precedence relation of their communications.

INTERNET Protocol is the layer below TCP and interfaces with the drivers of the physical network. It allows the TCP to send and receive variable-length packets of information enclosed in INTERNET datagram "envelopes". Inter-network communication is provided by the addressing employed in the IP envelope. IP also handles fragmentation and reassembly of packets. For example, if a datagram arrives containing 2K bytes of information and must be sent over a network that can only handle 1.5K in one packet, then the IP will fragment the datagram into two datagrams and provide necessary information to reassemble them at a later point.

Application processes rely on TCP and pass to TCP a buffer containing data to be sent to the other process on the connection. TCP serializes the data with sequence numbers, checksums, etc., and sends the packet to the IP. The IP, in turn, determines the proper route for the packet to take across the network by the addresses listed in its header. It also fragments the packet or combines several fragments as necessary to comply with the requirements of the route the packet is taking.

A User Datagram protocol is used to send messages to other programs with a minimum of protocol mechanism. The protocol is used above IP and is transaction oriented. It
does not guarantee protection against duplicate packets being sent. Format of User Datagram Protocol is shown in Figure 2.3.

```
<table>
<thead>
<tr>
<th>source</th>
<th>destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>port</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>length</td>
<td>checksum</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>data bytes...</td>
<td></td>
</tr>
</tbody>
</table>
```

Figure 2.3 User Datagram Header Format

D. NETWORK HARDWARE

To implement a network system, the hardware used to construct the network must be understood. The CCITT (Consultative Committee for International Telegraph And Telephone) specification X.21 is a standard for connecting terminals and networks, a general purpose interface for synchronous operations on public data networks. The X.21 (15 pin connectors) interface applies to the first level of the ISO model and is served by other interface standards such as RS232C (25 pin connectors).

The CCITT specifies an X.25 standard interface protocol for a Data Terminal Equipment (DTE) to attach to a packet-switch network using Data Circuit-terminating Equipment
(DCE). The interface between the DTE and the DCE is described in [Ref. 4].

The Electronic Industries Association standard RS-232 was originally developed to foster data communications on public telephone networks with use of a modem (modulator-demodulator). Since development in the mid-60s, the RS-232 has been used to directly connect terminals to computers without use of the phone lines and modems (except for truly remote connections). Figure 2.4 shows the RS-232 interface with communications equipment.

<table>
<thead>
<tr>
<th>terminal or computer</th>
<th>modem or other equip</th>
</tr>
</thead>
<tbody>
<tr>
<td>ring indicator</td>
<td>&lt;--22</td>
</tr>
<tr>
<td>data term ready</td>
<td>--&gt;20</td>
</tr>
<tr>
<td>carrier detect</td>
<td>&lt;--8</td>
</tr>
<tr>
<td>data set ready</td>
<td>&lt;--6</td>
</tr>
<tr>
<td>clr to send</td>
<td>&lt;--5</td>
</tr>
<tr>
<td>req to send</td>
<td>--&gt;4</td>
</tr>
<tr>
<td>receive</td>
<td>--&gt;2</td>
</tr>
<tr>
<td>transmit</td>
<td>--&gt;2</td>
</tr>
<tr>
<td>shield gnd</td>
<td>1---</td>
</tr>
</tbody>
</table>

Figure 2.4 RS-232 Pin Connections

An ETHERNET network is a local area network (LAN) that is capable of transferring data at 10 megabits / second over a 2500 meter coax cable. The ETHERNET cable and the associated transceivers that connect to it make up the physical layer of the ISO model for a network.
Interlan's MULTIBUS ETHERNET communications controller board (NI3010) is a single computer board that provides a host with a connection to an ETHERNET network. It complies fully with the Xerox/Intel/Digital ETHERNET specification, version 1.0. Figure 2.5 depicts the controller board's implementation.

![MULTIBUS ETHERNET Connection](image)

Some of the NI3010 Board modes:

1. Go offline - Logically disconnects the board's transmitter and receiver from the network.

2. Go online - Logically connects the board's transmitter and receiver to the network.

3. Run Onboard Diagnostics - Executes an onboard diagnostic program. Figure 2.6 lists the diagnostic outputs.
<table>
<thead>
<tr>
<th>Field</th>
<th>Number of Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Null</td>
<td>2</td>
</tr>
<tr>
<td>2. Frame Length</td>
<td>2</td>
</tr>
<tr>
<td>3. Physical Address</td>
<td>6</td>
</tr>
<tr>
<td>4. Number of Frames Received</td>
<td>2</td>
</tr>
<tr>
<td>5. Number of Frames in Receive FIFO</td>
<td>2</td>
</tr>
<tr>
<td>6. Number of Frames Transmitted</td>
<td>2</td>
</tr>
<tr>
<td>7. Number of Excess Collisions</td>
<td>2</td>
</tr>
<tr>
<td>8. Number of Collision Fragments</td>
<td>2</td>
</tr>
<tr>
<td>9. Number of Times 1 or More Lost</td>
<td>2</td>
</tr>
<tr>
<td>10. Number of Multicast Frames Accepted</td>
<td>2</td>
</tr>
<tr>
<td>11. Number of Multicast Frames Rejected</td>
<td>2</td>
</tr>
<tr>
<td>12. Number Received with CRC Error</td>
<td>2</td>
</tr>
<tr>
<td>13. Number Received with Alignment Err</td>
<td>2</td>
</tr>
<tr>
<td>14. Number of Collisions</td>
<td>2</td>
</tr>
<tr>
<td>15. Number of Out-of-Window Collisions</td>
<td>2</td>
</tr>
<tr>
<td>16. Reserved for Future Use</td>
<td>16</td>
</tr>
<tr>
<td>17. Module ID</td>
<td>6</td>
</tr>
<tr>
<td>18. Null</td>
<td>1</td>
</tr>
<tr>
<td>19. Firmware ID</td>
<td>6</td>
</tr>
<tr>
<td>20. Null</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 2.6 ETHERNET Report/Reset Format

4. Load Transmit Data and Send - Informs the board that it now has a block of transmit data and commands it to transmit it.

5. Reset - Goes to power-up state.

6. Insert Source Address Mode - Causes the board to insert its own physical address into the source field of the ETHERNET frame.

ETHERNET physical addresses are 6 bytes in length. The first 3 bytes are assigned by Xerox and the last 3 are assigned by the manufacturer of the ETHERNET board. The ETHERNET frame format is shown in Figure 2.7.

Programming requirements of the NI3010 board given by the manufacturer fall in 4 categories:

1. Handling an interrupt by the NI3010.

2. Issuing an NI3010 command
Figure 2.7 Receive Data Block in MULTIBUS Memory
3. Transmitting data to the ETHERNET

4. Receiving a status block from the NI3010

There are 7 kinds of interrupts possible, only 3 are discussed: receive-block-available, receive-DMA-done and transmit-DMA-done. Interrupts are enabled by writing to the interrupt enable register with the proper interrupt code. The state of the interrupt processor, identified by the type of the last interrupt received, is recorded in a variable since the interrupt enable register is write only. Interrupts must be disabled prior to handling the interrupt enable register because an interrupt may occur at any time. After a command is issued to the NI3010, the status register must be read. The NI3010 documentation contains code specification for interrupt handling is shown in Figure 2.8.

A command is issued to the NI3010 by writing to the command register, then waiting until the interrupt status register shows that the status register is full (SRF bit = 1). The status register is then read.

The data to be transmitted by the NI3010 is transferred to it then a command to transmit the data is issued. The host must first allow the NI3010 to finish any DMA in progress before trying to transfer data to it. The code listed in Figure 2.9 details the manufacturer's algorithm to transmit data.
disable CPU interrupts
get current IE_REG contents
set IE_REG to 0

if IE_REG was a 4 then
  load bus address registers
  load byte count registers
  set IE_REG to 7
end if

else if IE_REG was a 7
  wake up receive packet process
  give it this packet
  set IE_REG to 4
end else

else if IE_REG was a 6
  set IE_REG to 4
end else

enable CPU interrupts

Figure 2.8 Ethernet Interrupt Handler Code
disable CPU interrupts

while IE_REG is not a 0 or 4 do
  enable CPU interrupts
  repeat
    read IE_REG
    until IE_REG is a 0 or a 4
  disable CPU interrupts
  read IE_REG
end while

set IE_REG to 0

enable CPU interrupts
load bus address registers
load byte count registers

disable CPU interrupts
set IE_REG to 6
enable CPU interrupts

if IE_REG is a 6 then
  wait until it is not a 6
end if

issue a load-transmit-data-and-send command

Figure 2.9 Transmitting Data to ETHERNET

After issuing a command for status, the host reads the interrupt status register (IS-REG) until the status-block-available (SBA/) bit is high, indicating that no more status information is available. The status register is read when the SRF indicates the status register is full. Figure 2.10 lists the algorithm.

The 86/12A single board computer is a complete computer system on a single printed-circuit board. It includes a 16 bit 8086 CPU. 32K expandable to 64K bytes of dynamic RAM,
repeat
  read IS_REG
  if SRF is 1 then
    read s_REG
  end if
until SBA/ is 1

Figure 2.10 Receiving a Status Block

A serial communications interface, three programmable parallel I/O ports, programmable timers, priority interrupt control, MULTIBUS interface control logic, bus expansion drivers for interface with other MULTIBUS interface-compatible expansion boards, and up to 16K bytes of ROM.

Of primary importance is the I/O addressing assignments for the iSBC86/12A. Table 2.1 lists the possible port assignments.

The Zenith Z-100 computer is a dual processor 8085/8088 unit with several on-board hardware capabilities. Some of the hardware features include:

<table>
<thead>
<tr>
<th>model number</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8259A</td>
<td>Programmable interrupt controller</td>
</tr>
<tr>
<td>68A21</td>
<td>Peripheral interface adapter</td>
</tr>
<tr>
<td>2661</td>
<td>Enhanced programmable communications interface</td>
</tr>
<tr>
<td>8253</td>
<td>Programmable interval timer</td>
</tr>
</tbody>
</table>
The Z-100 has two serial ports (J1 and J2), both of which are connected through the 2661 communications interface. J1 is the primary printer port while the J2 port is the primary modem port.

**TABLE 2.1**

<table>
<thead>
<tr>
<th>I/O address</th>
<th>IC</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>00C0</td>
<td>PIC</td>
<td>Programmable</td>
</tr>
<tr>
<td>00C4</td>
<td>PIC</td>
<td>Read: status and poll</td>
</tr>
<tr>
<td>00C2</td>
<td>PIC</td>
<td>Read: OCW1 (mask)</td>
</tr>
<tr>
<td>00C6</td>
<td>Programmable</td>
<td>write: ICW2, ICW3, ICW4, OCW1</td>
</tr>
<tr>
<td>00C8</td>
<td>Programmable</td>
<td>write: port A (j1)</td>
</tr>
<tr>
<td>00CA</td>
<td>Programmable</td>
<td>read: port A (j1)</td>
</tr>
<tr>
<td>00CC</td>
<td>Programmable</td>
<td>write: port C (j1)</td>
</tr>
<tr>
<td>00CF</td>
<td>Programmable</td>
<td>read: port C(j1) or status</td>
</tr>
<tr>
<td>00D0</td>
<td>Programmable</td>
<td>write: counter0(load cnt/N)</td>
</tr>
<tr>
<td>00D2</td>
<td>Programmable</td>
<td>read: counter 0</td>
</tr>
<tr>
<td>00D4</td>
<td>Programmable</td>
<td>write: counter2(load cnt/N)</td>
</tr>
<tr>
<td>00D6</td>
<td>Programmable</td>
<td>read: counter 2</td>
</tr>
<tr>
<td>--D8</td>
<td>Programmable</td>
<td>write: data (j2)</td>
</tr>
<tr>
<td>00DC</td>
<td>Programmable</td>
<td>read: data (j2)</td>
</tr>
<tr>
<td>00DA</td>
<td>Programmable</td>
<td>write: mode or command</td>
</tr>
<tr>
<td>or 00DE</td>
<td>Programmable</td>
<td>read: status</td>
</tr>
</tbody>
</table>

The 8538 8 Channel Communication Expansion Board is a fully programmable synchronous or asynchronous serial communication channel with RS232C interfaces. The 8538 contains IC2651 USARThs for serial communications with other
devices. The 8538 is compatible with the MULTIBUS system. The board's addressing registers in each USART are optionally addressed as memory mapped locations or port addresses. There are 4 locations for each USART that we may be concerned about, the data register, the status register, the mode register and the command register. These memory locations are 0-3 respectively for port 0, 4-7 for port 1, etc. These address locations are added to a base address that is selectable by DIP switches on board the 8538. The total address space given to one board is 64, therefore, a second board would start at 40 hex if consecutive address locations are desired and the first board started at address 0. The four register addresses for each USART extends only to 20 hex, however, the remaining port addresses are given to interrupt handling, which is not used in the implementation of the system. The port addressing is shown in Table 2.2.
TABLE 2.2

USART ADDRESSING

<table>
<thead>
<tr>
<th>address (hex)</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-3</td>
<td>r/w data, status, sync/sync2/dle, mode, cmd</td>
</tr>
<tr>
<td>4-7</td>
<td>&quot;</td>
</tr>
<tr>
<td>8-B</td>
<td>&quot;</td>
</tr>
<tr>
<td>C-F</td>
<td>&quot;</td>
</tr>
<tr>
<td>10-13</td>
<td>&quot;</td>
</tr>
<tr>
<td>14-17</td>
<td>&quot;</td>
</tr>
<tr>
<td>18-1B</td>
<td>&quot;</td>
</tr>
<tr>
<td>1C-1F</td>
<td>&quot;</td>
</tr>
<tr>
<td>20,28,30,38</td>
<td>port reset register (write only)</td>
</tr>
<tr>
<td>21,29,31,39</td>
<td>n/a</td>
</tr>
<tr>
<td>22,2A,32,3A</td>
<td>transmit interrupt register</td>
</tr>
<tr>
<td>23,2B,33,3B</td>
<td>transmit interrupt requests</td>
</tr>
<tr>
<td>24,2C,34,3C</td>
<td>transmit interrupt mask</td>
</tr>
<tr>
<td>25,2D,35,3D</td>
<td>transmit interrupt requests</td>
</tr>
<tr>
<td>26,2E,36,3E</td>
<td>ring detects</td>
</tr>
<tr>
<td>27,2F,37,3F</td>
<td>n/a</td>
</tr>
</tbody>
</table>
III. PROTOCOLS FOR REMOTE LOGIN

The Protocol used for a remote login into the VAX-11/780 Unix system is that of TELNET described in [Ref. 2]. Lower level protocols use TCP/IP and ETHERNET for the transportation, physical, data link, and network levels of the ISO model. TELNET is a host to host communication protocol to allow a user to login onto a remote computer after first logging in on another, perhaps local computer. Once logged in to the remote host, a user can then enter data, run programs or do any operation that is allowed had he logged in directly. A typical remote login sequence is [Ref. 3]:

1. Login to an initial host
2. Invoke the TELNET program on that host
3. Identify the remote host you wish to access by host name or host address.
4. Once connected to the remote host, login with username and password for that host.
5. When finished working on the remote host, logout, then break the connection (if not done so by logging out). Return to the initial host for further processing.

A specialized use of TELNET is to connect to a particular well-known socket (assigned port) on a remote host. A connection such as this takes a user to the program or service offered on that socket. For example, to perform a remote login, socket number 23 is used. 23 is the well-known socket for such service. To transfer files between hosts, the well-known socket of 21 is used. To
make the connection to the host, the complete address or socket is used, which consists of the host's INTERNET address as well as the TCP address or well-known socket. An example of a socket is the well-known socket used by the Vax Unix of IP address of C009C803 hex and TCP address of 0017 hex.

On a more detailed level, to initiate a connection to a remote host, the local host performs what is termed a three-way handshake. To do the handshake, the initiator sends a packet to the remote host with a control code of 'syn' (synchronize). The remote host should recognize the 'syn' and issue an 'acknowledgement' and 'syn' together. These signals are simply a single bit set in a 6 bit control code (see Figure 3.1 for protocol details). Once the 'syn_ack' is received, the initiator sends an acknowledgement, completing the three-way handshake. When the handshake is complete, the state of the connection on each host is 'established'. This is when the user can use the remote host as if he were directly connected to it. In a typical connection, each character that the user enters at his/her terminal is sent in a packet to the remote host. The remote host will process that character and optionally send or not send it back to the user's terminal. Most entries are returned; however, passwords and such are not. Every entry, therefore, is sent individually, wrapped in the TCP/IP protocol as well as the physical network
protocol (ETHERNET protocol). An attempt was made in our implementation to send more than one character at a time. However, the BSD 4.2 Unix system did not recognize more than one character.

![TCP/IP protocol headers](image)

When completing a remote login session, the user logs out from the remote computer, which also causes the remote computer to signal termination of the connection to the
originating host. Once the connection has been terminated by transversing the intermediate states [Ref. 2], the user is returned to the environment of the local host.

TELNET is a very simple protocol above the TCP level. Once 'established' the local host simply sends the characters entered at the keyboard to the remote host by passing them down to the TCP level. Any data returned from the remote host is displayed on the user's terminal.
IV. FILE TRANSFER PROTOCOL

A. INTRODUCTION

1. FTP PURPOSE

File Transfer Protocol or FTP is a well documented software protocol for transferring information between computers within a network. The specifications for FTP are contained in the INTERNET Protocol Transition Workbook [Ref. 2, RFC-765].

This implementation of FTP is used to effect file transfer and related operations between computers on the NPS LAN. This process does not allow exchange between Z-100's, but only between a Z-100 and one of the minicomputers on ETHERNET. The NPS LAN is not directly connected to any external network such as ARPANET, so file transfer beyond the local network can only be accomplished by logging in to a computer on the local network that has external access, in this case the VAX 11-780 operating under UNIX. Once logged in, the user may utilize the version of FTP implemented under UNIX to access computers on the ARPANET and on other networks.

The FTP implementation for this thesis did not require all the features described in the FTP documentation. The goal here is to allow only active data transfers to remote sites, meaning no computer can initiate
a data transfer to a Z-100. This eliminates the need for an FTP server process to handle incoming requests to a Z-100. Additionally, the mail passing facilities of FTP were not programmed. A user of this FTP system may request transfer of a file to or from the remote computer, list the directory on the remote computer, change the working directory on the remote computer, ask for help, or terminate the process. The specific FTP commands, replies, and parameters that are included in this implementation are described in the Program Maintenance Manual [Appendix B].

2. FTP Description

FTP operates using two connections to effect information transfer. A command connection is initiated by the FTP requestor to begin the FTP process. This connection is used to send control information between the two sites before data is transferred. The requestor, or user, sends FTP commands to the remote host, or server. The commands request the desired mode, file type, data connection address, or service required, or reset or abort the connection. The server returns FTP replies which either acknowledge or decline the parameters or requests. The discourse continues until agreement on acceptable parameters is reached. When a data connection is required, the server process initiates a TELNET connection to the using site. When the data transfer (which includes a request for a directory listing) is triggered by the
requesting site, the data is transferred over the data connection. The data connection is closed at the conclusion of each file transfer to indicate that all data has been sent. The data connection then must be reinitiated if another transfer is desired. A typical command/reply sequence is provided in Figure 4.1.

```
***********************************************************************
*   USER TO SERVER       |    SERVER TO USER      *
*   EST CMD CONNECTION   |   REPLY: 220 READY FOR SERVICE *
*   CMD: USER <USERNAME> |   REPLY: 331 NEED PASSWORD *
*   CMD: PASS <PASSWORD>  |   REPLY: 230 USER OK *
*   CMD: PORT <ADDRESS>  |   REPLY: 200 COMMAND OK *
*   CMD: NLST (LIST DIR) |   REPLY: 150 OPENING DATA CONN *
*   SEND LIST            |   *
*   REPLY: 226 TRANSFER COMPLETE |
*   CLOSING DATA CONNECTION *
*   CMD: QUIT            |   REPLY: 221 CLOSING COMMAND *
*   REPLY: 221 CLOSING COMMAND CONNECTION *
***********************************************************************
```

Figure 4.1 FTP Command/Reply Sequence

B. SYSTEM DESCRIPTION

1. The Concentrator

The role of the concentrator is to route FTP commands replies, data, and to establish and maintain the command and data connections. When the concentrator polls
a port that is in the FTP state, there may be three types of information to be transferred in either direction. The three types are: control characters, FTP commands and replies, and data.

Control characters are used to pass coordinating information between the concentrator and a Z-100. Actions such as aborting the process and establishing a new connection are triggered by control codes. When a port operating under FTP is polled by the concentrator, the concentrator checks first to see if a control character is coming from the Z-100.

Incoming data from the network is queued for the Z-100 by attaching it to a pointer within the Port Control Block (PCB) entry for the connection. If the data connection is open, the concentrator checks for any packets received from the remote site queued for the Z-100 and sends any waiting data in its entirety. If no data is waiting from the remote site, the Z-100 is checked for data to transfer to the remote site. If data is waiting from the Z-100, a block is transferred to the remote host over the data connection. If the data connection is not open, bytes received from the Z-100 are presumed to be a command for the remote site and are transmitted to the remote site via the command connection.

The concentrator acts only as a go-between for the Z-100's and the network. The concentrator makes no effort
to recognize FTP level information or generate data on its own. Its responsibility is to pass data, maintain the connections at the TCP level and coordinate with the Z-100 concerning data origin.

2. The Z-100

The Z-100 maintains the dialogue with the remote FTP server process. The host-to-host FTP transfer utilizes separate virtual connections for control and data transfer so control information cannot be intermixed with data. The Z-100's do not have access to the two connections in the cluster configuration. Control and data must be passed over the same serial line connecting the concentrator and the Z-100. Clearly, additional means of communication between the concentrator and Z-100 must be implemented. This is done through the use of a header field implanted as the first byte in reply, command, and data transmission data streams. There is also a presumption of some degree of sequencing. For example, an FTP command can only be followed by an FTP reply.

The structure of the Z-100 process closely resembles the structure of the FTP system. The Z-100 process, termed the user process, is driven by the sequence of FTP commands and replies. The process is begun by initiating the FTP command connection which results in an FTP reply from the remote server process. This reply is captured and an appropriate command is sent to the remote
server. A reply to that command will ensue and the dialogue continues until the user or the server process terminate the dialogue and the process ends.

The Z-100 is primarily concerned with maintaining the FTP dialogue. The FTP command/reply cycle is not perfectly one-to-one. Several peculiarities may be encountered. For instance, all commands sent to the server will trigger a reply, however, some commands will trigger more than one reply. Similarly, some, but not all, replies require a command be sent in response. For example, the reply '331' means that the user name is accepted and a password command is needed, while the reply '200' indicates that the previous command was accepted but does not clearly suggest any further course of action. To further complicate the issue, many FTP replies are acceptable responses to several different FTP commands and the necessary action may be dependent upon which command was sent. The state of the process is identified by knowing the last reply and the last command.

The peculiarities noted are handled in the procedure that processes replies. When a reply is received and conditions are right to receive another reply without sending a command, preliminary action is taken without a command being generated. When no action is indicated by the system, the user is prompted to select an option which triggers an FTP command. The system is designed to be
robust. Even if unexpected replies are received to a particular command, the system will continue to transfer data and converse with the remote site.

3. The Connection

The connection between the Z-100 and the concentrator is a six wire line connected to the Z-100 auxiliary port. The connection is an RS-232 standard and DTR/DSR handshaking; as described in [Ref. 2], is used to pass commands, replies and data. Control characters are passed under cleared DTR/DSR. Two specially coded subroutines handle all handshaking and perform actual data transfers. Due to speed considerations, data is passed only to and from memory in the Z-100 in blocks of not more than five hundred and twelve bytes. When a complete packet has been received, the data is either displayed on the screen as a directory list or delivered to the destination file.
This chapter deals with the microcomputer-to-microcomputer connections used for transferring files, sending messages etc., within the Aegis star cluster network configuration. Since there was no requirement to interface the local connection system with any other systems, it provided the opportunity to implement a totally original scheme for networking. The following requirements were considered in our design phase:

1. Files are to be transferred between two computers with error detection and correction.
2. Files are to be transferred between one computer and two or more other computers with error detection.
3. Files are to be transferred between any computer and the local printer. The printer is to be connected to the concentrator identical to the computers.
4. Only one computer can transfer files to the printer at any given time (a non-sharable asset).

To carry out the above requirements, a new protocol was developed similar to the 'user datagram' described in Chapter II. A layout of the protocol is given in Figure 5.1.

Once a connection is established between two microcomputers, an application program running on one microcomputer simply sends data to the other by specifying the destination terminal in the first byte sent. The source, type, checksum and length fields are also
available, however, are not required in sending data. The fields in the header allow enough flexibility in programming application programs to enable some level of sophistication. The packets sent from one micro can be broadcast to all other microcomputers connected in the same 'group' by using FF hex in the destination field. A 'group' connection is created in the concentrator when a terminal initially connects to another. If the other terminal already has several terminals connected with it, the new terminal is simply inserted into the 'group'. A terminal remains in a 'group' until 1) it terminates the local connection, 2) all other terminals in the 'group' terminate, or 3) the terminal performs a group transfer.
(command 'Change group') to attach itself to another 'group'.

To fully discuss the local connections, a discussion of the process running on the concentrator will be followed by a discussion of a sample application program that allows multiway transfers of files, directory listings and message exchanges, all somewhat concurrently.

The concentrator executes a 'local' process during microcomputer-to-microcomputer transfers. A microcomputer can invoke a local connection by sending a control code 'code_loc' to the concentrator. The concentrator returns that code and changes 'pcb.state' to 'loc_init.' The next byte expected by the concentrator from the microcomputer is the destination port address. Once the destination port address is received, the concentrator checks the state of the destination to verify that a connection can be made. If the connection can be made, the concentrator sends 'code_estab' to the microcomputer. If the connection cannot be made, the 'pcb.state' is set to 'listen.' If the destination terminal was in state 'listen' then it too will be changed to 'local' and 'code_estab' sent to it. If the destination state was already local, no code is sent to it. When the terminals are established in a local connection, their Port Control Blocks (PCB) are linked together with pointers. Any additional terminals that connect to one of these terminals are simply inserted in the linked chain of
PCB's. All PCB's linked together in this fashion and their respective terminals are considered to be in a 'group' connection. There may be several 'group' connections existing at the same moment yet not associated with each other.

In addition to the connection link (PCB field 'loc_con' is used in each PCB to do the linking) each PCB is linked in the 'poller' routine to enable polling of each terminal. There are, therefore two linked chains, one for local connections and one for polling all active terminals. The active terminals are using various processes such as TELNET or File Transfers to other hosts on the ETHERNET. The poller routine polls each PCB individually, calling the appropriate process to handle the particular state of the PCB.

The method of transferring packets over a local connection is divided into three categories:

1. Direct microcomputer-to-microcomputer
2. Broadcast to all in the local connection

The microcomputer-to-microcomputer communication is implemented by receiving a packet from a terminal and determining who to send it to by looking at the first byte (destination field). The first byte is overlayed in the window(1) byte of the memory block (see Appendix B). If the first byte is out of range for the number of
destinations available (0..num_prts) then the packet is discarded. If, however, the destination is a valid port number and the destination state is 'local', then two tasks must be performed:

1. A bit is set in the originating PCB to indicate who must receive that packet.
2. A bit is set in the destination PCB to indicate who has a packet to send to its terminal.

On subsequent polls of these terminals, the local routine checks the originating PCB to see if the bit has been reset. If it has not, no new packet will be received (only one packet at a time). When polling the destination PCB, the bit that was set is found and an attempt is made to send the packet to the terminal. If successful, then both previously set bits are reset. This is the signal to the originator that the packet has been received. During the polling process, the bit reset in the originating PCB will be detected and the packet discarded. A check can then be made of the terminal for another packet to be sent.

The method of broadcasting packets is similar to the previous discussion on microcomputer-to-microcomputer transfers. A broadcast packet is one in which the destination field is FF hex. To effect a transfer to all terminals in the local connection, the local routine traverses the loc_con link and sets a bit in the PCB for each terminal as well as the appropriate bit in the PCB of the originator. As previously described, each terminal will
receive the packet because the correct bit will be set in their PCB entry. When all terminals have received the packet, all bits in the originating PCB are reset.

The transfer of packets to the printer is somewhat different in that only one byte is sent to the printer at a time rather than a block of data. A pointer and counter is maintained to keep track of where the next byte is in the memory block and how many bytes are left. The setting and resetting of bits remain the same as above. An additional mechanism is used to keep track of the number of characters on a line so that a 'tab' code can be replaced by a series of spaces.

The next discussion relates to the application program running on the microcomputer. A need was indicated for transferring files between two or more microcomputers, as well as for communication between users of the microcomputers. An application program has been implemented that carries out these functions. The 'networking environment' that is designed in the program is menu driven. The main features include single stroke key entries for commands. A help feature is incorporated allowing the user to view available commands at any non-text-input point by entering '?'. The following commands are available:
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 1 | All | 10 | Print |
| 2 | Bell | 11 | Quit |
| 3 | Change group | 12 | Send |
| 4 | Directory | 13 | Talk |
| 5 | Get | 14 | Verbose |
| 6 | Information | 15 | Who's there |
| 7 | List | 16 | <destination> |
| 8 | Mailbox | 17 | # |
| 9 | Netstat | 18 | ? |

To send a message to a specific destination, the corresponding terminal number must appear in front of the screen prompt such as:

```
14>
```

The above prompt indicates that anything sent will be sent to terminal number 14. The message could also go to all terminals on the connection with a prompt like this:

```
all>
```

A message is sent by entering a 't'(talk) and typing out the message. Up to 512 bytes can be sent in one message. Typing more than 512 characters will cause the transfer of the first 512 characters followed by another message. All characters entered will be sent including carriage returns, except cntl-Z, cntl-R, cntl-H, cntl-Q and delete. Cntl-Z terminates input and sends the message. When the message is received by the destination microcomputer the the originator is identified by login name and/or terminal number and the message is displayed on the console. The output looks something like the following:
msg fr <name> Nr
<text of message here>

While entering text of a message, all incoming messages are held until the text entry is completed. To review the contents of a message cntl-R is used. The only correction capability is back space (cntl-H) or delete followed by retyping the character. A message can be cancelled before sending it by typing cntl-Q.

To 'send' a file to the destination, an 's' (send) is entered. A prompt is displayed asking for the file name(s). Entering the file names is exactly the same as entering text of a message. Up to 512 bytes can be entered. A comma must be between the file names. An example entry would be:

Send <filename> enter text, ^Z to send:
b:test.com,a:command.com,e:.*

Once this entry is made it is parsed for each file entry. A search is conducted to find the file(s) and to send them one at a time. Since the packets have a checksum value in the header, a receiving microcomputer can determine if the packet arrived without error. If the packet is not sent in a broadcast mode, the receiving microcomputer will acknowledge receipt of the packet as either good or bad. The packet is sent again if it was received with errors. For broadcast packets no
acknowledgement is sent. If a broadcast packet arrives with errors, then the file is closed and an appropriate error message is presented on the screen.

To 'get' file(s), the same method is used as sending files except that the file names entered are sent to the destination computer (no broadcast capability is allowed nor desired) and the destination computer performs the same functions as the send command discussed previously.

To find out who is connected in the same group, (Anyone can connect to any established connection or anyone in the listen state) a 'w' is entered for the command 'Who's there'. A packet is broadcast to everyone in the group asking for user's name. As the names are returned, they are displayed on the screen.

To list all the lastest known active terminals, an 'l' is entered for the 'List' command. This list will include any known terminals in the 'group' connection as well as any other previously active terminals. When the 'Who's there' command is executed, all terminals are removed from the linked chain of active terminals. The terminals that respond to the 'Who's there' command are reconnected to the linked chain. Any transactions with terminals not in the active link chain cause the terminal to be added to the chain.

The 'Bell' on/off command controls the computers bell when incoming messages are received. With bell-ON, the
bell will beep when a message arrives, for use when the user is pre-occupied with other business and not looking at the terminal screen.

The 'Directory' command will search the destination computer for the specific files requested. Entry of file names is the same as that for sending or getting files. A maximum of 32 file names are returned in a single packet, therefore, if many file names are being sent, each packet arriving will indicate the source of the packet and list at most 32 file names. A search of all microcomputers in the connection can be performed by asking for the directory of 'all'. This will cause the directories of all connected computers to be displayed on the console.

The 'Netstat' command queries the concentrator for the status of all the terminals in the network. This information is displayed on the terminal. The 'netstat' command also includes status of the ETHERNET connection, such as number of frames (packets) transmitted, received, etc. To turn off the ETHERNET information when getting 'netstat', use 'verbose-OFF'.

THE 'All' command changes the destination to broadcast rather than a single terminal. The prompt will appear as 'all>'. Any transmissions to follow this prompt will be sent to all users in the group.

To change the destination to any other terminal, simply enter the terminal's number. To determine the number of a
terminal, use the 'List', 'Netstat' or '#' command. The '#' changes the destination terminal to the terminal itself.

The 'Information' command gives helpful information concerning each command available. The information presented is contained in a file on disk. If the file is not on disk the command will fail and an error message will appear. If the file does exist, the information is presented at the user's pace, each segment is presented one at a time following a space bar entry by the user. To implement this feature, the file data is written to the screen one character at a time until a 'tab' is encountered. Once finding a 'tab', space bar continues the screen output until the next 'tab' or end-of-file. The 'tab' is used for easy editing of the file with a typical text editor.

To print out a file, the 'p' command is used and the file name(s) are entered in the same format as a get or send command. The files requested are printed one at a time with automatic form feeds and file names inserted between file outputs.

To change a group connection a 'c' is entered which terminates the old connection and waits for a terminal number to be entered.

The 'Mailbox' command allows a user to keep a microcomputer connected to the network and accessible by
other users indefinitely. When a local connection is normally terminated, the concentrator will close the port. If 'mailbox' has been set, however, the terminal will re-establish a 'listen' connection, allowing any other terminal to connect with it at its leisure.

To quit a networking session, a 'q' is entered. The application program asks to 'confirm' the input by entering a carriage return.

The 'Verbose-ON' command allows a microcomputer to perform screen output during file transfers and full use of the 'netstat' command. No progress on file transfers will be shown on the screen when 'Verbose' is OFF. When multiple file transfers are occurring at the same time from different computers, the information displayed may be slightly confusing. The 'Verbose' feature will turn the screen output off, alleviating the user confusion. A user may also desire to turn off the Verbose feature when another user starts a file transfer with the former's terminal.
VI. IMPLEMENTATION SUMMARY

A. THE HARDWARE CONFIGURATION

The hardware involved in this project was selected and procurement was arranged before the project began. As depicted in Figure 1.1, the microcomputers that are the users of the network are Zenith, model Z-100 microcomputers with dual processor chips. The concentrator is a combination of three types of VLSI boards. The hub of the concentrator is an Intel 86-12A single board computer. It is connected to the microcomputers via three National Semiconductor model BLC 8548 I/O Expansion boards. Also part of the concentrator is the Intel NI3010 ETHERNET controller board. The concentrator is to be housed in an Intel Micromainframe System 432/600 or similar MULTIBUS frame which will serve as the communications medium between the boards.

ETHERNET is a broadcast network communications medium. Implemented as coaxial cable, it is interfaced directly to the ETHERNET controller at each node via an inductive connection. ETHERNET operates at 10 megabits per second (MBPS). The connection between the microcomputers and the concentrator is a serial line using RS232 standards and operating at 9600 BAUD. The pin connection of the cables is shown in Figure 6.1. The effect produced by this
The connection is to allow symmetrical handshaking. The handshaking protocol may be represented as a state transition model. Figure 6.2 gives the corresponding state diagram.

<table>
<thead>
<tr>
<th>peripheral gnd</th>
<th>TxD</th>
<th>RxD</th>
<th>DTR</th>
<th>DSR</th>
<th>CHASIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>gnd</td>
<td>RxD</td>
<td>TxD</td>
<td>DSR</td>
<td>DTR</td>
<td>CHASIS</td>
</tr>
</tbody>
</table>

**Figure 6.1 Serial Port Cable Pin Connections**

A prose description of the transitions follows:

If a site is clear, that is, DSR is low, and the site has not set DTR since last sending CLR, then either of two actions may occur:

1. The site may set DTR to indicate to the other end that it has data to transmit. It must then wait for the other end to signal it is ready to receive before transmitting data. This will be indicated by DSR going high.

2. DSR may go high indicating the other end has data to transmit. To receive the data, the site must set DTR to high. Once the data has been transmitted, the sender must send CLR, which will reset the receivers DSR and indicate that all data has been sent. The receiver will then send CLR and the sites will be back in the clear state.
Figure 6.2 Serial Connection Handshaking States

B. THE SOFTWARE CONFIGURATION

1. The Operating System

The single board computer driving the concentrator will not operate under a commercial operating system.
There will be no auxiliary storage to access and memory will be managed by the network software.

The software for the Z-100 was written to accommodate users under MSDOS version 2.11.

2. The Ada Programming Language
   a. Why The Ada Language?

According to MacLennan [Ref. 5], "The Ada language is the result of a Department of Defense initiative to find a language suitable for embedded computer applications that began in the mid 1960's. Specifications were written as a sequence of five documents between 1975 and 1979 culminating in a competitive language design effort that produced the Ada programming language. The Ada language was revised once and reached its final form in September 1980."

Since the Ada language is very large and complex, the first commercial compilers are only now becoming operational. The Janus implementation is not fully Ada language compatible in its implementation of strings, ASM statements, or type byte and does not support the Ada language standard exception handling or tasking capabilities. Janus/Ada was selected for this project largely due to its versatility in systems programming tasks and due to the Ada languages destiny as the Department of Defense standard language for embedded and systems programming applications.
b. Useful Features

The Ada language tool for iteration, 'LOOP', is a very pleasant addition to the high level operators repeated from PASCAL, COBOL, FORTRAN, and others. The ability to begin, end, and exit a 'LOOP', whenever desired provides the programmer the ability to create programs with the same structure as the problem.

Another extremely useful feature of the Ada language is packages. Packages are program units that contain data, procedures and/or functions. Data and routines within packages are made accessible to a user by providing to the requestor, a specification complete with all the information necessary to use the data or routines in the package. Since the specification is all that can be seen by the requestor, details of implementation may be withheld, supporting the principle of information hiding.

Packages also support separate compilation. Unless specifications are changed, recompiling of a package does not require recompiling packages referencing or being referenced by the recompiled package.

Janus/Ada's resident assembly language and assembly language interface also are very useful characteristics. Routines that require high efficiency may be coded in assembly language and called from high level code. The ease with which assembly code may be used in conjunction with high level Janus/Ada code facilitates
creation of routines that emulate high level operators.

c. Problems

(1) Janus/Ada is a New Language. A major problem in implementing the project in Janus/Ada was the lack of access to an instructor or programmer with experience in Janus/Ada to assist in resolving problems and questions. This problem will diminish rapidly as more projects are conducted in the Ada language throughout DoD.

(2) Janus/Ada is a Large Language. The specification of the Ada language dictates the size of an Ada language compiler. Many of the important features of Janus/Ada are not possible without a large compiler. This size does create software development problems relating to long compute times and large storage requirements. The speed of the compiler is comparatively slow, taking approximately 90 seconds to compile a 100 line package. The compiler is large, requiring disk storage for some 254K of command and overlay files. The execution time of compiled and linked modules is relatively slow and compiled modules are also very large. When Janus is upgraded to a full Ada language status these problems may even worsen.

C. SYSTEM PROGRAMMING FUNCTIONS ON THE CONCENTRATOR

The single board computer housed within the concentrator is a communications processor for the Z-100's. There is no auxiliary memory access required. The system
functions of managing internal memory (RAM) and resource sharing will be effected by the network software.

1. Resource Sharing

Requests to the concentrator for processing time will originate from twenty-five possible sources: The microcomputers, printers, Gemini system, and the ETHERNET interface. Each device is attached to one of the concentrator's twenty-four serial ports, and is polled for requests to send and receive data. Also connected to the concentrator, the ETHERNET controller board issues interrupts across the MULTIBUS to trigger direct memory access (DMA) transfer of packets destined for its ETHERNET address. These packets are linked to a queue for the destination Z-100 and are sent when the Z-100 is polled.

2. Managing Memory

It is projected that the programs that will execute in the concentrator will occupy approximately 50K bytes of the 64K bytes of memory on board the SBC. The rest of the memory will be declared as an array of memory blocks, each of which is large enough to contain one 'datagram' with all protocol headers and five hundred and twelve bytes of data. As the size of the program increases or the memory is expanded, the number of blocks declared may be modified by changing the max_mem_blk constant in the package 'Global'.

The blocks are used to hold incoming frames and to build and hold outgoing frames. The blocks are allocated
and returned by routines that manage the blocks using a memory management table and pointers to identify the next available block. Though most blocks will be returned through normal processing, there is a 'garbage collector' to return blocks left by abnormally terminated processes.

D. THE SYSTEM DESIGN

1. The Structure of the Problem

The problem of implementing a concentrator based NPS local area network may be decomposed into two halves:
   a. Programming the concentrator.
   b. Programming the Z-100 micro-computers.

These two separate computers perform many functions that are clearly assignable to one or the other. For example, routing messages to the correct Z-100 from the network and communicating with the ETHERNET controller board are definitely concentrator functions, while communicating with the user is the responsibility of the Z-100. There are many functions that could be performed by either computer.

The problem closely resembles the level structure of most network implementations. The physical network protocol level, network address resolution protocol, INTERNET interface, site to site coordination, and user interaction layers are encountered within this project. Figure 6.3 is a graphic description of the layers addressed
in this implementation. Generally, the concentrator handles the lower layers of protocol while the Z-100 performs functions nearer the user level.

Within the concentrator, the problem may again be decomposed into two halves. The concentrator must be able to send messages onto ETHERNET and receive messages from ETHERNET. These two halves are independant of each other though they do handle the same protocols. Figure 6.4 is the structure chart for the packages resident in the concentrator. The protocol layers are the more specific description of the problem since each message sent out must

<table>
<thead>
<tr>
<th>USERS</th>
<th>Z-100s</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLICATIONS</td>
<td>Z-100s</td>
</tr>
<tr>
<td>LOCAL/TELNET/FTP</td>
<td>Z-100s</td>
</tr>
<tr>
<td>PORTs</td>
<td>CONCENTRATOR/Z-100</td>
</tr>
<tr>
<td>LOCAL/TELNET/FTP</td>
<td>CONCENTRATOR</td>
</tr>
<tr>
<td>TCP</td>
<td>CONCENTRATOR</td>
</tr>
<tr>
<td>IP/ADDRESS RESOLUTION</td>
<td>CONCENTRATOR</td>
</tr>
<tr>
<td>ETHERNET</td>
<td>CONCENTRATOR</td>
</tr>
</tbody>
</table>

Figure 6.3 System/Protocol Layers
pass each protocol layer 'downward' while received messages must clear each layer 'upward.'

Figure 6.4 Concentrator Package Structure Chart

From the Z-100 level the problem is very specific: 'Match the protocol of the network application software on the VAX UNIX.' In the case of local transfer, the problem
was to create a protocol. The application software to be matched are TELNET and File Transfer Protocol (FTP).

2. **Principles**
   
a. **Remain standard**

   The primary source of information regarding the network protocols was the Stanford Research Institute's publication [Ref. 2]. This document contains a specification of the INTERNET Protocol and Transmission Control Protocol. Since there is no governing authority to enforce meticulous adherence to the specification, variations to the standard exist. Each of these variations is a source of error and aggravation for the programmer who is trying to match the non-standard system. In an attempt to facilitate future maintenance programming for this system, the documentation was followed as closely as possible. Most design and implementation decisions that the references addressed were made to follow the documented standard. Only when allowances had to be made to match non-standard systems was the system intentionally allowed to vary from the documentation. For this reason, the programs were allowed to carry the standard names TELNET and FTP. The local transfer program also follows the lead set by the guidelines for TELNET and FTP.

b. **Modularize**

   The scope of this project is in the range of a small to medium sized software engineering project with
between five and ten thousand lines of code. This, coupled with the nature of this project as a joint thesis, demanded a thorough decomposition of the problem. The modularization was attained based on the structure of the system. Each protocol layer was handled independently of the others and the modules were separated accordingly. Libraries were created to store frequently used modules such as handshaking routines, special adders, conversion functions and others, and modules written in assembly language were separated from high level Ada code.

3. Methodology
   a. Prototype

   The starting point for the work done in this thesis was the thesis by LtCol Reeke [Ref. 1], which contains the code to trigger a remote login to the VAX UNIX using the ETHERNET communication interface. The sequence was accomplished by 'listening' to ETHERNET and mimicking another station that had remote login capabilities. The first goal of this project was to complete the login sequence on that system. Reeke's system, programmed in PL/I, provided many tools used to effect the login. The program to 'listen' to ETHERNET was invaluable and his research concerning the checksum and other algorithms is implemented into the system. Many problems were yet to be overcome in order to effect a complete login. The problems of address resolution,
retransmission of lost or mishandled packets, sharing of data between interrupting and interrupted processes, and managing memory without the assistance of an operating system were just some of the major hurdles to be overcome.

Though the prototype system did not resolve all these problems, it did provide many answers and provide a very good basis to design the final product of this thesis.

b. Top Down

Figure 6.5 is the Level 0 diagram for the final system. This diagram was designed on the basis of

![Level Zero Diagram]

Figure 6.5 Level Zero Diagram
knowledge gained from the prototype and has survived the project with minimal modification. The blocks representing the highest level modules were decomposed into smaller functional modules. The decomposition continued in hierarchical fashion until the lowest functional level was attained. An attempt was made to minimize interface between modules by specifying all input and output data as parameters.

4. The Modules
   a. Global

   One of the taboos in structured design is the use of common data or global data areas. This project avoided these to a large degree. The package 'global' is an exception. Beyond the usual need for global types and constants which are made visible at all scoping levels, this problem required some other shared data. The ETHERNET controller is an interrupt based processor that communicates with the host computer it is supporting by triggering interrupts over a MULTIBUS configuration. All data addressed to the supported host is passed via this interrupt process. The host computer continuously queries or polls the ports and services requests including the transmission of packets onto ETHERNET. The polling process, considered the steady state process, and the interrupt process must be able to access and modify the transmission and port control tables. The necessary dual
access is effected using the global data area 'Global1' where these two control block tables are declared.

b. Poller

Poller is the driver of the SBC in the concentrator. It polls ports for service requests using the port control block entries as reference. Only active ports, which are identified in a linked list, are polled in each cycle. Periodically, with the time period specified by system maintenance personnel, inactive ports are checked for requests. In addition to the remote login, remote file transfer, and local file transfer requests, net status, port number identification, and passive listen are valid requests. Poller updates the state of each port as it progresses from process to process to attain requested service.

c. TELNET

Once a user has triggered a connection with a remote host, the only function of the concentrator is to pass data between the user and the network. This is done asynchronously. The port from the Z-100 is checked and if data is ready, it is read, stored in memory, and the transfer triggered. If there is data queued for the Z-100, at most one packet is transmitted per cycle to the Z-100. Checks are also performed to allow the user to terminate the process.
The Z-100 acts as a 'dumb' terminal once the process is begun. Each stroke of the keyboard is transferred to the concentrator and data echoed from the concentrator is displayed on the screen.

d. FTP

FTP is very similar to the TELNET process at the concentrator level. Data is simply passed between the Z-100 and the host. One difference is that FTP utilizes two network connections to effect data transfers. The concentrator must manage both connections and coordinate with the Z-100 to determine which line to use when.

The Z-100 is programmed to implement the file transfer protocol layer. FTP operates as a series of commands from the user (active requestor) and replies from the server (inactive servicer) to coordinate transfer parameters and trigger the transfer. Sequencing is presumed and close coordination with the concentrator is required. No provision has been made to allow a remote host to initiate an FTP connection with a Z-100, hence, no server process is coded.

e. Local

For the requirements of local file transfer, the concentrator acts as a packet switcher. A control block for each port identifies which packets are to be sent or received, and when the ports are active, the transfers
take place. Printing is also handled using this packet switching mode.

The software on the Z-100 makes this local transfer process a very powerful tool. Capabilities include message passing, transfer of multiple files to multiple users, passive listening to allow network users access to one library, network status query, and port identification. Minimal foreknowledge is required by a user of this system. In order to transfer files between two computers on the net, both must be operating under the local transfer software and the sender must know the port identification number of the receiver. Many of these features will prove very helpful to instructors and system maintenance personnel as well as students and other system users.

f. TCP

The purpose of these modules is to perform the telecommunication control protocol functions. These functions include opening, closing, and maintaining connections, monitoring and acknowledging packets to prevent data loss, updating the telecommunications control blocks, and interacting with the higher and lower layer protocols. TCP also provides an address to allow identification of separate users within a single network host.
g. IP

The INTERNET Protocol is concerned with the operation of the telecommunications network. The IP address identifies a unique node on the net. On ETHERNET, this address is resolved to select the controller board which will read the packet.

h. ETHERNET

At the physical layer, the protocols are handled by the controller board that must be used to access ETHERNET. The purpose of the software in these modules is to communicate with the controller board to coordinate and effect data transfers to and from the ETHERNET.

i. Library

Network transfer of data requires preservation of all eight bits of each data byte. This precludes representing the bytes as type 'character' and the type integer does not lend itself well to the necessary individual byte manipulation. Many fields within the protocol header are represented as arrays of bytes which require mathematical computation to be performed on them. Special routines to add two and four byte arrays were written and reside in the library package. The library package also contains the routines that provide memory management functions for the single board computer.
VII. CONCLUSIONS

This thesis is mainly an implementation of the TELNET, FTP and Local transfer processes on the NPS LAN. The research objectives of coding in Janus/Ada, navigating the protocols to allow remote login to the VAX UNIX system over ETHERNET, implementing protocol requirements of FTP, allowing single or multiple local transfers, and sharing of local resources were satisfactorily completed as evidenced by the systems in operation. The network communications systems created in the course of this thesis enhance the NPS AEGIS laboratory systems development in several areas. First, it is a demonstration of the ability of Janus/Ada to effectively perform complex operating system and embedded type functions. Second, it allows creation and integration of program code for the NPS AEGIS laboratory system on microcomputers. Third, it gained direct access to DDN, MILNET, and ARPANET from within the development system itself. Finally, it demonstrated the viability of clustering processors to share expensive resources and otherwise enhance data communication and transfer.
APPENDIX A

PROGRAMMING NOTES

A. INTRODUCTION

The objective of this appendix is to provide programmers maintaining the system with information that will be helpful in modifying or adding to this system. The largest and most complicated portion of this system is the process that executes in the concentrator, the LAN controller program. The major areas of the system to be discussed are program compilation and loading, the concentrator program, interface to the concentrator, TELNET program, the FTP program, and the local connection program.

A good source of information about the concentrator from the standpoint of overall design for TCP/IP protocols, and a 'must' reading for anyone doing maintenance on the program, is the SRI INTERNET Protocol Transition Workbook, reference 2 of the thesis.

B. PROGRAM COMPILATION AND LOADING

Under Janus/Ada, the compilation order of packages and specifications is critical to system operation. Included with program listing in the thesis is a listing of the .sub files used to compile the programs. Linking using 'Jlink' will produce an executable .com file for FTP, TELNET, and Local programs executing on the Z-100(under MS-DOS). Loading the concentrator is slightly more complex.

Loading the concentrator program is effected by a program named 'Boot.com'. Boot should be resident on the boot drive of each Z-100 and should execute each time each Z-100 is turned on. The Read Only Memory (ROM) of the 86/12 A has been configured to handshake with the boot program. If the concentrator is not executing the control program and is in the 'reset' mode, boot will transfer the control program to the concentrator. If the control program in the concentrator is executing or the concentrator is turned off, handshaking will preclude a boot attempt.

The control program loaded by 'Boot.com' contains the executable code created by linking the .jrl files from all
concentrator packages. When reprogramming a particular package, if the specification is not changed, then only the changed package need be compiled. If one or more package specifications is changed, the compilation should be accomplished using the .sub file provided with the concentrator software. To produce the controller program, compile concentrator programs (under CPM-86) as necessary, then use Jlink on the package 'Poller'. This will create the file 'Poller.cmd'. Rename 'Poller.cmd' to 'Control.prg' and, convert the CPM-86 .cmd file into MS-DOS format using the program 'Rdcpm', and place the file on the boot disk of each machine (under MS-DOS) along with 'Boot.com'.

C. CONCENTRATOR PROGRAM

The general function of the concentrator can be thought of as passing packets of information from one port to another. Along with the process of passing packets it must maintain the status of the connections as well as managing memory. The concentrator does not contain an operating system, therefore the customary functions that are normally handled by the operating system had to be avoided. Before going into detail about the individual procedures in each packet, you will benefit by a general one on the overall program.

The concentrator begins execution by initializing the data structures to appropriate values, setting appropriate initialization of the UARTS, activates the NI3010 ETHERNET controller board and obtains the ETHERNET physical address from the ETHERNET controller board.

Understanding the data structure is important to knowing the details of the system. Starting with the terminal ports are the Port Control Blocks (PCB) for maintaining the status of the connections with the terminals. The PCB structure is:

```pascal
TYPE pcb_rec IS RECORD
  is_print : BOOLEAN;
  data_prt : INTEGER;
  stat_prt : INTEGER;
  cmd_prt : INTEGER;
  prtQ : INTEGER;
  s_prtq : integer;
  sent : BOOLEAN;
  Pstate : Pstates;
  time_wait : INTEGER;
  act : BOOLEAN;
END;
```
The PCB state is used to determine which process to call to handle transactions going to and from the port. The close state is just that, the terminal connected to the port is inactive as far as the concentrator is concerned. The terminal may very well be executing an application program that does not require the use of the concentrator. When the users desire to do some networking, they must execute one of the programs to interact with the concentrator. The interaction begins by the terminal sending a control code down to the concentrator specifying what the users desires is. The concentrator reacts by sending the same control code back and changes the state of the PCB. Normally one state leads to another, for instance, when a TELNET process is initiated by the terminal, the PCB state goes from close to r_init. The next information expected from the terminal is the address of the destination host. Once the address is received the connection is attempted by the concentrator.

When the foreign host connects with the concentrator, the state of the PCB is changed from r_init to rlogn. The following is a summary of the state transitions:

```
closed
  +---(receive rlogn)----+ | +---(receive loc)---+
  |                     | +---+---+
  |                     |       |
  |                     | (receive ftp) |
  | r_init              | f_init   | (receive lsn)  | l_init
gestab)             | (estab)   | lsn      | (estab)
  |                     |           |           | |
  | rlogn               | ftp       | (estab)----+ |
  |                     |           |           | |
  | (disconnect)       |           |           | |
  | closing            |           |           |
  |                   |           |           |
  | ++-(PCB cleared)---|           |
  |                   |           |           |
  | closed            |           |           |
```
These are the fields in the PCB:

Is_print - A boolean that initializes that PCB to state lstn if the boolean is true.

Data_prt and stat_prt - Contains the port address of the data and status registers on the respective UART.

PrtQ and s_prtQ - Indexes to memory blocks in the particular queue. The prtQ is the primary queue and s_prtQ is the secondary queue. During TELNET communications only the prtQ is used. During FTP communications both queues are used. During local connections the s_prtQ is used to queue a memory block and the prtQ is used to count the characters between tabs for printing purposes. For a more thorough discussion of how the queues are implemented see the discussion on the memory management table.

Sent - A boolean used to remember if a packet was sent or not. It is also used as a flag in local connections.

Time_wait - is a loop counter for timing out a connection once a certain state is reached.

Act - A Boolean used to specify either an active or passive connection (see the TCP/IP handbook for details).

L_prt_ad and s_prt_ad - Field to store the TCP addresses are stored in the PCB. These are used to make the connection between the PCB and TCB. More than one TCB can be associated with one PCB.

sec_act - A boolean to indicate if a second connection is made to the same port. Used only in FTP when a primary connection is made and then a secondary connection for passing file data.

loc_con - Used as a pointer to link PCBs in the same 'group' (see ch 5 of [HART/YAS86] for group discussion) so that local broadcast packets are easily sent to all members of the group.

dst_ad - A record containing an IP address and a TCP address of a destination host.

dst_ad_rcv - A boolean to know if the destination address has been received or not.

pcb_ptr - A pointer to link all the active PCBs together to speed the polling processes. The inactive ports are polled once out of a value equal to 'loops_to_poll' (recommended: 1000).

snd - An array of flag bits that every terminal marks to indicate that a packet is ready at the respective bit position terminal. For example, bit 7 is set when terminal 7 has a packet to send that particular terminal. The packet is stored at the originator's s_prtQ. These flags are used only during local connections.
ack - An array of flag bits exactly like 'snd' above, only these bits keep track of all terminals that need to acknowledge receipt of a packet stored at that PCB.

When communications occur between a terminal and one of the hosts on the ETHERNET, the TCP/IP protocol is used to pass packets. TELNET, FTP and local all use the PCBs to maintain status of each port. TELNET and FTP use the TCP layer (see ch 2 of the thesis) to maintain status of the connection across ETHERNET. To store the state of communication a Transmission Control Block is used (see the TCP/IP handbook for details). The following fields are stored in the TCB:

prt_num - An integer index for the associated PCB number. This is the means of relating a TCB to a PCB when a packet is received. This field is set to 99 when no connection exits.

Tstate - Maintains the state of the particular connection, if there is one (see the TCP/IP handbook for details).

loc_sock - A record containing the local TCP and IP addresses of a connection.

rem_sock - A record containing the remote TCP and IP addresses of a connection.

snd - A record containing the necessary information about proper sequencing of packets over the ETHERNET.

rcv - A record similar to 'snd' above (see the TCP/IP handbook for details on both snd and rcv).

retrnsQ - An integer containing a memory block for the beginning of the retransmission queue. If a packet is not acknowledged by the receiving host after a number of times around the polling process, it is considered timed out and is retransmitted again.

Address resolution is a means of finding the physical ETHERNET address of a foreign host. Once the address is found it is placed in a table along with its IP address. The table also contains a value to identify how recent an address is should the table becomes full and one of the addresses is removed to allow room for another. There is more on address resolution in RFC826.

Eth_pck - This is a memory block much like the TCP/IP memory blocks only the fields are different. Two independent implementations cause a particular problem in handling address resolution packets. (1) All the memory blocks are identical; (2) Any packet received is put in any one of the standard memory blocks. The Ada language does not have overlay capability because of its strong typing. To allow a memory block to be transformed into another type an assembly language routine is called (convert block) that
does nothing but jump to a high level routine that expects a different type memory block. Once this is done all the fields in the memory block can be addressed normally. The following is a comparison of the two kinds of memory blocks:

<table>
<thead>
<tr>
<th>TCP/IP</th>
<th>ETH_PCK</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ETHERNET HEADER</strong></td>
<td><strong>ETHERNET HEADER</strong></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ver</td>
<td>serv</td>
</tr>
<tr>
<td>len(1)</td>
<td>len(2)</td>
</tr>
<tr>
<td>id(1)</td>
<td>id(2)</td>
</tr>
<tr>
<td>flag(1)</td>
<td>flag(2)</td>
</tr>
<tr>
<td>ttl</td>
<td>prot</td>
</tr>
<tr>
<td>ip_cksum1</td>
<td>ip_cksum2</td>
</tr>
<tr>
<td>ip_scr(1)</td>
<td>ip_scr(2)</td>
</tr>
<tr>
<td>ip_scr(3)</td>
<td>ip_scr(4)</td>
</tr>
<tr>
<td>ip_dst(1)</td>
<td>ip_dst(2)</td>
</tr>
<tr>
<td>ip_dst(3)</td>
<td>ip_dst(4)</td>
</tr>
<tr>
<td>scr(1)</td>
<td>scr(2)</td>
</tr>
<tr>
<td>dst(1)</td>
<td>dst(2)</td>
</tr>
<tr>
<td>seq(1)</td>
<td>seq(2)</td>
</tr>
<tr>
<td>seq(3)</td>
<td>seq(4)</td>
</tr>
</tbody>
</table>

The memory management table (MMT) enables full management of memory by use of pointers. The memory blocks are all equal in size (576 bytes). The MMT is an array of integers which is indexed by integers. The indexes correspond to the indexes of the memory blocks. For example, memory block number 7 would be index number 7 in the MMT. The integers stored in the MMT are pointers (indexes) to the next memory block in succession. If, for instance, a queue is used to store a number of packets, the
first memory block in the queue would be the first packet. The second memory block (or packet) could be found by checking the MMT with the index of the first one to get the second. The following is the way the MMT is initialized during boot-up:

```
free_blk---+------+
       1  |  2  |
       +------+
       2  |  3  |
       +------+
       3  |  4  |
       +------+
        .   |
        .    |
        .    |
        +------+
       N  |  0  |
       +------+
```

Free_blk is a pointer to the next available memory block. When the first memory block is used the free_blk pointer is moved to the next one as indicated in the MMT.

Appendix B will focus on individual procedures and how they function. The discussion will follow packet by packet and cover each procedure in each packet. The following is an outline of all the packets and what procedures are in each:

I. Poller.
   A. Poll.
   B. Rem_init.
   C. Rlog.
   D. Ftp.
   E. Initialize

II. Locxfer.
   A. Loc_init
   B. Loc

III. TCPsend
   A. TCP_open
   B. TCP_send
   C. TCP_close
   D. Check_retrnsQ

IV. IPsend
   A. IP_send
V. ETHsend
   A. ETH_send

VI. RCV
   A. Ntrpt_hdl

VII. ETHrec
   A. Eth_pck

VIII. IPrec
   A. IP_rcv

IX. TCPrec
   A. TCP_rcv
   B. pcb_clsing
   C. Conv_blk_snd
   D. Send_ack
   E. Update_retrns

X. PCBrec
   A. PCB_rcv
   B. Adv_PCB_state

XI. Convblk
   A. Conv_blk

XII. Ntrp_thd
   A. Assy_ntrpt_hdl
   B. Init_ntrpt

XIII. Lib
   A. Get_memory
   B. Give_memory
   C. Perf_cmd
   D. Trn_pck
   E. Resolve_ad
   F. Get_TCB_ndx
   G. PCB_cls
   F. PCB_abort
   H. T_CB_cls
   I. Ac_tivate_prt
   J. Giv_e_status

XIV. Assylib
   A. Cksum
   B. Wr_ad
   C. Outp rt
   D. Arr_to_int
   E. Ohi
   F. Olo
   G. Inp rt
   H. Otstbit
I. Oclrbit
J. Osetbit
K. Gt_equ
L. Lt_equ
M. Inc_arr
N. Grtr_of
O. Upper_nibble
P. Inc_nxt_prt_ad
Q. Prt_hex
R. Send_trns
S. Get_trns
T. Oput
U. Onew_line
V. Xsum

XV. FTP
A. FTP

XVI. TELNET
A. TELNET

XVII. LOCAL
A. Handle_kybk_input
B. Handle_incoming_packet
C. Established

XVIII. ASMLIB
A. Byte_to_char
B. Byte_to_chr
C. Prntdata
D. Getch
E. Delete_file
F. Create_file
G. Open_file
H. Write_file
I. Close_file
J. CKsum
K. Setdma
L. NO_echo
M. Search_first
N. Search_nxt
O. Get_trns
P. Send_trns
Q. Read_file
R. Capital
S. Lower_case
T. Arr_to_strg
U. Conv_byt
V. Two_bytes
W. Current_dsk
X. Get_strg
Y. Prnt_buf
XIX. LIB(Z-100)
   A. Send_cmd
   B. Send_data
   C. Get_data
   D. User_options
   E. Get_data_line
   F. Process_reply

XX. FUNCS
   A. Get_opt
   B. Get_password
   C. Get_username
   D. Get_portnum
   E. Get_filename
   F. Get_paramet

XXI. GET_IP
   A. GET_ADDR

XXII. LIBRARY
   A. Activate
   B. Deactivate
   C. Get_memory
   D. Arr_to_int
   E. Give_memory
   F. Put_In_trnsQ
   G. Inc_arr
   H. Prompt
   I. Add_to_Q

XXIII. FILEXFER
   A. Send_file
   B. Create_FCB
   C. Receive_file
   D. Close_FCB
   E. Send_dir
   F. Information
APPENDIX B

PROGRAM MAINTENANCE MANUAL

A. PACKAGE poller

1. CONFIGURATION
   a. Language - Janus/Ada
   b. Compiler version - 1.47
   c. Linker version - 1.47
   d. Target hardware - Intel 86/12A SBC
   e. Operating system - CP/M-86
   f. Package description:

Poller - The poller package consists of the initialization sequences of the entire program and polling routines that poll each terminal for transfer of data or execution of commands. Poller begins by setting up the data structures and initializing the hardware such as the NI3010 ETHERNET controller board and the terminal UARTS. There are 4 port addresses for the RS232 ports, each consecutive port are addressed next to the previous, however, a second set of port addresses, for interrupt use, are addressed after the first. See table 2.4 of [HARTMAN/YASINSAC 86] for port addresses on each board.

A total of 64 address locations are used for each RS232 controller board. In order to accommodate all the port addresses of the three RS232 boards as well as the iSBC86/12 and NI3010 boards 16 bit addresses had to be used. The RS232 port addresses range from 0000 hex to 01BF hex. The iSBC86/12 uses port address 00C0 hex to 00FF. The NI3010 board uses port address 00B0 hex to 00BF hex.

<table>
<thead>
<tr>
<th>Port Address Table (in Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000-00AF</td>
</tr>
<tr>
<td>00B0-00BF</td>
</tr>
<tr>
<td>00C0-00FF</td>
</tr>
<tr>
<td>0100-013F</td>
</tr>
<tr>
<td>0140-017F</td>
</tr>
<tr>
<td>0180-01BF</td>
</tr>
<tr>
<td>01C0-FFFF</td>
</tr>
</tbody>
</table>
Package Poller initializes all the PCBs to either a printer or terminal by setting the boolean is_print to true or false respectively. When the NI3010 ETHERNET controller board is initialized it is commanded to perform command 'receive status'. From the 'receive status' command the physical address of the controller board is obtained, enabling changing of this board without affecting operation of the system (ensure system is turned off before changing board). See the manufacturer's manual for more information on the NI3010 board. The ETHERNET board is set to receive packets over the ETHERNET. Only packets that are addressed to the physical address of the ETHERNET board or 'broadcast' packets are captured for processing even though the board has capabilities of capturing other packets.

2. SUBROUTINES

a. Poll

(1) Type - Procedure.

(2) Purpose - Poll all serial ports and call appropriate processes to handle the particular state of each port.

(3) Description of parameters - no parameters.

(4) External references:
(a) Get_tcbndx
(b) Tcb.cls
(c) Tcp_close
(d) Givememory
(e) Send_trns
(f) Inprt
(g) Outprt
(h) Otstbit
(i) Rem_init
(j) Rlog
(k) Ftp
(l) Loc_init
(m) Loc
(n) Check_retrnsq
(o) Give_status

(5) Process description:

The polling routine is an infinite loop that is divided into two phases. In the first phase only the ports that are active are polled. An active port is one whose state is anything other than 'closed' or 'listen'. These particular ports are selected by a linked chain of ports beginning at the 'head' PCB. When a port becomes active it is inserted into the linked chain and when inactive it is removed. The polling process simply follows the linked chain until it returns to the 'head' PCB. The state of the PCB is the important field for the poller. The state determines what process to call on (if any) to handle the connection. The only state that does not require calling another procedure is the closing state, when all data is
being flushed out of the queues, which, once done, the state is returned to closed.

The second phase of the polling process happens once every so many loops depending on the value of 'loops_to_poll' (constant 1000). During this phase all the closed or listening ports are checked for any control codes for which to change states. A garbage collection routine is also included in this phase.

b. Rem init

(1) Type - Procedure.

(2) Purpose - To obtain the foreign address from the particular terminal passed as a parameter. Once the address is obtained, the three-way handshake is initiated.

(3) Description of parameters -
    (a) prtnum - port number.
    (b) rem_tcp_addr - the two byte TCP address of the remote socket.

(4) External references
    (a) Inprtr
    (b) Otstbit
    (c) Outprtr
    (d) Tcp_open
    (e) Get_tcb_ndx
    (f) Pcb_cls

(5) Process description:

To establish a connection with a foreign host a sequence called the 'three way handshake' is initiated. The rem_init procedure is used for the handshake which is used in TELNET and FTP connections. When a terminal commands a TELNET or FTP process, then subsequent polls of that port calls rem_init to get the foreign INTERNET Protocol (IP) address from the port. The address is a 4 byte address which is concatenated with a 2 byte TCP address of the well-known socket. For TELNET the well-known TCP socket is 0017 hex and for FTP it is 0015 hex. The well-known TCP socket is passed in as a parameter to rem_init. The address it depends on whether a TELNET process or FTP process is desired. Once rem_init has the entire socket (TCP/IP address) it calls TCP_open with the socket. The sent parameter returns whether the packet was actually sent or not (if the physical address of the foreign host is not known then the packet is not sent and an 'address resolution' packet is sent instead. Rem_init will attempt again after a time_wait period, or close out the port if no reply is received from the foreign host. The state of the connection (PCB state) will be change by receipt of a packet from the foreign host. The state is changed in procedure 'adv_PCB_state'.

90
c. Rlog
(1) Type - Procedure.
(2) Purpose - To process and send data from the terminals to a remote host. Any control codes sent by the terminal is also processed.
(3) Description of parameters -
    (a) prt_num - port number.
(4) External references:
    (a) Inprt
    (b) Otstbit
    (c) Outprrt
    (d) Tcp_close
    (e) Get_tcbndx
    (f) Pcb_cls
    (g) Tcb_cls
    (h) Tcp_send
    (i) Get_trns
    (j) Give_memory
    (k) Get_memory
    (l) Send_trns

(5) Process description:
The TELNET process was formally called 'remote login' or rlog, hence the name rlog. When the three way handshake is complete for a TELNET process, subsequent polls of the PCB will call 'rlog'. Rlog does three types of checks, (1) checks for a control code from the terminal (2) checks for data from the terminal and (3) checks for data to the terminal. A control code will be found by checking the status port for 'receive ready'. To determine if the terminal is trying to send something, the status port is checked for Data Set Ready (DSR). If data is to be sent to the terminal then the prtQ field will contain a value other than zero (the value being what memory block is held in the queue).

d. FTP.
(1) Type - Procedure.
(2) Purpose - To process all data and control codes to and from the terminals in an FTP connection.
(3) Description of parameters -
    (a) prt_num - port number.
(4) External references:
    (a) Inprt
    (b) Otstbit
    (c) Outprrt
    (d) Tcp_close
    (e) Send_trns
    (f) Pcb_abort
    (g) Tcp_open
    (h) Tcp_send
    (i) Get_trns
(j) Give_memory
(k) Get_memory

(5) Process description:
The FTP process is a bit more complicated than the TELNET process since two connections must be handled at the same time. One connection is used to control the FTP process between the hosts and the second connection is used to pass the data. To implement a dual connection to a single port we use a boolean value (sec_act) to designate single or dual connections. If a dual connection exists then no data is passed to the terminal over the control connection until the data connection is terminated. The basic checks as in rlog are also made (1) check for control code (2) check for data from port (3) check for data to port. To determine if data from the terminal is for the control or data connection a one byte header is used which designates either control, data, or other.

e. Initialize_mem.
   (1) Type - Procedure.
   (2) Purpose - To initialize certain portions of memory at the beginning of execution and during periods when no terminals are active.
   (3) Description of parameters - none.
   (4) External references: NA.
   (5) Process description:
   To set up the data structures at the beginning of execution and also as a housekeeping function during periods when no terminals are active, the initialize mem routine is used to reset everything back to an inactive state, ensuring all memory blocks and TCBs are available for use.

B. PACKAGE locxfer.
1. CONFIGURATION.
   a. Language - Janus/Ada.
   b. Compiler version - 1.47.
   c. Linker version - 1.47.
   d. Target hardware - Intel 86/12A SBC.
   e. Operating system - CP/M-86.
   f. Package description:
   Package Locxfer handles all terminals in the states of 1_init (local initial) and local. Local connections can be from any terminal in the 'local' state to any in the 'local' or '1stn' state. 'Group' connections are simultaneously maintained, but the use of a group connection is solely for broadcast packets. For instance, if you want to send a message from terminal 5 to terminal 12 then you can do so if 5 is in the 'local' state and 12 is in either 'local' or '1stn'. If, however, you want to
send the same message to terminals 12, 13, 14 and 15, without repeating yourself, then the terminals 5, 12, 13, 14, and 15 must be in a 'group' connection. What designates a 'group' are the links that connect PCBs together using the loc_con field in the PCB. Every PCB that is in a 'local' state has a link to another PCB. The links are set up by the port number that is passed down from the terminal designating what terminal to link to. The queues for local connections, unlike TELNET and FTP, are maintained in the terminal themselves. The terminal sending packets out may have several in its transmission queue. Only one packet at a time resides in a PCB for transfer to another terminal. Another difference with local transfers and TELNET or FTP is that packets are stored at the originator's PCB and not at the destination PCB.

2. SUBROUTINES.
   a. loc_init.
      (1) Type - Procedure.
      (2) Purpose - To obtain the destination port number for connecting two ports together.
      (3) Description of parameters -
          (a) prt - port number.
      (4) External references:
          (a) Inprt
          (b) Otstbit
          (c) Activate_pr
          (d) Outprt
          (e) PcbCls
      (5) Process description:

      When a terminal initiates a local connection it first sends the control code 'code_loc'. The polling routine responds by sending back 'code_loc' and sets the state to '1 init'. In loc_init the desired destination is expected from the terminal. That destination is sent to the concentrator like a control code. When loc_init receives the number (one byte) it error checks it, then reacts according to the state of the destination PCB:

      lstn - switches both PCBs to local state and forms a 'group', outputs null to both.
      local - changes the single PCB to local state and inserts it into the 'group', outputs null.
      1 init - changes the single PCB to lstn, outputs null.
      others - changes the single PCB to lstn.

      The null is a byte of all zeros to trigger the terminals into another phase of their execution. The null byte was chosen so a printer terminal would not print a character when receiving it.
b. loc.

(1) Type - Procedure.
(2) Purpose - To get and send data packets to and from ports which are in local connections. Local connections are established as well as the handling of control codes.
(3) Description of parameters: 'Prt' is the port number.
(4) External references:
   (a) Inprt
   (b) Otstbit
   (c) Give_memory
   (d) Pcb_cls
   (e) Give_status
   (f) Outprt
   (g) Get_memory
   (h) Get_trns
   (i) Send_trns
   (j) Osetbit
   (k) Oclrbit

(5) Process description:
During local transfers three conditions are checked on each poll of the terminal (1) control codes sent from the terminal (2) any packet to be cleared from the PCB and (3) any packets being sent to the terminal. Number 1 is handled similar to every other routine handling control codes, a case statement for all the options. Numbers 2 and 3 are handled very similarly by use of a single bit for each terminal. Two fields in each PCB are used to track whether any packets are ready to be sent to the terminal and if a packet from the terminal has been sent to all the appropriate destinations. In brief, this is what happens: when a packet comes down from a terminal it is stored in the s_prtQ, but only one packet at a time can be stored there unlike the FTP process. The first byte of the packet indicates its destination. The bit corresponding to that destination is set in the ack field. The bit corresponding to the source is set in the snd field of the destination PCB. Therefore, the destination will know it has a packet to send its respective terminal by the bit in its PCB and the sender will know when the packet has reached its destination by the same bit being reset in its own PCB. This method works for one destination or many destinations. In the case of a broadcast packet, the bits in each PCB that is in the link or 'group' is set similarly. The two bits mentioned are set for transmission of a packet, those same two bits are reset by the receiving PCB once the transmission takes place.
Example bit arrangement for transmission from terminal 11 to terminal 5:
C. PACKAGE TCPsend.

1. CONFIGURATION.
   a. Language - Janus/Ada.
   b. Compiler version - 1.47.
   c. Linker version - 1.47.
   d. Target hardware - Intel 86/12A SBC.
   e. Operating system - CP/M-86.
   f. Package description:

   Package TCPsend implements the TCP protocol (see the
   TCP/IP handbook for details) for the sending of packets to
   foreign hosts. A TCP connection consists of three phases
   (1) handshaking to establish a connection, (2) established
   and (3) closing out a connection. Associated with the
   three phases are several states as described in the TCP/IP
   handbook. The data sending portion of the three phases are
   handled by the following three procedures:

2. SUBROUTINES.
   a. TCP_open.
      (1) Type - Procedure.
      (2) Purpose - To prepare the TCP protocol of a
         'syn' packet to a foreign host as part of the three-way
         handshake.
      (3) Description of parameters -
         (a) prt - port number.
         (b) foreign_sock - the IP and TCP
            addresses in a record structure.
         (c) act - a boolean to set an active or
            passive connection.
         (d) loc_tcp_ad - the local TCP used in
            the connection is output to the calling routine.
         (e) sent - a boolean to signal whether
            lower level processes sent the packet, ie. ETHERNET address
            of the foreign host is known.
      (4) External references -
         (a) inc_nxt_prt_ad.
         (b) get_TCBndx.
         (c) get_memory.
         (d) cksüm.
         (e) ip_send.
JANUS/ADA IMPLEMENTATION OF A STAR CLUSTER NETWORK OF PERSONAL COMPUTERS

R L HARTMAN ET AL.

MONTEREY CA JUN 86
(f) give_memory.
(g) inc_arr.

(5) Process description:
When a connection is requested from a higher level protocol (TELNET or FTP) a 'SYN' (synchronize) packet must be sent to start the handshake process. The 'syn' is a bit set in the control field of the protocol. A local TCP address is obtained to use for the connection and stored in the TCB (transmission control block). The local TCP address must not be any of the reserved addresses, therefore, addressing begins at 0400 hex. Each subsequent connection will be incremented from the previous until FFFF hex at which time the address starts over at 0400 hex. A TCB table is created and initialized appropriately. If the connection is active then a memory block is obtained and the TCP portion of the block is completed. The block is sent to 'IP_send'. An active connection is when the foreign host is known and the connection must be initiated locally. A passive connection is one which waits for the other host to initiate the handshake. In either case, a TCB must be created. Even though no data from the user is sent with this procedure, the 'syn' bit is considered data.

b. TCP_send.
(1) Type - Procedure.
(2) Purpose - To prepare the TCP protocol of data packet to be sent to a foreign host.
(3) Description of parameters -
   (a) indx - a memory block index.
   (b) data_len - amount of data sending.
   (c) tcp_ad - Local TCP address used to locat the TCB.
   (d) sent - a boolean to signal whether lower level processes sent the packet, ie. ETHERNET address of the foreign host is known.
(4) External references -
   (a) get_TCB_ndx.
   (b) inc_arr.
   (c) cksüm.
   (d) ip_send.
(5) Process description:
When a connection is established then all data that is sent to the foreign host is sent by TCP send. Appropriate information is obtained from the TCB to complete the TCP portion of the packet to be sent. Once the packet is sent the TCB is updated to reflect current status. The packet is then placed in the retransmission queue. For a good explanation of the necessary protocol see the TCP/IP handbook.
c. TCP close.
   (1) Type - Procedure.
   (2) Purpose - To prepare the TCP protocol of a 'fin' packet to be sent to a foreign host.
   (3) Description of parameters -
       (a) tcp_ad - local TCP address used to locat the TCB.
   (4) External references -
       (a) get_TCB_ndx.
       (b) get_memory.
       (c) cksum.
       (d) ip_send.
       (e) give_memory.
   (5) Process description:
       When a connection is to be closed a 'FIN' is Sent much like a 'SYN' at the beginning. This procedure sends a fin and changes the state of the connection appropriately. Like TCP_send, all the necessary protocol information is inserted in the outgoing packet.

d. Check_retransQ.
   (1) Type - Procedure.
   (2) Purpose - To check packets on the retransmission queue for retransmission if the foreign host does not acknowledge receipt.
   (3) Description of parameters -
       (a) tcp_ad - local TCP address used to locat the TCB.
   (4) External references -
       (a) get_TCB_ndx.
       (b) trn_pck.
   (5) Process description:
       Even though ETHERNET is highly reliable there is no guarantee that a packet gets to its destination. The TCP/IP protocol allows for lost packets and is able to recover from it. One of the requirements for this amount of robustness is a retransmission queue to retransmit packets that have not been acknowledged. To implement the queue we save all outgoing packets sent by TCP_send in a queue in the TCB. When acknowledgements come in the packets are removed from the queue. During the polling of the ports every 'loops to poll' is used to check the retransmission queue for the connections in TELNET and FTP. If a packet remains in the queue for 10 of these checks then it is retransmitted.

D. PACKAGE IPsend.
   1. CONFIGURATION.
      a. Language - Janus/Ada.
      b. Compiler version - 1.47.
      c. Linker version - 1.47.
      d. Target hardware - Intel 86/12A SBC.
e. Operating system - CP/M-86.
f. Package description:

Package IP_send is used to implement the INTERNET Protocol in outgoing packets. The IP protocol is not a major influence in the operation of our connections since it is designed mainly for crossing to different networks by breaking up packets into smaller ones or combining packets into bigger ones. Since we are using only ETHERNET the IP portions has no significance other than having to implement it because the other hosts on ETHERNET use it.

2. SUBROUTINES.
   a. IP_send.
      (1) Type - Procedure.
      (2) Purpose - To implement the IP protocol of
in outgoing packet.
      (3) Description of parameters -
         (a) inx - memory block index.
         (b) rslt - a boolean to signal whether
lower level processes sent the packet.
      (4) External references -
         (a) arr_to_int.
         (b) cksum.
         (c) eth_send.
      (5) Process description: see package
description.

E. PACKAGE ETHsend.
   1. CONFIGURATION.
      a. Language - Janus/Ada.
      b. Compiler version - 1.47.
      c. Linker version - 1.47.
      d. Target hardware - Intel 86/12A SBC.
      e. Operating system - CP/M-86.
      f. Package description:

Package ETHsend is used to implement the ETHERNET protocol of a packet. To send a packet over ETHERNET an ETHERNET address must be used. A table is maintained with currently known addresses. ETH_snd checks the table for
the ETHERNET address. If found, it sends the packet out,
if not found it sends out a special broadcast packet to all
hosts on the ETHERNET requesting the particular host with
the IP address listed to report back its ETHERNET address.
If the latter case occurs then the original packet is
effectively lost and will have to be sent again later.
This normally occurs when connecting to a host for the
first time since the system was re-booted.

The control program executing in the concentrator can
be thought of as two independent processes, one for sending
packets out and polling the terminals, the other for
receiving packets from ETHERNET and distributing them to
the appropriate places. The receiving process, however, sends acknowledgements out on ETHERNET as well.

F. PACKAGE rcv.
   1. CONFIGURATION.
      a. Language - Janus/Ada.
      b. Compiler version - 1.47.
      c. Linker version - 1.47.
      d. Target hardware - Intel 86/12A SBC.
      e. Operating system - CM/M-86.
      f. Package description.

   Package Rcv is the first high-level routine that receives packets from the ETHERNET. Ntrpthdl (interrupt handler) is the procedure at the interrupt vector whenever an interrupt occurs. The only interrupt implemented is that of the NI3010 controller board. There are four cases for an interrupt by this board (1) upon receipt of a packet (rcv_pck), (2) when the DMA transfer is complete after receiving a packet (rcv_DMA_dn), (3) when the DMA transfer is complete when transmitting a packet (tx_DMA_dn), and (4) when the DMA transfer is complete when transmitting a packet after having already interrupted from a rcv_DMA_dn (disable). The difference between 3 and 4 will be discussed shortly.

   The programming of the NI3010 board for receiving and transmitting packets is discussed in the manufacturer's hardware manual. We have modified the manufacture's recommended algorithms in order to enhance concurrent processing. For instance, rather than have the concentrator idly loop waiting for a DMA transfer to end we continue processing other procedures until an interrupt occurs to let us know the process is complete. To implement this strategy fully an interrupt labeled 'disable' was created to allow transmissions from ether side of the interrupt (remember the two independent processes in the controlling program). Lets discuss these four different interrupts separately:

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>Discussion</th>
</tr>
</thead>
<tbody>
<tr>
<td>rcv_pck</td>
<td>The NI3010 is initialized to start with this type of interrupt. If a packet is received an interrupt occurs. In handling this type, a memory block is gotten for which to do the DMA transfer to. The address of the memory block is sent to the NI3010 board and the next type of interrupt is enabled, rcv_DMA_dn.</td>
</tr>
<tr>
<td>rcv_DMA_dn</td>
<td>Once this interrupt occurs the received packet can be looked at to determine where to send it for processing. As a result of this packet, another packet may be transmitted out</td>
</tr>
</tbody>
</table>
(ie. acknowledgement of the former). Therefore, upon return to this procedure one of three cases could be occurring (1) tx_DMA_dn (2) rcv_pck or (3) disable (no change). For the latter two the able register is set to receive another packet. This cannot be done before this point else multiple interrupts could cause the runtime stack to overflow.

_DMA_dn

When a memory block has been DMA'd to the NI3010 board it is ready to be sent by the ld_snd (load and send) command.

able

The only interrupt that can occur during execution of the interrupt side of the independent processes is a tx_DMA_dn. If tx_DMA_dn is used instead of disable, however, a second interrupt could occur on top of the first. To prevent this disable loads and sends the packet but does not enable receipt of another packet.

note: The variable 'ntrpt' stores what the NI3010 board was last enabled to interrupt to.

2. SUBROUTINES.
   a. ntrpt_hdl.
      (1) Type - Procedure.
      (2) Purpose - To handle NI3010 interrupts.
      (3) Description of parameters - none.
      (4) External references -
         (a) outp_r.
         (b) perf_cmd.
         (c) get_memory.
         (d) wr_ad.
         (e) ip_rec.
         (f) conv_blk.
         (g) give_memory.
      (5) Process description - see package description.

G. PACKAGE ETHrec.
   1. CONFIGURATION.
      a. Language - Janus/Ada.
      b. Compiler version - 1.47.
      c. Linker version - 1.47.
      d. Target hardware - Intel 86/12A SBC.
      e. Operating system - CM/M-86.
      f. Package description.

Package ETHrec is used strictly to respond to address resolution packets. A packet can be of two types (1) a remote host asking for our ETHERNET address or (2) a remote host responding to our asking it what its ETHERNET address is. The ad_tbl is updated appropriately upon receipt of the latter.
H. PACKAGE IPrec.
1. CONFIGURATION.
   a. Language - Janus/Ada.
   b. Compiler version - 1.47.
   c. Linker version - 1.47.
   d. Target hardware - Intel 86/12A SBC.
   e. Operating system - CM/M-86.
   f. Package description.

Package IPrec checks the IP protocol of incoming packets for appropriate fields being correct including the local IP address. If everything is correct it passes the packet on up to TCPrec.

I. PACKAGE TCPrec.
1. CONFIGURATION.
   a. Language - Janus/Ada.
   b. Compiler version - 1.47.
   c. Linker version - 1.47.
   d. Target hardware - Intel 86/12A SBC.
   e. Operating system - CM/M-86.
   f. Package description.

Package TCPrec is the most complex package in the control program due, primarily, to all the checks it must make for any incoming packet. The best source of information about what checks are made is the Internet Protocol Transition Workbook. Two checks that are omitted in our implementation are the precedence and security checks. In addition, we do not test the checksum fields since ETHERNET has a reliable CRC field that assures proper transmissions. The basic functions of TCP_rec is to update the TCB table of the respective connection, send proper acknowledgements and send any data up to the respective PCB for that connection.

2. SUBROUTINES.
   a. Conv_blk_snd.
      (1) Type - Procedure.
      (2) Purpose - To take a received packet and reverse all the fields for transmission back to the sender.
      (3) Description of parameters -
         (a) blk - memory block index.
         (b) sent - boolean to indicate if the packet was sent.
      (4) External references -
         (a) upper_nibble.
         (b) cksum.
         (c) ip_send.
         (d) give_memory.
      (5) Process description:

This procedure uses the memory block of an incoming packet to send a reply by changing the destination fields to source fields and vice versa.
b. Send_ack.
   (1) Type - Procedure.
   (2) Purpose - To send an acknowledgement to a received packet.
   (3) Description of parameters -
       (a) blk - memory block index.
       (b) nr - TCB index.
       (c) sent - boolean to indicate if the packet was sent.
   (4) External references -
       (a) cksum.
       (b) ip_send.
       (c) give_memory.
   (5) Process description:
       To acknowledge receipt of a packet this procedure takes the appropriate fields out of the TCB to fill out a packet that has no data but simply acknowledges new data received.

c. PCB_clsing.
   (1) Type - Procedure.
   (2) Purpose - To set the PCB to closing which allows clearing of the receive queues and termination of a connection.
   (3) Description of parameters -
       (a) prn - port number.
   (4) External references - none.
   (5) Process description:
       PCB_clsing - Upon receipt of a 'FIN' the PCB state is set to closing. Closing allows any undelivered data in the port queue to be sent up to the terminal.

d. update_retrnsQ.
   (1) Type - Procedure.
   (2) Purpose - To clear out any acknowledged packets from the retransmission queue.
   (3) Description of parameters -
       (a) nr - TCB index.
       (b) ack - lastest acknowledgement number.
   (4) External references - give_memory.
   (5) Process description:
       This procedure loops through the linked list of memory blocks on the retransmission queue looking for all packets that have been acknowledged with the lastest acknowledgement number.

J. PACKAGE PCBrec.

1. CONFIGURATION.
   a. Language - Janus/Ada.
   b. Compiler version 1.47.
   c. Linker version 1.47.
d. Target hardware - Intel 86/12A SBC.
e. Operating system - CP/M-86

2. SUBROUTINES.
   a. PCB_rcv.
      (1) Type - Procedure.
      (2) Purpose - To queue up the received data
      packets for further transmission to the respective port.
      (3) Description of parameters -
          (a) inx - memory block index.
          (b) prt - port number.
      (4) External references - none.
      (5) Process description: Receipt of a packet
      containing data to be sent to a terminal requires storing
      the memory block containing the packet in a queue so that
      subsequent polls of the PCB in 'poll' will find the packet
      and send it to the terminal. Since two connections can
      exist at the same time to the same port, PCB_rec must
      determine if the data is for the first connection or
      second. Two queues are used, 'prtQ' and 's_prtQ'.
   b. Adv_PCB_state.
      (1) Type - Procedure.
      (2) Purpose - To change the PCB state to
      either rlogn or rftp.
      (3) Description of parameters -
          (a) nr - port number.
      (4) External references - none.
      (5) Process description:
          Adv_PCB_state - When a packet is received with a 'SYN'
          bit set then this procedure is called. If the PCB state is
          r_init or f_init then the state is advanced to their
          respective established state.

K. PACKAGE CONVBLK

1. CONFIGURATION.
   a. Language - Janus/Assembly.
   b. Compiler version 1.4.6.
   c. Linker version 1.4.7.
   d. Target hardware - Intel 86/12A SBC.
   e. Operating system - CP/M-86
   f. Comments - Package Convblk is the smallest and
      simplest package in this system. It is used to implement
      an overlay while evading the strong typing of the Ada
      language.

2. SUBROUTINES
   a. Conv_blk
      (1) Type - Procedure.
      (2) Purpose - To allow an overlay type
      conversion on the data structure 'mem' allowing two
different packet formats to be handled by the same physical memory area.

(3) Description of parameters -
   (a) 'Nr': The memory block array index to be used as an ETHERNET packet.
(4) External references: Eth_rcv.
(5) Process description: When a packet is received it can be one of two kinds. The memory blocks which a packet is put in is a record with fields for one of the two types. To transform a memory block to the other type a simple jump command is executed in assembly language. A procedure calls the assembly routine with a memory block as the parameter. The assembly routine jumps to another high-level Ada routine that expects a memory block of the other type to be passed in. None of the procedures know the difference and the transform is made.

L. PACKAGE NTRPTHD

1. CONFIGURATION.
   a. Language - Janus/Assembly.
   b. Compiler version 1.4.6.
   c. Linker version 1.4.7.
   d. Target hardware - Intel 86/12A SBC.
   e. Operating system - CP/M-86
   f. Initialization - The initialization routine places the 20 bit address of the interrupt routine in the interrupt vector section of memory.

2. SUBROUTINES
   a. Assy_ntrpt_hdl
      (1) Type - Procedure.
      (2) Purpose - Save the state of the machine and call the routine to resolve the ETHERNET controller interrupt.
      (3) Description of parameters - NA
      (4) External references: Ntrpt_hdl
      (5) Process description: The interrupt routine saves all the registers then calls the high-level routine 'rcv' to handle the interrupt. Remember, when a Janus/Ada assembly package first executes, any assembly code not jumped over is executed before any main program is begun.

M. PACKAGE LIB

1. CONFIGURATION.
   a. Language - Janus/Ada
   b. Compiler version 1.4.7.
   c. Linker version 1.4.7.
   d. Target hardware - Intel 86/12A SBC.
e. Operating system - CP/M-86

f. Comments - Package Lib contains all the high-level library routines we have developed for the system. Just about every other package 'withs' the lib package and uses one or more of the procedures.

2. SUBROUTINES
   a. Get_memory
      (1) Type - Procedure.
      (2) Purpose - Get_memory is called when a process has a need to store a packet in the main memory of the concentrator. Get_memory allocates memory blocks and performs other memory management functions.
      (3) Description of parameters:
          (a) 'Next' is the array index of the memory block requested. If 'next' is returned as '0', no memory is available.
      (4) External references: NA.
      (5) Process description: To allocate memory in which to store packets we have declared an array of records, the records being individual memory blocks. Pointers are used to keep track of which blocks are in use and which are not. The get_memory routine takes the first available block (if any) and returns the index to that block. It also increments the used_blk variable which counts how many blocks are in use at any given time. The routine also manages the rcvWnd variable which is used to tell foreign hosts how much data we are willing to accept at any given time in a packet. As soon as the used blocks is above 50% of the total number of blocks available, the window is changed to zero indicating the remote should not send anything else until we have a chance to clear out memory.

   b. Give_memory
      (1) Type - Procedure.
      (2) Purpose - Give_memory is called when a process has completed processing of all data within a memory block and is ready to return that block to availability. Give_memory inserts the index to the block into the availability queue and performs other memory management functions.
      (3) Description of parameters:
          (a) 'Inx' is the array index of the memory block to be returned.
      (4) External references: NA.
      (5) Process description: As with get_memory, the give_memory procedure manages the rcvWnd, only the window is opened back up to normal size (512 bytes) when the used_blk variable is 33% of the total. 'Inx' is inserted in the front of the queue and the used_blk counter is decremented.

105
c. Perf_cmd
(1) Type - Procedure.
(2) Purpose - To send an instruction to the ETHERNET controller board. Commands and procedures are detailed in the Interlan ETHERNET Controller Handbook.
(3) Description of parameters:
   (a) 'Cmd' is a byte representing an ETHERNET command.
(4) External references:
   (a) Inprt
   (b) oTstbit
(5) Process description: - To instruct the NI3010 board to perform a command the command register of the board is written to. The interrupt register is then read until bit zero is set, at which time the status register is read. If the status register is greater than one an error has occurred in the board.

d. Trn_pck
(1) Type - Procedure.
(2) Purpose - To initiate a DMA transfer from memory in the 86/12A to the ETHERNET controller board.
(3) Description of parameters:
   (a) 'Ad' is the address of the first byte of data to be transferred.
   (b) 'Size' is the number of bytes to be transferred.
(4) External references:
   (a) Outprt
   (b) oTstbit
(5) Process description: - If the state of the controller is 'disable' then the input address is converted into the 20 bit address necessary to perform DMA transfer over the MULTIBUS and written to the proper ports to allow immediate transfer. Otherwise, this procedure wait for the state to change before performing its transfer function. The algorithm for the former case is outlined in the manufacture's manual for the NI3010 board.

e. Resolve_ad
(1) Type - Procedure.
(2) Purpose - Convert the INTERNET address of an alleged ETHERNET host into an ETHERNET address.
(3) Description of parameters:
   (a) 'Ip_ad' is the INTERNET address to be resolved. It is 'in out' status to save space and is not modified by this procedure.
   (b) 'Eth_ad' is the address of the ETHERNET controller board assigned to this host. The address table is a dynamic structure maintained by another procedure.
(c) 'Rslt' indicates whether the IP address was found in the table or not.
(4) External references: NA.
(5) Process description: - This process looks up the input IP address in the dynamic address table 'ad_tbl' which is declared in Global.

g. Get_tcbndx
(1) Type - Procedure.
(2) Purpose - Establish a one-to-one mapping between local TCP addresses in use and indices to the TCB table. More simply, to find the TCB index for a connection from the local TCP address of the connection.
(3) Description of parameters:
   (a) 'Arr' is the TCP address to be used to find the TCB table entry.
   (b) 'Index' is the array index of the TCB entry corresponding to the input TCP address.
   (c) 'Found' indicates whether the TCP address was found or not.
(4) External references: NA.
(5) Process description: - The TCP address to TCB index mapping is accomplished by use of a hashing function.

h. Pcb_cls
(1) Type - Procedure.
(2) Purpose - Reinitialize and normally terminate a Port Control Block entry.
(3) Description of parameters:
   (a) 'Prt_num' is the PCB table index to be closed out.
(4) External references: Outprt
(5) Process description: - The pstate, time_wait, and buf_in_cnt fields in the PCB record are reinitialized and a control character is sent to the Z-100 to ensure termination.

i. Pcb_abort
(1) Type - Procedure.
(2) Purpose - Reinitialize, clear out data and transmission queues, and terminate a Port Control Block entry for the specified port.
(3) Description of parameters:
   (a) 'Prt' is the PCB table index to be closed out.
(4) External references:
   (a) Outprt
   (b) Give_memory
(5) Process description: Pcb_abort will return memory locations attached to the port's primary queue, change the state to allow final close out, reinitialize the
time_wait field in the PCB entry, and send the close code to the Z-100. The state is set to closing to allow an FTP process to clear data from its secondary connection before the memory is returned.

j. Tcb_abort
   (1) Type - Procedure.
   (2) Purpose - Clear out the retransmission queue and reinitialize the port number field of the TCB entry for a TCP connection that is being closed.
   (3) Description of parameters:
       (a) 'Ndx' is the index to the TCB table entry to be closed out.
   (4) External references: NA.
   (5) Process description: The retransmission queue is traversed from front to rear and each memory location returned. The PCB port number field is set to '99' to indicate the port is inactive.

k. Activate_prt
   (1) Type - Procedure.
   (2) Purpose - Add a port that has requested service from the concentrator to the active ports list. Inactive ports are only polled every 10,000 or so loops for activity while active ports are polled on each loop.
   (3) Description of parameters:
       (a) 'Prt' is the port number to be activated.
   (4) External references: NA.
   (5) Process description: The port specified is added to the queue to be polled.

L. Give_status
   (1) Type - Procedure.
   (2) Purpose - Supply information concerning the activity within the concentrator to the user and maintenance programmer.
   (3) Description of parameters: 'Port' is the terminal number.
   (4) External references:
       (a) Get_tcb_ndx
       (b) Get_memory
       (c) Outprt
       (b) Osetbit
   (5) Process description:

This procedure produces a packet which contains the state of execution for all the terminals and includes a status block from the NI3010 board for ETHERNET transmissions. Codes are used in the packet to identify the various states of the PCBs and TCBs. The first byte in the packet is the number of terminals there are, enabling various implementations of the system. Status can only be
requested by a terminal in the local or listen state.

Interface to the Concentrator

To understand the interface to the concentrator requires understanding the RS232 serial communications hardware.

<table>
<thead>
<tr>
<th>terminal or computer</th>
<th>concentrator</th>
</tr>
</thead>
<tbody>
<tr>
<td>(connected via modular phone connector)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>signal gnd</th>
<th>signal gnd</th>
</tr>
</thead>
<tbody>
<tr>
<td>data set ready</td>
<td>data terminal ready</td>
</tr>
<tr>
<td>receive</td>
<td>transmit</td>
</tr>
<tr>
<td>transmit</td>
<td>receive</td>
</tr>
<tr>
<td>shield gnd</td>
<td>shield gnd</td>
</tr>
<tr>
<td>carrier detect</td>
<td>carrier detect</td>
</tr>
<tr>
<td>clr to send</td>
<td>clr to send</td>
</tr>
<tr>
<td>req to send</td>
<td>req to send</td>
</tr>
<tr>
<td>ring indicator</td>
<td>n/a</td>
</tr>
</tbody>
</table>

The line numbers represent pin connections on a 25 pin 'D' connector.

Since communication is bi-directional there is no master-slave relationship or DCE-DTE correspondence between the concentrator and the connected computers. Communication comes in two forms from each end of the line:

1. Control codes to effect action or pass acknowledgement. Control codes are sent at any time necessary by simply writing to the data port of the connected UART. See global1.spc file for a list of control codes.

2. Packets of data to be sent on to a destination. Packets are sent with the use of handshaking signals. Because the communication is bi-directional and control codes are used, some rather unique problems had to be overcome. To send a control code at any time the transmitter had to be available without relying on the receiver to enable it. To receive control codes at any time the receiver had to be enabled at all times. Therefore the request to send (RTS) and clear to send (CTS) signals were not utilized due to their side effects. One signal line is used for a dual purpose of requesting to send data and acknowledging preparation to receive data. This signal line is the Data Terminal Ready (DTR) out line which is connected to the Data Set Ready (DSR) in line. No other signal line is available on the RS232 in our hardware configuration that could function as one of these purposes without having a side effect. Therefore, the problem was
to use the signal line without it being mistaken for the
wrong signal. For example, if the concentrator wanted to
send data to one of the terminals it would set DTR. If
that terminal also wanted to send data to the concentrator
it would also set DTR. Receipt of DSR on the other end
would tell the receiver that its DTR has been acknowledged,
therefore both the concentrator and terminal would proceed
to transmit data at the same time. Of course the data
would be lost in this situation. A means was devised to
ensure that receipt of a DSR could only mean one thing at
that particular moment. Two common assembly routines was
devised for such purpose.

N. PACKAGE LOCAL.

1. CONFIGURATION.
   a. Language - Janus/Ada.
   b. Compiler version 1.5.
   c. Linker version 1.5.
   d. Target hardware - Zenith model 100.
   e. Operating system - MS-DOS.
   f. Package description:
      This package initializes memory, sets the interrupt
      mask, gets the login name, establishes connections, polls
      the keyboard for user inputs, polls the RS232 port for any
      packets or control codes, polls the local control blocks
      for any needed actions, and polls the transmission queue
      for outgoing packets.

2. SUBROUTINES.
   a. Handle kybd_input.
      (1) Type - procedure.
      (2) Purpose - identify input characters from
      the keyboard and process as necessary.
      (3) Description of parameters -
      (a) ch - character to process.
      (4) External references -
      (a) get_memory.
      (b) prompt.
      (c) prntdata.
      (d) give_memory.
      (e) deactivate.
      (f) cKsum.
      (g) information.
      (h) put_in_trnsQ.
      (i) Activate.
      (5) Process description:
         Inputs are handled from the keyboard as bytes and are
         processed in accordance with the state of the current LCB
         (current means the prompt number or destination terminal
         number). A case statement is used for the state of the
         LCB, then, within each case is another case statement for
the byte input. Commands are initiated and states are changed as necessary depending on the input.

b. Handle incoming packet.
   (1) Type - Procedure.
   (2) Purpose - To act on any packets that are received over the RS232 port.
   (3) Description of parameters -
      (a) blk - a memory block containing a packet that was received.
   (4) External references -
      (a) add_to_Q.
      (b) activate.
      (c) prntdata.
      (d) create_FCB.
      (e) put_in_trnsQ.
      (f) receive_file.
      (g) close_FCB.
      (h) prompt.
      (i) give_memory.
   (5) Process description:
      This procedure processes any packets received from the RS232 port or calls an appropriate procedure to handle the packet. The key to the processing is the type field in the packet and what state the LCB is in of the source terminal.

c. Established.
   (1) Type - Procedure.
   (2) Purpose - Polls the keyboard, LCBs, RS232 port and transmission queue.
   (3) Description of parameters - none.
   (4) External references -
      (a) give_memory.
      (b) close_file.
      (c) prompt.
      (d) get_trns.
      (e) send_trns.
      (f) get_memory.
   (5) Process description:
      This process is continually polling all connection ports for any needed processing. A continual polling routine such as this allows many transactions to be carried out simultaneously because the user is not locking up the system with slow inputs from the keyboard. One of the primary concerns was to maintain a continuous poll of the RS232 port for any incoming packets which frees up the concentrator process once a packet is sent. During the polling process, appropriate routines are called when action becomes necessary.
O. PACKAGE FILEXFER.

1. CONFIGURATION.
   a. Language - Janus/Ada.
   b. Compiler version 1.5.
   c. Linker version 1.5.
   d. Target system - Zenith model 100.
   e. Operating system - MS-DOS.
   f. Package description:
      This package handles all commands that require file
      access. A parser is implemented to parse a user input into
      8 character filenames and 3 character extensions. The
      status of file transfers are maintained in the LCBs. If a
      file data packet is sent to a single terminal it is also
      queued until an acknowledgement is received indicating
      proper transmission, else the packet is set up for
      retransmission.

2. SUBROUTINES.
   a. Parse.
      (1) Type - procedure.
      (2) Purpose - To parse a user's input into
      filenames for access to the system disk files.
      (3) Description of parameters -
          (a) blk - a memory block containing the
          user's input.
          (b) FCB - a file control block.
          (c) EOL - End of Line boolean output.
      (4) External references -
          (a) capital.
      (5) Process description:
          When this procedure executes, it takes the user input
          from the memory block and puts a pointer on the beginning
          character, one on any decimal point designating the
          extension, and one on the filename separator (',',) or at
          the end of line. The process then begins to validate the
          filename and writes it to the name field of the FCB. Once
          the name is finished, the extension is validated the same
          way. This procedure was written because the CP/M operating
          system does not have a parse system call as the MS-DOS
          system has. It is desired to have the local system written
          for CP/M-86 as is MS-DOS.

   b. Create_FCB.
      (1) TYPE - procedure.
      (2) Purpose - To initialize a file control
      block and open the file.
      (3) Description of parameters-
          (a) blk - a memory block containing a
          packet received.
      (4) External references -
          (a) put_in_trnsQ.
          (b) prntdata.
(c) create_file.
(d) give_memory.

(5) Process description:
If the state of the LCB for the sender of the packet is ready to open a file, then the file is created which also deletes any file existing by the same name. The file name is contained in the packet received. An acknowledgement packet is prepared with success or failure of this process. If the state is not proper for opening a file, a packet is prepared for transmission to the sender of type 'unable'.

(1) TYPE - procedure.
(2) Purpose - To write to disk file data that is received from another terminal.
(3) Description of parameters - (a) blk - a memory block containing file data.
(4) External references - (a) setDMA.
(b) write_file.
(c) put_in_trnsQ.
(d) give_memory.
(e) cksum.
(5) Process description:
File data is received and written in an opened file. One of four responses will occur, no acknowledgement packet is sent because it is a broadcast packet that was received. A packet indicating a good transmission is sent. A packet indicating a bad transmission is sent. A packet indicating unable to write to file because the file is not open.

d. Close_file.
(1) TYPE - Procedure.
(2) Purpose - To close a file.
(3) Description of parameters - (a) blk - a memory block.
(4) External references - (a) put_in_trnsQ.
(b) close_file.
(c) give_memory.
(5) Process description:
This procedure closes an opened file and sends a reply to the sender if the packet received is not a broadcast packet.

e. Send_file.
(1) Type - Procedure.
(2) Purpose - To read a file at the next sequential point and send the data to a receiver.
(3) Description of parameters -
   (a) prt - the terminal number of the receiver.

(4) External references -
   (a) close_file.
   (b) get_memory.
   (c) setDMA.
   (d) read_file.
   (e) give_memory.
   (f) search_nxt.
   (g) open_file.
   (h) search_frst.

(5) Process description:

When a file is being transmitted to another terminal, this procedure creates the necessary packets to open a file, send file data and close a file. The file names to be sent are contained in a memory block held in the LCB namQ. The memory block is parsed for each file name, the name is then searched for and if found, a packet is sent to open the file. Each subsequent packet contains the file data of 512 bytes. Once all the data is read from the file the file is closed and a packet is sent to close the file on the receiver's terminal. File names are searched for the first occurrence then next occurrence.

f. Send_dir.
   (1) Type - procedure.
   (2) Purpose - To send a directory listing to another terminal.
   (3) Description of parameters -
      (a) prt - the terminal number of the receiver.
   (4) External references -
      (a) setDMA.
      (b) search_nxt.
      (c) parse.
      (d) search_frst.
      (e) give_memory.
      (f) put_in_trnsQ.
   (5) Process description:

This procedure will place up to 32 file names in a memory block for transmission to another terminal when a directory listing is requested from the other terminal. Files are searched the same as procedure send_file.

g. Information.
   (1) Type - Procedure.
   (2) Purpose - To display a text file to the user containing useful information about using the system.
   (3) Description of parameters -
      (a) prt - the terminal number currently being used.
(4) External references -
(a) setDMA.
(b) read_file.
(c) prntdata.
(d) close_file.
(e) give_memory.

(5) Process description:
This procedure reads data from a file that is open and displays the text of the file on the screen until an ascii character 'tab' is found or end of file. If end of file then the file is closed.

P. PACKAGE NAME: FTP.

1. CONFIGURATION
   a. Language: JANUS/Ada
   b. Compiler version: 1.5.0
   c. Linker version: 1.5.0
   d. Target hardware: Zenith model 100 micro-computer
   e. Operating system:
      (1) Name: MS-DOS
      (2) Version: 2.11

2. SUBROUTINE
   a. FTP.
      (1) Type subroutine: Procedure.
      (2) Purpose: This procedure drives the remote file transfer process on the NPS local area network. A signal and an address are sent to the concentrator triggering the FTP command connection establishment. The command/reply sequence then drives the process.
      (3) Description of parameters: NA.
      (4) External references:
         (a) Lib.send_cmd
         (b) Get_ip.get_addr
         (c) Lib.process_reply
         (d) Lib.make_reply
         (e) Lib.get_data_line
         (f) Asmlib.send trns
         (g) Asmlib.tstbit
         (i) Bit.inport
         (j) IO.open
         (k) IO.write
         (l) IO.close
         (m) IO.ioreult

   (5) Process description: The IP address of the destination is returned by get_address. Once the control code has been sent to and answered from the auxiliary port, the address is sent out the auxiliary port.
The process then becomes a cycle of sending commands and processing replies. The dataline received may contain either data or an FTP reply. FTP must inspect the first character of the dataline to determine its content. That first character is set by the concentrator before the data is transmitted. If irregularities occur, 'get dataline' may insert a control code in the first byte. A control byte is also attached to outgoing data.

The replies and commands used in this implementation of FTP are a subset of the system specification in the Stanford Research Institute, RFC-765. The possible responses to commands listed on pages 46 and 47 of RFC-765 are followed very closely. If a reply is received that is not allowed in response to the command issued most recently, the reply is ignored. This allows this system to interface with different implementations of FTP. The first acceptable reply to a command drives the system.

The state diagram for command/reply exchange from [pg. 55 of RFC-765] of the thesis and reproduced below, is followed as closely as possible. Variations to this diagram from the remote site have been detected when in communication with the VAX when a second 500 level reply is sent to clarify the first 500 level reply.

```
*Begin
* |cmd = Send an FTP command.
* |W = Wait for reply.
* |S = Command executed successfully.
* |F = Command failed.
* |1,2,3,4,5 = The first digit of the reply received.

The cycle ends when the user enters 'quit' and the quit command is sent or when the connection is aborted by the remote host.
```
Q. PACKAGE NAME: Lib.pkg

1. CONFIGURATION
   a. Language: JANUS/Ada
   b. Compiler version: 1.5.0
   c. Linker version: 1.5.0
   d. Target hardware: Zenith model 100 micro-computer
   e. Operating system:
      (1) Name: MS-DOS
      (2) Version: 2.11

2. Subroutines.
   a. Send cmd.
      (1) Type subroutine: Procedure.
      (2) Purpose: Send cmd prepares a string in FTP command format and passes that string out the auxiliary port.
      (3) Description of parameters
      (a) 'Cmd' is the enumerated type that represents the FTP command to be sent.
      (b) 'Parameter' is the string that represents the FTP paramater that accompanies 'cmd'.
      (4) External references:
      (a) Asmlib.send_trns
      (b) Bit.inport
      (c) Strlib.length
      (d) Strlib.insert
      (e) Bit.tstbtt
      (f) Asmlib.byte_to_chr
      (5) Process description: Send_command calls internal subprocedure 'convert' to convert the enumerated type 'cmd' into a string, concatenates that string with the input string 'parameter', attaches a control byte as the first character, and sends the resultant string out the auxiliary port.

   b. User options.
      (1) Type subroutine: Procedure.
      (2) Purpose: User_options is called to allow a user to enter his desired file transfer or maintenance request. The FTP command corresponding to that request is sent.
      (3) Description of parameters
      (a) 'Opt' represents the option that the user selected and the command that this procedure transmitted.
      (4) External references:
      (a) IO.is_open
      (b) IO.close
(c) Funcs.get_filename
(d) Funcs.get_opt
(e) Funcs.get_parameter
(f) IO.ioresult
(g) IO.purge
(h) IO.open
(i) Lib.send_cmd
(j) IO.create

(5) Process description: User_options is called when a reply is received that does not in itself require some action be taken. It is expected that if this procedure is called, the user is logged in to the system. From here, the user can request a file transfer, change directory on the remote host, list the directory on the remote host or terminate the process. The user_options procedure also opens and closes locfile for retrieving or sending data to/from the remote host.

User_options displays the options that a user may select, and prompts the user for a selection, attains a parameter for the selected option and sends the command.

c. Send_data
   (1) Type subroutine: Procedure.
   (2) Purpose: Send data to a remote host on ETHERNET.
   (3) Description of parameters
        (a) 'Lst_cmd' is the variable that keeps track of the state of this user FTP process.
   (4) External references
        (a) outport
        (b) inport
        (c) tstbit
        (d) keypress
        (e) getch
        (f) send_cmd
        (g) read
        (h) end_of_file
        (i) eof
        (j) send_trns
        (k) close

(5) Process description: Send_data is passed control after a reply has been received indicating a data connection is being established. Using control code communication with the concentrator, send_data determines when the connection has been established, and sends the data through the auxiliary port.

The data is transmitted in packets of 512 bytes because this is the max packet size of transmission for the concentrator. The user is queried to determine if the file to be transmitted is a text file to allow correct end of file identification.
d. Get_data
   (1) Type subroutine: Procedure.
   (2) Purpose: Get_data is the routine that accepts data from the remote site and dispenses it appropriately.
   (3) Description of parameters
        (a) 'Opt' represents the last command that was transmitted.
        (b) 'Ctr' is the number of bytes passed in the parameter 'byte_array'.
        (c) 'Byte_array' contains the data received from the remote site.
   (4) External references:
        (a) IO.write
        (b) Asmlib.prntdata
   (5) Process description: Get_data identifies the data as a directory listing to be printed on the console or as file data to be written to the global file 'locfile' by the last command that was transmitted ('lst cmd'). The file is opened in 'user_options' and closed in 'process_reply' when a reply indicates the transfer is complete. If abnormal termination occurs the file is closed in 'FTP'. Data may be several packets long. The display of a listing on the console will be continuous from packet to packet. The opening and closing of 'locfile' in 'user_options' allows the data from subsequent packets to be added at the end of the file.

e. Get_dataline.
   (1) Type subroutine: Procedure.
   (2) Purpose: Get_dataline receives data and control characters from the concentrator and passes the results to the caller.
   (3) Description of parameters
        (a) 'Dataline' contains the data received from the concentrator.
        (b) 'Ctr' is the number of bytes passed out in the parameter 'data_line'.
   (4) External references:
        (a) IO.keypress
        (b) IO.is_open
        (c) IO.close
        (d) Asmlib.get_trans
        (e) Asmlib.prntdata
        (f) Bit.outport
        (g) Bit.tstbit
        (h) Typpkg.locfile
        (i) Bit.inport
        (j) Asmlib.getch

119
(5) Process description: Procedure 'get_dataline' will wait for the user to enter control right bracket, timeout to be reached, a control character received, or data received from the concentrator. Timeout does not terminate the process but is included to allow future expansion. Its major function is to clear any handshaking signals that may have been inadvertently set. If a control character is received or control right bracket detected, the first character of 'dataline' is set to the appropriate control code. Code_abort tells the caller to stop the process immediately and code_cls means terminate the process normally.

f. Make_reply.
   (1) Type subroutine: Procedure.
   (2) Purpose: Make_reply receives the reply as an array of bytes and converts that array into an integer 'reply' and a string 'parameter'. The results are returned and displayed on the console.
   (3) Description of parameters
      (a) 'Dataline' contains the data bytes from the concentrator that are the FTP reply and parameter.
      (b) 'Ctr' is the number of bytes in the parameter 'data_line'.
      (c) 'Reply' is the integer representation of the FTP reply identification number.
      (d) 'Parameter' is the string representation of the FTP parameter to the reply.
   (4) External references:
      (a) Asmlib.prntdata
      (b) Asmlib.byte_to_char
      (c) Strlib.insert
      (d) Strlib.str_to_int
   (5) Process description: The conversion of the first fifth through last bytes to a string is done first. Each byte is converted to a character and inserted in to the string. The second through the fourth bytes are converted into an integer by converting each byte into a character, adding the three characters to a string and converting the string into an integer. The first byte in the array is a control code.

g. Process_reply
   (1) Type subroutine: Procedure.
   (2) Purpose: Process_reply is the workhorse of FTP. All replies received from the concentrator are passed to this process for action. This procedure must determine what command to send if any command is required.
   (3) Description of parameters
      (a) 'Reply' is the integer representation of the FTP reply identification number. FTP
replies are described in detail in [Internet Protocol Transition Workbook, pg. 278-281].

(b) 'Parameter' is the string representation of the FTP parameter to the reply. This parameter is not generally used in determining the course of action. It is displayed for the user.

c) 'State' tracks the last FTP command issued. This is used as the state of the process.

(4) External references:
(a) Lib.user_options
(b) Asmlib.byte_to_char
(c) IO.close
(d) IO.is_open
(e) Bit.outport
(f) Bit.tstbit
(g) IO.read
(h) Asmlib.prntdata
(i) Asmlib.send_trns
(j) IO.close
(k) Funcs.get_username
(l) Lib.send_cmd
(m) Get_portnum
(n) Get_password

(5) Process description: Process_reply takes a course of action determined by the reply received and the last command that was sent. Any reply listed in [Ref. 2] of the thesis is handled.

The last command issued may be considered the state of the process. Each state combined with the reply received is assigned a response. If a reply is received that is inappropriate for the state of the process, the reply is ignored. This situation is the result of the different implementations of FTP. Since a server may or may not return more than one reply to a particular command and varying implementations have been experienced even in the limited scope of this thesis, the user system must be able handle many possible occurrences. This process simplifies the problem by using the first acceptable reply to a command as the key to its next action. Generally, the second reply is only information for the user anyway so the second and subsequent replies are displayed on the console. Printing of the multiple replies in sequence is ensured by issuing a 'noop' command before prompting the user to enter his option. A description of the states and their responses follows.

(a) Send username. The FTP command connection is established and the login sequence has begun. If a username has been requested, only the user and quit commands will be accepted by the remote server.

(b) Send password. Follows the 'send username' state. A user must have an account assigned and know the password to access files.
(c) Send portnum. In order for the server to initiate the data connection, the concentrator must be issue a port number. This information is retrieved from the concentrator and transmitted to the remote server via the 'port' command. The port command is sent whenever the server does not have an updated port number, i.e., at the beginning of the process or when the data connection has been open and is closed.

(d) Send user option command. The goal of FTP is to send and receive files. Once the preliminary commands to set up the account have been accomplished, the user is allowed to select his option. The appropriate parameter is attached to this command, and the command is sent to the remote site. The commands issued include:

- (1) List the working directory (nlst)
- (2) Change the working directory (cwd)
- (3) Send a file (stor)
- (4) Get a file (retr)
- (5) Get help (help)
- (6) Delete a file (dele)
- (7) Quit the process (quit).

(e) Send data. Various data types, modes, and formats are accepted by FTP. This implementation allows only the defaults in these areas. The defaults are:

- (1) Format: Ascii non-print
- (2) File structure: File
- (3) Mode: Stream

Before sending data, FTP coordinates with the concentrator to ensure that the data connection is open. When this is confirmed, the file is sent to the concentrator in blocks of five hundred and twelve bytes. The procedure 'process_reply' does not relinquish control of the processor until the entire file is transmitted. The user is prompted to indicate whether the file is textural or binary to allow accurate end of file detection.

When the entire file has been transferred, the local file is closed and a control code is sent to the concentrator to trigger closing of the data connection. This indicates end of file to the remote server.

(f) Get data. The process enters this state after the user has requested a directory list or retrieve data and the server has responded by indicating the data is on its way by sending an appropriate reply.
R. PACKAGE NAME: FUNCS.PKG

1. CONFIGURATION
   a. Language: JANUS/Ada
   b. Compiler version: 1.5.0
   c. Linker version: 1.5.0
   d. Target hardware: Zenith model 100 microcomputer
   e. Operating system:
      (1) Name: MS-DOS
      (2) Version: 2.11

2. SUBROUTINE
   a. Get_opt
      (1) Type subroutine: Function
      (2) Purpose: Get_opt will display possible
                   file maintenance and transfer requests and return the
                   user's selection.
      (3) Description of parameter: The command
                   that is returned is an enumerated type. 'Cmd type' is
                   declared in 'Typ.pkg.spc'. This represents an FTP command
                   as described in [Ref. 2] of the thesis.
      (4) External references:
           (a) IO.get_line

      (5) Process description: Get_opt displays the options that are available to the user on the screen.
          The user selects an option by entering the first letter and
          carriage return. The selected letter corresponds to only
          one command. That command is returned.

   b. Get_password
      (1) Type subroutine: Function
      (2) Purpose: Prompt the user to enter the
                   appropriate password and return that password in string
                   representation.
      (3) Description of parameter: The password
                   that is returned is to be used as a parameter to an FTP
                   command. It is represented as a string of characters.
      (4) External references:
           (a) Asmlib.no_echo
           (b) Strlib.insert
           (c) Strlib.char_to_str
           (d) Asmlib.byte_to_chr

      (5) Process description: Get_password prompts the user to enter his password and reads the
                   keystrokes as bytes from the keyboard without echo to the
                   screen. The bytes are converted to characters, the
                   characters to strings, and the single character strings
inserted into the password. The characters are inspected to ensure only alphabetic characters have been entered.

c. Get_username
   (1) Type subroutine: Function
   (2) Purpose: Prompt the user to enter the valid user id and return the entered string.
   (3) Description of parameter: The username that is returned is to be used as a parameter to an FTP command. It is represented as a string of characters.
   (4) External references:
      (a) IO.get_line
   (5) Process description: The user enters his account id name followed by a carriage return. Only alphabetic characters are allowed.

d. Get_portnum
   (1) Type subroutine: Function
   (2) Purpose: The goal of get_portnum is to attain a valid port number to pass to the remote host in the port command.
   (3) Description of parameter: The port number that is returned is to be used as a parameter to an FTP command. It is represented as a string of characters.
   (4) External references:
      (a) Bit.outport
      (b) Bit.inport
      (c) Bit.tstbit
      (d) Asmlib.get_trns
      (e) Strlib.int_to_str
      (f) Strlib.insert
   (5) Process description: In order for the remote server process to initiate a connection to a particular TCP (or port) address, the concentrator must select the sequentially correct port number and perform some initialization. Get_portnum sends a control character to the concentrator requesting a port number which triggers this initialization. The port address that the concentrator sends is four bytes long. The FTP format for the 'port' command parameter requires the port address be a string of characters with the four bytes represented as characters in a string separated by commas. The bytes received are converted into integers which are converted into strings. The four strings are concatenated with commas between them to form the string acceptable as the 'port' command parameter.

e. Get_filename
   (1) Type subroutine: Function
   (2) Purpose: Get_filename returns a string containing a file name that meets the format required by CPM and MS-DOS for file names.
(3) Description of parameter: The file name that is returned is to be used as a parameter to an FTP command. It is represented as a string of ASCII characters.

(4) External references:
(a) IO.get_line
(b) Strlib.char_to_str
(c) Strlib.insert
(d) Strlib.length

(5) Process description: Get_filename reads the characters entered by the user when the carriage return is detected. Each character of the string is then scrutinized to ensure proper file name format. Leading and trailing spaces are ignored. A string with a space in the middle of the name will result in only the part of the string before the space being recognized. If a drive designator is included, a colon must be the second non-blank character. The number of characters in the primary file name are counted by the local variable 'name_len'. If nine characters are counted, not counting the drive designator, before a period, space, or end of line is reached, the file name is rejected as too long. If a period is encountered, the extension is validated. Only leading spaces, alphanumeric characters, one colon, and one period are allowed in a file name.

f. Get_parameter
(1) Type subroutine: Function
(2) Purpose: The purpose of get_parameter is to attain a parameter for a command corresponding to an FTP command.
(3) Description of parameters:
(a) The option that is passed in represents an FTP command. Each FTP command accept a unique type of parameter.
(b) The parameter that is returned is to be used as a parameter to an FTP command. It is represented as a string of ASCII characters.
(4) External references:
(a) Funcs.get_filename
(b) IO.get_line
(5) Process description: Only seven of the FTP commands implemented in this system require parameters other than the null string. The file name required as parameter to the 'retr' and 'stor' commands is a filename for the remote site. It is parsed by the remote site and errors identified via FTP replies.
S. PACKAGE NAME: GET_IP.PKG

1. CONFIGURATION
   a. Language: JANUS/Ada
   b. Compiler version: 1.5.0
   c. Linker version: 1.5.0
   d. Target hardware: Zenith model 100 micro-computer
   e. Operating system:
      (1) Name: MS-DOS
      (2) Version: 2.11

2. SUBROUTINE
   a. Get_addr
      (1) Type subroutine: Procedure
      (2) Purpose: Get_addr will display available remote destinations to the user return the address of the user's selected destination.
      (3) Description of parameters: The four integers returned by this procedure represent the four byte IP address of the desired destination.
      (4) External references:
          (a) Hosts.fil
          (b) IO.open
          (c) IO.close
          (d) IO.get
          (e) IO.end_of_file
          (f) IO.read
          (g) IO.end_of_line
          (h) IO.skip_line

      (5) Process description: Get_ip printsthe contents of the file 'hosts.fil' along with a selector number and prompts the user to select his destination by keying in a number. Get_addr then interprets the addr and returns the selected address to the calling routine. The address is read from the file as an array of integers and the name as a string. The address is stored in four arrays representing each byte of the address. The user selection number then acts as the index of these arrays to identify the correct address.

      Additions to the hosts file may be required as hosts are added to the ETHERNET. The correct IP address may be obtained from the file 'hosts' on the VAX Unix or from a technical representative. The address must be entered as four integers separated by spaces. Each integer represents one byte so each must be less than 256. The name is a string of not more that 21 characters. The new entry must be made in the following format:
          (a) IP address byte one (<= 256)
          (b) Space
T. PACKAGE NAME: ASMLIB.ASM

1. CONFIGURATION
   a. Language: JANUS/ASSEMBLER
   b. Compiler version: 1.5
   c. Linker version: 1.5.0
   d. Target hardware: Zenith model 100 micro-computer
   e. Operating system:
      (1) Name: MS-DOS
      (2) Version: 2.11

2. Comments
   a. As stated in the Janus/Ada Users Man, the discrete type input parameters for Janus/assembly modules are stored on the stack with the last parameter closest to the top. Output and other type parameters are addressed by the stack.
   b. Also stated in the Janus/Ada Users Man, the discrete value to be returned from Janus/assembly functions must be placed in the al register just before returning. Word values are returned in the ax register, and the address of non-discrete types returned is returned in the AX register.
   c. The interrupts and function calls used are standard to the operating system. Descriptions may be found in the commercial documentation.

3. SUBROUTINES
   a. Byte_to_char
      (1) Type subroutine: Function
      (2) Purpose: Allow assignment of a variable of type byte to be assigned in to a variable of type character. Bit seven is masked to ensure the byte corresponds to an ascii character.
      (3) Description of parameters:
          (a) A value of type byte to be converted is the input parameter.
          (b) The input value is returned as a character.
      (4) External references: NA.
b. Byte_to_char
   (1) Type subroutine: Function
   (2) Purpose: Allow assignment of a variable of type byte to be assigned in to a variable of type character. The byte is not modified, allowing control characters to be assigned into strings.
   (3) Description of parameters:
       (a) A value of type byte to be converted is the input parameter.
       (b) The input value is returned as a character.
   (4) External references: NA.
   (5) Process description: This function returns the input byte as a character by moving the input "value into the ax register.

c. Prntdata
   (1) Type subroutine: Procedure
   (2) Purpose: Display a value of type byte on the console.
   (3) Description of parameters
       (a) A value of type byte to be displayed on the console device is the input parameter.
   (4) External references: Interrupt 21h
   (5) Process description: This procedure moves the input parameter to the dx register, masks bit seven, sets the ah register and invokes the operating system function call '21h'. This interrupt identifies the function desired from the ah register and reads its input from the dx register. The ascii representation of input value will be displayed on the console.

d. Getch
   (1) Type subroutine: Procedure
   (2) Purpose: Return the value most recently entered through the keyboard.
   (3) Description of parameters
       (a) The value of type byte most recently entered through the keyboard is returned.
   (4) External references: Interrupt 21h
   (5) Process description: The registers are set and a call is made to the operating system function to return the byte representation of the character entered to the keyboard. This value is placed at the address pointed to in the di register to be returned.
e. Delete file
   (1) Type subroutine: Procedure
   (2) Purpose: Delete a file.
   (3) Description of parameters
       (a) The address of the file control
           block of the file to be deleted is input to this procedure.
           In Janus/Ada, addresses are represented as integers.
   (4) External references: Interrupt 21h
   (5) Process description: The registers are
       set and a call made to the operating system that will
       perform the desired file maintenance function.

f. Create file
   (1) Type subroutine: Procedure
   (2) Purpose: Initialize a file control
           block for an unopened file.
   (3) Description of parameters
       (a) The address of the file control
           block of the file to be created is input to this procedure.
           In Janus/Ada, addresses are represented as integers.
           (b) An integer indicating the status
               of the function upon completion is returned.
   (4) External references: Interrupt 21h
   (5) Process description: The registers are
       set and a call made to the operating system that will
       perform the desired file maintenance function. The file
       control block must be declared by the calling routine or an
       address obtained from an existing FCB. FCB format and a
       description of the system function may be found in the
       Zenith/Heath Programmer's Utility Pack, chapters three and
       four.

g. Open file
   (1) Type subroutine: Procedure
   (2) Purpose: Initialize a file control
           block for an unopened file.
   (3) Description of parameters
       (a) The address of the file control
           block of the file to be opened is input to this procedure.
           In Janus/Ada, addresses are represented as integers.
           (b) Found indicates if the file named
               in the File Control Block was found in the disk directory.
   (4) External references: Interrupt 21h
   (5) Process description: The registers are
       set and a call made to the operating system that will
       perform the desired file maintenance function. The file
       control block must be declared by the calling routine or an
       address obtained from an existing FCB. The FCB must be
       correctly initialized in order for this procedure to work
       correctly. FCB format and a description of the system
       function may be found in the operating system
       documentation.
Found will be set to false if the file identified in the file name field of the FCB does not exist.

h. Write_file
   (1) Type subroutine: Procedure
   (2) Purpose: Write a record to a disk file.
   (3) Description of parameters
       (a) The address of the file control block of the file to be written to is input to this procedure. In Janus/Ada, addresses are represented as integers.
       (b) 'Succ' indicates if the write was successfully completed.
   (4) External references: Interrupt 21h
   (5) Process description: The registers are set and a call made to the operating system that will perform the desired file maintenance function. The file control block must be declared by the calling routine or an address obtained from an existing FCB. FCB format and a description of the system function may be found in the operating system documentation.
       'Succ' will be set to false if value returned in the AL register is not equal to zero.

i. Close_file
   (1) Type subroutine: Procedure
   (2) Purpose: Close a file.
   (3) Description of parameters
       (a) The address of the file control block of the file to be closed is input to this procedure. In Janus/Ada, addresses are represented as integers.
   (4) External references: Interrupt 21h
   (5) Process description: The registers are set and a call made to the operating system that will perform the desired file maintenance function. A description of the MS-DOS system kernel function may be found in MS-DOS Programmer's Utility Pack.

j. Cksum
   (1) Type subroutine: Function
   (2) Purpose: Compute the check sum of a designated number of consecutive bytes.
   (3) Description of parameters
       (a) 'Addr' is the address of the first of the bytes to be part of the check sum process. In Janus/Ada, addresses are represented as integers.
       (b) 'Amt' is the number of bytes to compute the check sum for.
       (c) The result of the check sum process is returned.
   (4) External references: NA.
   (5) Process description: Compute_cksum performs an XOR of 'amt' bytes beginning at 'addr' and the
result is returned as 'cksm'. This check sum algorithm is a simple check done only on data transmitted across the RS232 serial lines connection to verify data.

k. Setdma
   (1) Type subroutine: Procedure
   (2) Purpose: Set the disk data transfer address.
   (3) Description of parameters
   'Addr' is the address at which the disk transfer is to begin. In Janus/Ada, addresses are represented as integers.
   (4) External references: Interrupt 21h.
   (5) Process description: The registers are set and a call made to the operating system that will perform the desired file maintenance function. A description of the MS-DOS system kernel function may be found in MS-DOS Programmer's Utility Pack.

l. No_echo
   (1) Type subroutine: Function
   (2) Purpose: Return a character from the keyboard without displaying the character on the console.
   (3) Description of parameters
   No_echo returns the character as type byte.
   (4) External references: Interrupt 21h.
   (5) Process description: The registers are set and a call made to the operating system that will perform the desired console operation. A description of the MS-DOS system kernel function may be found in MS-DOS Programmer's Utility Pack.

m. Search_first
   (1) Type subroutine: Procedure
   (2) Purpose: Verify the existence of a file or match a filename that has wild card characters.
   (3) Description of parameters
   (a) 'Addr' is the address of an unopened FCB.
   (b) 'Fnid' is a boolean that indicates if the file was found or not.
   (4) External references: Interrupt 21h.
   (5) Process description: The registers are set and a call made to the operating system that will perform the desired function. A description of the MS-DOS system kernel function may be found in MS-DOS Programmer's Utility Pack.

n. Search_nxt
   (1) Type subroutine: Procedure
   (2) Purpose: Used after 'search_first' to
find additional entries that match a file name that contains wild card characters.

(3) Description of parameters
   (a) 'Addr' is the address of an unopened FCB. Addresses are represented as integers in Janus\Ada.
   (b) 'Fnd' is a boolean that indicates if the file was found or not.

(4) External references: Interrupt 21h.

(5) Process description: The registers are set and a call made to the operating system that will perform the desired function. A description of the MS-DOS system kernel function may be found in MS-DOS Programmer's Utility Pack.

o. Get_trns
   (1) Type subroutine: Procedure
   (2) Purpose: Receive one or more characters across the RS232 connection between the Z-100's and the concentrator.
   (3) Description of parameters
       (a) 'Addr' is the address that the first byte of the data is to be stored into. Addresses are represented as integers in Janus\Ada.
       (b) 'Dprt' is the port data port address the data is to be received from.
       (c) 'Amt' is the maximum number of bytes to be received on input and is returned as the number of bytes received.

   (4) External references: NA.

   (5) Process description: The data is read one byte at a time until the amount count is reached. DSR/DTR handshaking is performed before each character is read. For a state diagram of the handshaking, see [Hart\YAS86].

p. Send_trns
   (1) Type subroutine: Procedure
   (2) Purpose: Send one or more bytes across the RS232 connection between the Z-100's and the concentrator.
   (3) Description of parameters
       (a) 'Addr' is the address of the first byte of the data to be transmitted. Addresses are represented as integers in Janus\Ada.
       (b) 'Dprt' is the port data port address the data is to be transmitted to.
       (c) 'Amt' is the number of bytes to be transmitted on input and is returned as the number of bytes actually sent.

   (4) External references: NA.
(5) Process description: The data is sent one byte at a time until the amount count is reached. DSR/DTR handshaking is performed before each byte is sent. For a state diagram of the handshaking, see [Hart\Yas86].

q. Read_file
(1) Type subroutine: Procedure
(2) Purpose: Read a record from a disk file.
(3) Description of parameters
   (a) 'Addr' is the address of the file control block of the file to be read. In Janus/Ada, addresses are represented as integers.
   (b) 'Rslt' is an integer that identifies the result of the read. The details of the System Kernel Function may be found in the programmer's utility pack.
(4) External references: Interrupt 21h
(5) Process description: The registers are set and a call made to the operating system that will perform the desired file maintenance function. The file control block must be declared by the calling routine or an address obtained from an existing FCB. FCB format and a description of the system function may be found in the operating system documentation.

r. Capital
(1) Type subroutine: Function
(2) Purpose: Convert a byte representing a lower case letter into a byte representing the corresponding upper case letter.
(3) Description of parameters
   (a) 'Char' is the byte representation of a letter to be converted to upper case.
   (b) If the byte input was a letter, the byte returned will be the upper case representation of that letter.
(4) External references: NA
(5) Process description: Capital performs an 'and' operation between the input value and 5f hex and returns the result. No check is made to ensure the input is in the range of the ascii letters. An upper case letter will not be modified.

s. Lower_case
(1) Type subroutine: Function
(2) Purpose: Convert an upper case letter into the corresponding lower case letter.
(3) Description of parameters
   (a) 'Char' is the upper case letter to be converted to lower case.
   (b) If the character input was a letter, the character returned will be the lower case
representation of that letter.
(4) External references: NA
(5) Process description: Lower_case performs an or' operation between the input value and 20h and returns the result. No check is made to ensure the input is in the range of the ascii letters. A lowercase letter will not be modified.

t. Arr_to_strg
(1) Type subroutine: Function
(2) Purpose: Convert an array of bytes into a string.
(3) Description of parameters
   (a) 'Addr' is the address of the first byte of the array to be converted into a string. Since the first byte of a string contains the length of the string, the first byte of the array passed in must identify the number of bytes in the array.
   (b) The function returns the array unchanged.
(4) External references: NA
(5) Process description: Arr_to_strg returns the byte that was passed in as a string. The array is not modified in any way and it is assumed that the programmer has set the first byte of the array as the length of the array (that is, length not including the length byte).

u. Conv_byt
(1) Type subroutine: Function
(2) Purpose: Allow assignment of a variable of type character to be assigned in to a variable of type byte. The value is not modified.
(3) Description of parameters:
   (a) A value of type character to be converted is the input parameter.
   (b) The input value is returned as a byte.
(4) External references: NA.
(5) Process description: This function returns the input character as a byte by moving the input value into the ax register.

v. Two_bytes
(1) Type subroutine: Function
(2) Purpose: Convert a two byte array into an integer.
(3) Description of parameters:
   (a) The address of the array to be converted is input to the function.
   (b) The input value is returned as an integer.
(4) External references: NA.
(5) Process description: The address of the array is used to move the two bytes into the AX register to be returned.

w. Dec cnt
   (1) Type subroutine: Procedure
   (2) Purpose: No idea.
   (3) Description of parameters:
       (a)
       (b)
       (c)
   (4) External references: NA.
   (5) Process description:

x. Current_dsk
   (1) Type subroutine: Procedure
   (2) Purpose: Identify the currently selected disk drive.
   (3) Description of parameters: A byte is returned representing the currently selected disk drive (0=A, 1=B, etc.).
   (4) External references: Int 21h.
   (5) Process description: This procedure only calls the System Kernel function that performs this service. See the Programer's Utility Pack for details of the function's operation.

y. Get_strg
   (1) Type subroutine: Procedure
   (2) Purpose: Allow a user to enter a string of characters into the keyboard.
   (3) Description of parameters: 'Addr' is the address of a memory buffer. The byte addressed must contain the maximum number of bytes that may be entered into the buffer. The second byte will be set to the actual number of bytes entered from the keyboard. Characters entered from the keyboard will be sequentially stored after the second byte of the buffer until the maximum length is reached or carriage is entered.
   (4) External references: Int 21h.
   (5) Process description: This procedure calls the System Kernel function that performs this service. See the Programer's Utility Pack for details of the function's operation.

z. Prnt_buf
   (1) Type subroutine: Procedure
   (2) Purpose: Display one or more consecutive characters in memory on the console.
   (3) Description of parameters: 'Addr' is the address of the memory buffer containing the data to be displayed.
(4) External references: Int 21h.
(5) Process description: This procedure calls the System Kernel function that will display one byte and loops until all bytes are displayed. The first byte of the buffer must contain the length number of bytes to be displayed.

U. PACKAGE NAME: ASSYLIB.ASM

1. CONFIGURATION
   a. Language: JANUS/ASSEMBLER
   b. Assembler Version: 1.4.6
   c. Linker Version: 1.4.7
   d. Target Hardware: Intel 80/12A SBC
   e. Operating system:
      (1) Name: Cpm-86
      (2) Version: 1.1
      (3) Release: 1.4

2. Comments
   a. As stated in the Janus/Ada Users Man, the discrete type input parameters for Janus/assembly modules are stored on the stack with the last parameter closest to the top. Output and other type parameters are addressed by the stack.

   b. Also stated in the Janus/Ada Users Man, the discrete value to be returned from Janus/assembly functions must be placed in the al register just before returning. Word values are returned in the ax register, and the address of non-discrete types returned is returned in the AX register.

   c. The interrupts and function calls used are standard to the operating system. Descriptions may be found in the documentation supplied by Zenith Data Systems for CPM-86.

   d. Many of the functions and procedures in this package perform the same function as a supplied Janus/Ada tool. In order to access the Janus supplied modules, other modules that may not be used must be linked into the command file. These modules were coded by the authors to preclude inclusion of excess modules.

3. SUBROUTINES
   a. Cksum
      (1) Type subroutine: Procedure
      (2) Purpose: Calculate the 'checksum' value of a specified number of bytes.
      (3) Description of parameters:
(a) 'Addr' is the address of the first byte to be included in the checksum calculation.
(b) 'Num_wrds' specifies the number of sixteen bit words to include in the checksum calculation.
(c) 'Rslt' is the result of the calculation.

(4) External references: NA.
(5) Process description: The checksum of a network packet is defined as being the ones complement of the one's complement sum of all sixteen bit words. For the purposes of computing the checksum, the checksum field is set to zero. 'Cksum' begins the calculation at the address specified and computes the next 'Num_wrds' sequential sixteen bit words. Checksum is used to verify accuracy of datagram headers transmitted over networks. The headers used in this application do have the checksum field within the header. A detailed description of checksum computation is contained in Stanford Research Institute, Request For Comments number 793, p 16.

b. Wr_ad
(1) Type subroutine: Procedure
(2) Purpose: Send the memory address to be used for a block data transfer to the ETHERNET controller board.
(3) Description of parameters:
   (a) 'Ad' is the offset address of the first byte to be used by for the data transfer.
(4) External references: NA.
(5) Process description: The offset address that is input is converted to a 20 bit address needed to perform a DMA transfer across the MULTIBUS. The address is computed by shifting the extension byte to the left four bits and adding it to the lower two bytes. The three bytes of the 20 bit address are written to the ports declared in the procedure. H_ad_prt and l_ad_prt are for the high and low bytes of the address, and e_ad_prt is the port address to send the extended portion of the 20 bit address. Since the these addresses are hard coded, the program would have to be modified and reassembled and linked if the NI3010 port addresses change (which is not very likely).

c. Inprt/outprt
(1) Type subroutines: Procedure
(2) Purpose: Get/send value to/from an IO port.
(3) Description of parameters:
   (a) 'Prt' is the port number the data is to be accessed.
   (b) 'Byt' is the value to be written to or read from the port.
(4) External references: NA.
(5) Process description: The assembly 'in' and 'out' instructions are used to get/send the value through the designated port.

d. Addarr, subarr
   (1) Type subroutines: Procedure
   (2) Purpose: Add/subtract two four byte arrays.
   (3) Description of parameters:
        (a) 'Arr1' and 'arr2' are the two arrays to be operated on. The result is returned in 'Arr1'.
   (4) External references: NA.
   (5) Process description: Each of the eight input bytes are moved into registers. The corresponding bytes are added/subtracted as though the array represented a long integer.

e. Arr_to_int
   (1) Type subroutines: Function
   (2) Purpose: Convert the value represented in a two byte array into a two byte integer representation.
   (3) Description of parameters:
        (a) 'Arr' contains the value to be converted.
   (4) External references: NA.
   (5) Process description: The input value is not modified. The value of each byte of the input array is moved into the output area and returned.

f. Ohi/olo
   (1) Type subroutines: Function
   (2) Purpose: Convert the high/low byte of an integer into a byte.
   (3) Description of parameters:
        (a) 'Int' is the integer from which the high byte will be copied.
   (4) External references: NA.
   (5) Process description: Integers are represented as two bytes. In these functions, the value of the high/low byte of the input integer is assigned to the AL register and returned.

g. Osttbit
   (1) Type subroutine: Function
   (2) Purpose: Determine if a specific bit of an eight bit byte is set (equal to one).
   (3) Description of parameters:
        (a) A value of type byte to be inspected.
(b) An integer identifying the bit number of the byte that is to be inspected. The range is 0..7.

(4) External references: NA.
(5) Process description: To test a particular bit of a byte and return true if set.

h. Oclrbit/osebit
(1) Type subroutines: Function
(2) Purpose: Set or clear a specific bit of a specific byte. Most often used to set values of control words.
(3) Description of parameters:
   (a) 'Num' is the byte in which the bit is to be set/cleared.
   (b) 'Bit' is the bit number of the bit to be set/cleared. The range is 0..7.
(4) External references: NA.
(5) Process description: ???

i. Gt equ, lt equ, g than, l than
(1) Type subroutines: Function
(2) Purpose: Determine the logical relationship between two four byte arrays.
(3) Description of parameters:
   (a) 'Arr1' and 'Arr2' are the arrays to be compared.
   (b) A boolean value is returned indicating if the tested condition holds.
(4) External references: NA.
(5) Process description: ???

j. Inc arr
(1) Type subroutines: Function
(2) Purpose: Increase the value of an array by one as if it were an integer.
(3) Description of parameters:
   (a) 'Arr1' is the array to be incremented.
   (b) 'Int' is the ???
(4) External references: NA.
(5) Process description: ???

k. Grtr of
(1) Type subroutines: Function
(2) Purpose: Identify the integer with the larger numerical value.
(3) Description of parameters:
   (a) 'Int1' and 'Int2' are the integers to be compared.
   (b) The larger integer is returned as an array of two bytes.
1. **Upper_nibble**
   (1) Type subroutines: Function
   (2) Purpose: To return the integer value of the upper nibble of a specified byte.
   (3) Description of parameters:
       (a) 'Byt' is a byte;
       (b) an integer is returned.
   (4) External references: NA.
   (5) Process description:
       A field in the TCP/IP header is only 4 bits wide and is contained in the upper nibble of a particular byte. This function shifts that byte to the right 4 bits, then returns that value.

m. **Inc_nxt_prn_ad**
   (1) Type subroutines: Function
   (2) Purpose: Advance the value of the buffer pointing at the next TCP address to be used.
   (3) Description of parameters:
       (a) 'Addr' is the integer representation of the last TCP addressed.
       (b) The incremented input value is and returned.
   (4) External references: NA.
   (5) Process description: The input value is incremented and returned.

n. **Prntch**
   (1) Type subroutines: Function
   (2) Purpose: Output a value on the console.
   (3) Description of parameters: NA.
   (4) External references: Int 21h.
   (5) Process description: A system kernel function is called to perform the desired function. The registers must be set prior to calling this function.

o. **Prt_hex**
   (1) Type subroutines: Function
   (2) Purpose: Output the hexadecimal representation of a value on the console.
   (3) Description of parameters:
       (a) 'Addr' is the integer representation of the last TCP addressed.
       (b) 'Num' is
   (4) External references: NA.
   (5) Process description: The input value is incremented and returned.
p. Get_trns
   (1) Type subroutine: Procedure
   (2) Purpose: Receive one or more characters across the RS232 connection between the Z-100's and the concentrator.
   (3) Description of parameters
       (a) 'Addr' is the address that the first byte of the data is to be stored into. Addresses are represented as integers in Janus/Ada.
       (b) 'Dprt' is the port data port address the data is to be received from.
       (c) 'Amt' is the maximum number of bytes to be received on input and is returned as the number of bytes received.
   (4) External references: NA.
   (5) Process description: The data is read one byte at a time until the amount count is reached. DSR/DTR handshaking is performed before each character is read. For a state diagram of the handshaking, see [Hart/YAS86].

q. Send_trns
   (1) Type subroutine: Procedure
   (2) Purpose: Send one or more bytes across the RS232 connection between the Z-100's and the concentrator.
   (3) Description of parameters
       (a) 'Addr' is the address of the first byte of the data to be transmitted. Addresses are represented as integers in Janus/Ada.
       (b) 'Dprt' is the port data port address the data is to be transmitted to.
       (c) 'Amt' is the number of bytes to be transmitted on input and is returned as the number of bytes actually sent.
   (4) External references: NA.
   (5) Process description: The data is sent one byte at a time until the amount count is reached. DSR/DTR handshaking is performed before each byte is sent. For a state diagram of the handshaking, see [Hart/YAS86].

r. Oput
   (1) Type subroutine: Procedure
   (2) Purpose: Display one or more characters on the console.
   (3) Description of parameters
       (a) 'Strg' is the string to be displayed.
   (4) External references: NA.
   (5) Process description: The first byte of the input string is expected to be the length of the string to be displayed. That number of characters are displayed
to the console using the 'out' instruction to the monitor data port address. The monitor data and status port addresses are specified in the Z-100 hardware documentation.

s. Onew_line
   (1) Type subroutine: Procedure
   (2) Purpose: Advance the 'next display' position on the console to the beginning of a new line.
   (3) Description of parameters: NA.
   (4) External references: NA.
   (5) Process description: The ascii characters 'carriage return' and 'line feed' are sent to the monitor data port using the 'out' instruction.

t. Xsum
   (1) Type subroutine: Function
   (2) Purpose: Perform an XOR operation on the specified number of bytes. This is used as a primitave checksum for local network transmissions.
   (3) Description of parameters:
      (a) 'Addr' is the address of the first byte of data to be included in the checksum operation.
      (b) 'Cnt' is the number of consecutive bytes to process.
      (c) The result of the multiple XOR operations is returned.
   (4) External references: NA.
   (5) Process description: The address is incremented as each byte is XOR'ed against the register holding the return value.

u. Get_data
   (1) Type subroutine: Procedure
   (2) Purpose: ???
   (3) Description of parameters:
      (a) 'Port' is the port number to be read.
      (b) 'Addr' is the address in which to store the first byte of data.
      (c) 'Len' is the number bytes received.
   (4) External references: NA.
   (5) Process description: ???
APPENDIX C

USER MANUAL FOR TELNET

SECTION 1. GENERAL

1.1 Purpose of the Users Manual.

The purpose of the Users manual for the NPS Local Area Network TELNET is to allow students with minimal experience in computer science to effectively use the system.

1.2 Project References.


1.3 Terms and Abbreviations.

a. TELNET. The name for the software standard remote login protocol.

b. LAN. Acronym for Local Area Network.

c. Z-100. Short name for the Zenith model 100 micro-computer.

d. TCP. Telecommunications Protocol.

e. IP. INTERNET Protocol.

f. NPS. Naval Postgraduate School, Monterey, Ca.

1.4 Security and Privacy.

The Users Manual, programs, and files used to implement the NPS TELNET process are unclassified and contain no information covered by the Privacy Act.
SECTION 2. SYSTEM SUMMARY

2.1 System Application.

a. Purpose of TELNET. As stated in the SRI RFC-764, the purpose of the TELNET Protocol is to provide a general, bi-directional, eight-bit byte oriented communications facility. Its primary goal is to allow a standard method of interfacing terminal devices and terminal-oriented processed to each other.

b. Capabilities of the system.

TELNET allows a user to act as a terminal to the VAX 11-780, 11-750, Iris1, and Iris2 computers attached to ETHERNET. To login to one of these systems a user must have an account on the desired system. When logged in, a user has all capabilities of a directly connected terminal including file edit, copy, directory inquiry and maintenance, and network access via FTP.

c. Additional features. None.

d. Functions of the system. TELNET will allow the user to select a remote destination, will establish a network connection to the desired destination and pass the transmitted characters between the user and the remote location. Once the system has established the connection, the Z-100 will function as a remote terminal to the remote host.

2.2 System operation.

In order to use TELNET, the files telenet.com (.cmd if under CPM-86) and hosts.fil must reside on the users auxiliary storage device.

2.3 System Configuration.

TELNET was designed to operate on Zenith model 100 microcomputers connected to the NPS local area network.

2.4 System Organization.

TELNET operates as an information passing station when logged in to a remote host. Characters entered in to the keyboard are sent to the remote host and received bytes are displayed on the screen.

2.5 Performance.

a. Input.
The only user input to TELNET is the selection of the desired destination.

b. Output.

There is no output generated from TELNET.

c. Response Time.

Response time will vary due to three primary reasons:

1. Function.

A request to list the directory will generally be accomplished quicker than a request to edit a file.

2. System usage.

ETHERNET is a broadcast network operating at ten megabits per second. Even at this high bit rate, the medium becomes quickly overloaded when the number of users increases. Additionally, the local and remote front end processors slow down significantly when use increases. With the current configuration, it is suggested that a maximum of four Zenith users operate under TELNET/FTP concurrently.

3. Error occurrence.

User caused error such as misspelling a password or system error caused by transmission medium malfunction will be corrected by the system. However, response time may be degraded.

d. Limitations.

NPS TELNET can not be used to log in to a computer outside the NPS LAN. ARPANET access may be achieved by utilizing NPS TELNET to log in to a computer with ARPANET access and utilizing TELNET on that system to access ARPANET.

2.6 Data Base.

The only file used by TELNET is the file 'HOSTS.FIL'. This file contains the name and INTERNET address of remote hosts connected to the NPS LAN. The hosts file is a text file that is maintained by programmers of the Aegis project and is write protected.
2.7 General Description of Inputs, Processing, and Outputs.

2.7.1 Inputs.

a. User input.

The only user input to TELNET is the selection of the desired remote host. Once the user is logged in to a remote host, the console input is considered input to the operating system of the remote computer. Control right bracket may be entered by the user as a signal to TELNET to terminate the process.

b. File input.

The file 'HOSTS.FIL' contains the name and INTERNET address of the computers directly accessible from the NPS LAN.

2.7.2 Output.

a. Console output.

1. The available destinations are displayed when a user initiates TELNET. The name of the desired destination is the important element to the TELNET user. The address is displayed for system maintenance purposes.

2. Data received from the remote computer is considered to be information from the remote host operating system to the user and is displayed on the console.

b. Network connection.

Every keystroke by the user is transmitted individually to the remote host.

2.7.3 Process.

TELNET initiates a network connection with the selected remote host and then acts as an information passer between the micro-computer user and the remote host.
ATTACHMENT 1 TO APPENDIX C

TELNET RUN SHEET

A. Getting started.

TELNET is programmed to operate on the Zenith model 100 attached to the cluster of micro-computers in the NPS micro-computer lab. All computers in the lab should have the files 'TELNET.COM' ('TELNET.CMD' if under CPM 86) and 'HOSTS.FIL' needed to utilize TELNET resident on the Z-100 hard disk. If under MSDOS the files will be in directory 'LOCAL.NET'.

To use TELNET, an MSDOS user must enter the directory 'LOCAL.NET'. To initiate TELNET the user will enter 'TELNET<cr>'. The first message displayed to the console by TELNET will be 'ENTERING THE TELNET PROCESS.'. The user will then be prompted to select the destination. Once the destination is selected, the first user of the system may experience a short delay of up to one minute while the Z-100 transmits the control program to the concentrator. No action is required by the user until another message is displayed to the screen. From this point, the user only need respond to messages displayed on the screen and to the operating system of the remote host.

B. SELECTING A DESTINATION.

TELNET will display a list of possible destinations for an TELNET connection. Selecting the desired destination is accomplished by entering the number corresponding to the desired system name followed by a carriage return.

The destinations displayed include the recognized INTERNET name and address of computers connected to the NPS LAN. The user may select any computer on the list. However, TELNET will not allow remote login unless the user has an account on the remote computer. If a user is not sure which computer he may connect to, he should contact an instructor or the computer science department technical representative responsible for system accounts.

C. SELECTING AN OPTION.

TELNET will prompt the user to enter an option and will display a list of valid options. The option list and further messages are self explanatory. Selection is effected by entering the number corresponding to the desired option followed by carriage return.

147
D. WHEN TROUBLE OCCURS.

TELNET is designed to be totally robust. If a user desires to terminate the system abnormally, enter control right bracket (\]) or the prompted character for termination. If this does not work, the user may terminate the process at any time without destroying files or causing system damage by utilizing control reset. Some specific problems and response descriptions follow.

1. Excessive wait occurring. The NPS LAN is designed for a small number of users and will backup quickly as the number of users rise. Terminating while waiting can usually be accomplished by entering \] (control right bracket). If this is not successful, enter control reset. Terminating the system abnormally in this fashion may cause a longer than normal wait required to reenter the system.

2. Keyboard does not accept characters. If the keyboard is 'frozen' a short wait may allow the system to recover. If this is not effective, the only recourse is control reset.

3. System will not accept a file name. If the system will not accept a filename, refer to the messages produced and documentation for the operating system in use as to proper filename format.
APPENDIX D

USER MANUAL FOR FTP

SECTION 1. GENERAL

1.1 Purpose of the Users Manual.

The purpose of the Users manual for the NPS Local Area Network file transfer process is to allow students with minimal experience in computer science to effectively use the system.

1.2 Project References.


1.3 Terms and Abbreviations.

a. FTP. The acronym for the software standard File Transfer Process.

b. LAN. Acronym for Local Area Network.

c. Z-100. Short name for the Zenith model 100 microcomputer.

d. TCP. Telecommunications Protocol.

e. IP. INTERNET Protocol.

f. NPS. Naval Postgraduate School, Monterey, Ca.

1.4 Security and Privacy.

The Users Manual, programs, and files used to implement the FTP process are unclassified and contain no information covered by the Privacy Act.
SECTION 2. SYSTEM SUMMARY

2.1 System Application.

a. Purpose of FTP.

FTP is a well documented software protocol for transferring information between computers within a network. The specifications for FTP are contained in the INTERNET Protocol Transition Workbook and Stanford Research Institute Request for Comments number 765 dated June, 1980.

FTP is used to effect file transfer and related operations between computers on the NPS local area network. The NPS local area network is not directly connected to any external network such as ARPANET, so file transfer beyond the local network can only be accomplished by logging in to a computer on the local network that has external access, in this case the VAX 11-780 operating under UNIX. Once logged in the user may utilize the version of FTP implemented under UNIX to access computers on ARPANET and other networks.

The FTP implementation for this thesis did not require all the features described in the FTP documentation. The goal here is to allow only active data transfers to remote sites, meaning no computer can initiate a data transfer to a Z-100. This eliminates the need for an FTP server process to handle incoming requests to a Z-100. Additionally, the mail passing facilities of FTP were not programmed. A user of this FTP system may request transfer of a file to or from the remote computer, list the directory on the remote computer, change the working directory on the remote computer, ask for help, or terminate the process. The specific FTP commands, replies, and parameters that are included in this implementation are listed in the Program Maintenance Manual [Appendix ?].

b. Capabilities of the system.

FTP is a general process for transferring files across data networks. In the NPS LAN its capabilities are limited to transfer of files only between computers operating under TCP/IP attached to ETHERNET.

c. Additional features. None.

d. Functions of the system.

FTP allows a user to copy, send, and delete files from any directory he has access to on a remote host computer.
2.2 System operation.

In order to use FTP, the files ftp.com (.cmd if under CPM-86) and hosts.fil must reside on the user's auxiliary storage device.

2.3 System Configuration.

FTP was designed to operate on Zenith model 100 microcomputers connected to the NPS local area network.

2.4 System Organization.

FTP operates as a dialogue between the FTP process on the user's micro-computer and an FTP process on the remote computer. When the user selects an option, including starting FTP, FTP will generate and send an FTP command to the remote computer. The remote computer will respond with replies that identify the state of the remote FTP process.

2.5 Performance.

a. Input.

FTP prompts the user for information including his remote user name, password, account number if required, and request. A local file to be transmitted may also be considered input to FTP. Input received from the network connection includes data, FTP replies, and coordinating information from the communication front end processor (concentrator).

b. Output.

The same type of information that is received as input is also output of FTP.

c. Response Time.

Response time will vary due to three primary reasons:

1. Function.

A request to change the working directory will generally be accomplished quicker than transfer of a large file.

2. System usage.

ETHERNET is a broadcast network
operating at ten megabits per second. Even at this high bit rate, the medium becomes quickly overloaded when the number of users increases. Additionally, the local and remote front end processors slow down significantly when use increases. With the current configuration, it is suggested that a maximum of four Zenith users operate under FTP concurrently.

3. Error occurrence.

User caused error such as misspelling a password or system error caused by transmission medium malfunction will be corrected by the system. However, response time will be severely diminished.

d. Limitations.

FTP can not be used to transfer a file to another micro-computer on the cluster. Text or command files may be transferred.

2.6 Data Base.

The only file used by FTP is the file 'HOSTS.FIL'. This file contains the name and INTERNET address of remote hosts connected to the NPS LAN. The hosts file is a text file that is write protected.

2.7 General Description of Inputs, Processing, and Outputs.

2.7.1 Inputs.

a. User input.

1. Username.

This is the user name that identifies the account to be connected to on the remote computer.

2. Password.

Password is the password that must be entered in order to connect to the account identified by 'username'.

3. Filename.

a) Local.

The local file name must be a valid file name under CPM or MSDOS. Improperly formatted file names are not accepted. If the filename is for a file to be
sent, the file must exist on the device specified in the file name.

b) Remote.

The remote file name is a string of not more than eighty characters. If the file name entered is not acceptable or does not exist in the case of getting a file, FTP will so notify the user.

4. Option.

The option selected identifies the type of request the user desires. The possible options are displayed on the screen and the user selects the letter of the desired option.

b. File data.

Text or command files may be transferred.

c. Network connection.

1. FTP replies.

These replies are textural data that provide information to the user. These replies are displayed on the user's console.

2. File data.

Text or command files may be received.

2.7.2 Output.

a. Console output.

1. FTP replies.

FTP replies received from the network connection are displayed on the console.

2. Prompts for option, user name, password, and file name are displayed on the console.

b. Network connection.

FTP commands triggered by a user specified option or by an FTP reply are send to the network connection.

c. File data.
Data received from the network is stored into the file specified by the user.

2.7.3 Process.

The process maintains the dialogue with the FTP process on the remote computer by responding to replies with commands. The appropriate command is selected by following the documented FTP protocol and prompting the user when information is needed.
ATTACHMENT 1 TO APPENDIX D

FTP RUN SHEET

A. Getting started.

FTP is programmed to operate on the Zenith model 100 attached to the cluster of micro-computers in the NPS micro-computer lab. All computers in the lab should have the files 'FTP.COM' ('FTP.CMD' if under CPM 86) and 'HOSTS.FIL' needed to utilize FTP resident on the Z-100 hard disk. If under MSDOS the files will be in directory 'LOCAL.NET'.

To use FTP, an MSDOS user must enter the directory 'LOCAL.NET'. To initiate FTP the user will enter 'FTP<cr>'. The first message displayed to the console by FTP will be 'ENTERING THE FTP PROCESS'. The user will then be prompted to select the destination. Once the destination is selected, the first user of the system may experience a short delay of up to one minute while the Z-100 transmits the control program to the concentrator. No action is required by the user until another message is displayed to the screen. From this point, the user only need respond to messages displayed on the screen.

B. SELECTING A DESTINATION.

FTP will display a list of possible destinations for an FTP connection. Selecting the desired destination is accomplished by entering the letter corresponding to the desired system name followed by a carriage return.

The destinations displayed include the recognized INTERNET name and address of computers connected to the NPS LAN. The user may select any computer on the list. However, FTP will not allow transfer of files unless the user has an account on the remote computer. If a user is not sure which computer he may connect to, he should contact an instructor or the computer science department technical representative responsible for system accounts.

C. SELECTING AN OPTION.

FTP will prompt the user to enter an option and will display a list of valid options. The option list and further messages are self explanatory. Selection is effected by entering the letter corresponding to the desired option followed by carriage return.
D. WHEN TROUBLE OCCURS.

FTP is designed to be totally robust. If a user desires to terminate the system abnormally, enter control right bracket (^]) or the prompted character for termination. If this does not work, the user may terminate the process at any time without destroying files or causing system damage by utilizing control reset. Some specific problems and response descriptions follow.

1. Excessive wait occurring. The NPS LAN is designed for a small number of users and will backup quickly as the number of users rise. Terminating while waiting can usually be accomplished by entering ^] (control right bracket). If this is not successful, enter control reset. Terminating the system abnormally in this fashion may cause a longer than normal wait required to reenter the system.

2. Keyboard does not accept characters. The system is designed to allow a user to enter data only when prompted. If the keyboard is frozen when a user prompt appears on the screen, the only recourse is control reset. At other times, a screen requesting the user to wait may appear for a substantial period. See the previous paragraph.

3. System will not accept a file name. Local filenames entered by the user will be parsed by the system to ensure proper format. If the system will not accept a filename, refer to the messages produced and documentation for the operating system in use as to proper filename format.
APPENDIX E

USER MANUAL FOR LOCAL

SECTION 1. GENERAL

1.1 Purpose of the User's Manual.

The purpose of the User's Manual for the NPS Local Area Network Local connection process is to allow students with minimal experience in computer science to effectively use the system.

1.2 Project References.


1.3 Terms and Abbreviations.

a. Local. The command file used to connect two or more terminals together.

b. Group. When two or more terminals are connected together under the 'local' process, those terminals with a common 'link' in the concentrator are considered in a group. The group defines the destinations of broadcast packets in a local connection. A terminal is connected to a particular group when it initially connects to another terminal, becoming connected to the other terminal's group. More than one group can exist at once. Two or more terminals constitute a group.

c. Link. Terminals are linked together in groups using pointers implemented in the concentrator program.

d. LAN. Local Area Network.

1.4 Security and Privacy.

The Users Manual, programs, and files used to implement the Local process are unclassified and contain no information covered by the Privacy Act.
SECTION 2. SYSTEM SUMMARY

2.1 System Application.
   a. Purpose of Local.

   Local is used to transfer files, send messages and print files among the different terminals in the LAN.

   b. Capabilities of the system.

   The local communication network system can be thought of as potentially connecting, simultaneously, all terminals in the LAN. Each remote terminal has its own connecting port in each local terminal. The remote terminals may be simultaneously getting files, sending files, exchanging messages or using the printer, all from the same terminal. The system has been designed for ease of use. For instance, the command '1' can be entered any time text is not being entered, to find out what commands are available. Multiple files can be transferred with a single text input.

   c. Additional features.

   Directory listings can be obtained from remote terminals by use of the 'directory' command. User names are passed upon command. Network status is available. Terminals can be used as mailboxes for other terminal users. Helpful information on using the system is readily available to the user.

2.2 System operation.

   Getting started - The system is started by executing the command file 'local'. Successful boot-up and initial communication with the concentrator is observed by your terminal number being displayed. Continued boot-up will display the message 'Login:'. At this time you should enter your name. If another terminal connects to yours before you enter your name, the connection will be established but you will not be logged in under your name. The automatic login feature allows a single user to connect to multiple terminals without logging in at each one.

   Once logged in, a destination terminal should be selected. Enter the terminal number for any of the other terminals. If the destination terminal you pick is not booted up in 'local', your terminal will be set to 'listen', which listens for another terminal to connect with it. The other terminal can and must log into yours to establish the connection. Once established, full use to the system is
available. The following is a summary of what can be performed:

Send files
Get files
Send messages
Receive messages
Get directory listings
Get status
Print files

Many of the commands can be executed with all the other terminals at once. For instance, to send a file to all terminals simultaneously, the 'all>' prompt needs to be on the screen. If 'send files' is selected and one or more file names entered, the files (assuming the files are available on disk) will be broadcast to all terminals in the same connection or 'group' connection (more about 'groups' later). A message, likewise, can be sent to 'all', as well as getting a directory listing from 'all'. The prompt is the terminal number that will receive an outgoing packet (if one is sent) when a command is entered. The prompt:

  15>

for instance, will direct any transmitted data, as a result of a command, to terminal number 15. If terminal 15, however, is not executing 'local', then the data goes nowhere.

To find out which terminals are in local or listen states (waiting for a local connection) enter 'n' for the netstat command. All command entries, by the way, are by a single keystroke. When the netstat information appears on your screen, all terminal numbers will be listed along with their state. The PCB state is the one you are concerned about.

To obtain a summary of all the available commands use the command information. This command opens the file 'info.txt' and presents it to you. Here is a summary of that file.

all - used to broadcast transmissions to the 'group'.

bell ON/OFF - when a message arrives to your terminal the bell will either sound or not, depending on this setting. The default setting is OFF.

change group - once established in a 'group', to change to a different group without 'quitting', use this command followed by a terminal number in the other group.
directory - to obtain a directory listing on one or more other computers use this command followed by the listing desired. ie:

[drive:] <filename | wildcard>. <ext | wildcard>, ...

When entering the filenames you are in 'enter text' mode which means to terminate use a cntl-Z. To abort the command enter cntl-Q and to review what has been entered use cntl-R.

get - to get files use this command followed by the file(s) you want to get. You are in 'enter text' mode after issuing the command so the same rules apply as above. The file(s) will be stored on the current logged disk.

information - this command displays a text file called 'info.txt' to you describing each command, one at a time.

list - a list of all the terminals which have communicated with yours is displayed. This list will be only those in the 'group' connection if a 'whose's there?' command is issued prior to 'list'.

print - used to print out one or more text files. After issuing the command the 'enter text' mode is again used to enter the file name(s).

send - to send file(s) to another terminal. 'Enter text' mode is used to enter the file name(s).

talk - to send a message to another terminal. 'Enter text' mode is used to enter the message. If more than 512 characters are entered then one message is sent and another is automatically started so that continuous entries can be made. This command can also be used to directly interact with the printer rather than creating a file to print.

Verbose - to turn on and off certain screen output when files are being transferred.

2.3 System Configuration.

Local was designed to operate on the Zenith model 100 microcomputers connected to the NPS Aegis local area network.

2.4 System Organization.

Local can have multiple connections existing with other computers simultaneously. Each connection executes independently of the others unless broadcast packets are used to send duplicate packets to all terminals in a 'group'. The system continuously monitors input from the
keyboard and the concentrator while making repeated attempts
to send any outgoing packets to the concentrator. Very
rarely does the system wait in a non-executing loop waiting
for an input to trigger the next execution.

2.5 Performance.

All communication is via RS232 9600 baud connections
which means large files will take a minimum of 1.2K bytes
per second to transfer packets to the concentrator and the
same amount of time from the concentrator to another
terminal or .6K per second. A 64K byte file will,
therefore, take more than 100 seconds. If the system is
performing multiple transfers simultaneously, obviously a
slower performance time will be experienced. Approximately
20% overhead exists in going through the concentrator
processor.

2.6 Data Base.

The only file used by Local is info.txt which is a text
file available to the user for helpful information in using
the system. The file can be accessed while executing
'local'.

2.7 General Description of Inputs, Processing, and Outputs.

2.7.1 Inputs.

a. User inputs.

1. Login name. At the present time the user
name is not used to protect access, only informational to
who's on the system. The user's name is set to upper case
upon entry.

2. Commands. The commands available to the
user are entered by a single keystroke. The first character
of the command is needed for execution of the command (upper
or lower case).

3. Text input. After certain commands text is
input from the user. All text input modes are executed and
terminated the same. If, for instance, a message is to be
sent, after entering the command 'talk', the text is input
until the message is complete. At completion of the text
input, control-Z is used to send the message. To review the
message control-R is used. To exit the text input mode
without sending the message control-Q is used. Control-H or
Delete is used to delete the last character. Full screen
editing is available, therefore, trying to delete characters
up one line will not appear on the screen, however, a review
of the text input will show any deletions. File names are
entered as text. Commas must be used between file names for
separation. Wildcards (?,* ) may be used in file names.
2.7.2 Output.

a. Messages. When messages arrive at a terminal they are displayed on the screen unless the user is in a text input mode, then they are saved until out of the text mode.

b. File transfers. When files are transferred the name of each file is presented on the screen at the beginning of transfer unless 'verbose' is OFF. In addition, each 512 bytes of the file sent or received is indicated by either a 'G'/B' when receiving or '.' when sending. The 'G'/B' indicates whether the 512 bytes was received with a good checksum or bad checksum, respectively.

2.7.3 Process. The process manages the connections, ensuring against multiple commands over the same connection.

2.8 When trouble occurs.

Most of the problems will occur when a terminal does not know what state it is in. For instance, if a connection is established then the user enters 'C at any time, the local program is terminated, however, the concentrator is unaware of the termination. Subsequent execution of 'local' may not boot-up properly. In this case, resetting the terminal (control reset) should re-initialize the terminal's state in the concentrator. If it still doesn't boot-up, the concentrator may have malfunctioned. Trying a different terminal would better confirm the latter.

The printer can be connected to as another terminal or by use of the 'print' command. The print command is recommended since the printer will be freed up at termination of printing, where as, making a connection to it will prevent others from using the printer until the connection is broken. The printer is normally terminal number 0 and should always be in either listen or local state (when using netstat).

It is possible, but rare, that all the memory blocks in either the concentrator or on a terminal, are used. The latter could be due to a packet not being received by a destination terminal while packets continue to be made and queued behind the first. In this case the terminal not accepting any more packets must be found and reinitialized.

Error messages will appear on the screen when a checksum field is not correct upon receipt of a packet. If a terminal to terminal file transfer is taking place, retransmissions will resolve the problem automatically. Checksum errors are very rare. During testing, for instance, 40,000 packets were sent without error. The key to this success are the send_trns and get_trns routines.
which ensure no conflict occurs when bi-directional transmissions occur. A possible cause of error is if a control code is sent just prior to transmitting a packet. In this case the control code could be mixed in with a packet. Any problems in using the system is directed to 'problems', a file containing observed problems (or compliments) that may help on any revision of the program. This file can be created on any terminal.
PACKAGE globall is

--CONSTANTS:
--control codes:
term : CONSTANT BYTE := BYTE(16#9D#); --]
code_cls : CONSTANT BYTE := BYTE(16#C3#); --C
code_abort : CONSTANT BYTE := BYTE(16#C1#); --A
code_status : CONSTANT BYTE := BYTE(16#D3#); --S
code_Arlog : CONSTANT BYTE := BYTE(16#D2#); --R
code_Ptlog : CONSTANT BYTE := BYTE(16#D0#); --P
code_Ftp : CONSTANT BYTE := BYTE(16#C6#); --F
code_Loc : CONSTANT BYTE := BYTE(16#CC#); --L
code_lstn : CONSTANT BYTE := BYTE(16#CF#); --O
code_reqPrt : CONSTANT BYTE := BYTE(16#F0#); --p
code_quit : CONSTANT BYTE := BYTE(16#D1#); --Q

--interrupt control codes for ni3010:
disable : CONSTANT BYTE := BYTE(16#00#);
stat_blk : CONSTANT BYTE := BYTE(16#02#);
rcv_pck : CONSTANT BYTE := BYTE(16#04#);
tx_dma_dn : CONSTANT BYTE := BYTE(16#06#);
rcv_dma_dn : CONSTANT BYTE := BYTE(16#07#);

--ni3010 port addresses:
cmd_reg : CONSTANT INTEGER := 16#00b0#;--note:if changing
stat_reg : CONSTANT INTEGER := 16#00b1#;--port addr also
tx_reg : CONSTANT INTEGER := 16#00b2#;--change bus addr
ntrp_reg : CONSTANT INTEGER := 16#00b5#;--regs in assembly
able_reg : CONSTANT INTEGER := 16#00b8#;--routine 'wr_ad'
h_cnt_reg : CONSTANT INTEGER := 16#00bc#;
l_cnt_reg : CONSTANT INTEGER := 16#00bd#;

--ni3010 control codes:
interface : CONSTANT BYTE := BYTE(16#01#);
internal : CONSTANT BYTE := BYTE(16#02#);
clear : CONSTANT BYTE := BYTE(16#03#);
go_off : CONSTANT BYTE := BYTE(16#08#);
go_on : CONSTANT BYTE := BYTE(16#09#);
diagnostic : CONSTANT BYTE := BYTE(16#0a#);
rcv_stat : CONSTANT BYTE := BYTE(16#18#);
1d_tx_dat : CONSTANT BYTE := BYTE(16#28#);
1d_snd : CONSTANT BYTE := BYTE(16#29#);
reset : CONSTANT BYTE := BYTE(16#3f#);
prom_mode : CONSTANT BYTE := BYTE(16#04#);
c1_insert_mode : CONSTANT BYTE := BYTE(16#0e#);

-- iSBC86/12A port addresses
monitor_data_prt: CONSTANT INTEGER := (16#D8#);
monitor_stat_prt: CONSTANT INTEGER := (16#DA#);
max_ad : CONSTANT INTEGER := 11;
max_tcb : CONSTANT INTEGER := 29;
max_mem_blk : CONSTANT INTEGER := 30;
num_prts : CONSTANT INTEGER := 23;
pcb_head : CONSTANT INTEGER := num_prts + 1;
threshold : CONSTANT INTEGER := 1000;
min_size : CONSTANT INTEGER := 60;
blk_size : CONSTANT INTEGER := 576;
max_flag_byt : CONSTANT INTEGER := num_prts / 8;

-- ASM machine instructions:
c1i : CONSTANT BYTE := BYTE(16#FA#);-- clear ints
sti : CONSTANT BYTE := BYTE(16#FB#);-- start ints
pushF : CONSTANT BYTE := BYTE(16#9C#);-- push flags
popF : CONSTANT BYTE := BYTE(16#9D#);-- pop flags

-- ASCII codes:
asciiA : CONSTANT BYTE := BYTE(16#41#);
asciiO : CONSTANT BYTE := BYTE(16#4F#);
asciIS : CONSTANT BYTE := BYTE(16#53#);
asciiI : CONSTANT BYTE := BYTE(16#49#);
asciIE : CONSTANT BYTE := BYTE(16#45#);
asciIM : CONSTANT BYTE := BYTE(16#4D#);
asciID : CONSTANT BYTE := BYTE(16#64#);
asciIr : CONSTANT BYTE := BYTE(16#72#);
asciix : CONSTANT BYTE := BYTE(16#78#);
asciiv : CONSTANT BYTE := BYTE(16#76#);
asciT : CONSTANT BYTE := BYTE(16#54#);
CR : CONSTANT BYTE := BYTE(16#0D#);
LF : CONSTANT BYTE := BYTE(16#0A#);
TxRdy : CONSTANT INTEGER := 0;
RxRdy : CONSTANT INTEGER := 1;
DSR : CONSTANT INTEGER := 7;
DTR : CONSTANT BYTE := BYTE(16#27#);
clr : CONSTANT BYTE := BYTE(16#25#);

-- programmable interrupt controller ports and codes:
icw1_prt : CONSTANT INTEGER := 16#00C0#; -- initialize
icw2_prt : CONSTANT INTEGER := 16#00C2#; -- cntl word:
icw4_prt : CONSTANT INTEGER := 16#00C2#; -- icw
ocw_prt : CONSTANT INTEGER := 16#00C2#; -- oper cw

icw1 : CONSTANT BYTE := BYTE(16#13#);
icw2 : CONSTANT BYTE := BYTE(16#40#);
icw4 : CONSTANT BYTE := BYTE(16#0F#);
ocw : CONSTANT BYTE := BYTE(16#DF#); -- mask other
sba : CONSTANT INTEGER := 1; -- interrupts
srf : CONSTANT INTEGER := 0;

-- TYPES:
TYPE Pstates IS
  (cls, r_init, rlogn, f_init, rftp, lstn, l_init, local, clsing);
TYPE Tstates IS
  (listen, syn_sent, syn_rcv, estab, fin_wait_1, fin_wait_2,
   close_wait, closing, last_ack, time_wait);
TYPE array2 IS ARRAY (1..2) OF byte;
TYPE array4 IS ARRAY (1..4) OF byte;
TYPE array6 IS ARRAY (1..6) OF byte;
TYPE array512 IS ARRAY (1..512) OF byte;
TYPE flg_array IS ARRAY (0..max_flag_byt) OF byte;
TYPE socket_rec IS RECORD
  ip_ad : array4;
  tcp_ad : array2;
END RECORD;
TYPE send IS RECORD
  una : array4;
  nxt : array4;
  wnd : array2;
  w11 : array4;
  w12 : array4;
  iss : array4;
END RECORD;
TYPE receive IS RECORD
  nxt : array4;
  wnd : array4;
  irs : array4;
END RECORD;
TYPE pcb_rec IS RECORD
  is_print : BOOLEAN;
  data_prt : INTEGER;
  stat_prt : INTEGER;
  cmd_prt : INTEGER;
  prtQ : INTEGER;
  s_prtq : integer;
  sent : BOOLEAN;
  Pstate : Pstates;
  time_wait : INTEGER;
  act : BOOLEAN;
  l_prt_ad : array2; -- local port address
  s_prt_ad : array2; -- secondary port address
  sec_act : BOOLEAN; -- true if sec port active
  loc_con : INTEGER;
  buf_in : socket rec;
  buf_in_cnt : INTEGER;
  pcb_ptr : INTEGER;
  snd : flg_array;
  ack : flg_array;
166
f1g_byt : INTEGER;
f1g_bit : INTEGER;
END RECORD;

TYPE tcb_rec IS RECORD
prtnum : INTEGER;
Tstate : Tstates;
loc_sock : socket_rec;
rem_sock : socket_rec;
snd : send;
rcv : receive;
ctl : BYTE;
retrnsQ : INTEGER;
END RECORD;

TYPE ad_resol_rec IS RECORD
ip_ad : array4;
eth_ad : array6;
update : INTEGER;
END RECORD;

TYPE ethpck IS RECORD
frm_stat : array2;
frm_len : INTEGER;
to_eth_ad : array6;
fm_eth_ad : array6;
type_pck : array2;
  ar_hrd : array2;--see RFC 826, Network
  ar_pro : array2;--Information Center
  ar_len : array2;--publication for details
  nul : BYTE;
  ar_op : BYTE;
  fm_eth : array6;
  fm_ip : array4;
to_eth : array6;
to_ip : array4;
END RECORD;

TYPE mem_blk IS RECORD
frm_stat : array2;
frm_len : INTEGER;
to_eth_ad : array6;
fm_eth_ad : array6;
type_pck : array2;
  ver : byte;
  serv : byte;
  len : array2;
  id : array2;
  flag : array2;
  ttl : byte;
  prot : byte;
ip_cksum : array2;
ip_scr : array4;
ip_dst : array4;
scr : array2;
dst : array2;
seq : array4;
ack : array4;
off : byte;
ctl : byte;
wnd : array2;
tcp_xsum : array2;
urg : array2;
data : array512;
crc : array4;
spare : integer;

END RECORD;

-- VARIABLES:
loc_ip_ad : array4;
--INITIALIZED TO C0 09 C8 04 IN init_mem
mem_manag_tbl : ARRAY (1..max_mem_blk) OF INTEGER;
pcb : ARRAY (0..pcb_head) OF pcb_rec;
tcb : ARRAY (0..max_tcb) OF tcb_rec;
mem : ARRAY (1..max_mem_blk) OF mem_blk;
eth : eth_pck;
ad_tbl : ARRAY (1..max_ad) of ad_resol_rec;
rêtWnd : array2;--how many bytes we can receive
nxt_prt_ad : INTEGER;--next tcp port address to use
used_blk : INTEGER;--counts blocks in use
free_blk : INTEGER;--points to free blocks of mem
loc_eth_ad : array6;
wrd : INTEGER;--used by rcv.pkg for
start_of_loop : INTEGER;--memory block ptrs
end_of_loop : INTEGER;
time_cnt : INTEGER;
ni3010_ok : BOOLEAN;
ntrpt : BYTE;

END globalll;
with globalll;
PACKAGE assylib is
use globalll;

PROCEDURE wr_ad(ad : IN INTEGER);

PROCEDURE outprt(prt : IN INTEGER; byt : IN BYTE);

PROCEDURE addarr(arr1 : IN OUT array4; arr2 : IN array4);

PROCEDURE subarr(arr1, arr2 : IN OUT array4);

168
PROCEDURE cksum(addr, num_wrds : IN INTEGER;
    reslt : OUT array2);
FUNCTION arr_to_int(arr : IN array2) RETURN INTEGER;
PROCEDURE inprt(prt : IN INTEGER; byt : OUT BYTE);
FUNCTION otstbit(num : IN BYTE; bit : IN INTEGER)
    RETURN BOOLEAN;
PROCEDURE oclrbit(num : IN OUT BYTE; bit : IN INTEGER);
PROCEDURE osetbit(num : IN OUT BYTE; bit : IN INTEGER);
FUNCTION ohi(int : IN INTEGER) RETURN BYTE;
FUNCTION olo(int : IN INTEGER) RETURN BYTE;
FUNCTION gt_equ(arr1, arr2 : IN array4) RETURN BOOLEAN;
FUNCTION lt_equ(arr1, arr2 : IN array4) RETURN BOOLEAN;
FUNCTION g_than(arr1, arr2: IN array4) RETURN BOOLEAN;
FUNCTION l_than(arr1, arr2 : IN array4) RETURN BOOLEAN;
PROCEDURE inc_arr(arr1 : IN array4; int : IN INTEGER;
    arr2 : OUT array4);
FUNCTION grtr_of(int1, int2 : IN INTEGER) RETURN INTEGER;
FUNCTION upper_nibble(byt : IN BYTE) RETURN INTEGER;
FUNCTION inc_nxt_prt_ad(addr : IN INTEGER) RETURN INTEGER;
PROCEDURE get_data(prt, addr IN INTEGER;
    len : OUT INTEGER);
PROCEDURE prt_hex(addr, num : IN INTEGER);
PROCEDURE send_trns(addr, Data_prt : IN INTEGER;
    amt : IN OUT INTEGER);
PROCEDURE get_trns(addr, Data_prt: IN INTEGER; 
    amt : IN OUT INTEGER);
PROCEDURE oput(strg : IN STRING);
PROCEDURE onew_line;
FUNCTION xsum(addr, cnt : IN INTEGER) RETURN BYTE;
END assylib;

PACKAGEx ASSEMBLY assylib;
jmp init
;--asm package must jump code not intended as initialization

PROC cksum;
;--the checksum field is the 16 bit one's complement of the
;--one's complement sum of all 16 bit words; for purposes
;--of computing the checksum, the checksum field is zero,
;--ref RFC 793 pg16, sep81

POP bx ;return address
POP di ;resultant array address
POP cx ;# of words to cksum
POP si ;starting addr
PUSH si ;restore stack
PUSH cx
PUSH di
PUSH bx

MOV dx,0 ;zero total
CLC
AGAIN:
	MOV al,[si]
	INC si
	MOV ah,[si]
	ADC dx,ax ;add to total
	INC si
	LOOP AGAIN
 NOT dx ;1's complement of total
 MOV [di],dl ;put result in array
 INC di
 MOV [di],dh
 RET

END PROC cksum;

PROC wr ad;
;--tested ok on 17 feb 86
;--this procedure writes the 20 bit address of the item whos
;--offset is passed in as a parameter to the NI3010 bus
;--address registers

e_ad_prt EQU 0B9h ;--if NI3010 port addr
h_ad_prt EQU 0BAh ;--are changed, change
l_ad_prt EQU 0BBh ;--these as well

POP di ;--return address
POP ax ;--address offset of memory block
PUSH di ;--put return address back
MOV bx,ds
MOV dx,bx
MOV  cl, 12
SHR  dx, cl
MOV  cl, 4
SHL  bx, cl
ADD  ax, bx
JNC  no_add
INC  dx

no_add: OUT  1_ad_prt, al
MOV  al, al
OUT  h_ad_prt, al
MOV  al, dl
OUT  e_ad_prt, al
RET

END PROC wr_ad;

PROC outprt;
;--tested ok on 16 feb 86
;--this procedure outputs the byte sent in parameter 2 to
;--port address in parameter 1
;--parameters are: 1. IN port address
;--                  2. IN byte to output

    POP  bx ;return addr
    POP  ax ;byte to output in al
    POP  dx ;port addr
    PUSH bx ;put return address on the stack
    OUT  dx, al ;output the byte
RET

END PROC outprt;

PROC addarr;
    POP  ax
    POP  si
    POP  di
    PUSH di
    PUSH si
    PUSH ax
    MOV  ch, [di]
    INC  di
    MOV  cl, [di]
    INC  di
    MOV  ah, [di]
    INC  di
    MOV  al, [di]
    MOV  dh, [si]
    INC  si
    MOV  dl, [si]
    INC  si
    MOV  bh, [si]
    INC  si
    MOV  bl, [si]
    ADD  bx, ax
PROC subarr;

POP ax
POP si
POP di
PUSH di
PUSH si
PUSH ax
MOV ch,[di]
INC di
MOV cl,[di]
INC di
MOV ah,[di]
INC di
MOV al,[di]
MOV dh,[si]
INC si
MOV dl,[si]
INC si
MOV bh,[si]
INC si
MOV bl,[si]
SUB bx,ax
JNC no_car
DEC dx

no_car:
SUB dx,cx
MOV [si],bl
DEC si
MOV [si],bh
DEC si
MOV [si],dl
DEC si
MOV [si],dh
RET

END PROC subarr;

172
PROC arr_to_int;
;--parameters are: 1.IN 2 byte array, note: array is on
;--stack vice address return integer value of array
  POP  bx ;--rtn addr
  POP  cx ;--arr
  PUSH bx
  MOV  al,ch
  MOV  ah,cl
  RET
END PROC arr_to_int;

PROC ohi;
      POP  bx
      POP  ax
      PUSH bx
      MOV  al,ah
      RET
END PROC ohi;

PROC olo;
      POP  bx
      POP  ax
      PUSH bx
      RET
END PROC olo;

PROC inprt;
;--tested ok on 16 feb 86
;--this procedure inputs a byte from the port address
;--in parameter 1
;--parameters are: 1.IN port address
;--                  2.OUT byte read in from port
  POP  bx ;--return address
  POP  di ;--output byte address
  POP  dx ;--input port address
  PUSH dx
  PUSH di
  PUSH bx
  IN   al,dx
  MOV  [di],al
  RET
END PROC inprt;

PROC otstbit;
;--this procedure checks to see if a bit specified in
;--parameter 2
;--is set in the byte passed in parameter 1
;--parameters are: 1.IN byte to test
;--                  2.IN bit to test
;--RETURN: T/F
  POP  di ;return address
  POP  cx ;bit
POPBx ;byte
PUSHdi
MOVdx,1
ANDcl,07H ;mask numbers > 7
SHLdx,cl ;shift left until bit is found
ANDbx,dx
JZfalsetstbit
MOVax,1 ;leave value for true in ax
RET
falsetstbit:
MOVax,0 ;leave value for false in ax
RET
ENDPROCfalsetstbit;

PROCoclrbit;
;--this procedure resets a bit specified in parameter 2
;--for the byte passed in parameter 1
;--parameters are: 1.IN byte to reset bit in
;--2.IN bit to reset
POPDi ;return address
POPCx ;bit
POPSi ;address of number
PUSHsi
PUSHcx
PUSHdi
MOVdx,1
ANDcl,07H ;mask numbers > 7
SHLdx,cl ;shift left until bit is found
MOVbl,[si]
NOTdx ;1's compliment
ANDbx,dx
MOV[si],bl
RET
ENDPROCoclrbit;

PROCosetbit;
;--this procedure sets a bit specified in parameter 2
;--for the byte passed in parameter 1
;--parameters are: 1.IN byte to reset bit in
;--2.IN bit to reset
POPDi ;return address
POPCx ;bit
POPSi ;address of number
PUSHsi
PUSHcx
PUSHdi
MOVdx,1
ANDcl,07H ;mask numbers > 7
SHLdx,cl ;shift left until bit is found
MOVbl,[si]
OR    bx, dx
MOV    [si], bl
RET
END PROC osetbit;

PROC gt_equ;
    POP    ax      ; return
    POP    si      ; second array
    POP    di      ; first array
    PUSH   ax
    MOV    cx, 2
lagain: MOV    ah, [di]
INC    di
MOV    al, [di]
INC    di
MOV    bh, [si]
INC    si
MOV    bl, [si]
INC    si
SUB    ax, bx
JC      false1
JNZ    true1
LOOP   lagain
true1:  MOV    ax, 1
        RET
false1: MOV    ax, 0
        RET
END PROC gt_equ;

PROC lt_equ;
    POP    ax      ; return
    POP    si      ; second array
    POP    di      ; first array
    PUSH   ax
    MOV    cx, 2
back:   MOV    ah, [di]
INC    di
MOV    al, [di]
INC    di
MOV    bh, [si]
INC    si
MOV    bl, [si]
INC    si
SUB    bx, ax
JC      false2
JNZ    true2
LOOP   back
true2:  MOV    ax, 1
        RET
false2: MOV    ax, 0
        RET
END PROC lt_equ;

PROC g_than;
    POP ax ;return
    POP si ;second array
    POP di ;first array
    PUSH ax
    MOV cx,2
    lback: MOV ah,[di]
          INC di
          MOV al,[di]
          INC di
          MOV bh,[si]
          INC si
          MOV bl,[si]
          INC si
          SUB ax,bx
          JC false3
          JNZ true
          LOOP lback
false3: MOV ax,0
        RET
true: MOV ax,1
        RET
END PROC g_than;

PROC l_than;
    POP ax ;return
    POP si ;second array
    POP di ;first array
    PUSH ax
    MOV cx,2
    again1: MOV ah,[di]
            INC di
            MOV al,[di]
            INC di
            MOV bh,[si]
            INC si
            MOV bl,[si]
            INC si
            SUB bx,ax
            JC false4
            JNZ true3
            LOOP again1
false4: MOV ax,0
        RET
true3: MOV ax,1
        RET
END PROC l_than;

PROC inc_arr;  --(arr1 : IN array4,
                ;--int : IN INTEGER; arr2 OUT array4)
POP dx ;--return addr
POP di ;--output array address
POP bx ;--int
POP si ;--input array address
PUSH si
PUSH bx
PUSH di
PUSH dx
MOV ch, [si]
INC si
MOV cl, [si]
INC si
MOV ah, [si]
INC si
MOV al, [si]
ADD ax, bx
JNC no_car_over
INC cx

no_car_over:
MOV [di], ch
INC di
MOV [di], cl
INC di
MOV [di], ah
INC di
MOV [di], al
RET

END PROC inc_arr;

PROC grtr_of;--function grtr_of(intl, int2) return intx
POP dx ;rtn addr
POP ax ;int2
POP bx ;int1
PUSH dx
CMP bx, ax
JG intl_big
RET ;int2 bigger

intl_big:
MOV ax, bx
RET

END PROC grtr_of;

PROC upper_nibble; --function upper_nibble
(byt : IN byte) return byte
POP dx ;rtn addr
POP ax ;byt
PUSH dx
AND ax, 00f0H
MOV cl, 4
SHR ax, cl
RET

END PROC upper_nibble;
PROC inc_nxt_prt_ad;      --function returns an integer
;--tested ok on 27 feb 86
    POP   di
    POP   ax
    PUSH  di
    INC   ax
    JNZ   no_ovrflw
    MOV   ax,0400H

no_ovrflw:
    RET
END PROC inc_nxt_prt_ad;

PROC get_data;(prt : IN INTEGER; addr : IN INTEGER;
;  len : OUT INTEGER);

thrshld EQU 100
DSR    EQU 80H
rxRdy  EQU 2H

    POP   ax     ;rtn
    POP   si     ;addr of len
    POP   di     ;addr of storage area
    POP   dx     ;dataport
    PUSH  dx
    PUSH  di
    PUSH  si
    PUSH  ax
    MOV   bx,0
    MOV   cx,thrshld

nextbyt:INC  dx
NotRdy: IN   al,dx
            AND  al,DSR
            JZ    done
            DEC   cx
            JZ    done
            IN    al,dx
            AND  al,rxRdy
            JZ    NotRdy
            DEC   dx
            IN    al,dx
            MOV   [di],al
            INC   di
            INC   bx
            CMP   bx,512
            JZ    done
            MOV   cx,thrshld
            JMP   nextbyt

done:  MOV   [si],bx
    RET

END PROC get_data;
PROC prntch;
    MOV   dl,al
    MOV   ah,02H
    INT   224
    RET
END PROC prntch;

PROC prnt_hex;(addr : IN INTEGER; num : IN INTEGER);
asciisspace EQU 20H
    POP   ax
    POP   cx
    POP   si
    PUSH  ax
again2: MOV   al,[si]
    SHR   al,1
    SHR   al,1
    SHR   al,1
    SHR   al,1
    CMP   al,10
    JL    lower1
    ADD   al,31H
    CALL  prntch
    JMP   nibble2
lower1:  ADD   al,30H
    CALL  prntch
nibble2:MOV   al,[si]
    AND   al,0FH
    CMP   al,10
    JL    lower2
    ADD   al,31H
    CALL  prntch
    JMP   next
lower2:  ADD   al,30H
    CALL  prntch
next:    MOV   al,asciisspace
    CALL  prntch
    INC   si
    LOOP  again2
END PROC prnt_hex;

PROC send_trns;(addr, Data_prt : IN INTEGER;
    ;amt : IN OUT INTEGER) is
    wait_time EQU 1000
    rs232_delay EQU 400
    DTR    EQU 27H
    TxRdy  EQU 1
    RxRdy_DSR EQU 82H
    clr    EQU 25H
    CLI
    POP   ax;rt
    POP   di;amt
    POP   dx;Data_prt
POP si ;addr
PUSH si
PUSH dx
PUSH di
PUSH ax
INC dx
IN al,dx
AND al,DSR
JNZ send_trnsD2
MOV al,DTR
INC dx
INC dx
OUT dx,al
DEC dx
DEC dx
IN al,dx
AND al,DSR
JNZ send_trnsD ;--too soon for DSR
MOV bx,wait_time
MOV cx,[di]

send_trnsL1:
  IN al,dx
  AND al,DSR
  JNZ send_trnsL5
  DEC bx
  JZ send_trnsD
  JMP send_trnsL1

send_trnsL5:
  ;--this routine was inserted
  NOP
  IN al,dx ;--after repeated tests in which
  AND al,DSR ;--an occasional timing problem
  JZ send_trnsD ;--would appear

send_trnsL2:
  IN al,dx
  AND al,DSR
  JZ send_trnsD
  MOV al,[si]
  DEC dx
  OUT dx,al
  INC si
  INC dx

send_trnsL3:
  IN al,dx
  AND al,TxRdy
  JZ send_trnsL3
  LOOP send_trnsL2
  MOV [di],cx
  MOV cx,rs232_delay

send_trnsL4:
  NOP
  LOOP send_trnsL4

send_trnsD:
MOV al, clr
INC dx
INC dx
OUT dx, al
DEC dx
DEC dx
MOV cx, wait_time

send_trnsD1:
IN al, dx
AND al, DSR
JZ send_trnsD2
LOOP send_trnsD1

send_trnsD2:
STI
RET

END PROC send_trns;

PROC get_trns; (addr, data_prt : IN INTEGER,
amt : IN OUT INTEGER) is
CLI
POP ax ; --rtn
POP si ; --amt
POP dx ; --data_prt
POP di ; --addr
PUSH di
PUSH dx
PUSH si
PUSH ax
MOV cx, [si]
MOV bx, 0
INC dx
IN al, dx
AND al, DSR
JZ get_prt_dataD
INC dx
INC dx
MOV al, DTR
OUT dx, al
DEC dx
DEC dx
MOV ah, 255

get_prt_dataD:
IN al, dx
AND al, RxRdy_DSR
JZ get_prt_dataD1
AND al, RxRdy
JNZ get_prt_dataD1
DEC ah
JNZ get_prt_dataL
JMP get_prt_dataD1
get_prt_dataL1:
  DEC dx
  IN al,dx
  MOV [di],al
  INC di
  INC bx
  INC dx
  MOV ah,255
  LOOP get_prt_dataL
get_prt_dataD1:
  MOV al,clr
  INC dx
  INC dx
  get_prt_dataD:
  MOV [si],bx
  STI
  RET
END PROC get_trns;

PROC oput;(strg : IN STRING) is
  monitor_data EQU 0d8H
  monitor_stat EQU 0daH
  POP ax
  POP si
  PUSH ax
  MOV cl,[si]
  MOV ch,0
  AND cx,cx
  JZ oputD
  oputL2: INC si
  oputL1: IN al,monitor_stat
  AND al,TxRdy
  JZ oputL1
  MOV al,[si]
  OUT monitor_data,al
  LOOP oputL2
  oputD: RET
END PROC oput;

PROC onew_line;() is
  CR EQU 0dH
  LF EQU 0aH
  MOV bl,CR
  MOV cx,2
  onew_lineL:
  IN al,monitor_stat
  AND al,TxRdy
  JZ onew_lineL
  MOV al,bl
  OUT monitor_data,al
  MOV bl,LF

  182
LOOP onew_lineL
RET
END PROC onew_line;

PROC xsum; (addr : IN INTEGER, cnt : IN INTEGER) is
   POP ax
   POP cx
   POP si
   PUSH ax
   MOV al, 0
   xsuml: MOV bl, [si]
   XOR al, bl
   INC si
   LOOP xsuml
   RET
END PROC xsum;

init:
END assylib;
with globall;
PACKAGE lib is
   use globall;
   PROCEDURE oPUT(num IN INTEGER);
   PROCEDURE get_memory (next: OUT INTEGER);
   PROCEDURE give_memory(inx: IN INTEGER);
   PROCEDURE perf_cmd(cmd: IN BYTE);
   PROCEDURE trn_pck(ad: IN INTEGER; size: IN INTEGER);
   PROCEDURE resolve_ad(ip_ad: IN OUT array4;
      eth_ad: OUT array6; rslt: OUT BOOLEAN);
   PROCEDURE get_tcb_ndx(arr: IN OUT array2;
      tbl: OUT INTEGER; found: OUT BOOLEAN);
   PROCEDURE pcb_cls(prt_num: in integer);
   PROCEDURE pcb_abort(prt_num: IN INTEGER);
   PROCEDURE tcb_cls(ndx: IN INTEGER);
   PROCEDURE activate_prt(prt: IN INTEGER);
   PROCEDURE give_status(port: IN INTEGER);
END lib;
PRAGMA condcomp(ON);

WITH assylib, globall;
PACKAGE BODY lib is
USE assylib, global;
-- last updated 7 June 86

PROCEDURE oPUT(integr : IN INTEGER) is
  int  : INTEGER;
  num  : INTEGER;
  byt  : BYTE;
  started : BOOLEAN;

PROCEDURE prntnum(num : IN INTEGER) is
  zero  : CONSTANT BYTE := BYTE(16#30#);
  one   : CONSTANT BYTE := BYTE(16#31#);
  two   : CONSTANT BYTE := BYTE(16#32#);
  three : CONSTANT BYTE := BYTE(16#33#);
  four  : CONSTANT BYTE := BYTE(16#34#);
  five  : CONSTANT BYTE := BYTE(16#35#);
  six   : CONSTANT BYTE := BYTE(16#36#);
  seven : CONSTANT BYTE := BYTE(16#37#);
  eight : CONSTANT BYTE := BYTE(16#38#);
  nine  : CONSTANT BYTE := BYTE(16#39#);
  question: CONSTANT BYTE := BYTE(16#3F#);
BEGIN
  LOOP
    inprt(monitor_stat_prt,byt);
    EXIT WHEN otstbit(byt,TxRdy);
  END LOOP;
  CASE num is
    WHEN 0 => outprt(monitor_data_prt,zero);
    WHEN 1 => outprt(monitor_data_prt,one);
    WHEN 2 => outprt(monitor_data_prt,two);
    WHEN 3 => outprt(monitor_data_prt,three);
    WHEN 4 => outprt(monitor_data_prt,four);
    WHEN 5 => outprt(monitor_data_prt,five);
    WHEN 6 => outprt(monitor_data_prt,six);
    WHEN 7 => outprt(monitor_data_prt,seven);
    WHEN 8 => outprt(monitor_data_prt,eight);
    WHEN 9 => outprt(monitor_data_prt,nine);
    WHEN others =>
      outprt(monitor_data_prt,question);
  END CASE;
END prntnum;

BEGIN
  int := integr;
  started := FALSE;
  IF int < 0 THEN
    oPUT("-");
  END IF;
  IF int / 10000 > 0 THEN
    num := int / 10000;
  END IF;
procedure get_memory (next: out integer) is
  -- AUTHOR: ALEC YASINSAC    -- DATE: 16 FEB 86
  -- INPUT: 1. GLOBAL TABLE MEM_MANAG_TBL.
  -- OUTPUT: 1. THE INDEX OF THE MEM ARRAY RECORD TO BE USED.
              2. THE GLOBAL TABLE MEM MANAG_TBL IS UPDATED.
              3. GLOBAL VARIABLES 'FREE_BLK' AND 'USED_BLK'.
              4. GLOBAL VAR SNDWND IS MODIFIED IF MEMORY USAGE GOES ABOVE 50%.
  -- EXTERNAL MODULES CALLED: 1. NONE.
  -- DESCRIPTION: GET_MEMORY WILL RETURN THE INTEGER FROM THE GLOBAL VARIABLE FREE_BLK. FREE_BLK IS THEN SET EQUAL TO MEM_MANAG_TBL(FREE_BLK) WHICH POINTS TO THE NEXT AVAILABLE BLOCK. THE GLOBAL USED_BLK IS INCREMENTED. CONTENTS OF THE USED INDEX MEM_MANAG_TBL IS SET TO ZERO. INTS ARE DISABLED AT THE BEGINNING AND ENABLED AT THE END.
  --used_blk is global var that counts the number of memory blocks in use.
  --free_blk is a global var that points to next available memory record.

one_half_mem_blk : CONSTANT INTEGER := max_mem_blk / 2;

begin    --BEGIN GET_MEMORY
  prntnum(num);
  int := int rem 10000;
  started := TRUE;
  END IF;
  IF int / 1000 > 0 OR started THEN
    num := int / 1000;
    prntnum(num);
    int := int rem 1000;
    started := TRUE;
  END IF;
  IF int / 100 > 0 OR started THEN
    num := int / 100;
    prntnum(num);
    int := int rem 100;
    started := TRUE;
  END IF;
  IF int / 10 > 0 OR started THEN
    num := int / 10;
    prntnum(num);
    int := int rem 10;
    started := TRUE;
  END IF;
  num := int;
  prntnum(num);
END oPUT;

---
asm pushF; -- save state of interrupts.
asm cli; -- DISABLE INTERRUPTS.
if used_blk > one_half_mem_blk then-- MEANS MEMORY IS
  rcvWnd(1) := byte(0); --HALF FULL. RCV WND = 0 STOPS A
  rcvWnd(2) := byte(0); --REMOTE FROM SENDING. LAG TIME
end if; -- WILL OCCUR BEFORE REMOTE HOST GETS THE MSG.

if free_blk > 0 then
  next := free_blk; --NEXT TO POINT TO NEXT AVAIL BLOCK.
  free_blk := mem_manag_tbl(next);
  mem_manag_tbl(next) := old; --FREE_BLK TO SUBSEQUENT AVAILABLE BLOCK.
  used_blk := used_blk + 1;
else
  next := 0; --IF MEMORY IS FULL, RETURN ZERO.
end if;
asm popF; -- RESTORE STATE OF INTERRUPTS.

procedure give_memory(inx: in integer) is
  -- AUTHOR: ALEC YASINSAC -- DATE: 16 FEB 86
  -- INPUT: 1. INDEX OF MEMORY BLOCK TO BE RETURNED.
  -- OUTPUT: 1. UPDATED GLOBAL ARRAY 'MEM'.
  -- 2. UPDATED GLOBAL ARRAY 'MEM_MANAG_TBL'.
  -- 3. UPDATED VARS FREE_BLK AND USED_BLK.
  -- 4. GLOBAL VAR SND WND IS MODIFIED IF MEMORY
     -- USAGE DROPS BELOW 30%
  -- EXTERNAL MODULES CALLED: 1. NONE.
  -- DESCRIPTION: GIVE_MEMORY SETS FREE_BLK EQUAL TO THE INPUT
     -- PARAMATER AND SETS THE MEM_MANAG_TBL ENTRY INDEXED BY
     -- THE INPUT PARAMETER TO THE OLD FREE_BLK. THE 'USED_BLK'
     -- COUNTER IS DECREMENTED.

old: integer;
--used_blk is a global var that counts the number of
--memory blocks in use
--free_blk is a global var that points to the next
--available memory record
one_third_mem_blk : CONSTANT INTEGER := max_mem_blk / 3;

begin
asm pushF; -- save state of interrupts
asm cli:
old := free_blk;
free_blk := inx; --SET FREE_BLK TO POINT TO RETURNED BLOCK.
mem_manag_tbl(inx) := old; --SET MEM_MANAG_TBL ENTRY OF
  --RETURNED BLOCK TO POINT TO THE OLD FREE BLOCK.
used_blk := used_blk - 1; --DECREMENT USED_BLK COUNTER.
if used_blk < one_third_mem_blk then -- MEMORY IS
  rcvWnd(1) := byte(02); --LESS THAT 1/3 FULL. RCVWnd=
PROCEDURE perfcmd(cmd : IN BYTE) is
  err : CONSTANT BOOLEAN := FALSE;
  val : BYTE;
  prt : INTEGER;
BEGIN
  outprt(cmd_reg, cmd);
  LOOP
    inpnt(ntrpt_reg, val);
    EXIT WHEN otstbit(val, 0);
  END LOOP;
  inpnt(stat_reg, val);
  IF INTEGER(val) > 1 THEN
    ni3010_ok := err;
  END IF;
END perfcmd;

PROCEDURE trn_pck(ad : IN INTEGER; size : IN INTEGER) is
  -- author      r l hartman
  -- date        15 feb 86
  -- input parameters address of block to transmit
  -- size of block to transmit (# of bytes)
  -- this procedure performs a DMA transfer of the block
  -- designated to the NI3010 ethernet controller board
  val : BYTE;
BEGIN
  IF ntrpt = disable THEN
    ASM  sti;
    wr_ad(ad);
    outprt(h_cnt_reg, ohi(size));
    outprt(l_cnt_reg, olo(size));
    outprt(able_reg, tx_dma_dn);
  ELSE
    ASM  cli;
    LOOP
      EXIT WHEN ntrpt = rcv_pck;
      ASM  sti;
      LOOP
        EXIT WHEN ntrpt = rcv_pck;
      END LOOP;
      ASM  cli;
    END LOOP;
    ntrpt := disable;
    outprt(able_reg, disable);
  ASM  sti;
END trn_pck;
PROCEDURE resolve_ad(ip_ad : IN OUT array4;
  eth_ad : OUT array6; rslt : OUT BOOLEAN) is
  -- author r l hartman
  -- date 15 feb 86
  -- input parameter  internet protocol address
  -- output parameters  physical ethernet address
  --  boolean indicating if the address was found
  -- this procedure resolves the physical addr of a destination
  -- Ethernet controller board by looking up the ip address
  -- in the table. if the physical address is not known the
  -- result will be false.
  found  CONSTANT BOOLEAN := TRUE;
  ndx  INTEGER;
BEGIN
  ndx := 1;
  rslt := NOT found;
  LOOP
    EXIT WHEN ndx > max_ad;
    IF ad_tbl(ndx).update /= 0 THEN
      IF ip_ad = ad_tbl(ndx).ip_ad THEN
        eth_ad := ad_tbl(ndx).eth_ad;
        rslt := found;
        EXIT;
      ELSE ndx := ndx + 1;
      END IF;
    ELSE
      ndx := ndx + 1;
    END IF;
  END LOOP;
END resolve_ad;

PROCEDURE get_tcb_ndx(arr : IN OUT array2;
  index : OUT INTEGER; found : OUT BOOLEAN) is
  -- author r l hartman
  -- date 18 feb 86
  -- this procedure performs a double hashing function to find
  -- the tcb record in the array. note: the max_tcb constant
  -- in global.spc must be a prime number in order to maximize
  -- the number of records available.
  incr  : INTEGER;
int : INTEGER;
BEGIN
  int := arr_to_int(arr); -- change array to integer addr
  incr := 0;
  index := int MOD max_tcb;
  found := TRUE;
  LOOP
    IF tcb(index).prt_num > num_prt THEN
      found := FALSE;
      EXIT;
    ELSE
      EXIT WHEN tcb(index).loc_sock.tcp_ad = arr;
    END IF;
    incr := int MOD max_tcb - 1;
    index := (index + incr) MOD max_tcb;
  END LOOP;
END get_tcb_ndx;

procedure pcb_cls( prt_num: in integer) is
  -- AUTHOR: ALEC YASINSAC  -- DATE: FEB 1986
  -- INPUT: PORT NUMBER OF CONNECTION TO BE TERMINATED
  -- OUTPUT: 1. MODIFIED GLOBAL VARIABLES FROM THE PCB RECORD
  --     A. PSTATE
  --     B. TIME WAIT
  --     C. BUF IN CNT
  -- 2. CNTL Character TO Z100 TO TERMINATE RLOGIN.
  -- DESCRIPTION: THIS PROCEDURE REINITIALIZES FIELDS IN PCB
  -- TABLE TO ALLOW A NEW CONNECTION TO BE ESTABLISHED AND
  -- SENDS A CONTROL CHARACTER TO THE Z-100 TO TERMINATE THE
  -- APPLICATION PROGRAM ON THAT MACHINE. PRTQ AND
  -- BUF_OUT_PTR FIELDS MUST BE RESET BY TERMINATING ROUTINE
  -- AND STORED PACKETS HANDLED APPROPRIATELY.
begin
  outprt(pcb(prt_num).data_prt,code_cls);
  pcb(prt_num).pstate := cls;
  pcb(prt_num).time_wait := 0;
  pcb(prt_num).buf_in_cnt := 0;
end pcb_cls;

procedure pcb_abort(prt_num : in integer) IS
  -- DISCRPTION: THIS PROCEDURE RETURNS ALL MEMORY LOCATIONS
  -- CONTAINING DATA FOR THE PRIMARY CONNECTION OF THE PORT #
  -- TO BE ABORTED, CHANGE THE STATE TO CLOSED, INITIALIZE PCB
  -- TIMWAIT FIELD, AND SEND THE CHARACTER TO THE Z100 TO
  -- TERMINATE THE CONNECTION AS APPROPRIATE.
  qadd, inx: integer;
found: boolean;
begin
while pcb(prt_num).prtq /= 0
loop --DELETE DATA STORED FOR PORT AND RETURN MEMORY.
  qadd := mem_manag_tbl(pcb(prt_num).prtq);
  give_memory(pcb(prt_num).prtq);
  pcb(prt_num).prtq := qadd;
end loop;
if pcb(prt_num).sec_act then
  pcb(prt_num).pstate := clsing;
  while pcb(prt_num).s_prtq /= 0
  loop --DELETE DATA ON SECONDARY CONNECTION.
    qadd := mem_manag_tbl(pcb(prt_num).s_prtq);
    give_memory(pcb(prt_num).s_prtq);
    pcb(prt_num).s_prtq := qadd;
  end loop;
else
  pcb(prt_num).pstate := cls;
end if;
pcb(prt_num).time_wait := 0;
outprt(pcb(prt_num).data_prt, code_cls);
end pcb_abort;

PROCEDURE tcb_cls(ndx IN INTEGER) is
  ptr : INTEGER;
BEGIN
  ASM
  pushF;
  ASM
  cli;
  LOOP
    ptr := tcb(ndx).retrnsQ;
    EXIT WHEN ptr = 0;
  tcb(ndx).retrnsQ := mem_manag_tbl(ptr);
  give_memory(ptr);
  END LOOP;
  tcb(ndx).prt_num := 99;
  ASM
  popF;
  END tcb_cls;

PROCEDURE activate_prt(prt IN INTEGER) is
BEGIN
  pcb(prt).pcb_ptr := pcb(pcb_head).pcb_ptr;
  pcb(pcb_head).pcb_ptr := prt;
END activate_prt;

PROCEDURE give_status(port IN INTEGER) is
  hdr_len : CONSTANT INTEGER := 6;
  ndx : INTEGER;
  prt : INTEGER;
  found : BOOLEAN;
  listed : BOOLEAN;
  box : ARRAY (1 .. max_mem_blk) of BOOLEAN;

ptr : INTEGER;
blk : INTEGER;
amt : INTEGER;
val : BYTE;

BEGIN
  @oPUT("s_prtQ should be 0, = "); oPUT(pcb(port).s_prtQ);
  @oNEW_LINE;
  @ASM cli;                     --the remaining code is temporary
  @listed := FALSE;

  @IF used_blk /= 0 THEN
    @FOR i IN 1..max_mem_blk LOOP
      @box(i) := TRUE;
    @END LOOP;
    @prt := free_blk;
    @LOOP
      @EXIT WHEN prt = 0;
      @box(prt) := FALSE;
      @prt := mem_manag_tbl(prt);
    @END LOOP;
    @oPUT("---The following memory blks are not free--");
    @oNEW_LINE;
    @oPUT("dst scr seq ack ");
    @oPUT("llen crit wnd");
    @oNEW_LINE;
    @FOR i IN 1..max_mem_blk LOOP
      @IF box(i) THEN
        @oNEW_LINE;
        @oPUT(INTEGER(mem(i).dst(1)));  
        @oPUT(" ");
        @oPUT(INTEGER(mem(i).dst(2)));  
        @oPUT(" ");
        @oPUT(INTEGER(mem(i).scr(1)));  
        @oPUT(" ");
        @oPUT(INTEGER(mem(i).scr(2)));  
        @oPUT(" ");
        @FOR j IN 1..4 LOOP
          @oPUT(INTEGER(mem(i).seq(j)));  
          @oPUT(" ");
        @END LOOP;
        @oPUT(" ");
        @FOR j IN 1..4 LOOP
          @oPUT(INTEGER(mem(i).ack(j)));  
          @oPUT(" ");
        @END LOOP;
        @oPUT(" ");
        @oPUT(INTEGER(mem(i).len(1)));  
        @oPUT(" ");
        @oPUT(INTEGER(mem(i).len(2)));  
        @oPUT(" ");
        @oPUT(INTEGER(mem(i).ctl));
      @END IF
    @END LOOP;
  @END IF

  191
@oPUT(" ");
@oPUT(INTEGER(mem(i) .wnd(1)));
@oPUT(" ");
@oPUT(INTEGER(mem(i) .wnd(2)));
@oNEW_LINE;
@END IF;
@END LOOP;
@END IF;
IF pcb(port).s_prtQ = 0 THEN
get_memory(blk);
IF blk /= 0 THEN
ptr := 1;
mem(blk) .data(ptr) := BYTE(num_prts);
prr := ptr + 1;
FOR i IN 0..num_prts LOOP
CASE pcb(i) .Pstate is
WHEN cls => mem(blk) .data(ptr) := BYTE(0);
WHEN r_init => mem(blk) .data(ptr) := BYTE(1);
WHEN r_logn => mem(blk) .data(ptr) := BYTE(2);
WHEN f_init => mem(blk) .data(ptr) := BYTE(3);
WHEN rftp => mem(blk) .data(ptr) := BYTE(4);
WHEN lsmtp => mem(blk) .data(ptr) := BYTE(5);
WHEN l_init => mem(blk) .data(ptr) := BYTE(6);
WHEN local => mem(blk) .data(ptr) := BYTE(7);
WHEN closing => mem(blk) .data(ptr) := BYTE(8);
WHEN others => mem(blk) .data(ptr) := BYTE(9);
END CASE;
prr := prrr + 1;
get_tcb ndx(pcb(i) .1_prt_ad,ndx,found);
IF found THEN
mem(blk) .data(ptr) := pcb(i) .1_prt_ad(1);
prr := prrr + 1;
mem(blk) .data(ptr) := pcb(i) .1_prt_ad(2);
prr := prrr + 1;
CASE tcb(ndx) .Tstate is
WHEN listen => mem(blk) .data(ptr) := BYTE(1);
WHEN syn_sent =>
mem(blk) .data(ptr) := BYTE(2);
WHEN syn_rcv =>
mem(blk) .data(ptr) := BYTE(3);
WHEN estab => mem(blk) .data(ptr) := BYTE(4);
WHEN fin_wait_1 =>
mem(blk) .data(ptr) := BYTE(5);
WHEN fin_wait_2 =>
mem(blk) .data(ptr) := BYTE(6);
WHEN close_wait =>
mem(blk) .data(ptr) := BYTE(7);
WHEN closing =>
mem(blk) .data(ptr) := BYTE(8);
WHEN last_ack =>
mem(blk) .data(ptr) := BYTE(9);
WHEN time_wait =>

192
mem(blk).data(ptr) := BYTE(10);
WHEN others=>mem(blk).data(ptr) := BYTE(0);
END CASE;
ptr := ptr + 1;
ELSE
FOR i IN 1..3 LOOP
  mem(blk).data(ptr) := BYTE(0);
  ptr := ptr + 1;
END LOOP;
END IF;
END LOOP;
mem(blk).data(ptr) := BYTE(used_blk);
ptr := ptr + 1;
mem(blk).data(ptr) := BYTE(max_mem_blk);
ptr := ptr + 1;
amt := 0;
FOR i IN 0..max_tcb LOOP
  IF tcb(i).prt_num <= num_prts THEN
    amt := amt + 1;
  END IF;
END LOOP;
mem(blk).data(ptr) := BYTE(amt);
ptr := ptr + 1;
mem(blk).data(ptr) := BYTE(max_tcb);

ASM cli;
outprt(able_reg disable);
ASM sti;
perf_cmd(rcv_stat);
ptr := ptr + 1;
LOOP
  inprt(ntrpt_reg,val);
  EXIT WHEN otstbit(val,1);
  IF otstbit(val,0) THEN
    inprt(stat_reg,val);
    mem(blk).data(ptr) := val;
    ptr := ptr + 1;
  END IF;
END IF;
END LOOP;
outprt(able_reg,ntrpt);
mem(blk).wnd(1) := BYTE(port);
mem(blk).wnd(2) := BYTE(port);
mem(blk).tcp_xsum(1) := code_status;
mem(blk).tcp_xsum(2) := BYTE(0);
mem(blk).urg(1) := ohi(ptr);
mem(blk).urg(2) := olo(ptr);
mem(blk).tcp_xsum(2) := xsum(mem(blk).wnd'ADDRESS,
ptr+hdr_len);
pcb(port).s_prtQ := blk;
osetbit(pcb(port).ack(pcb(port).flg_byt),
  pcb(port).flg_bit);
osetbit(pcb(port).snd(pcb(port).flg_byt),
193
PACKAGE ntrpthd is

PROCEDURE assy_ntrpt_hdl;

END ntrpthd;

with rcv;
PACKAGE ASSEMBLY ntrpthd;
jmp init_ntrpt;
--asm package must jump code not intended as initialization
PROC assy_ntrpt_hdl;
  CLI
  PUSHF
  PUSH ax
  PUSH bx
  PUSH cx
  PUSH dx
  PUSH si
  PUSH di
  PUSH bp
  PUSH ds
  PUSH es
  CALL ntrpt_hdl
  POP es
  POP ds
  POP bp
  POP di
  POP si
  POP dx
  POP cx
  POP bx
  POP ax
  POPF
  STI
  IRET
END PROC assy_ntrpt_hdl;

init_ntrpt:
  ;--Initialization of interrupt vector into main memory
  PUSH ds
  MOV bx,assy_ntrpt_hdl
  MOV ax,0
MOV ds, ax
MOV di, 114h
MOV [di], bx
MOV bx, cs
INC di
INC di
MOV [di], bx
POP ds

END ntrpthd;
with global;
PACKAGE convblk is
use global;

PROCEDURE conv_blk(blk : IN OUT mem_blk);

END convblk;
with ethrec;
PACKAGE ASSEMBLY convblk;
;--this procedure used to allow converting memory from type
;--mem to type eth (see global.spc)
jmp init
PROC conv_blk;

            JMP    eth_rcv;

END PROC conv_blk;
init:
END convblk;

package pcbrec is

    procedure pcb_rcv(inx, prt: in integer);

    procedure adv_pcb_state(nr : IN INTEGER);

end pcbrec;

with assylib, lib, global1;
package body pcbrec is

--FILE NAME: PCBREC_PKG
--PROCEDURES CONTAINED: 1. PCB_RCV.
-- 2. ADV_PCB_STATE.
-- 3.
-- 4.
--AUTHOR: ALEC YASINSAC
--DATE: FEB 1986
--EXTERNAL REFERENCES:
    --GLOBAL1.SPC CONTAINS ALL GLOBAL VARIABLES AND TYPES.

195
--LIB_PKG WHICH CONTAINS OUR UTILITY PROCEDURES.
--ASSYLIB WHICH CONTAINS ASSEMBLY UTILITY PROCEDURES.

--INPUT: 1)
-- 2)
--OUTPUT: 1)
--COMPILER: THIS PACKAGE WAS CODED TO COMPILE ON JANUS ADA
  --UNDER CPM 86.
--DESCRIPTION:

procedure pcb_rcv(inx, prt: in integer) is
  --AUTHOR: ALEC YASINSAC  --DATE: FEB 86
  --INPUT: 1. INX IS THE MEMORY BLOCK INDEX OF INCOMING DATA.
          2. PRT IS THE PORT NUMBER BEING PROCESSED.
  --OUTPUT: 1. FIELDS MODIFIED IN THE GLOBAL TABLE PCB:
             PSTATE, PRTQ, BUF_OUT_CNT, BUF_OUT_PTR,
             2. FIELD MODIFIED IN THE GLOBAL TABLE TCB: TSTATE
  --EXTERNAL MODULES CALLED: 1. NONE.
  --DESCRIPTION:
  -- THE DATA FROM THE PACKET WILL BE ADDED TO THE END OF THE
  -- QUE FOR THE PRIMARY OR SECONDARY CONNECTION.

use global, assylib;
ind : integer;

BEGIN

if pcb(prt).l_prt_ad = mem(inx).dst then
  ind := pcb(prt).prtq;
  IF ind = 0 THEN --NOTHING ON QUE FOR PRI CONNECTION.
    pcb(prt).prtq := inx;
  ELSE
    while mem_manag_tbl(ind) /= 0
      loop --FIND END OF QUE FOR THIS PORT.
        ind := mem_manag_tbl(ind);
      end loop;
      mem_manag_tbl(ind) := inx; --ATT NEW DATA TO QUE.
    END IF;
  else --PACKET NOT FROM PRIMARY CONNECTION.
    if pcb(prt).s_prt_ad = mem(inx).dst then
      --PACKET IS FROM FTP DATA CONNECTION.
      ind := pcb(prt).s_prtq;
      IF ind = 0 THEN --NOTHING ON SEC CONNECTION QUE.
        pcb(prt).s_prtq := inx;
      ELSE
        while mem_manag_tbl(ind) /= 0
          loop --FIND END OF QUE FOR THIS PORT.
            ind := mem_manag_tbl(ind);
          end loop;
          mem_manag_tbl(ind) := inx; --ATT DATA TO QUE.
        END IF;
      end put(" s_prtq = ");
    end if;
  end if;

END;
@ind := pcb(prt).s_prtq;
@loop
  @oput(ind); oput(" ");
  @exit when ind = 0;
  @ind := mem_manag_tbl(ind);
@end loop;
end if;
end if;
end pcb_rcv;

PROCEDURE adv_pcb_state(nr : IN INTEGER) is
  use assylib, globall;
BEGIN
  CASE pcb(nr).Pstate is
  WHEN r_mnit =>
    oPUT("advancing state to rlogn"); oNEW_LINE;
    pcb(nr).Pstate := rlogn;
  WHEN f_mnit =>
    oPUT("advancing state to rftp"); oNEW_LINE;
    pcb(nr).Pstate := rftp;
  WHEN rftp =>
    pcb(nr).sec_act := TRUE;
  WHEN others =>
    oPUT("***error in 'adv_pcb_state'"); oNEW_LINE;
  END CASE;
END adv_pcb_state;
end pcbrec;

PACKAGE tcprec is
  PROCEDURE tcp_rcv(blk : IN INTEGER);
END tcprec;

with tcpsend,pcbrec,ipsend,assylib,lib,globall;
PACKAGE BODY tcprec is
  use tcpsend,pcbrec,ipsend,assylib,lib,globall;

PROCEDURE pcb_clsing(prt : IN INTEGER) is
BEGIN
  pcb(prt).Pstate := clsing;
  pcb(prt).time_wait := 0;
  pcb(prt).sent := FALSE;
END pcb_clsing;

PROCEDURE tcp_rcv(blk : IN INTEGER) is
  fin_bit : CONSTANT INTEGER := 0;
  syn_bit : CONSTANT INTEGER := 1;
  rst_bit : CONSTANT INTEGER := 2;
  psh_bit : CONSTANT INTEGER := 3;

  -- Remaining code

ack_bit       : CONSTANT INTEGER := 4;
fin           : CONSTANT BYTE := BYTE(1);
syn           : CONSTANT BYTE := BYTE(2);
rst           : CONSTANT BYTE := BYTE(4);
rst_ack       : CONSTANT BYTE := BYTE(16#14#);
psh           : CONSTANT BYTE := BYTE(8);
ack           : CONSTANT BYTE := BYTE(16);
syn_ack       : CONSTANT BYTE := BYTE(16#12#);
off5          : CONSTANT BYTE := BYTE(16#50#);
off6          : CONSTANT BYTE := BYTE(16#60#);
fin_rec, syn_rec, rst_rec, ack_rec   : BOOLEAN;
int, nr, ackn, blk1: INTEGER;
sent          : BOOLEAN;
found         : BOOLEAN;
seg_len       : INTEGER;
ptr           : integer;
byt           : BYTE;

PROCEDURE conv_blk_snd(blk: IN INTEGER; sent: OUT BOOLEAN) is
BEGIN
  mem(blk).ip_dst := mem(blk).ip_scr;
  mem(blk).ip_scr := loc_ip_ad;
  mem(blk).ip_cksum := mem(blk).dst;  --temp storage
  mem(blk).dst := mem(blk).scr;
  mem(blk).scr := mem(blk).ip_cksum;
  mem(blk).ttl := BYTE(0);
  int := 6+(upper_nibble(mem(blk).off)*2);  
  --num of words to cksum
  mem(blk).ip_cksum(1) := BYTE(0);  --pg. 17 of TCP manual
  mem(blk).ip_cksum(2) := BYTE((int - 6)*2);  
  --tcp header len in byt
  mem(blk).wnd := rcv_wnd;
  mem(blk).tcp_xsum(1) := BYTE(0);
  mem(blk).tcp_xsum(2) := BYTE(0);
  cksum(mem(blk).ttl,address,int,mem(blk).tcp_xsum);
  ip_send(blk,sent);
  give_memory(blk);
END conv_blk_snd;

PROCEDURE send_ack(blk : IN INTEGER; nr : IN INTEGER; sent : OUT BOOLEAN) is
BEGIN
  mem(blk).ip_dst := mem(blk).ip_scr;
  mem(blk).ip_scr := loc_ip_ad;
  mem(blk).dst := mem(blk).scr;
  mem(blk).scr := tcp(nr).loc_sock.tcp_ad;
  mem(blk).seg := tcp(nr).snd.nxt;
  mem(blk).ack := tcp(nr).rcv.nxt;
  mem(blk).off := off5;

  198
mem(blk).ctl := ack;
mem(blk).ip_cksum(1) := BYTE(0);
mem(blk).ip_cksum(2) := BYTE(20);
mem(blk).wnd := rcv_wnd;
cksum(mem(blk).ttl, address, 16,
     mem(blk).tcp_xsum);
ip_send(blk, sent);
give_memory(blk);
END send_ack;

PROCEDURE update_retrnsQ(nr IN INTEGER;
ack : IN OUT array4) is
ptr : INTEGER;
BEGIN
  LOOP
    ptr := tcb(nr).retrnsQ;
    EXIT WHEN ptr = 0;
    IF 1 than(mem(ptr).seq, ack) THEN
      tcb(nr).retrnsQ := mem_manag_tbl(ptr);
      give_memory(ptr);
      END IF;
  END LOOP;
END update_retrnsQ;

BEGIN -- begin procedure tcprec
get_tcb_ndx(mem(blk).dst, nr, found);
seg_len := arr_to_int(mem(blk).len) - 20 -
     (upper_nibble(mem(blk).off) * 4);
IF found THEN
  byt := mem(blk).ctl;
  fin_rec := otstbit(byt, fin_bit);
  syn_rec := otstbit(byt, syn_bit);
  rst_rec := otstbit(byt, rst_bit);
  ack_rec := otstbit(byt, ack_bit);
  CASE tcb(nr).Tstate is
  WHEN listen =>
    if rst_rec then
      give_memory(blk);
      RETURN;
    end if;
    if ack_rec then
      mem(blk).seq := mem(blk).ack;
      mem(blk).off := off5;
      mem(blk).ctl := rst;
      conv_blk_snd(blk, sent);
      RETURN;
    END IF;
    if syn_rec then
      inc_arr(mem(blk).seq, 1,
        tcb(nr).rcv.nxt);
      END;
  END CASE;
END tcprec;
tcb(nr).rcv.irs := mem(blk).seq;
tcb(nr).Tstate := syn_rcv;
tcb(nr).rem_sock.ip_ad := mem(blk).ip_scr;
tcb(nr).rem_sock.tcp_ad := mem(blk).scr;
mem(blk).seq := tcb(nr).snd.iss;
mem(blk).ack := tcb(nr).rcv.nxt;
mem(blk).off := off6;
mem(blk).ctl := syn_ack;
mem(blk).data(1) := BYTE(2);
mem(blk).data(2) := BYTE(4);
mem(blk).data(3) := BYTE(4);
mem(blk).data(4) := BYTE(0);
conv_blk_snd(blk,sent);
inc_arr(tcb(nr).snd.iss,1,
- tcb(nr).snd.nxt);
tcb(nr).snd.una:= tcb(nr).snd.iss;
RETURN;
end if;

WHEN syn_sent =>
  --first check for ack
  IF ack_rec AND NOT rst_rec THEN
    IF lt_equ(mem(blk).ack,tcb(nr).snd.iss) OR
      g_than(mem(blk).ack,tcb(nr).snd.nxt) THEN
      mem(blk).seq := mem(blk).ack;
      mem(blk).off := off5;
      mem(blk).ctl := rst;
      conv_blk_snd(blk,sent);
      RETURN;
    END IF;
  END IF;
  IF lt_equ(tcb(nr).snd.una,mem(blk).ack) AND
    lt_equ(mem(blk).ack,tcb(nr).snd.nxt) THEN
    --second ckeck for reset
    IF rst_rec THEN
      pcb_cls(tcb(nr).prt_num);
tcb_cls(nr);
      oPUT("abort connection due to rst");oNEW_LINE;
give_memory(blk);
      ntrpt := rcv_pck;
      RETURN;
    END IF;
  END IF;
  --skip security and precedence
  --fourth check for syn
  IF syn_rec THEN
    inc_arr(mem(blk).seq,1,tcb(nr).rcv.nxt);
tcb(nr).rcv.irs := mem(blk).seq;
  IF ack_rec THEN
    tcb(nr).snd.una := mem(blk).ack;
    IF g_than(tcb(nr).snd.una,
tcb(nr).snd.iss) THEN
    
200
tcb(nr).Tstate := estab;
adv_pcb_state(tcb(nr).prt_num); -- in pcbrec
mem(blk).seq := tcb(nr).snd.nxt;
mem(blk).ack := tcb(nr).rcv.nxt;
mem(blk).off := off5;
mem(blk).ctl := ack;
conv_blk_snd(blk,sent);
RETURN;
ELSE
  tcb(nr).Tstate := syn_rcv;
  mem(blk).seq := tcb(nr).snd.iss;
  mem(blk).ack := tcb(nr).rcv.nxt;
  mem(blk).off := off6;
  mem(blk).ctl := syn_ack;
  mem(blk).data(1) := BYTE(2);
  mem(blk).data(2) := BYTE(4);
  mem(blk).data(3) := BYTE(4);
  mem(blk).data(4) := BYTE(0);
  conv_blk_snd(blk,sent);
  RETURN;
END IF;
END IF;
END IF;

WHEN syn_rcv..time_wait =>
  --first check the seq number
  IF seg_len < 0 OR seg_len > 512 THEN--error fm sender
    oPUT("seg_len out of range"); oNEW_LINE;
    send_ack(blk,nr,sent);
    RETURN;
  END IF;
  IF seg_len = 0 THEN
    IF mem(blk).seq /= tcb(nr).rcv.nxt THEN
      IF NOT rst_rec THEN
        oPUT("seg.seq /= rcv.nxt, seg_len = 0");
        oNEW_LINE;
        send_ack(blk,nr,sent);
        RETURN;
      ELSE
        give_memory(blk);
      RETURN;
      END IF;
    ELSE
      --seg_len > 0.
      IF arr_to_int(rcvWnd) = 0 THEN
        IF NOT rst_rec THEN
          oPUT("sending ack, rcvWnd = 0"); oNEW_LINE;
          send_ack(blk,nr,sent);
          RETURN;
        ELSE
          give_memory(blk);
        END IF;
      END IF;
    END IF;
  END IF;
  END WHEN;
RETURN;
END IF;
ELSE
IF tcb(nr).rcv.nxt /= mem(blk).seq THEN
IF NOT rst_rec THEN
  oPUT("seq_seq /= rcv.nxt, seq_len > 0");
  oNEW_LINE;
  send_ack(blk,nr,sent);
  RETURN;
ELSE
give_memory(blk);
RETURN;
END IF;
END IF;
END IF; --ends if rcv wnd = 0.
END IF; --ends if seg_len = 0.
--second, check for rst
IF rst_rec THEN
  --pg206
  CASE tcb(nr).Tstate is
    WHEN syn_rcv =>
      tcb(nr).Tstate := listen;
    WHEN estab..close_wait =>
      if mem(blk).dstE = pcb(tcb(nr).prt_num).s_prt_ad
        then
          pcb(tcb(nr).prt_num).sec_act := false;
        else
          pcb_abort(tcb(nr).prt_num);
        end if;
      tcb_cls(nr); -- aborting the tcb
      oPUT("aborting connection rst_rec");
      oNEW_LINE;
    WHEN closing..time_wait =>
      tcb_cls(nr);
      oPUT("aborting connection rst_rec");
      oNEW_LINE;
  END CASE;
give_memory(blk);
ntrpt := rcv_pck;
RETURN;
END IF; --ends if rst_rec.
--skip security and precedence
--fourth, check for syn_rec
IF syn_rec THEN
  --pg 207
  CASE tcb(nr).Tstate is
    WHEN syn_rcv..time_wait =>
      oPUT("sending reset - condition 2");
      oNEW_LINE;
      mem(blk).seq := tcb(nr).snd.iss;
      mem(blk).ack := tcb(nr).rcv.nxt;
      mem(blk).off := off6;
      mem(blk).ctl := rst;
      conv_blk_snd(blk,sent);
      RETURN;
  END CASE;
END IF;
--fifth, check for ack
IF ack_rec THEN
CASE tcb(nr).Tstate is

WHEN syn rcv =>
  IF NOT(lt_equ(tcb(nr).snd.una,mem(blk).ack) AND
      lt_equ(mem(blk).ack,tcb(nr).snd.nxt)) THEN
    oPUT("sending reset - condition 3");
oNEW_LINE;
    mem(blk).seq := mem(blk).ack;
    mem(blk).ack := tcb(nr).rcv.nxt;
    mem(blk).off := off5;
    mem(blk).ctl := rst;
    conv_blk_snd(blk,sent);
    RETURN;
  ELSE
    tcb(nr).Tstate := estab;
    adv_pcb_state(tcb(nr).prt_num);
  END IF;

WHEN estab..closing =>
  IF 1_than(tcb(nr).snd.una,mem(blk).ack) AND
      mem(blk).ack = tcb(nr).snd.nxt THEN
    tcb(nr).snd.una := mem(blk).ack;
    update_retransQ(nr,mem(blk).ack);
    IF 1_than(tcb(nr).snd.w11,mem(blk).seq)
      OR (tcb(nr).snd.w11 = mem(blk).seq AND
           lt_equ(tcb(nr).snd.wl2,mem(blk).ack))
    THEN
      tcb(nr).snd.wnd := mem(blk).wnd;
      tcb(nr).snd.wl1 := mem(blk).seq;
      tcb(nr).snd.wl2 := mem(blk).ack;
    END IF;
  END IF;
END CASE;

WHEN estab => null;

WHEN fin_wait_1 =>
  IF fin_rec THEN
    tcb(nr).Tstate := fin_wait_2;
  END IF;

WHEN fin_wait_2 =>
  pcb_cls(tcb(nr).prt_num);

WHEN close_wait => null;

WHEN closing =>
  IF mem(blk).ack = tcb(nr).snd.nxt THEN
    tcb(nr).Tstate := time_wait;
  END IF;
END CASE;
END IF;

WHEN last_ack =>
    IF mem(blk).ack = tcb(nr).snd.nxt THEN
        pcb_cls(tcb(nr).prt_num);
        tcb_cls(nr);
        oPUT("aborting in last_ack"); oNEW_LINE;
        ntrpt := rcv_pck;
        give_memory(blk);
        RETURN;
    END IF;

WHEN time_wait =>
    pcb_cls(tcb(nr).prt_num);
    tcb_cls(nr);
    oPUT("aborting in time_wait"); oNEW_LINE;
    ntrpt := rcv_pck;
    give_memory(blk);
    RETURN;

WHEN others => null;
END CASE;
ELSE
    --no ack received.
    give_memory(blk);
    RETURN;
END IF;

    --ends if ack_rec.
    --skip check for urg
    --seventh, process the segment
    --text pg.210
    IF seg_len > 0 THEN
        CASE tcb(nr).Tstate is

WHEN estab..fin_wait_2 =>
    inc_arr(tcb(nr).rcv.nxt,seg_len,tcb(nr).rcv.nxt);
    inc_arr(tcb(nr).rcv.wnd,seg_len,tcb(nr).rcv.wnd);
    get_memory(ackn);
    if ackn > 0 then
        mem(ackn).ttl := BYTE(0);
        mem(ackn).prot := BYTE(6);
        mem(ackn).ip_cksum(1) := BYTE(0);
        mem(ackn).ip_cksum(2) := BYTE(20);
        mem(ackn).ip_scr := loc_ip_ad;
        mem(ackn).ip_dst := mem(blk).ip_scr;
        mem(ackn).scr := tcb(nr).loc_sock.tcp_ad;
        mem(ackn).seq := tcb(nr).snd.nxt;
        mem(ackn).ack := tcb(nr).rcv.nxt;
        mem(ackn).off := off5;
        mem(ackn).ctl := ack;
        mem(ackn).wnd := rcv_wnd;
        mem(ackn).tcp_xsum(1) := BYTE(0);
        mem(ackn).tcp_xsum(2) := BYTE(0);
        mem(ackn).urg(1) := BYTE(0);
mem(ackn).urg(2) := BYTE(0);
cksum(mem(ackn).ttl, address, 16,
        mem(ackn).tcp_xsum);
ip_send(ackn, sent);
give memory(ackn);
mem(blk).frm_len := seg_len; -- # of data bytes
mem(blk).spare := 1; -- start of data
pcb_rcv(blk, tcb(nr).prt_num);
else
    null; -- $$$
end if;

WHEN close_wait..time_wait =>
give memory(blk);
RETURN;

END CASE;
END IF;

-- eighth, check for fin

IF fin_rec THEN
    IF tcb(nr).Tstate=listen OR tcb(nr).Tstate=syn_sent
        THEN
            get_memory(blk);
            ntrpt := rcv_pck;
            RETURN;
        ELSE
            inc_arr(tcb(nr).rcv.nxt, 1, tcb(nr).rcv.nxt);
            oPUT("Connection closing"); oNEW_LINE;
            IF seg_len > 0 THEN
                get_memory(blk1);
                IF blk1 /= 0 THEN
                    mem(blk1) := mem(blk);
                ELSE
                    RETURN;
                END IF;
            ELSE
                blk1 := blk;
            END IF;
    END IF;
CASE tcb(nr).Tstate is
    WHEN syn_rcv..estab =>
        send_ack(blk1, nr, sent);
        tcb(nr).Tstate := close_wait;
        if tcb(nr).loc_sock.tcp_ad =
            pcb(tcb(nr).prt_num).s_prt_ad then
            pcb_close(pcb(tcb(nr).prt_num).s_prt_ad); oceans
            tcb_cls(nr);
        ELSE
            pcb_close(tcb(nr).prt_num);
        end if;
    WHEN fin_wait_1 =>

IF mem(blk).ack = tcb(nr).rcv.nxt THEN
  tcb(nr).Tstate := time_wait;
ELSE
  tcb(nr).Tstate := closing;
END IF;

WHEN fin_wait_2 =>
  tcb(nr).Tstate := time_wait;
WHEN close_wait..time_wait => null;
END CASE;

RETURN;
END IF;
END IF; -- end if fin_rec.

IF seg_len = 0 THEN
  give_memory(blk);
END IF;
END CASE;
ELSE -- no tcb entry for tcp dst address.
  PUT("sending reset - tcb not found"); NEW LINE;
  mem(blk).ip_dst := mem(blk).seq; -- temp storage
  mem(blk).seq := mem(blk).ack;
  mem(blk).ack := mem(blk).ip_dst;
  inc_arr(mem(blk).ack,seg_len,mem(blk).ack);
  mem(blk).off := off5;
  mem(blk).ctl := rst_ack;
  conv_blk_end(blk,sent);-- sending bad tcb index!? RETURN;
END IF;
E tcp_rcv;
E tcp_rec;
PACKAGE iprec is

PROCEDURE ip_rcv(blk : IN INTEGER);
END iprec;

with tcprec,lib,global1;
PACKAGE BODY iprec is
use tcprec,lib,global1;

PROCEDURE ip_rcv(blk : IN INTEGER) is
  arpa_ver : CONSTANT BYTE := BYTE(16#45#);
  arpa_prot : CONSTANT BYTE := BYTE(16#06#);
  int : INTEGER;
BEGIN
  IF mem(blk).ver /= arpa_ver THEN
    give_memory(blk);
    RETURN;
  END IF;
  IF mem(blk).ack = tcb(nr).rcv.nxt THEN
    tcb(nr).Tstate := time_wait;
  ELSE
    tcb(nr).Tstate := closing;
  END IF;
  WHEN fin_wait_2 =>
    tcb(nr).Tstate := time_wait;
  WHEN close_wait..time_wait => null;
  END CASE;
  RETURN;
  END IF;
  END IF; -- end if fin_rec.
  IF seg_len = 0 THEN
    give_memory(blk);
  END IF;
  END CASE;
  ELSE -- no tcb entry for tcp dst address.
    PUT("sending reset - tcb not found"); NEW LINE;
    mem(blk).ip_dst := mem(blk).seq; -- temp storage
    mem(blk).seq := mem(blk).ack;
    mem(blk).ack := mem(blk).ip_dst;
    inc_arr(mem(blk).ack,seg_len,mem(blk).ack);
    mem(blk).off := off5;
    mem(blk).ctl := rst_ack;
    conv_blk_end(blk,sent);-- sending bad tcb index!? RETURN;
  END IF;
END tcp_rcv;
END tcp_rec;
PACKAGE iprec is

PROCEDURE ip_rcv(blk : IN INTEGER);
END iprec;

with tcprec,lib,global1;
PACKAGE BODY iprec is
use tcprec,lib,global1;

PROCEDURE ip_rcv(blk : IN INTEGER) is
  arpa_ver : CONSTANT BYTE := BYTE(16#45#);
  arpa_prot : CONSTANT BYTE := BYTE(16#06#);
  int : INTEGER;
BEGIN
  IF mem(blk).ver /= arpa_ver THEN
    give_memory(blk);
    RETURN;
  END IF;
  IF mem(blk).ack = tcb(nr).rcv.nxt THEN
    tcb(nr).Tstate := time_wait;
  ELSE
    tcb(nr).Tstate := closing;
  END IF;
  WHEN fin_wait_2 =>
    tcb(nr).Tstate := time_wait;
  WHEN close_wait..time_wait => null;
  END CASE;
  RETURN;
  END IF;
END tcp_rcv;
END tcp_rec;
IF mem(blk).frm_len > 576 THEN
  give_memory(blk);
  RETURN;
END IF;

IF mem(blk).prot /= arpa_prot THEN
  give_memory(blk);
  RETURN;
END IF;

IF mem(blk).ip_dst. /= loc_ip_ad THEN
  give_memory(blk);
  RETURN;
END IF;

tcp_rncv(blk);
END ip_rncv;
END iprcv;
with globall;
PACKAGE ethrec is
use globall;

PROCEDURE eth_rncv(blk : IN OUT eth_pck);
END ethrec;

with lib, globall;
PACKAGE BODY ethrec is
use lib, globall;
-- this package handles the packets that don't have an arpa
-- protocol structure. there are two types to handle,
-- 1. broadcast packets sent to us asking for our ethernet
-- physical address and
-- 2. packets addressed to us giving us the sender's
-- physical address (ie. in response to our request for
-- their address

-- TYPE eth_pck IS RECORD
--   fmt_stat : array2; --see RFC826.TXT,
-k48H-fmt_len : INTEGER; --Network Info Center,
--   to_eth_ad : array6; --for details:
--   fm_eth_ad : array6; --arpanet SRI-NIC
--   type_pck : array2;
--   ar_hrd : array2;
--   ar_pro : array2;
--   ar_len : array2;
--   nul : BYTE;
--   ar_op : BYTE;
--   fm_eth : array6;
--   fm_ip : array4;
PROCEDURE eth_rcv(blk : IN OUT eth_pck) is
  max_int : CONSTANT INTEGER := INTEGER(16#7FFF#);
  ndx    : INTEGER := 1;
  oldest : INTEGER;
BEGIN
  IF blk.to_ip = loc_ip_ad THEN
    IF blk.ar_op = BYTE(1) THEN
      eth.to_eth_ad := blk.fm_eth;
      eth.ar_op := BYTE(2);
      eth.to_eth := blk.fm_eth;
      eth.to_ip := blk.fm_ip;
      trn_pck(eth.to_eth_ad'address,min_size);
    ELSE
      ntrpt := rcv_pck;
    END IF;
  ELSE
    ntrpt := rcv_pck;
  END IF;
END IF;
END LOOP outer;
ELSE
  ntrpt := rcv_pck;
END IF;
END IF;
with convblk, iprec, assylib, lib, globall;
PACKAGE BODY rcv is
use convblk, iprec, assylib, lib, globall;

PROCEDURE ntrpt_hdl is
--author r. l. hartman
--date 22 feb 86
--this procedure handles interrupts from the NI3010 ethernet controller board. the 4 types of interrupts are:
--1. rcvpck received a packet from ethernet
--2. rcv DMA dn DMA done on incoming packet
--3. tx DMA dn DMA done on outgoing packet
--4. disable when the interrupt hdlr xmits a packet
val : BYTE;
BEGIN
outprt(able_reg, disable);
CASE ntrpt IS
WHEN disable =>
perfcmd(ld_snd);
ntrpt := rcv_pck;

WHEN rcv_pck =>
get_memory(wrd);
IF wrd = 0 THEN --no space avail
out("no memory blocks available!"); oneNew_LINE;
ntrpt := rcv_pck;
outprt(able_reg, rcvpck);
ELSE
wr_ad(mem(wrd) 'address);
outprt(h_cnt_reg, ohi(blk_size));
outprt(l_cnt_reg, olo(blk_size));
ntrpt := rcv_dma_dn;
outprt(able_reg, rcv_dma_dn);
END IF;

WHEN rcv_DMA_dn =>
ntrpt := disable; --for possible trns
IF mem(wrd).type_pck(1) = BYTE(16#08#) THEN
IF mem(wrd).type_pck(2) = BYTE(16#00#) THEN
ip_rcv(wrd);
ELSE IF mem(wrd).type_pck(2) = BYTE(16#06#) THEN
conv_blk(mem(wrd));
give_memory(wrd);—eth_rcv can't do
END IF;
ELSE
  give_memory(wrd);
END IF;
ASM cli; —-clear interrupts
IF ntrpt = disable OR ntrpt = rcv_pck THEN
  ntrpt := rcv_pck;
  outprt(able_reg,rcv_pck);
END IF;

WHEN tx_DMA_dn =>
  perf_cmd(ld_snd);
  ntrpt := rcv_pck;
  outprt(able_reg,rcv_pck);

END CASE;
END ntrpt_hdl;
END rcv;
PACKAGE ethsend is
PROCEDURE eth_snd(blk IN INTEGER; size: IN INTEGER;
resit OUT BOOLEAN); END
ethsend;

with assylib,lib,globall;
PACKAGE BODY ethsend is
USE assylib,lib,globall;
PROCEDURE eth_snd(blk IN INTEGER; size: IN INTEGER;
resit OUT BOOLEAN) is
  sent CONSTANT BOOLEAN := TRUE;
  not_sent CONSTANT BOOLEAN := FALSE;
  found BOOLEAN;
BEGIN
  resolve_ad(mem(blk).ip_dst,mem(blk).to_eth_ad,found);
  IF found THEN
    mem(blk).fm_eth_ad := loc_eth_ad;
    mem(blk).type_pck(1) := BYTE(8); — standard for
    mem(blk).type_pck(2) := BYTE(0); — arpanet packets
    trn_pck(mem(blk).to_eth_ad'address,
      grtr_of(size + 14, min_size));
    resit := sent;
  ELSE
    PUT("cannot find ethernet addr"); NEW_LINE;
    FOR i IN 1..6 LOOP
      eth.to_eth_ad(i) := BYTE(16#FF#);
    END LOOP;
  END IF;
eth.ar_op := BYTE(1);
eth.to_ip := mem(blk).ip_dst;
trn_pck(eth.to_eth_ad'address,min_size);
reslt := not_sent;
END IF;
END eth_snd;
END ethsend;
package ipsend is
  procedure ip_send(inx: in integer; rslt: out BOOLEAN);
end ipsend;

with assylib, lib, ethsend, globall;
PACKAGE body ipsend IS

procedure ip_send(inx: in integer; rslt: out BOOLEAN) is
  -- AUTHOR: ALEC YASINSAC
  -- DATE: FEB 1986
  -- INPUT: 1.INX IS THE MEMORY BLOCK INDEX TO BE TRANSMITTED.
  -- OUTPUT: 1.RSLT IS AN ERROR FLAG
  -- DESCRIPTION: IP_SEND SETS THE IP HEADER FIELDS OF THE
  -- PACKET TO BE TRANSMITTED TO A REMOTE HOST, AND CALLS
  -- THE PROCEDURE THAT WILL PASS THE PACKET OUT ONTO
  -- ETHERNET.
  use assylib, lib, ethsend, globall;
  ip_hdr_len : CONSTANT INTEGER := 20;
totlen: integer;
begin
  mem(inx).ver := byte(16#45#);--4 is protocol version,
  mem(inx).serv := byte(0);--5 is # of 32 bit words in hdr
  mem(inx).id(1) := byte(0);
  mem(inx).id(2) := byte(0);
  mem(inx).flag(1) := byte(0);
  mem(inx).flag(2) := byte(0);
  mem(inx).ttl := byte(16#0F#);
totlen := arr_to_int(mem(inx).ip_cksum) + ip_hdr_len;
  mem(inx).len(1) := ohi(totlen);
  mem(inx).len(2) := olo(totlen);
  mem(inx).ip_cksum(1) := BYTE(0);
  mem(inx).ip_cksum(2) := BYTE(0);
cksum(mem(inx).ver'address,10,mem(inx).ip_cksum);
  -- THE TCP_OPEN PROCEDURE SETS THE IP_CKSUM FIELD TO
  -- CONTAIN THE LENGTH OF THE TCP HEADER AND DATA. THE
  -- LEN FIELD CONTAINS THE LENGTH OF THE THE IP HEADER.
  -- THE LENGTH OF THE PACKAGE IS THE SUM OF THESE TWO
  -- FIELDS.
  eth_snd(inx,totlen,rslt);
  --TOTLEN IS PACKAGE LENGTH IN BYTES.
end ip_send;
end ipsend;
with globall:
package tcpsend is
  use globall;

211
PROCEDURE tcp_open(prt: IN INTEGER;
    foreign_sock: IN OUT socket_rec; act: IN BOOLEAN;
    loc_tcp_ad: OUT array2; rslt: OUT BOOLEAN);

PROCEDURE tcp_send(indx: IN INTEGER;
    data_len : IN INTEGER; tcp_ad: IN OUT array2;
    rslt: OUT BOOLEAN);

PROCEDURE tcp_close(tcp_ad : IN OUT array2);

PROCEDURE check_retrnsQ(tcp_ad : IN OUT array2);

end tcpsend;

with ipsend, lib, assylib, globall;
PACKAGE body tcpsend IS
    use ipsend, lib, assylib, globall;
    --last updated 29 Apr 86
    hdr_len : CONSTANT INTEGER := 16;
datawds : INTEGER;
---------------------------------------------------------------------------
PROCEDURE check_retrnsQ(tcp_ad : IN OUT array2) is
    ndx : INTEGER;
exists : BOOLEAN;
    ptr : INTEGER;
BEGIN
    get_tcb_ndx(tcp_ad,ndx,exists);
    IF exists THEN
        ptr := tcb(ndx).retrnsQ;
        LOOP
            EXIT WHEN ptr = 0;
            mem(ptr).spare := mem(ptr).spare + 1;
            IF mem(ptr).spare >= 10 THEN
                mem(ptr).ttl := byte(0);
                mem(ptr).ip_cksum(1) := ohi(mem(ptr).frm_len+20);
                mem(ptr).ip_cksum(2) := olo(mem(ptr).frm_len+20);
                mem(ptr).tcp_xsum(1) := byte(0);
                mem(ptr).tcp_xsum(2) := byte(0);
                IF mem(ptr).frm_len < 512 THEN
                    mem(ptr).data(mem(ptr).frm_len+1) := BYTE(0);
                END IF;
                datawds := hdr_len + (mem(ptr).frm_len + 1)/2;
                cksum (mem(ptr).ttl'address,datawds,
                    mem(ptr).tcp_xsum);
                ip_send(ptr, exists);
                tcb(ndx).retrnsQ := mem_manag_tbl(ptr);
                give_memory(ptr);
                oPUT("retransmit blk # "); oPUT(ptr):oNEW_LINE;
                oPUT("seq ");
                FOR i IN 1..4 LOOP
                    oPUT(INTEGER(mem(ptr).seq(i)));
                END LOOP;
            END IF;
            ptr := tcb(ndx).retrnsQ;
            tcb (ndx).retrnsQ := mem_manag_tbl (ptr);
            give_memory (ptr);
        END LOOP;
    END IF;
END;

212
END LOOP; oNEW_LINE;
oPUT(" ");
END LOOP; oNEW_LINE;

FOR i IN 1..4 LOOP
  oPUT(INTEGER(mem(ptr).ack(i))); 
oPUT(" ");
END LOOP; oNEW_LINE;
END IF;

ptr := mem_manag_tbl(ptr);
END LOOP;
END IF;
END check_retrnsQ;

PROCEDURE tcp_open (prt: in integer;
foreign_sock: IN OUT socket rec; act: in boolean;
loc_tcp_ad: OUT array2; sent: OUT BOOLEAN) is
  -- AUTHOR: ALEC YASINSAC -- DATE: FEB 86
  --INPUT: 1. INX IS THE INDEX FOR THE TCB ARRAY RECORD.
  -- 2. FOR_IP_AD IS THE IP ADDRESS OF THE REMOTE HOST
  -- 3. ACT INDICATES WHETHER THE CONNECTION IS ACTIVE
  -- OR PASSIVE. THE PASSIVE, OR HOST, CONNECTION
  -- MAY BE IMPLEMENTED AT A LATER DATE.
  --
  --OUTPUT: 1. GLOBAL ARRAY 'MEM'
  -- 2. PARAMETER LOCAL TCP ADDRESS
  -- 3. GLOBAL ARRAY PCB
  -- 4. GLOBAL ARRAY TCB
  -- 5. PARAMETER RESULT
  -- 6. 
  --EXTERNAL MODULES CALLED: 1. oHI, oLO
  -- 2. GET_MEMORY
  -- 3. ADD_4BYT_ARRS
  -- 4. IP_SEND
  -- 5. CKSUM
  --DESCRIPTION: TCP_OPEN OPENS A TCP CONNECTION BETWEEN THE
  -- SELECTED FOREIGN HOST AND THE Z100. A TCB RECORD WILL BE
  -- BUILT AND A SYN SIGNAL PACKET IS BUILT AND PASSED TO
  -- IP SND FOR TRANSMISSION TO THE DESTINATION ADDRESS.
  -- THIS PROCESS IS EXPLICITLY DESCRIBED IN THE STANFORD
  -- RESEARCH CENTER REQUEST FOR COMMENT MANUALS, SRI-RFC.
  -- RFC-793 IS THE TCP MANUAL. SOME IMPORTANT PAGES ARE
  -- 54, 45, 31, 16, AND 17.
  --
  --DECLARATIONS FOR PROCEDURE TCP_OPEN
indx, inx: integer;
overflo, exists: boolean;

213
begin --BEGIN PROCEDURE TCP_OPEN.
    loc_tcp_ad(1) := ohi(nxt_prt_ad);
    loc_tcp_ad(2) := olo(nxt_prt_ad);
    nxt_prt_ad := inc_nxt_prt_ad(nxt_prt_ad);--betw 0400-ffffH
    get_tcbndx(loc_tcp_ad, inx, exists);
    if exists then
        sent := false;
        return;
    end if;
    tcb(inx).prt_num := prt;
    tcb(inx).loc_sock.tcp_ad := loc_tcp_ad;
    tcb(inx).loc_sock.ip_ad := loc_ip_ad;
    tcb(inx).rem_sock := foreign_sock;
    tcb(inx).snd.iss(1) := byte(0);--MAKE FIRST SEQ # EQUAL
    tcb(inx).snd.iss(2) := byte(0);--TO TCP ADDR TO BE USED.
    tcb(inx).snd.iss(3) := tcb(inx).loc_sock.tcp_ad(1);
    tcb(inx).snd.iss(4) := tcb(inx).loc_sock.tcp_ad(2);
    --ASSIGNMENT OF ISS FIELD MADE ARBITRARILY.
    tcb(inx).snd.una := tcb(inx).snd.iss;
    tcb(inx).snd.w1l := tcb(inx).snd.iss;
    tcb(inx).ctl := byte(2);
    if act then
        get_memory(indx);
        if indx = 0 then
            sent := false;
            return;
        end if;
        mem(indx).scr(1) := loc_tcp_ad(1);--SET LOCAL SOCK #.
        mem(indx).scr(2) := loc_tcp_ad(2);
        tcb(inx).tstate := syn_sent;
        mem(indx).ttl := byte(0);--MUST BE ZERO TO COMPUTE
        mem(indx).prot := byte(6);--TCP CKSUM.
        mem(indx).ip.cksum(1) := byte(0);
        mem(indx).ip.cksum(2) := byte(24);--SET TO TCP LENGTH
        --FOR COMPUTATION OF TCP CHECKSUM.SEE p17 TCP MANUAL.
        mem(indx).ip_scr := loc_ip_ad;--INIT TO LOCAL IP ADDR.
        mem(indx).ip.dst := tcb(inx).rem_sock.ip_ad;
        mem(indx).dstat := tcb(inx).rem_sock.tcp_ad;
        mem(indx).seq := tcb(inx).snd.iss;
        mem(indx).ack(1) := byte(0);
        mem(indx).ack(2) := byte(0);
        mem(indx).ack(3) := byte(0);
        mem(indx).ack(4) := byte(0);
        mem(indx).off := byte(16#60#);
        mem(indx).ctl := byte(16#2#);
        mem(indx).wnd := rcv_wnd;   --MAX PACKET SIZE TO REC.
        mem(indx).tcp_xsum(1) := byte(0);
        mem(indx).tcp_xsum(2) := byte(0);-- p16 TCP MANUAL.
        --ZERO OUT XSUM FIELD BEFORE COMPUTING TCP CHECKSUM.
        mem(indx).urg(1) := byte(0);
    end if;
mem(indx).urg(2) := byte(0);
mem(indx).data(1) := byte(2); -- TELNET RESET CODE.
mem(indx).data(2) := byte(4);
mem(indx).data(3) := byte(4);
mem(indx).data(4) := byte(0);--make even num for cksum
cksum (mem(indx).ttl'address, 18, mem(indx).tcp_xsum);
-- THERE ARE EIGHTEEN 16 BIT WORDS IN THE TCP AND
-- PSEUDO HEADERS. CKSUM USES THE STARTING ADDRESS
-- AND LENGTH TO COMPUTE CHECKSUM.
ip_send(indx, sent); --IF RSLT OF IP_SEND IS GOOD,
give_memory(indx); --TCP_SEND IS ALSO GOOD.
IF NOT sent THEN
tcb(inx).prt_num := 99;
ELSE
  tcb(inx).snd.una := tcb(inx).snd.iss;
  inc_arr(tcb(inx).snd.iss,1,tcb(inx).snd.nxt);
END IF;
else -- PASSIVE CONNECTION.
tcb(inx).tstate := listen;
  pcb(tcb(inx).prtnum).s_prtq := 0;
end if;

end tcp_open;

procedure tcp_send(indx: IN INTEGER; data_len : IN INTEGER;
tcp_ad: IN OUT array2; sent: OUT BOOLEAN) is
  -- AUTHOR: ALEC YAISINAC
  -- DATE: FEB 86
  -- INPUT: 1. INDX IS INDEX OF MEMORY BLOCK TO BE TRANS.
  -- 2. DATA_LEN IS THE NUMBER OF DATA BYTES IN THE PACKET.
  -- 3. TCP_AD IS THE LOCAL TCP ADDRESS SENDING THE PACKET.
  -- 4. RSLT IS THREE VALUED ERROR FLAG.
  --
  -- OUTPUT: 1. GLOBAL ARRAY 'MEM'
  -- 2.
  -- 3. GLOBAL ARRAY PCB
  -- 4. GLOBAL ARRAY TCB
  -- 5. PARAMETER RSLT
  --
  -- EXTERNAL MODULES CALLED: 1. IP SEND
  -- 2. ADD_4BYT_ARRS
  -- 3. GET_TCB_INDEX
  -- 4. CKSUM
  -- 5.
  -- DESCRIPTION: TCP_SEND SENDS A PACKET TO REMOTE HOST
  -- FROM 2100. THE TCB RECORD WILL BE UPDATED. MUCH OF
  -- THIS PROCESS IS EXPLICITLY DESCRIBED IN THE STANFORD
  -- RESEARCH CENTER REQUEST FOR COMMENT MANUALS, SRI-RFC.
  -- RFC-793 IS THE TCP MANUAL. SOME IMPORTANT PAGES ARE
  -- 54, 45, 31, 16, AND 17.
  --
  -- DECLARATIONS FOR PROCEDURE TCP_SEND

215
inx: integer;
overflo: boolean;
exists: boolean;

begin --BEGIN PROCEDURE TCP_SEND.
get_tcb_nidx(tcp_ad, inx, exists);
IF exists THEN
  mem(indx).frm_len := data_len;
  mem(indx).ttl := byte(0); -- SET TO 0 TO COMPUT TCP CKSUM.
  mem(indx).prot := byte(6);
  mem(indx).ip_cksum(1) := ohl(data_len+20); -- SEE P17
  mem(indx).ip_cksum(2) := olo(data_len+20); -- TCP MAN.
  mem(indx).ip_scr := tcb(inx).loc_sock.ip_ad;
  mem(indx).ip_dst := tcb(inx).rem_sock.ip_ad;
  mem(indx).ip_prot := byte(6);
  mem(indx).ip_dst := tcb(inx).loc_sock.tcp_ad;
  mem(indx).ip_scr := tcb(inx).loc_sock.tcp_ad;
  mem(indx).seq := tcb(inx).snd.nxt; -- PAGE 40, TCP MAN.
  inc_arr(tcb(inx).snd.nxt, data_len, tcb(inx).snd.nxt);
  -- THE SND.NXT FIELD IS THE SUM OF THE SEQUENCE NUMBER
  -- AND THE NUMBER OF DATA BYTES IN THE PACKET. p40.
  mem(indx).ack := tcb(inx).rcv.nxt;
  mem(indx).off := byte(16#50#);
  -- TCP HEADER IS 5 32 BIT WORDS LONG. p16.
  mem(indx).ctrl := byte(16#18#);--WHILE CONNECTION IS
  -- ESTABLISHED, WILL ALWAYS SET ACK BIT. p16.
  mem(indx).wnd := rcv wnd; -- MAX PACKET SIZE TO RECEIVE.
  mem(indx).tcp_xsum(1) := byte(0); -- ZERO TO COMPUT TCP CHECKSUM, p16.
  mem(indx).tcp_xsum(2) := byte(0);
  mem(indx).urg(1) := byte(0);
  mem(indx).urg(2) := byte(0);
  IF data_len < 512 THEN
    mem(indx).data(data_len+1) := BYTE(0);
  END IF;
  datawrds := hdr_len + (data_len + 1)/2;
  cksum (mem(indx).ttl'address, datawrds,
          mem(indx).tcp_xsum);
  ip_send(indx, sent);
  ASM cli;
  IF NOT sent THEN
    oPUT("packet not sent, in proc tcp_send"); oNEW_LINE;
  END IF;
  mem(indx).spare := 0; -- reset counter for
  IF tcb(inx).retrnsQ = 0 THEN -- retransmission
    tcb(inx).retrnsQ := indx;
  ELSE -- need proc add_to_Q
    datawrds := tcb(inx).retrnsQ;
    LOOP
      EXIT WHEN mem_manag_tbl(datawrds) = 0;
      datawrds := mem_manag_tbl(datawrds);
    END LOOP;
    mem_manag_tbl(datawrds) := indx;
  END IF;
END IF;

216
ASM
END IF;
end tcp_send;

PROCEDURE tcp_close(tcp_ad : IN OUT array2) is
indx,inx: INTEGER;
exists : BOOLEAN;
asciic : CONSTANT BYTE := BYTE(16#43#);
sent : BOOLEAN;
ptr : INTEGER;
BEGIN
get_tcb_ndx(tcp_ad,inx,exists);
IF exists THEN
  get_memory(indx);
  IF indx /= 0 THEN
    mem(indx).ttl := byte(0);
    mem(indx).prot := byte(6);
    mem(indx).ip_cksum(1) := BYTE(0);
    mem(indx).ip_cksum(2) := BYTE(20);
    mem(indx).ip_scr := tcb(inx).loc_sock.ip_ad;
    mem(indx).ip_dsr := tcb(inx).rem_sock.ip_ad;
    mem(indx).dst := tcb(inx).rem_sock.tcp_ad;
    mem(indx).seq := tcb(inx).snd.nxt;
    inc_arr(tcb(inx).snd.nxt,l,tcb(inx).snd.nxt);
    mem(indx).ack := tcb(inx).rcv.nxt;
    mem(indx).off := byte(16#50#);
    mem(indx).ctl := byte(16#11#);
    mem(indx).wnd := rcv_wnd;
    mem(indx).tcp_xsum(1) := byte(0);
    mem(indx).tcp_xsum(2) := byte(0);
    mem(indx).urg(l) := byte(0);
    mem(indx).urg(2) := byte(0);
    cksum (mem(indx).ttl'address,16,
           mem(indx).tcp_xsum);
    ip_send(indx, sent);
    CASE tcb(inx).Tstate is
      WHEN estab => tcb(inx).Tstate := fin_wait_1;
      WHEN others => tcb(inx).Tstate := last_ack;
    END CASE;
    give_memory(indx);
    tcp_ad(l) := byte(0);
    tcp_ad(2) := byte(0);
  END IF;
END IF;
END tcp_close;

END tcpsend;

PACKAGE locXfer is
PROCEDURE loc_init(prt : IN INTEGER);

PROCEDURE loc(prt : IN INTEGER);
END locXfer;

WITH lib, assylib, global;
PACKAGE BODY locXfer is
USE lib, assylib, global;
code_print: CONSTANT BYTE := BYTE(16#D4#);
code_endprint: CONSTANT BYTE := BYTE(16#F4#);
nullbyte : CONSTANT BYTE := BYTE(0);
byt : BYTE;
int_input : INTEGER;
amt : INTEGER;

--------------------------------------------------------------------------------
PROCEDURE loc_init(prt: in integer) is
begin
  inprt pcb(prt).stat prt, byt);
  IF otstbit(byt, RxRdy) then
    inprt pcb(prt).dataprt, byt);
    int_input := INTEGER(byt);
    IF Int input <= num_prts AND int_input >= 0 THEN
      CASE pcb(int_input).Pstate is
        WHEN lstn =>
          IF prt /= int_input THEN
            pcb(prt).loc_con := int_input;
            pcb(int_input).loc_con := prt;
            pcb(prt).Pstate := local;
            pcb(int_input).Pstate := local;
            activate_prt(int_input);
            outprt pcb(int_input).data_prt, nullbyte);
            outprt pcb(prt).data_prt, nullbyte);
          END IF;
        WHEN local =>
          IF NCT pcb(int_input).is_print THEN
            pcb(prt).loc_con := pcb(int_input).loc_con;
            pcb(int_input).loc_con := prt;
            pcb(prt).Pstate := local;
            outprt pcb(prt).data_prt, nullbyte);
          END IF;
        WHEN lInit =>
          pcb(prt).Pstate := lstn;
        WHEN CLS =>
          pcb(prt).Pstate := lstn;
        WHEN others =>
          pcb_cls(prt);
        END CASE;
      ELSE
        pcb_cls(prt);
      END IF;
    ELSE
      pcb_cls(prt);
      END IF;
    ELSE
      218
pcb(prt).time_wait := pcb(prt).time_wait + 1;
IF pcb(prt).time_wait = threshold THEN
   pcb_cls(prt);
END IF;
END IF;
end loc_init;

PROCEDURE loc(prt: in integer) is
--
--this is a user datagram designed for local transfers:
-- mem_blk
-- wnd(1)  | dest |
-- wnd(2)  | source |
-- tcp_xsum| type |
-- tcp_xsum| cksum |
-- urg(1)  | length1|
-- urg(2)  | length2|
-- data |
--       | data |
--       | (512) |
--

page : CONSTANT BYTE := BYTE(16#0C#);
hdr_len : CONSTANT INTEGER := 6;
blk,ptr,bytcnt : INTEGER;
ndx : INTEGER;
found : BOOLEAN;

PROCEDURE snd_data_to_printer(prt, blk : IN INTEGER) is
BEGIN
   inprt(pcb(prt).statprt,byt);
   IF otstbit(byt,TxRdy) AND otstbit(byt,DSR) THEN
      IF mem(blk).spare > 0 THEN
         CASE mem(blk).data(mem(blk).spare) is
            WHEN BYTE(16#1A#) =>
               mem(blk).frm_len := 0;
               RETURN;
            WHEN BYTE(16#20#).BYTE(16#7E#) =>
               pcb(prt).prtQ := pcb(prt).prtQ + 1;
               outprt(pcb(prt).data_prt,
                  mem(blk).data(mem(blk).spare));
            WHEN BYTE(16#0D#) =>
pcb(prt).prtQ := 0;
output pcb(prt).data prt,
mem(blk).data(mem(blk).spare));
WHEN BYTE(16#09#) =>
FOR i IN 1..(8-pcb(prt).prtQ mod 8) LOOP

loop

inprt pcb(prt).stat prt,byt);
IF otstbit(byt,TxRdy) THEN

outprt pcb(prt).data prt,BYTE(16#20#));
EXIT;
END IF;
END LOOP;
END LOOP;
pcb(prt).prtQ := 0;
WHEN others =>

outprt pcb(prt).data prt,
mem(blk).data(mem(blk).spare));
END CASE;
mem(blk).spare := mem(blk).spare + 1;
mem(blk).frm_len := mem(blk).frm_len - 1;
ELSE

mem(blk).frm_len := 0;
END IF;
END IF;
END snd_data_to_printer;

PROCEDURE remove_link(prt : IN INTEGER) is
ptr : INTEGER;
BEGIN

ptr := prt;
LOOP

EXIT WHEN pcb(ptr).loc_con = ptr;
ptr := pcb(ptr).loc_con;
END LOOP;
pcb(ptr).loc_con := pcb(ptr).loc_con;
IF pcb(ptr).loc_con = ptr THEN--only one left in link

IF pcb(ptr).s_prQ /= 0 THEN

give_memory pcb(ptr).s_prQ;
pcb(ptr).s_prQ := 0;
END IF;
IF pcb(ptr).is_print THEN

pcb(ptr).pstate := lsn;
pcb(ptr).prQ := 0;
ELSE

cpcb_cls(ptr);
END IF;
END IF;
END remove_link;

BEGIN

inprt pcb(ptr).stat pry, byt);

220
IF otstbit(byt,RxRdy) THEN -- check for cntl
    inpri(pc(prt).data_prty,byt);
    CASE byt is
        WHEN code_cls =>
            IF pcb(prt).sprtQ /= 0 THEN
                give_memory(pcb(prt).s_prtQ);
                pcb(prt).s_prtQ := 0;
            END IF;
            remove_link(prt);
            pcb_cls(prt);
        WHEN code_status =>
            outp(pcb(prt).data_prty,code_status);
            give_status(prt);
        WHEN code_reqPrt =>
            outp(pcb(prt).data_prty,code_reqPrt);
            outp(pcb(prt).data_prty,BYTE(prt));
        WHEN code_loc =>
            outp(pcb(prt).data_prty,code_cls);
        WHEN code_print =>
            ptr := 0;
            LOOP
                IF pcb(ptr).is_print AND
                    pcb(ptr).state = 1stn THEN
                    outp(pcb(ptr).data_prty,BYTE(ptr));
                    activate_prt(ptr);
                    pcb(ptr).loc_con := pcb(ptr).loc_con;
                    pcb(ptr).loc_con := ptr;
                    pcb(ptr).state := local;
                    pcb(ptr).sent := FALSE;
                    pcb(ptr).prtQ := 0;
                    outp(pcb(ptr).cmd_prt,DTR);
                    EXIT;
                END IF;
                ptr := ptr + 1;
                IF ptr > num_prts THEN
                    outp(pcb(ptr).data_prty,code_quit);
                    EXIT;
                END IF;
            END LOOP;
        WHEN code_endprint =>
            ptr := pcb(ptr).loc_con;
            LOOP
                IF pcb(ptr).is_print THEN
                    pcb(ptr).sent := TRUE;
                    pcb(ptr).prtQ := 0;
                    outp(pcb(ptr).cmd_prt,clr);
                    EXIT;
                END IF;
                ptr := pcb(ptr).loc_con;
            EXIT WHEN ptr = prt;
            END LOOP;
        WHEN others =>
            A
outprt(pcb(prt).data_prt,code_cls);
END CASE;
END IF;
IF NOT pcb(prt).is_print THEN
IF pcb(prt).s_prtQ /= 0 THEN
blk := pcb(prt).s_prtQ;
ptr := 0;
LOOP
IF pcb(prt).ack(ptr) /= BYTE(0) THEN
EXIT;
ELSE
ptr := ptr + 1;
IF ptr > max_flag_byt THEN
give_memory(blk);
pcb(prt).s_prtQ := 0;
EXIT;
END IF;
END IF;
END LOOP;
END IF;
IF pcb(prt).s_prtQ = 0 THEN
inprt(pcb(prt).stat_prt,byt);
IF otstbit(byt,DSR) THEN
give_memory(blk);
IF blk /= 0 THEN
bytcnt := 518;
get_trns(mem(blk).wnd'ADDRESS,
pcb(prt).data_prt, bytcnt);
IF bytcnt > 0 THEN
mem(blk).frm_len := bytcnt - hdr_len;
mem(blk).spare := 1;
pcb(prt).s_prtQ := blk;
IF mem(blk).wnd(1) = BYTE '16#FF#' THEN
ptr := pcb(prt).loc_con;
LOOP
EXIT WHEN ptr = prt;
osetbit(pcb(ptr).ack pcb(ptr).flg_byt),
pcb(ptr).flg_bit);
osetbit(pcb(ptr).snd pcb(ptr).flg_byt),
pcb(ptr).flg_bit);
ptr := pcb(ptr).loc_con;
END LOOP;
ELSE
ptr := INTEGER(mem(blk).wnd(1));
IF ptr <= num_prts THEN
CASE pcb(ptr).Pstate is
WHEN local ! lstn =>
osetbit(pcb(ptr).ack pcb(ptr).flg_byt),
pcb(ptr).flg_bit);
ELSE
osetbit(pcb(ptr).
snd(pcb(prt).flg_byt),
pcb(prt).flg_bit);
IF pcb(ptr).Pstate = lstn
   AND ptr /= prt THEN
   activate_ptr(ptr);
   pcb(ptr).Pstate := local;
   pcb(ptr).loc_con :=
      pcb(prt).loc_con;
   pcb(ptr).loc_con := ptr;
   outptr(pcb(ptr).data_ptr,
      nullbyt);
END IF;
WHEN others =>
give_memory(blk);
pcb(prt).s_prtQ := 0;
END CASE;
ELSE
   give_memory(blk);
pcb(prt).s_prtQ := 0;
END IF;
ELSE  -- if byt = FF
   give_memory(blk);
   IF NOT pcb(prt).is_print THEN
      pcb(prt).time_wait :=
         pcb(prt).time_wait + 1;
      IF pcb(prt).time_wait=threshold THEN
         oPut("closing local connection");
oPut(" . time out."); oNEW LINE;
         IF pcb(prt).s_prtQ /= 0 THEN
            give_memory(pcb(prt).s_prtQ);
            pcb(prt).s_prtQ := 0;
         END IF;
         remove_link(prt);
         pcb_cls(prt);
      END IF;
      END IF;
   -- if blk /= 0
   END IF;
   -- tstbit DSR
   END IF;
ELSE  -- if byt = FF
   IF pcb(prt).sent THEN
      oPut("pcb(prt).sent is TRUE"); oNEW_LINE;
      found := FALSE;
      FOR i IN 0..max_flag_byt LOOP
         IF pcb(prt).snd(i) /= BYTE(0) THEN
            found := TRUE;
         END IF;
      END LOOP;
      IF NOT found THEN
         oPut("pcb(prt).sent is FALSE"); oNEW_LINE;
      END IF;
   END IF;
END IF;
END IF; -- if byt = FF
ELSE
   IF pcb(ptr).sent THEN
      oPut("pcb(ptr).sent is TRUE"); oNEW_LINE;
      found := FALSE;
      FOR i IN 0..max_flag_byt LOOP
         IF pcb(ptr).snd(i) /= BYTE(0) THEN
            found := TRUE;
         END IF;
      END LOOP;
      IF NOT found THEN
         oPut("pcb(ptr).sent is FALSE"); oNEW_LINE;
      END IF;
   END IF;
END IF; -- if blk /= 0
END IF;
-- tstbit DSR
END IF;
END IF;
--if blk /= 0
END IF;
--tstbit DSR
END IF;
--s_prtQ = 0
ELSE
   IF pcb(prt).sent THEN
      oPut("pcb(prt).sent is TRUE"); oNEW_LINE;
      found := FALSE;
      FOR i IN 0..max_flag_byt LOOP
         IF pcb(prt).snd(i) /= BYTE(0) THEN
            found := TRUE;
         END IF;
      END LOOP;
      IF NOT found THEN
         oPut("pcb(prt).sent is FALSE"); oNEW_LINE;
      END IF;
   END IF;
END LOOP;
IF NOT found THEN
remove_link(prt);
pcb(prt).Pstate := 1stn;
END IF;
END IF;
END IF;
END IF;  --NOT is_print
FOR i IN 0..max_flag_byt LOOP
  IF pcb(prt).snd(i) /= BYT(0) THEN
    FOR j IN 0..7 LOOP
      IF otsbit(pcb(prt).snd(i),j) THEN
        ptr := (8*i)+j;
        IF pcb(ptr).s_prtn /= 0 THEN
          IF pcb(ptr).is_print THEN
            snd_data_to_printer(prt,
                                pcb(ptr).s_prtn);
            IF mem(pcb(ptr).s_prtn).frm_len=0 THEN
              oclrbit(pcb(ptr).ack(pcb(prt).
                          flg_byt),pcb(prt).flg_bit);
              oclrbit(pcb(ptr).snd(pcb(ptr).
                          flg_byt),pcb(ptr).flg_bit);
            END IF;
          ELSE
            amt := arr_to_int(mem(pcb(ptr).s_prtn).
                               urg) + hdr_len;
            IF amt <= 518 THEN
              send_trns(mem(pcb(ptr).s_prtn).
                        wnd'ADDRESS,
                        pcb(prt).data_prtn,amt);
              IF amt = 0 THEN
                oclrbit(pcb(ptr).ack(pcb(prt).
                           flg_byt),pcb(prt).flg_bit);
                oclrbit(pcb(ptr).snd(pcb(ptr).
                           flg_byt),pcb(ptr).flg_bit);
              END IF;
            ELSE
              oclrbit(pcb(ptr).ack(pcb(prt).
                          flg_byt),pcb(prt).flg_bit);
              oclrbit(pcb(ptr).snd(pcb(ptr).
                          flg_byt),pcb(ptr).flg_bit);
            END IF;
          END IF;
        ELSE
          oclrbit(pcb(ptr).ack(pcb(prt).
                         flg_byt),pcb(ptr).flg_bit);
          oclrbit(pcb(prt).snd(pcb(ptr).
                         flg_byt),pcb(ptr).flg_bit);
        END IF;
      END IF;
    END LOOP;
  END IF;
END LOOP;
END IF;
END LOOP;
END loc;
with locXfer, ntrpthd, assylib, lib, globall, tcpSend;
package body poller is
use locXfer, assylib;

--FILE NAME: POLLER.PKG
--PROCEDURES CONTAINED:
-- 1. POLL 5. FTP 9. PCB_CLS
-- 2. SND_DATA_TO_PORT 6. LOC
-- 3. RLOG 7. GET_PRT_DAT
-- 4. REM_INIT 8. LOC_INIT
--AUTHOR: ALEC YASINSAC
--DATE: JAN 1985
--EXTERNAL REFERENCES:
--   --GLOBAL1.SPC CONTAINS ALL GLOBAL VARIABLES AND TYPES.
--   --LIB.PKG WHICH CONTAINS OUR UTILITY PROCEDURES.
--   --BIT.PKG CONTAINS THE ADA BIT MANIPULATION Routines
--   --IO.PKG
--INPUT: 1) PORT STATUS BYTE FOR EACH PORT
--   2) IP ADDR FOR REMOTE LOGIN AND FTP DESTINATION
--OUTPUT: 1) PORT NUMBER TO CALLED ROUTINES
--COMPILER: THIS PACKAGE WAS CODED TO COMPILE ON JANUS/ADA
--   UNDER CPM 86.
--DESCRIPTION:
--   --POLLER CONTAINS THE PROCEDURE 'POLL', THE CONTROLLING
--   --PROGRAM OF THE CONCENTRATOR. IT CHECKS EACH PORT FOR
--   --ACTIVITY FROM ITS CORRESPONDING PORT AND PASSES
--   --CONTROL TO THE APPROPRIATE SUBROUTINE BASED ON STATE.
--
--handshake signals betw concentratr and peripheral devices
--per gnd TxD RxD DTR DSR CHASIS
--|
| gnd RxD TxD DSR DTR CHASIS |
--concentrator
--
--principals of communication
--data data
--from remote DTR, wait a short time for DSR
--from concentrator DTR, wait a short time for DSR

use assylib, lib, globall;
prt_addr : INTEGER := INTEGER(16#0100#);
code_print : CONSTANT BYTE := BYTE(16#D4#);
code_endprint : CONSTANT BYTE := BYTE(16#F4#);
ndx : INTEGER; -- index to pcb tables
pred_ndx : INTEGER; -- predecessor of ndx
val : BYTE; -- input byte from port
ptr : INTEGER; -- pointer index
loopthrshld : CONSTANT INTEGER := 200;
hdr_len : CONSTANT INTEGER := 6;

--initialize rs232 UARTs
model : CONSTANT BYTE := BYTE(16#4E#);
mode2 : CONSTANT BYTE := BYTE(16#3E#);--3F 19.2K
commd : CONSTANT BYTE := clr;--txEN, RxEN and RTS
loopcnt : INTEGER;
blk : INTEGER;
stat : INTEGER;
bytstat : BYTE;
byt : BYTE;

procedure rem_init(prt_num:IN INTEGER; rem_tcp_addr:array2)is
--EXTERNAL CALLS TO: 1. TCPOPEN, TCP_ABORT
-- 2. PCB_CLS
-- 3. OTSTBIT
-- 4. IOPORT, OUTPORT
--This procedure initiates a remote login to the address
--received from the port. The ip address is four bytes and
--are stored in an array to send to tcp_open. Buf_in_cnt
--counts how many bytes of the address has been received.
--If the connection has not been established within the
--number of cycles indicated by the global constant
--'threshold', then the port will be closed.
use global1, assylib, lib, tcpsend;

rslt : BOOLEAN;
data : BYTE;
indx : INTEGER;
found : BOOLEAN;
loopcnt : INTEGER;

BEGIN --BEGIN PROCEDURE REM_INIT.
IF pcb(prt_num).buf_in_cnt = 0 THEN
inpct(pcb(prt_num).STAT_prt, data);--CHECK FOR CHR FM PRT.
if otstbit(data, DSR) then --THERE IS DATA TO BE READ.
outprt(pcb(prt_num).cmd_prt, DTR); --ready to receive
loopcnt := 0;
pcb(prt_num).buf_in_cnt := 1;
LOOP
   --GET ALLBYTES OF THE REMOTE IP ADDRESS.
   EXIT WHEN loopcnt = loopthrshld;
   inpnt(pcb(prtnum).stat prtn, data);
   IF otstbit(data,RxRdy) THEN
      inpnt(pcb(prtnum).data prtn, data);
      pcb(prtnum).buf_in.ip_ad(pcb(prtnum).buf_in_cnt)
         := data;
      pcb(prtnum).buf_in_cnt := pcb(prtnum).buf_in_cnt
         + 1;
   EXIT WHEN pcb(prtnum).buf_in_cnt = 5;
   END IF;
   loopcnt := loopcnt + 1;
END LOOP;
outprnt(pcb(prtnum).cmd prtn, clpr);
IF pcb(prtnum).buf_in_cnt /= 5 THEN
   pcb(prtnum).Pstate := cls;
ELSE
   pcb(prtnum).buf_in.tcp ad := rem tcp addr;
   --The pcb state will be changed by the int handler
   --when the tcp connection is established.
   tcp_open(prtnum,
      pcb(prtnum).buf_in,
      pcb(prtnum).act,
      pcb(prtnum).l_prtn_ad,
      pcb(prtnum).sent);
   IF NOT NI3010Ok THEN
      get_tcb_ndx(pcb(prtnum).l_prtn_ad,indx,found);
      tcb(indx).prt num := 99;
      pcb_cls(prtnum);
      pcb_cls(prtnum);
      pcb(predndx).pcb_ptr := pcb(prtnum).pcb_ptr;
   END IF;
   IF NOT pcb(prtnum).sent THEN
      pcb_cls(prtnum);
      pcb_cls(prtnum);
   END IF;
ELSE
   IF pcb(prtnum).time_wait>threshold THEN    --TIMED OUT.
      IF NOT pcb(prtnum).sent THEN
         tcp_open(prtnum,
            pcb(prtnum).buf_in,
            pcb(prtnum).act,
            pcb(prtnum).l_prtn_ad,
            pcb(prtnum).sent);
         IF NOT NI3010Ok THEN
            get_tcb_ndx(pcb(prtnum).l_prtn_ad,indx,found);
            tcb(indx).prt num := 99;
            pcb_cls(prtnum);
            pcblinkplain(prtnum).pcb_ptr := pcb(prtnum).pcb_ptr;
         END IF;
         IF NOT pcb(prtnum).sent THEN
            oPUT("'syn' packet not sent again"); oNEW_LINE;
            pcb_cls(prtnum);
         END IF;
         pcb(prtnum).time_wait := 0;
      ELSE

procedure ftp(prtnum: in integer) is
--CURRENT: 19 May 86
--AUTHOR: ALEC YASINSAC APRIL 86
--DESCRIPTION: FTP IS PASSED CONTROL WHEN PORT 'PORT NUM' IS
--POLLED IN STATE FTP. AT THIS STAGE, A COMMAND CONNECTION
--HAS BEEN ESTABLISHED BETWEEN THE Z-100 AND THE REMOTE
--SITE. THREE LEVELS OF COMMUNICATION ARE POSSIBLE BOTH
--TO AND FROM A Z-100:
-- 1. CONTROL CODES.
-- 2. DATA THRU THE SECONDARY OR DATA CONNECTION.
-- 3. COMMANDS/REPLYS THRU THE FTP CMD CONNECTION.
--CONTROL CODES FROM THE Z-100 ARE CHECKED FIRST. THEN
--ANY DATA FROM THE Z-100 IS ACCEPTED. DATA WAITING FOR
--THE Z-100 IS THEN SENT.

use tcpsend, assylib, globall1;
type port_rec is record
  typ_tran : byte;
  sock: socket_rec;
end record;
out_rec: port_rec;
code_addr: constant byte := byte(16#e1#); --a 225
code_cmd: constant byte := byte(16#e3#); --c 227
code_data: constant byte := byte(16#e4#); --d 228
code_qempty: constant byte := byte(16#e5#); --e 229
code_getcpad: constant byte := byte(16#e7#); --init tcb entry
code_check_replyq: constant byte := byte(16#e8#); --h 232
code_more: constant byte := byte(16#ed#); --m more data. 237
code_open: constant byte := byte(16#ef#); --o 239
code_closdata: constant byte := byte(16#f1#); --q 241
code_reply: constant byte := byte(16#f2#); --r 242
code_dprtstat: constant byte := byte(16#f4#); --t 244
bytstat: byte;
len, ndx, holdq, inxl, i: integer;
ind: integer; --stub.
found, sent: boolean;
byt_arr: array (1..512) of byte;

begin
-- IS THERE A CNTL CHAR FROM THE Z-100?
inprt(pcb(prt_num).stat_prt, bytstat);
if otstbit(bytstat,rxrdy) then --got a control char
inprt(pcb(prt_num).data_prt,byt);
case byt is
    when code ftp =>
        pcb_abort(prt_num);
    when code_check_replyq =>
        if pcb(prt_num).s_prtq > 0 then
            outprt(pcb(prt_num).data_prt,bytstat);
        else
            outprt(pcb(prt_num).data_prt,code_qempty);
        end if;
    when code_dprtstat => --CHECK STATUS OF DATA PORT.
        if pcb(prt_num).sec_act then
            outprt(pcb(prt_num).data_prt,code_open);
        else
            outprt(pcb(prt_num).data_prt,code_closdata);
        end if;
    when code_getcpad =>
        if pcb(prt_num).sec_act then null;
        --WILL SEND ACTIVE ADDRESS.
        --outprt(pcb(prt_num).data_prt,code_open);
        else
            tcp_open(pprt_num,
                pcb(prtnum).buf_in,
                false,
                pcb(prtnum).s_prt_ad,
                pcb(prtnum).sent);
        end if;
    end if;
out_rec.typ_tran := code_addr;
out_rec.sock.ip_ad := loc_ip_ad;
out_rec.sock.tcp_ad := pcb(prtnum).s_prt_ad;
outprt(pcb(prtnum).data_prt,code_getcpad);
for i in 1..30 loop
    len := 7;
    send_trns(out_rec'address,
        pcb(prtnum).data_prt,len);
    exit when len = 0;
end loop;
when code_open =>--ASKING TO OPEN DATA CONNECTION.
    null;
when code_abort =>
    outprt(pcb(prtnum).data_prt,code_abort);
    pcb_abort(prt_num);
when code_closdata =>
    if pcb(prtnum).sec_act then
        tcp_close(pcb(prtnum).s_prt_ad);
        pcb(prtnum).sec_act := false;
    end if;
when code_cls =>
229
outprt(pcb(prt_num).data_prt, code_cls);
tcp_close(pcb(prt_num).l_prt_ad);
if pcb(prt_num).sec_act then
tcp_close(pcb(prt_num).s_prt_ad);
end if;
pcb(prt_num).sent := FALSE;
pcb(prt_num).pstate := clsing;
when others => null;
end case;
else -- THERE IS NOT A CONTROL CHARACTER FROM THE Z-100.
-- IS THERE DATA OR A COMMAND FROM THE Z-100?
if otstbit(bytstat, dsr) then-- SOMETHING FROM Z_100.
if used_blk<max_mem_blk then
get_memory(inxl); -- GET A NEW PACKET INDEX.
len := 513;
get_trns(mem(inxl).urg(2)'address,
pcb(prtnum).data_prt, len);-- STOR IN PACKET.
if len > 0 then
if mem(inxl).urg(2) = code_cmd then
--SEND CMD FM Z-100 TO REM OVER CMD LINE.
@oput("cmd = ");
@for i in 1..4 loop
@oput(integer(mem(inxl).data(i)));
@end loop; one line;
tcp_send(inxl, len - 1,

pcb(prt_num).l_prt_ad, sent);
else -- NOT A COMMAND FROM THE Z-100.
if mem(inxl).urg(2) = code_data then
--SEND DATA FM Z TO REM OVER DATA CONN.
if pcb(prtnum).sec_act then
tcp_send(inxl, len - 1,

pcb(prtnum).s_prt_ad, sent);
else-- TRIED DATA W/ NO DATA CONNECTION.
give_memory(inxl);
end if;
else --BYTES FROM Z-100 NOT IDENTIFIED.MAY
-- MEAN USER HAS REBOOTED SO DSR IS HIGH
-- THOUGH NO DATA IS ACTUALLY BEING SENT.
give_memory(inxl);
pcb(prt_num).time_wait :=

pcb(prt_num).time_wait + 1;
if pcb(prt_num).time_wait > 100 then
outprt(pcb(prt_num).data_prt,

code_abort);
pcb_abort(prt_num);
end if;
end if;-- ENDS IF BYTES RECEIVED ARE DATA.
end if; -- ENDS IF COMMAND.
else
give_memory(inxl); -- NO DATA RECEIVED
end if;
else --ALL MEMORY BLOCKS ARE IN USE.
230
null; -- CANNOT GET DATA FROM THE Z-100.
end if; -- END IF NOT ALL MEMORY BLOCKS IN USE.
else
null; -- NOTHING WAITING. NO ACTION REQUIRED.
end if; -- END IF Z-100 TRYING TO SEND DATA.

-- IS THERE DATA FOR THE Z-100?
if pcb(prt_num).s_prtq > 0 then -- FTP DATA FOR Z-100.
  inx1 := pcb(prt_num).s_prtq;
  mem(inx1).urg(2) := code_data;
  len := mem(inx1).frm_len + 1;
  send_trns(mem(inx1).urg(2)'ADDRESS,
             pcb(prt_num).data_prt,len);
  IF len = 0 then
    pcb(prt_num).s_prtq := mem_manag_tbl(inx1);
    give_memory(inx1);
  END IF;
else -- NO DATA IS WAITING FROM DATA CONNECTION.
  --if not pcb(prt_num).sec_act then
  -- IS THERE A REPLY FOR THE Z-100?
  if pcb(prt_num).prtq /= 0 then -- FTP REPLY.
    inx1 := pcb(prt_num).prtq;
    mem(inx1).urg(2) := code_reply;
    len := mem(inx1).frm_len + 1;
    send_trns(mem(inx1).urg(2)'ADDRESS,
               pcb(prt_num).data_prt, len);
    IF len = 0 THEN
      pcb(prt_num).prtq := mem_manag_tbl(inx1);
      give_memory(inx1);
    end if; -- END IF LEN = 0.
  end if; -- END IF PRTQ /= 0.
  --end if; -- END IF NOT SEC ACT.
end if; --ENDS IF THERE IS A CONTROL CODE FROM THE Z-100.
end ftp;

---------------------------------------------
procedure rlog(prt_num: in integer) is
  -- AUTHOR: ALEC YASINSAC   -- DATE: FEB 86
  -- INPUT: 1. PRT_NUM IS THE PORT NUMBER CURRENTLY BEING
          -- PROCESSED AND 0 <= PRT_NUM <= 23
  -- OUTPUT: 1. FIELDS MODIFIED IN THE GLOBAL TABLE PCB.
          -- 2. FIELDS MODIFIED IN THE GLOBAL TABLE TCB.
  -- EXTERNAL MODULES CALLED: 1. GIVE_MEMORY
  -- 2. GET_MEMORY
  -- 3. TCP_SND
  -- 4. TCP_ABORT
  -- 5. CONV_HEXARR_INT
  -- 6. TST_BIT
  -- DESCRIPTION: RLOG IS PASSED CONTROL BY POLLER WHEN A PORT
  -- WITH PORT NUMBER 'PRT_NUM' IS POLLED AND IS IN THE RLOG
STATE (WHICH MEANS A REMOTE CONNECTION HAS BEEN ESTAB-
lished). RLOG WILL THEN SEND DATA WAITING FOR THE PORT
AND POLL THE PORT FOR DATA TO THE REMOTE HOST. THE
NUMBER OF CHARS IN A PACKET FOR THE PORT IS STORED IN
THE FRM_LEN FIELD OF THE MEMORY BLK.

use lib, global1, assylib, tcpsend;
arr4isl: array4;
max_used_blk : CONSTANT INTEGER :=
max_mem_blk - 1; --leave one spare
rcvRdy: CONSTANT integer := 1;
bytinp, bytstat: byte;
next, inp, status, ndx: integer;
found, sent: BOOLEAN;
ptr: integer;

BEGIN
  --WILL PROCESS DATA FROM Z-100 THEN DATA FROM ETHERNET
inprt(pcb(prtnum).stat_prt, bytstat);
if otstbit(bytstat, DSR) then -- INFORMATION FROM Z-100.
  IF used_blk < max_used_blk THEN
    loopcnt := 0;
    ptr := 1;
    get_memory(next);
    IF next /= 0 THEN
      mem(next).frm_len := 512;
      get_trns(mem(next).data(l)'ADDRESS,
      pcb(prtnum).data_prt,
      mem(next).frm_len);
      IF mem(next).frm_len > 0 THEN
        tcp_send(next, mem(next).frm_len,
        pcb(prtnum).l_prt_ad, sent);
        IF NOT NI3010 ok THEN
          get_tcb_ndx(pcb(prtnum).l_prt_ad,
          ndx, found);
          tcb_cls(ndx);
          pcb_cls(prtnum);
          pcb(predndx).pcb_ptr :=
          pcb(prtnum).pcb_ptr;
      END IF;
    ELSE
      give_memory(next);
      pcb(prtnum).time_wait :=
      pcb(prtnum).time_wait + 1;
      IF pcb(prtnum).time_wait = threshold THEN
        tcp_close(pcb(prtnum).l_prt_ad);
        pcb(prtnum).sent := FALSE;
        pcb(prtnum).Pstate := clsing;
      END IF;
    END IF;
  END IF;
END IF;
END IF;
END IF;
END IF;
ELSE IF otstbit(bytstat,RxRdy) THEN

  inp(rpcb(prt_num),data_prt,bytinp);

  case bytinp is
    when code_abort => pcbAbort(prt_num);
    when code_status => give_status(pcb(prt_num));
    when code_cls => tcp_close(pcb(prt_num),l_prt_ad);
    pcb(prt_num).sent := FALSE;
    pcb(prt_num).Pstate := closing;
    when code_Arlog => outp(rpcb(prt_num),data_prt,
                              code_Arlog);
    when others => null;
  end case;

  END IF;
end if; --END otstbit. END PROCESSING DATA FROM A Z-100.

IF pcb(prt_num).prtn > 0 THEN

  next := pcb(prt_num).prtn;
  send_trns(mem(next),data(1)'ADDRESS,
             pcb(prt_num),data_prt,
             mem(next),frm_len);

  IF mem(next).frm_len = 0 THEN
    pcb(prt_num).prtn := mem-manag-tbl(next);
    give_memory(next);
  END IF;
END IF;
END rlog;

---------------------------------------------------------------------

PROCEDURE initialize_mem is

BEGIN

  FOR i IN 0..num_prts LOOP
    pcb(i).prtn := 0;
    pcb(i).s_prt := 0;
    pcb(i).sent := FALSE;
    pcb(i).time_wait := 0;
    pcb(i).act := TRUE;
    pcb(i).sec_act := FALSE;
  END LOOP;

  FOR i in 1..max_mem_blk - 1 LOOP
    mem-manag-tbl(i) := i + 1;
  END LOOP;

  mem-manag-tbl(max_mem_blk) := 0;
  used_blk := 0;
  free_blk := 1;

  FOR i IN 0..max_tcb LOOP
    tcb(i).prt_num := 99;
    tcb(i).retrnsQ := 0;
  END LOOP;

  rcv_wnd(1) := BYTE(2);
  rcv_wnd(2) := BYTE(0);

END initialize_mem;

233
procedure poll is
  use lib, tcpsend, global1;
  rcvRdy : CONSTANT INTEGER := 1;
  loops_to_poll: CONSTANT INTEGER := 1000;
  bytcode, bytstat: byte;
  ndx, indx, inx: INTEGER;
  pred_ndx : INTEGER;
  loop_cnt, len: INTEGER;
  found : BOOLEAN;
  rlog_tcp : array2;
  ftp_tcp : array2;
  pntr, qadd : INTEGER;
  no_active : BOOLEAN;
begin
  rlog_tcp(1) := byte(0);
  rlog_tcp(2) := byte(16#17#);
  ftp_tcp(1) := byte(0);
  ftp_tcp(2) := byte(16#15#);
  loop_cnt := 0;
  loop
    pred_ndx := pcb_head;
    ndx := pcb(pcb_head).pcb_ptr;
    loop
      EXIT WHEN ndx > num_prts;
      case pcb(ndx).pstate is
        when clsing =>
          if pcb(ndx).sec_act then
            get_tcb_ndx(pcb(ndx).s prt ad, indx, found);
            tcb_cls(indx);--CLEAR RETRANSMISSION QUE.
            tcp_close(pcb(ndx).s prt ad); --SEND FIN.
            while pcb(ndx).s prtq /= 0
              loop--DELETE DATA ON SECOND CONNECTION.
                qadd := mem_manag_tbl(pcb(ndx).s prtq);
                give_memory(pcb(ndx).s prtq);
                pcb(ndx).s prtq := qadd;
                end loop;
            pcb(ndx).sec_act := FALSE;
          else
            IF pcb(ndx).sent THEN
              pcb(ndx).time_wait :=
                pcb(ndx).time_wait + 1;
            IF pcb(ndx).time_wait=threshold THEN
              get_tcb_ndx(pcb(ndx).1 prt ad,
                indx, found);
              tcb_cls(indx);
              pcb_cls(ndx);
              pcb(pred ndx).pcb_ptr :=
                pcb(ndx).pcb_ptr;
            END IF;
          END IF;
  end loop;
  loop
  END loop;
end poll;
ELSE
    if pcb(ndx).prtq = 0 then
        tcp_close(pcb(ndx).l_prt_ad);
        pcb(ndx).sent := TRUE;
        pcb(ndx).time_wait := 0;
    else
        pntr := pcb(ndx).prtQ;
        len := mem(pntr).frm_len;
        send_trns(mem(pntr).data(1)'ADDRESS,
                   pcb(ndx).data_prt, len);
        if len = 0 then
            pcb(ndx).prtQ :=
                mem_manag_tbl(pntr);
            give_memory(pntr);
        else
            mem(pntr).spare :=
                mem(pntr).spare + 1;
            if mem(pntr).spare>=threshold then
                pcb(ndx).prtQ :=
                    mem_manag_tbl(pntr);
                give_memory(pntr);
            end if;
        end if;
    end if;
END IF;

when cls =>
    pcb(pred_ndx).pcb_ptr := pcb(ndx).pcb_ptr;
when lstn=> pcb(pred'ndx) .pcb_ptr :=
    pcb (ndx) .pcb_ptr;
when r_init => rem_init(ndx,rlog_tcp);
when rlogn => rlog(ndx);
when f_init => rem_init(ndx,ftp_tcp);
when rftp => ftp(ndx);
when l_init => loc_init(ndx);
when local => loc(ndx);
when others => pcb(ndx).Pstate := cls;
end case;

pred_ndx := ndx;
ndx := pcb(ndx).pcb_ptr;
end loop;

pred_ndx := pcb_head;
ndx := pcb(pcb_head).pcb_ptr;
loop_cnt := loop_cnt + 1;

IF loop_cnt = loops_to_poll THEN
    no active := TRUE;
    FOR i IN 0..num_prts LOOP
        IF pcb(i).Pstate=cls OR pcb(i).Pstate=lstn THEN
            inp(rt(pcb(i).stat_prt, bytstat);

235
if otstbit(bytstat,rcvRdy) then
  inprt(pcb(i).data_prt, bytcode);
CASE bytcode IS
  WHEN code_Arlog =>
    oPUT("Code_Arlog received");
    oNEW_LINE;
    pcb(i).Pstate := r_init;
    outprt(pcb(i).data_prt,bytcode);
    activate_prt(i);
  WHEN code_ftp =>
    oPUT("code_ftp received");
    oNEW_LINE;
    pcb(i).Pstate := f_init;
    outprt(pcb(i).data_prt,bytcode);
    activate_prt(i);
  WHEN code_loc =>
    oPUT("code_loc received");
    oNEW_LINE;
    pcb(i).Pstate := l_init;
    outprt(pcb(i).data_prt,bytcode);
    activate_prt(i);
  WHEN code_Prlog =>
    pcb(i).Pstate := r_init;
    outprt(pcb(i).data_prt,bytcode);
    activate_prt(i);
  WHEN code_lstn =>
    pcb(i).Pstate := lstn;
    outprt(pcb(i).data_prt,bytcode);
  WHEN code_status =>
    IF pcb(i).Pstate = lstn THEN
      outprt(pcb(i).data_prt,
           code_status);
      give_status(i);
    END IF;
  WHEN code_reqPrt =>
    outprt(pcb(i).data_prt,bytcode);
    outprt(pcb(i).data_prt,BYTE(i));
  WHEN code_quit =>
    oput("SYSTEM ABORTED BY CONSOLE");
  WHEN code_cls =>--to close a listen
    pcb_cls(i);
  WHEN code_print =>
    ptr := 0;
    LOOP
      IF pcb(ptr).is_print AND
         pcb(ptr).Pstate = lstn THEN
        outprt(pcb(i).data_prt,
             BYTE(ptr));
        activate_prt(ptr);
        activate_prt(i);
        pcb(ptr).loc_con := i;
    END LOOP;
  RETURN;
pcb(i).loc_con := ptr;
pcb(ptr).Pstate := local;
pcb(i).Pstate := local;
pcb(ptr).sent := FALSE;
EXIT;
END IF;
ptr := ptr + 1;
IF ptr > num_prts THEN
  outprt(pcb(i).data_prt,
         code_quit);
  EXIT;
END IF;
END LOOP;
WHEN others =>--BAD CODE RECEIVED.
  outprt(pcb(i).data_prt,code_cls);
END CASE;
END IF;
IF pcb(i).s_prtQ /= 0 THEN
  blk := pcb(i).s_prtQ;
  mem(blk).frm_len :=
        arr_to_int(mem(blk).urg)+hdr_len;
  send_trans(mem(blk).wnd(1)\ADDRESS,
            pcb(i).data_prt,
            mem(blk).frm_len);
  IF mem(blk).frm_len=0 THEN
    give_memory(blk);
    pcb(i).s_prtQ := 0;
  END IF;
END IF;
ELSE IF pcb(i).Pstate = rlogn OR pcb(i).Pstate
         = rftp THEN
  check_retrnsQ(pcb(i).l_prt_ad);
END IF;
END IF;
IF pcb(i).Pstate /= cls AND NOT pcb(i).is_print
THEN
  no_active := FALSE;
END IF;
END LOOP;
IF no_active THEN
  initialize_mem;
END IF;
loop_cnt := 0;
END IF;
end loop;
end poll;

begin --INITIALIZATION FOR CONTROLLING PACKAGE.
  pcb(pcb_head).pcb_ptr := pcb_head;
  FOR i IN 0..num_prts LOOP
    pcb(i).data_prt := i * 4 + (32 * (i/8)) + 256;
  END LOOP;
  pcb(pcb_head).pcb_ptr := pca_head;
end poll;
pcb(i).statprt := pcb(i).dataprt + 1;
pcb(i).cmdprt := pcb(i).statprt + 2;
CASE i is
  WHEN 0 => pcb(i).is_print := TRUE;
  WHEN others => pcb(i).is_print := FALSE;
END CASE;
outprt(pcb(i).statprt+1,model);
outprt(pcb(i).statprt+1,mode2);
outprt(pcb(i).cmdprt,commd);

IF pcb(i).is_print THEN
  pcb(i).Pstate := lsn;
ELSE
  pcb(i).Pstate := cls;
  outprt(pcb(i).dataprt,code_cls);
END IF;
pcb(i).buf.in_cnt := 0;
FOR j IN 0..max_flag_byt LOOP
  pcb(i).snd(j) := BYTE(0);
  pcb(i).ack(j) := BYTE(0);
END LOOP;
pcb(i).flg_byt := i/8;
pcb(i).flg_bit := i REM 8;
END LOOP;

--initialize memory
initialize_mem;

--the following initialization is the internet protocol
--address assigned to the aegis system and listed in the
--VAX UNIX local host table for 'npacs-aegis'
loc_ip_ad(1) := byte(16#C0#); --decimal equivalent: 192
loc_ip_ad(2) := byte(16#09#); -- 9
loc_ip_ad(3) := byte(16#c8#); -- 200
loc_ip_ad(4) := byte(16#04#); -- 4

nxt_prt_ad := 1024;
ni3010_ok := TRUE;
eth.type_pck(1) := BYTE(8); --address resolution protocol
eth.type_pck(2) := BYTE(6); --see RFC 826, Ntwk Info Cntr
eth.ar_hrd(1) := BYTE(0);
eth.ar_hrd(2) := BYTE(1);
eth.ar_pro(1) := BYTE(8);
eth.ar_pro(2) := BYTE(0);
eth.ar_len(1) := BYTE(6);
eth.ar_len(2) := BYTE(4);
eth.null := BYTE(0);
eth.fm_ip := loc_ip_ad;

FOR i IN 1..max_ad LOOP
  ad_tbl(i).update := 0;

238
END LOOP;
ntrpt := disable;
outprt(able_reg,disable);
inprt(stat_reg,val);
perf_cmd(go_off);
perf_cmd(reset);
outprt(icw1_prt,icw1);
outprt(icw2_prt,icw2);
outprt(icw4_prt,icw4);
outprt(ocw_prt,ocw);

ASM
sti; -- set interrupt-enable flag

perf_cmd(go_on);

perf_cmd(rcv_stat);

ptr := 1;
LOOP
  inprt(ntrpt_reg,val);
  EXIT WHEN otstbit(val,sba);
  IF otstbit(val,srf) THEN
    inprt(stat_reg,val);
    mem(1).data(ptr) := val;
    ptr := ptr + 1;
  END IF;
END LOOP;
inprt(stat_reg,val);
FOR i IN 1..6 LOOP
  loc_eth_ad(i) := mem(1).data(i+3);
END LOOP; oNEW_LINE;

oPUT("RUNNING");
oneNEW_LINE;

eth.fm_eth_ad := loc_eth_ad;
eth.fm_eth := loc_eth_ad;
ad_tbl(max_ad).eth_ad := loc_eth_ad;
ad_tbl(max_ad).ip_ad := loc_ip_ad;
ad_tbl(max_ad).update := INTEGER(16#7FFF#);
perf_cmd(cl_insert_mode);
ntrpt := rcv_pck;
outprt(able_reg,rcv_pck);

poll;
perf_cmd(reset);
end poller;
BATCH FILE IN COMPILATION ORDER

The following is the contents of the batch (.bat or .sub) file used to compile the preceding programs.

era globall.sym
era assylib.sym
era lib.sym
era ntrpthd.sym
era convblk.sym
era pcbrec.sym
era tcprec.sym
era iprec.sym
era ethrec.sym
era rcv.sym
era ethsend.sym
era ipsend.sym
era tcpsend.sym
era locxfer.sym
janus globall.spc
janus assylib.spc
janus lib.spc
janus ntrpthd.spc
janus convblk.spc
janus pcbrec.spc
janus tcprec.spc
janus iprec.spc
janus ethrec.spc
janus rcv.spc
janus ethsend.spc
janus ipsend.spc
janus tcpsend.spc
janus locxfer.spc
jasm86 assylib.asm
janus lib.pkg
jasm86 ntrpthd.asm
jasm86 convblk.asm
janus pcbrec.pkg
janus tcprec.pkg
janus iprec.pkg
janus ethrec.pkg
janus rcv.pkg
janus ethsend.pkg
janus ipsend.pkg
janus tcpsend.pkg
janus locxfer.pkg
janus poller.pkg
jlink poller/re
APPENDIX G

LISTING OF Z-100 TELNET PROGRAMS

--PACKAGE: TELNET_PKG
--AUTHOR: ALEC YASINSAC
--DATE: DEC 1985
--SYSTEM NAME: TELNET
--EXTERNAL REFERENCES: 1. GET_ADDR
--INPUT: HOSTS.FIL

--DESCRIPTION:
-- TELNET ALLOWS A USER AT A Z-100 TO UTILIZE THE MULTIBUS
-- SYSTEM AND ETHERNET TO BECOME A REMOTE TERMINAL TO OTHER
-- HOSTS ON ETHERNET. THIS PROGRAM SENDS A CONTROL
-- CHARACTER TO ITS DESIGNATED OUTPUT PORT FOR THE 8612
-- CONCENTRATOR THAT IDENTIFIES THE FUNCTION TO BE
-- PERFORMED. THIS PROGRAM ALSO DETERMINES THE INTERNET
-- PROTOCOL ADDRESS AND FORWARDS THAT INFORMATION TO THE
-- 8612. THE 8612 PERFORMS ALL THE TELECOMMUNICATION
-- PROCESSES NECESSARY TO NAVIGATE PROTOCOLS. TELNET
-- MERELY PASSES DATA BETWEEN THE USER AND THE 8612. ONCE
-- LOGGED ON TO THE REMOTE HOST, THE Z-100 USER CAN
-- NAVIGATE ANY HOST ACCESSIBLE TO THE REMOTE HOST. FOR
-- EXAMPLE, A USER CAN LOG ON TO THE VAX UNIX SYSTEM AND
-- FROM THERE LOG ON TO NODES IN ARPA NETWORK WHICH IS SERVICED
-- BY VAX UNIX.

with bit, asmlib, get_ip, io;
procedure telnet is
  use bit, asmlib, get_ip, io;
  bytaddr: array (1..4) of byte;
  addr: array (1..4) of integer;
  ocwl_reg: constant integer := (16#00f3#);
  ocw1: constant byte := byte (16#aa#);
  auxprt: constant integer := (16#00ec#);
  term: constant byte := byte (16#1d#);  --^]
  code_cls: constant byte := byte (16#c3#);  --C
  code_abort: constant byte := byte (16#c1#);  --A
  code_status: constant byte := byte (16#d3#);  --S
  code_arlog: constant byte := byte (16#d2#);  --R
  dat: constant integer := (16#ec#);
  stat: constant integer := (16#ed#);
  cmd: constant integer := (16#ef#);
  clr: constant byte := byte (16#25#);
  DSR: constant integer := 7;
  DTR: constant byte := byte (16#27#);
RxRdy: constant integer := 1;
rs232_delay: constant integer := 100;
hosts, auxfile: file;
ch: character;
outcnt, len, cnt, ptr: integer;
datstrg: array(1..512) of byte;
ok, cnt_exit: BOOLEAN;
org_ocwl, byt, data, charbyt: BYTE;

function checkterm return boolean is
use asmlib, io;
byt: byte;

begin
if keypress() then
getch(byt);
case byt is
when byte(16#1d#) => return true;
when others => null;
end case;
return false;
end if;
end checkterm;

begin
--begin TELNET
inport(ocwl_reg,org_ocwl);
outport(ocwl_reg,ocwl);
outport(cmd,clr);
new_line; new_line;
crscreen;
put("WELCOME TO THE MULTIUSER SYSTEM TELNET PROCESS.");
new_line;
open(hosts,"hosts.fil",read-only);
if ioreresult = 255 then
put("FILE 'HOSTS.FIL' DOES NOT EXIST.");
ELSE
if end_of_file(hosts) then
put("NO DATA IN FILE 'HOSTS.FIL'.");
close(hosts);
else
close(hosts);
get_addr(addr(1),addr(2),addr(3),addr(4));
if ((addr(1) = 0) and (addr(2) = 0) and
(addr(3) = 0) and (addr(4) = 0)) then
new_line;--BY CONVENTION, telnet RECOGNIZES AN
new_line;--IP ADDR OF ZERO AS USER TERMINATION.
put("TELNET TERMINATED BY USER.");
else
new_line;
for i in 1..4 loop
bytaddr(i) := byte(addr(i));
end loop;
open(auxfile,"aux",read_write);
write(auxfile,code_arlog);
outport(auxprt,code_arlog);
PUT(" CONNECTING WITH CONCENTRATOR.");NEW_LINE;
loop
  inport(stat,data);
  IF tstbit(INTEGER(data),RxRdy) THEN
    clrscreen;
    PUT("trying ..."); NEW_LINE;
    inport(dat,data);
  case data is
    when code_arlog => exit;
      .when code_cls =>
        write(auxfile,code_arlog);
        when others=>write(auxfile,code_arlog);
    end case;
    if checkterm() then
      return;
    end if;
  END IF;
end loop;
close(auxfile);
loop
  len := 4;
  send_trns(bytaddr'address,dat,len);
  EXIT WHEN len = 0;
  if checkterm() then
    return;
  end if;
end loop;
outport(cmd,clr);
outcnt := 0;
LOOP -- MAIN LOOP SENDING DATA BTWN HOST & USER.
  IF keypress() AND outcnt = 0 THEN
    --OUTCNT = 0 MEANS LAST CHAR WAS SENT.
    getch(charbyt);
    EXIT WHEN charbyt = term;
    outcnt := 1;
  END IF;
  IF outcnt = 1 THEN
    inport(stat,data);
    IF NOT tstbit(INTEGER(data),DSR) THEN
      send_trns(charbyt'ADDRESS,dat,outcnt);
      --OUTCNT WILL BE 0 IF SEND SUCCESSFUL.
    END IF;
  END IF;
end loop;
inport(stat,data);
IF tstbit(INTEGER(data),RxRdy) THEN
  inport(dat,data);
  EXIT WHEN data = code_cls;
END IF;
  import(stat,data);
  IF tstbit(INTEGER(data),DSR) THEN
    ptr := 512;
    get_trns(datstrg'ADDRESS,dat,ptr);
    FOR i IN 1..ptr LOOP
      prntdata(datstrg(i));
    END LOOP;
  END IF;
END LOOP;
END IF;
end if; -- ENDS 'IF ADDR = 0
END IF; -- ENDS ' IF END_OF_FILE(HOSTS)'
end if; --ENDS 'IF IORESULT = 255
new_line;
outport(dat,code_cls);
outport(ocwl_reg,org_ocwl); --restore state
end telnet;
APPENDIX H

LISTING OF Z-100 FTP PROGRAMS

with typpkg;
package funcs is
  use typpkg;
  function checkterm return boolean;
  function get_opt return cmd_typ;
  function get_password return string;
  function get_username return string;
  function get_portnum return string;
  function get_filename return string;
  function get_parameter(opt:in cmd_typ) return string;
end funcs;

with typpkg, strlib, io, bit, asmlib;
package body funcs is
  use typpkg, strlib, io, bit, asmlib;

function checkterm return boolean is
  use asmlib, io;
  byt: byte;

begin
  if keypress() then
    @put("got keypress"); new_line; --stub.
    getch(byt);
    case byt is
      when byte(16#1d#) => return true;
      when others => null;
    end case;
    return false;
  end if; --END IF KEYPRESS.
end checkterm;

function get_opt return cmd_typ is
  --AUTHOR: ALEC YASINSAC APRIL 86
  --DESCRIPTION: GET_OPTION DISPLAYS THE POSSIBLE FTP OPTION
  -- SELECTIONS AND PROMPTS THE USER TO SELECT AN OPTION.
  -- THE OPTION IS RETURNED AS THE ONLY OUTPUT.
  --EXTERNAL CALLS TO: 1. IO.GET_LINE.
  use io;
  str: string;

  --AUTHOR: ALEC YASINSAC APRIL 86
  --DESCRIPTION: GET_OPTION DISPLAYS THE POSSIBLE FTP OPTION
  -- SELECTIONS AND PROMPTS THE USER TO SELECT AN OPTION.
  -- THE OPTION IS RETURNED AS THE ONLY OUTPUT.
  --EXTERNAL CALLS TO: 1. IO.GET_LINE.
  use io;
  str: string;

  --AUTHOR: ALEC YASINSAC APRIL 86
  --DESCRIPTION: GET_OPTION DISPLAYS THE POSSIBLE FTP OPTION
  -- SELECTIONS AND PROMPTS THE USER TO SELECT AN OPTION.
  -- THE OPTION IS RETURNED AS THE ONLY OUTPUT.
  --EXTERNAL CALLS TO: 1. IO.GET_LINE.
  use io;
  str: string;

  --AUTHOR: ALEC YASINSAC APRIL 86
  --DESCRIPTION: GET_OPTION DISPLAYS THE POSSIBLE FTP OPTION
  -- SELECTIONS AND PROMPTS THE USER TO SELECT AN OPTION.
  -- THE OPTION IS RETURNED AS THE ONLY OUTPUT.
  --EXTERNAL CALLS TO: 1. IO.GET_LINE.
  use io;
  str: string;

  --AUTHOR: ALEC YASINSAC APRIL 86
  --DESCRIPTION: GET_OPTION DISPLAYS THE POSSIBLE FTP OPTION
  -- SELECTIONS AND PROMPTS THE USER TO SELECT AN OPTION.
  -- THE OPTION IS RETURNED AS THE ONLY OUTPUT.
  --EXTERNAL CALLS TO: 1. IO.GET_LINE.
  use io;
  str: string;

  --AUTHOR: ALEC YASINSAC APRIL 86
  --DESCRIPTION: GET_OPTION DISPLAYS THE POSSIBLE FTP OPTION
  -- SELECTIONS AND PROMPTS THE USER TO SELECT AN OPTION.
  -- THE OPTION IS RETURNED AS THE ONLY OUTPUT.
  --EXTERNAL CALLS TO: 1. IO.GET_LINE.
  use io;
  str: string;

  --AUTHOR: ALEC YASINSAC APRIL 86
  --DESCRIPTION: GET_OPTION DISPLAYS THE POSSIBLE FTP OPTION
  -- SELECTIONS AND PROMPTS THE USER TO SELECT AN OPTION.
  -- THE OPTION IS RETURNED AS THE ONLY OUTPUT.
  --EXTERNAL CALLS TO: 1. IO.GET_LINE.
chr, junk: character;
valid: boolean;

begin
  loop
    valid := true; new_line;
    put("ENTER THE FIRST CHARACTER OF THE");
    put(" OPTION YOU PREFER."); new_line;
    put(" <S>END A FILE "); new_line;
    put(" <G>ET A FILE "); new_line;
    put(" <D>LETE A FILE "); new_line;
    put(" <L>IST THE WORKING DIRECTORY ");
    put("(<S will stop scroll."); new_line;
    put(" <C>HANGE THE WORKING DIRECTOY");new_line;
    put(" <Q>UIT FTP "); new_line;
  loop
    put("OPTION: ");
    str := get_line();
    exit when (length(str) > 0);
  end loop;
  chr := str(1);
  put(" ");
  case chr is
    when 'S'!'s' => put("SEND"); new_line;
      return stor;
    when 'G'!'g' => put("GET"); new_line;
      return retr;
    when 'D'!'d' => put("DELETE"); new_line;
      return dele;
    when 'L'!'l' => put("LIST"); new_line;
      return nlst;
    when 'C'!'c' => put("CHANGE"); new_line;
      return cwd;
    when 'H'!'h' => put("HELP"); new_line;
      return help;
    when 'Q'!'q' => put("QUIT"); new_line;
      return quit;
    when others => valid := false; new_line;
      put("THE ONLY VALID OPTIONS ARE: S, G, D, L, C, H AND Q.");
      new_line; new_line; put("PLEASE REENTER."); new_line;
  end case;
  exit when valid;
end loop;
end getopt;

function get password return string is
  -- AUTHOR: ALEC YASINSAC
  -- DATE: APRIL 1986
  -- DESCRIPTION: THIS PROCEDURE PROMPTS THE USER TO ENTER
  -- A VALID PASSWORD AND RETURNS THE ENTERED STRING.
  -- EXTERNAL CALLS TO: 1. IO.GET_LINE.

  246
use asmlib, io, strlib;
goodpw: boolean;
i : integer;
byt: byte;
pw: string;
cntl_rt_brack: constant byte := byte(16#1d#);

begin
  loop
    goodpw := true;
    put("ENTER YOUR PASSWORD ");
    put("[no special characters].");
    new_line;
    put("PASSWORD: ");
    pw := "";
    i := 0;
    loop
      i := i + 1;
      byt := no_echo();
      case byt is
        when byte(16#0d#) =>
          if (i > 1) then
            exit;
          else goodpw := false;
            exit;
          end if;
        when byte(65)..byte(90) => --A..Z
          pw := insert(pw, char_to_str(byte_to_chr(byt)), l);
        when byte(97)..byte(122) => --a..Z
          pw := insert(pw, char_to_str(byte_to_chr(byt)), l);
        when cntl_rt_brack =>
          return "";
        when others => goodpw := false;
          exit;
      end case;
    end loop; new_line;--END ONE TRY AT ENTERING A PASSWORD.
    exit when goodpw;
  end loop;
  return pw;
end get_password;

function get_username return string is
  -- AUTHOR: ALEC YASINSAC
  -- DATE: APRIL 1986
  -- DESCRIPTION: THIS PROCEDURE PROMPTS THE USER TO ENTER
  -- A VALID USER ID AND RETURNS THE ENTERED STRING.
  -- EXTERNAL CALLS TO: 1. IO.GET_LINE.
  -- 2. STRLIB.LENGTH.

  use io, asmlib, strlib;
goodname: boolean;

  end get_username;
username: string;
byt: byte;
cntl rt brack: constant byte := byte(16#ld#);

begin
  loop
    goodname := true;
    loop
      put("USER NAME: ");
      username := get_line();
      exit when (length(username) > 0);
    end loop;
    for i in 1..length(username) loop
      byt := conv byt(username(i));
      case byt is
        when byte(65)..byte(90) =>
          A-->Z null;
        when byte(97)..byte(122) =>
          a-->z null;
        when cntl rt brack =>
          return "";
        when others =>
          goodname := false;
          exit;
      end case;
    end loop;
  end loop;
  exit when goodname;
end loop;
return username;
end getusername;

function getportnum return string is
  --DESCRIPTION: THIS PROC ISSUES REQUEST TO THE 8612 ASKING
  --FOR A NEW TCB TABLE TO BE EST AND NEW PORT NUMBER
  --ASSIGNED. GET PORTNUM THEN READS NEW PORT NUMBER AND
  --CONVERTS IT INTO A STRING THAT CAN BE TRANSMITTED AS
  --THE PARAMETER TO THE FTP PORT CMD.
  --EXTERNAL CALLS TO: 1. BIT.OUTPORT/TSTBIT/INPORT.
  --          2. ASMB.BYTE TO CHAR.
  use bit, strlib, asmlib, typpkg;
  byt: byte;
  byt_arr: byte array;
  int, i, j, timer, amt : integer;
  portnum, coma, str: string;

begin --BEGIN PROCEDURE GET_PORTNUM.
  put("in get_portnum"); --stub.
  outport(aux_cmd_prtere, clr);
  byt_arr(1) := byte(0);
  outport(aux_data_prtere, code_getcpad);--REQUEST TCP ADDRESS.
  timer := 0;

  248
loop --WAIT FOR DATA RECEIVE READY.
if checkterm() then
    return "";
end if;
inport(aux_stat_prt,byt);
if tstbit(integer(byt),rxrdy) then
    inport(aux_data_prt,byt);
case byt is
    when code_getcpad => exit;
    when code_open => put("sending port w/ sec act");
        return "";
    when code_cls ! code_abort => return "";
    when others => null;
        @put("control code=");put(integer(byt));new_line;
        --outport(aux_data_prt,code_getcpad); --stub.
        outport(aux_cmd_prt,clr);
end case;
end if;
end loop;

loop --WAIT FOR DATA SET READY.
    amt := 513;
inport(aux_stat_prt,byt);
if tstbit(integer(byt),dsr) then
    get_trns(byt_arr'address,aux_data_prt,amt);
    exit when amt > 0; --$$$
end if;
end loop;
if byt_arr(l) = code_addr then
    j := 1;-- POINTER FOR BYT_ARR. BYPASS THE CONTROL BYTE.
    portnum := "";
    coma := ",";
loop
    --CONVERT BYTES FROM CONCENTRATOR INTO INTEGERS
    --AND THEN INTO A STRING WITH COMMAS.
    j := j + 1;
    int := integer(byt_arr(j));
    str := int_to_str(int);
    portnum := insert(portnum,str,1);
    exit when j = 7; --ADDRESS IS SIX BYTES LONG.
    portnum := insert(portnum,coma,1);
end loop;
else --THE PROCESS IS OUT OF SYNC. REVERT TO
    null; --USER OPTIONS. LEAVE PORTNUM AS ALL BLANKS.
    put("BAD PORT NUMBER FROM CONCENTRATOR. ABORTING.");
    new_line;
end if;
return portnum;
end get_portnum;

function get_filename return string is
    --AUTHOR: ALEC YASINSAC    DATE: APRIL 86

-----------------------------
--DESCRIPTION: THIS PROCEDURE PROMPTS THE USER TO ENTER
-- A VALID FILE NAME AND RETURNS THE ENTERED STRING.
--EXTERNAL CALLS TO: 1. IO.GET_LINE.
--          2. STRLIB.INSERT/LENGTH
--          3. ASMLIB.GETCH

use asmlib, strlib, io;
i, name_len, ext_len, ctr, strlen: integer;
good_name, has_ext, got_colon: boolean;
instring, filename: string;
temp: file;
byt: byte;
begin
  --BEGIN FUNCTION GET_FILENAME.
  loop
  --LOOP UNTIL GOOD_NAME.
    loop
      put("FILENAME: ");
instring := get_line();
      exit when length(instring) > 1;
    end loop;
good_name := true;
has_ext := false;
got_colon := false;
name_len := 0;
ext_len := 0;
i := 0;
filename := ";
loop
  --LOOP TO CHECK THE DRIVE DESIGNATOR AND NAME.
  i := i + 1;
  byt := conv_byt(instring(i));
  case byt is
    when byte(97)..byte(122) ! -- a..z
      byte(65)..byte(90) ! -- A..Z
      byte(48)..byte(57) => -- 0..9
        if name_len < 8 then
          filename := insert(filename,
                        char_to_str(instring(i)), 1);
          name_len := name_len + 1;
        else
          good_name := false;
          put("FILE NAME TOO LONG."); new-line;
        end if;
    when byte(32) => -- space
      if name_len = 0 then
        null; -- SKIP LEADING SPACES.
      else
        exit;
      end if;
    when byte(58) => -- colon (:)
      if (not got_colon) and (name_len = 1) then
        name_len := name_len - 1;
        filename := insert(filename, char_to_str(':'), 1);
        got_colon := true;
  end case;
end loop;
  if good_name then
    put(filename);
  else
    put("FILENAME NOT VALID.");
    new-line;
  end if;
end loop;
end --BEGIN FUNCTION GET_FILENAME.
else
  good_name := false;
  put("ONLY ONE COLON ALLOWED."); new_line;
end if;
when byte (46) => --period (.)
  filename := insert(filename,char_to_str('.'),1);
  has_ext := true;
when byte(16#'ld#) => return "";
when others => good_name := false;
  put("CONTROL CHARACTERS NOT ALLOWED.");new_line;
end case;
end loop;--END LOOP TO CHECK THE DRIVE DESIG AND NAME.
if name_len = 0 then
  good_name := false;
else
  if has_ext then
    loop
      exit when (ext_len > 2) or
        not good_name or (i = length(instring));
      i := i + 1;
      case instring(i) is
        when 'a'..'z' ! 'A'..'Z' ! '0'..'9' =>
          filename := insert(filename,
            char_to_str(instring(i)), 1);
          ext_len := ext_len + 1;
        when '.' => ext_len := 3;
        when others => good_name := false;
          put("UNIDENTIFIED CHARACTERS IN EXTENSION.");
          new_line;
      end case;
    end loop;
    --END IF HAS_EXT.
  end if;  
end if;
exit when good_name;
end loop;
return filename;
end get_filename;

--------------------------------------------------------------------------------
function get_parameter (opt: in cmd_typ) return string is
--DESCRIPTION: USER_OPTIOS ATTACHES THE PARAMETER TO THE
--OPTION SELECTED.
--EXTERNAL CALLS TO:
--  1. FUNCS.GET_FILENAME/GET_PASSWORD/GET_USERNAME.
use io;
  parm, dirname, remname, locname: string;
begin
  case opt is
    when nlst => parm := "";

when cwd =>
    put("ENTER THE REMOTE DIRECTORY NAME. "); new_line;
    parm := get_line();
when dele =>
    put("ENTER THE NAME OF THE REMOTE FILE TO DELETE. ");
    new_line;
    put("FILE NAME: ");
    parm := get_line();
when pass =>
    new_line;
    parm := get_password();
when port =>
    new_line;
    parm := get_portnum();
when retr =>
    put("ENTER THE NAME OF THE REMOTE FILE TO RETRIEVE. ");
    new_line;
    put("FILE NAME: ");
    parm := get_line();
when stor =>
    put("ENTER THE REMOTE FILE NAME TO STORE IN TO. ");
    new_line;
    put("FILE NAME: ");
    parm := get_line();
when user =>
    new_line;
    parm := get_username();
    when others => parm := ""
end case;
return parm;
end get_parameter;

end funcs;

with typpkg;
package lib1 is
    --WRITTEN FOR Z100 UNDER ZDOS
    use typpkg;
    procedure send_cmd(cmd: in out cmd_typ;
        parameter: in string);
    procedure user_options(opt: out cmd_typ);
    procedure get_dataline(dataline: out byte_array;
        ctr: out integer);
    procedure make_reply(dataline: in byte_array;
        ctr: in integer;
        reply: out integer; parameter: out string);
    procedure process_reply(reply: in integer;
        parm: in string; state: in out cmd_typ);
end lib1;
with asmlib, blkio, func$s, io, strlib, bit;
package body libl is
use typpkg;

-----------------------------------------------
procedure send_cmd(cmd: in out cmd_typ;
parameter: in string) is

--AUTHOR: ALEC YASINSAC
--DATE: APRIL 1986
--DESCRIPTION: SEND COMMAND CALLS INTERNAL PROC 'CONVERT'
-- TO CONVERT THE ENUMERATED TYPE "CMD" INTO A STRING AND
-- SENDS THE STRING WITH ITS PARAMETER OUT THE SERIAL PORT.
-- IF THE COMMAND CANNOT BE SENT OR THE USER TERMINATES,
-- CMD WILL BE SET TO ABORT FTP. OTHERWISE, CMD IS NOT
-- MODIFIED.
--EXTERNAL CALLS TO: 1. BIT.INPORT/OUTPORT/TSTBIT.
-- 2. STRLIB.LENGTH.
-- 3. IO.WRITE/OPEN/CLOSE.
-- 4. ASMB.BYTE_TO_CHAR.

use typpkg, asmlib, io, strlib, bit;
byt: byte;
cmdline, cmdstr: string;
suffix: string(2);
addr, len: integer;
chr: character;
timer: integer := 0;
timeout: constant integer := 500;
-----------------------------------------------
procedure convert(cmd: in cmd_typ; cmdstr: out string) is

--DESCRIPTION: CONVERT CONVERTS THE ENUMERATED TYPE "CMD" INTO A STRING.

begin
  case cmd is
    when abor =>
      cmdstr := "abor";
    when cwd =>
      cmdstr := "cwd ";
    when dele =>
      cmdstr := "dele ";
    when help =>
      cmdstr := "help";
    when nul =>
      cmdstr := "noop";
    when pass =>
      cmdstr := "pass ";--PARM IS THE PASSWORD.
  end case;
end;
when pasv =>
  cmdstr := "pasv";
when port =>
  cmdstr := "port ";
when quit =>
  cmdstr := "quit";
when rein =>
  cmdstr := "rein ";--REINITIALIZE.
when rest =>
  cmdstr := "rest ";--RESET.
when retr =>
  cmdstr := "retr ";--GET A FILE.
when stat =>
  cmdstr := "stat";
when stor =>
  cmdstr := "stor ";
when user =>
  cmdstr := "user ";--PARM IS THE USER ID.
when others =>
  cmdstr := "noop";
  put("ERROR OCCURRED. CMD NOT RECOGNIZED.");
  new_line;
end case;
end convert;

------------------------------------------------------------------
begin -- BEGIN PROCEDURE SEND COMMAND.
  @put("send cmd"); new_line;
  convert(cmd,cmdstr);
  cmdline := insert(cmdstr,parameter,1);--ATT CMD TO PARM.
  suffix := "bb";
  suffix(1) := byte_to_chr(cr);
  suffix(2) := byte_to_chr(lf);
  cmdline := insert(cmdline,suffix,1);--ATTACH CARRIAGE
     --RETURN AND LINE FEED TO THE COMMAND STRING.
  suffix := "b";
  suffix(1) := byte_to_chr(code_cmd);
  cmdline := insert(suffix,cmdline,1);
loop
  import(aux_stat_prt,byter);--WAIT UNTIL DSR GOES LOW.
  if not tstbit(integer(byter),dsr) then
  --THE FIRST BYTE OF A STRING IS ITS LENGTH, ADD ONE
  --TO THE ADDRESS OF THE STRING TO START AT THE FIRST
  --BYTE OF THE MESSAGE.
    addr := cmdline'address + 1;
    len := length(cmdline);
    send_trns(addr, aux_data_prt, len);
    exit when len = 0;
    timer := timer + 1;
  end if;
  @put("no cmd sent. cmd = ");put(cmdline);new_line;
end loop;

254
if timer > timeout then
    cmd := abort;
    put("NO RESPONSE FROM CONCENTRATOR."); new_line;
    exit;
else
    timer := timer + 1;
end if;
end loop;
@put("cmd = ");put(cmdline); put("$$");new_line;
@put("length = ");put(length(cmdline));new_line;
end send_cmd;

procedure user_options(opt: out cmd_typ) is
    -- AUTHOR: ALEC YASINSAC DATE: MAY 86
    -- OUTPUT: THE COMMAND THAT THIS PROCEDURE TRANSMITTED IS
    -- IDENTIFIED BY THE OUT PARAMETER.
    -- DESCRIPTION: USER OPTIONS IS CALLED WHEN ACTION IS
    -- ON ALL PREVIOUS COMMANDS. IT IS EXPECTED THAT IF THIS
    -- PROCEDURE IS CALLED, THE USER IS LOGGED IN TO THE
    -- SYSTEM. FROM HERE, THE USER CAN REQUEST A FILE
    -- TRANSFER, CHANGE DIRECTORY ON THE REMOTE HOST, LIST THE
    -- DIRECTORY ON THE REMOTE HOST, OR TERMINATE THE PROCESS.
    -- THE USER_OPTIONS PROCEDURE ALSO OPENS AND CLOSES LOCPF
    -- FOR RETRIEVING OR SENDING DATA TO/FROM THE REMOTE HOST.
    -- EXTERNAL CALLS TO: 1. IO.OPEN/CLOSE/CREATE/DELETE.
    use io, funcs;
    filename, parameter: string;
    got_opt: boolean;
begin
    if is_open(typpkg.locfile) then
        close(typpkg.locfile);
    end if;
    got_opt := false;
    opt := get_opt();
    case opt is
        when retr =>
            put("ENTER THE LOCAL DESTINATION FILE NAME.");
            new_line;
            filename := get_filename();
            purge(filename); -- PURGE WILL NOT ABORT IF THE
            create(typpkg.locfile,filename,write_only);
        when stor =>
            loop
                put("ENTER THE LOCAL SOURCE FILE NAME. ");
                new_line;
                filename := get_filename();
                open(typpkg.locfile,filename,read_only);
                exit when ioreresult /= 255;
                put("CANNOT OPEN FILE ");put(filename);put(".");
                new_line;
                255
end loop;
when others => null;
end case;
parameter := get_parameter(opt);
send_cmd(opt, parameter);
end user_options;

-----------------------------------------------

procedure get_dataline(dataline: out byte_array;
ctr: out integer) is

-- AUTHOR: ALEC YASINSAC
-- DATE: APRIL 1986
-- DESCRIPTION: PROCEDURE GET_DATALINE DOES THE NECESSARY
-- HANDSHAKING WITH THE 8612 AND READS ANY DATA OR CONTROL
-- CHARACTER IS COMING UP FROM THE FOREIGN SITE. OUTPUT IS
-- THE DATA AND THE BYTE COUNT. CONTROL INFO IS PASSED
-- BACK AS THE FIRST CHARACTER OF THE DATALINE.
-- EXTERNAL CALLS TO: 1. BIT.INPORT/OUTPORT/TSTBIT.
-- 2. ASMB.GET_TRNS.
-- 3. IO.KEYPRESS.

use typpkg, asmlib, io, bit;
tnl Chr_rec : boolean;
max wait: constant integer := 30000;
i : integer;
byt: byte;
inline : string;
ctr: integer;

begin
  _ _
@put("in get_dataline"); newline;
outport(aux dataprt,clr);
ct1 Chr_rec := false;
ctr := 0; cntr := 0; i := 0;
loop--WAIT FOR KEYPRESS,TIMEOUT,CONTROL CHARACTER,OR DSR.
  if func1.checkterm() then --CHECKS FOR ^].
    dataline(l) := code_cls;
    return;
  end if; --END IF CHECKTERM.
  cntr := cntr + 1; --TEST FOR TIMEOUT.
  if cntr > max_wait then
    @put("time-wait in get_dataline."); new_line;
    outport(aux_data_prt, code_check_replyq);
    -- ACTS AS AN 'ARE YOU THERE' REQUEST.
    cntr := 0;
  end if; --END IF TIMEWAIT.
  inport(aux stat_prt,byt);
  if tspbit(integer(byt),rxrdy) then--TEST FOR CNTL CHR.
    inport(aux_data_prt,byt);
    @put("got cntl chr"); new_line;
    ctr := 1;
case byt is
  when code cls ! code abort =>
    dataline(l) := code_abort;
  end case;
if isopen(typpkg.locfile) then
  close(typpkg.locfile);
end if;
exit; --EXIT LOOP TO GET DATA FROM AUX PORT.
when code_open => dataline(1) := code_open;
when code_closdata=> dataline(1):=code_closdata;
when code_qempty => dataline(1) := code_null;
when others => ctr := 0;
@put("don't recognize control character = ");
@put(integer(byt)); new_line;
loop --CLEAR AUXILLARY PORT.
  inport(aux_stat_prt,byt);
  exit when not tstbit(integer(byt),rxrdy);
  inport(aux_data_prt,byt);
  @prntdata(byt);
end loop;
end case;
end if; --END IF RXRDY.
inport(aux_stat_prt,byt);
if tstbit(Integer(byt),dsr) then --TEST FOR DSR.
  ctr := 513 ;
  get_trns(dataline'ADDRESS, aux_data_prt,ctr);
  @put("dsr");
  if ctr > 0 then
    exit;
  end if;
  end if; --END IF TSTBIT FOR DSR.
end loop;--ENDS LOOP WAITING FOR BYTES FM CONCENTRATOR.
end get_dataline;

-----------------------------------------------
procedure make_reply(dataline:in byte_array;ctr:in integer;
  reply: out integer; parameter: out string) is
  -- CURRENT: 3 MAY 1986
  -- AUTHOR: ALEC YASINSAC DATE: MAY 1986
  -- DESCRIPTION: PROCEDURE GET_DATALINE DOES THE NECESSARY
  -- HANDSHAKING WITH THE 8612 AND READS ANY DATA OR CONTROL
  -- CHARACTER IS COMING UP FROM THE FOREIGN SITE.
  --EXTERNAL CALLS TO: 1. BIT.INPORT/OUTPORT/TSTBIT.
  -- 2. ASMB.PRNTDATA/BYTE_TO_CHR.
  -- 3. INSERT.STRILIB.

  use strlib, typpkg, asmlib, bit;
  len, j : integer;
  chr : character;
  rep : string;

begin
  @put("in make_reply");
  len := ctr;
  parameter := "";
  for j in 2..len loop
    prntdata(dataline(j)); --DISPLAY REPLY ON SCREEN.
rep := " ";
if ((j > 5) and (j < 80)) then
    chr := byte_to_char(dataline(j));
    rep(1) := chr; -- TEMP STORAGE.
    parameter := insert(rep, parameter, 1);
end if;
end loop;
rep := " ";
for j in 1..3 loop
    rep(j) := byte_to_char(dataline(j + 1));
    if not (rep(j) in '0'..'9') then
        -- NOT A REPLY. COULD BE A HELP MSG.
        reply := 0;
        parameter := " ";
        @put("not a.reply "); put(rep); new_line;
        return;
    end if;
end loop;
reply := str_to_int(rep);
end make_reply;

---------------------------------------------------------------------
procedure get_data(lst_cmd: in out cmd_typ) is
    -- AUTHOR: ALEC YASINSAC DATE: APRIL 86
    -- INPUT: 1. LST_CMD IS THE LAST COMMAND THAT WAS SENT.
    -- DESCRIPTION:
    -- Get data calls get_trns to receive an expected data
    -- transfer from the concentrator. If the transfer is not
    -- received after ten tries, a code is sent asking for
    -- status of the data connection.
    -- EXTERNAL CALLS TO: 1. IO.WRITE.
    -- 2. ASMLIB.PRNTDATA.
    use typpkg, asmlib, io, strlib, bit;
    reply, amt, strlen: integer;
    byt: byte;
    parameter: string;
    timer, ctr: integer;
    dataline: byte_array;
begin
    @put("in get_data"); new_line;
    timer := 0;
    loop
        if funcs.checkterm() then
            lst_cmd := abor;
            return;
        end if;
        inport(aux_stat_prt, byt);
        if tstbit(integer(byt), dsr) then
            ctr := 513;
            get_trns(dataline'ADDRESS, aux_data_prt, ctr);
            if ctr > 0 then
                case dataline(1) is

258
when code_data =>
  if ((is_open(typpkg.locfile)) and (ctr > 1)) then
    for j in 2..ctr loop
      write(locfile,dataline(j));
      @prntdata(dataline(j));
      --LOCFILE IS OPENED IN USER_OPTIONS
      --WHEN THE RETR COMMAND IS SENT.
    end loop; new_line;
    @put("DATA RECEIVED ctr = "); put(ctr);
  else
    for j in 2..ctr loop--DISPLAY ON CONS.
      prntdata(dataline(j));
    end loop; new_line;
    end if; --END IF IS_OPEN.
  when code_reply =>--REPLY HERE OUT OF ORDER.
    make_reply(dataline, ctr, reply, parameter);
    case reply is
      when 221 ! 421 => lst_cmd := abor;
      when others => null;
    end case;
  when others =>
    for j in 2..ctr loop--DISPLAY ON CONS.
      prntdata(dataline(j));
    end loop; new_line;
  end case; --END CASE DATALINE(1) IS.
  end if; --END IF CTR > 0;
else
  if tstbit(integer(byt), rxrdy) then
    inport(aux_data_prt, byt);
    case byt is
      when code_closdata =>
        if is_open(typpkg.locfile) then
          close(typpkg.locfile);
        end if;
        @put("data connection is closed");
        exit;
      when code_open => null; --KEEP WAITING.
      when code_cls => lst_cmd := abor;
        exit;
      when code_abort => lst_cmd := abor;
        exit;
      when others => null;
    end case;
  else
    if timer > 3 then timer := 0;
      outport(aux_data_prt, code_dprtstat);
    else
      timer := timer + 1;
    end if;
  end if; --END IF RXRDY.
end if; --END IF DSR.

259
end loop;
if is_open(typpkg.locfile) then
  close(typpkg.locfile);
end if;
end get_data; --ENDS PROCEDURE PROCESS_DATA.

procedure senddata(ist cmd: in out cmd_typ) is
  use typpkg, io, asmlib, bit, funcs;
time_wait_exceeded: boolean := false;
len, i, amt, ctr: integer;
byt: byte;
byt_arr: byte array;
is_text: boolean;
begin

  len := 0;
  loop --MAKE SURE REMOTE SERVER IS READY TO RECEIVE.
    outport(aux_data_prt,code_dprtstat);
    i := 0;
    loop --WAIT FOR CODE RETURNED FROM CONCENTRATOR.
      if funcs.checkterm() then --CHECKS FOR ^].
        lst_cmd := abor;
        return;
      end if; --END IF CHECKTERM.
      i := i + 1;
      inport(aux_stat_prt,byt);
      exit when tstbit(integer(byt),rxrdy);
      if i > 3000 then --CONCENTRATOR IN LOOP.
        time_wait_exceeded := true;
        exit;
      end if;
    end loop;
  end loop;
inport(aux_data_prt,byt);
if len > 1000 then
  time_wait_exceeded := true;
else
  len := len + 1;
end if;
exit when ((byt = code_open) or time_wait_exceeded);
end loop; --END LOOP CHECKING DATA CONNECTION.
if byt = code_open then --SEND WHOLE FILE TO 8612.
  put("IS_FILE TO BE TRANSFERRED A TEXT FILE?");
  put(" (y/n): ");
  loop --DETERMINE IF TEXT FILE.
    if keypress() then
      getch(byt); new_line;
      case byt is
        when byte(16#1d#) =>
          lst_cmd := quit;
          send_cmd(lst_cmd,"");
          return;
      end when;
  end loop;
when byte(89) ! byte(121)==>  --Y,y
  is_text := true;
  exit;
when byte(78) ! byte(110)==>  --N,n
  is_text := false;
  exit;
when others => new_line;
  put(" (y/n): ");
end case; new_line;
end if;
end loop;

loop  --SEND AS MANY PACKETS AS REQUIRED.
@new line; put("data to be sent = ");
byt_arr(1) := code_data;
len := 1;
loop  --STORE FILE DATA IN MEMORY READY TO SEND.
  len := len + 1;
  read(typpkg.locfile,byt_arr(len));
  @printdata(byt_arr(len));
  exit when len > 511;
  if is_text then
    exit when end_of_file(typpkg.locfile);
  else
    exit when eof(typpkg.locfile);
  end if;
end loop;
@new line;
@put("num chrs = "); put(len);
import(aux_stat_prt,byt);
if tstbit(integer(byt),rxrdy) then--GOT CTL CHR
  import(aux_data_prt,byt);--FRM CONCENTRATOR.
case byt is
  when code_cls!code_abort =>
    put("ABORTED BY REMOTE HOST.");
    lst_cmd := null; new_line;
    return;
  when others => null;
end case;
end if;
end loop
amt := len;
send_trns(byt_arr'address,aux_data_prt,amt);
exit when amt = 0;
if keypress() then
  getch(byt);
  case byt is
    when byte(16#1d#)==> lst_cmd := quit;
      send_cmd(lst_cmd,"");
      return;
    when others => null;
  end case;
end if;  --END IF KEYPRESS.
end loop; --END LOOP WAITING FOR LOW DSR.
@put("packet sent"); new_line;
if is text then
    exit when end_of_file(typpkg.locfile);
else
    exit when eof(typpkg.locfile);
end if;
end loop;--END LOOP TO SND WHOLE FILE TO REM HOST.
@put("end of file reached"); new_line;
outport(aux_data prt,code_closdata);
close(typpkg.locfile);

else --COULD NOT OPEN DATA CONNECTION.
null; -- THE DATA CONNECTION COULD NOT BE
--OPENED. AN FTP REPLY SHOULD BE COMING.
end if; --END IF BYT = CODE_OPEN.
end send_data;

------------------------------------------------------------------------------------------------------------------
procedure process_reply(reply: in integer; parm: in string;
state: in out cmd typ) is

--AUTHOR: ALEC YASINSAC  DATE: APRIL 1986
--DESCRIPTION: PROCESS_REPLY USES THE INPUT PARAMETER
-- 'REPLY' TO DETERMINE THE COURSE OF ACTION FOR THE
-- SYSTEM TO TAKE. 'REPLY' IS THE FTP REPLY THAT A FOREIGN
-- SITE HAS GENERATED IN RESPONSE TO AN FTP COMMAND THAT
-- ORIGINATED IN THIS MACHINE. POSSIBLE ACTIONS INCLUDE
-- (but are not limited to) TRIGGERING A DATA
-- TRANSFER, REISSUING A COMMAND, AND CLOSING A CONNECTION.
-- ANY REPLY THAT IS NOT A REPLY THAT CAN BE TRIGGERED BY
-- THE COMMAND 'STATE' IS IGNORED. OFTEN, A NOOP COMMAND
-- IS SENT WHEN THE REMOTE HOST IS LIKELY TO SEND A SECOND
-- REPLY TO THE PREVIOUS COMMAND. THE SECOND REPLY WILL BE
-- DISPLAYED BEFORE THE REPLY TO 'NOOP' IS PROCESSED.
--EXTERNAL SUBROUTINES:
-- 1. FUNCS.GETFILENAME/GET_PARAMETER/GET_PORTNUM/
--      GET_PASSWORD/GET_OPT/GET_USERNAME/SEND_CMD.

use typpkg, io, funcs, strlib;
parameter: string;

begin  --PROCESS_REPLY.
@put("In process reply."); new_line;
case state is
    when acct =>
        case reply is
            when 202 => state := noop;
                send_cmd(state,""");
            when 230 =>
                parameter := get_portnum();
                if parameter = "" then
                    state := quit;
                else
                    -- do something
                end if;
            when others =>
            end if;
        end case;
    when others =>
        -- handle other states
    end when;
end case;
end process_reply;
state := port;
send_cmd(state, parameter);
end if;
when 421 => state := abor;
when 500 ! 501 =>
  put("ENTER YOUR ACCOUNT NUMBER: ");
  parameter := get_line();
  state := acct;
  send_cmd(state, parameter);
when 530 => parameter := get_username();
  if parameter = "" then
    state := abor;
  else
    state := user;
    send_cmd(state, parameter);
  end if;
  when others => null;
end case;
-- End of 'when acct'.

when cwd ! dele =>
case reply is
  when 200!250 => state := noop;
    send_cmd(state,"" );
  when 421 => state := abor;
  when 500 ! 501 ! 502 => state := noop;
    send_cmd(state,"" );
  when others => null;
end case;
-- END WHEN CWD ! DELE.

when help =>
case reply is
  when 211 ! 214 ! 500..502 => state := noop;
    send_cmd(state,"" );--2ND REPLY MAY FOLLOW
  when others => null; -- THE HELP COMMAND.
end case;

--when nlst => see when retr.

when noop ! port =>
case reply is
  when 200 => user_options(state);
  when 421 => state := abor;
  when 426 =>
    parameter := get_portnum();
    if parameter = "" then
      state := quit;
    else
      state := port;
      send_cmd(state,parameter);
    end if;
  end if;
when 500!501 => null;--THIS SYSTEM WILL NOT SEND
--AN INVALID PORT COMMAND OR PARAMETER.
    when others => null;
end case;
--END CASE NOOP!PORT.

when null =>
  -- NULL IS THE START STATE.
case reply is
  when 220 ! 530 =>
    parameter := get parameter(user);
    if parameter = "" then
      state := abor;
    else
      state := user;
      send_cmd(state,parameter);
    end if;
  when 221 ! 421 => state := abor;
  when others => null;
end case;

when port =>
  case reply is
    when 230 =>
      parameter := get portnum();
      if parameter = "" then
        state := quit;
      else
        state := port;
        send_cmd(state,parameter);
      end if;
    when 332 =>
      put("ENTER YOUR ACCOUNT NUMBER: ");
      parameter := get_line();
      state := acct;
      send_cmd(state,parameter);
    when 421 => state := abor;
    when 500 ! 501 => null;
    -- ASSUME THIS SYSTEM CANNOT SEND BAD PASSWORD.
    when 530 =>
      parameter := get parameter(user);
      if parameter = "" then
        state := abor;
      else
        state := user;
        send_cmd(state,parameter);
      end if;
    when others => null;
  end case;
--End of 'when pass'.

--when port => SEE WHEN NOOP.

264
when quit =>
  case reply is
    when 221 ! 421 => state := abor;
    when others => null;
  end case;
--END WHEN QUIT.
when retr ! nlst =>
  case reply is
    when 110 ! 125 ! 150 => null;--Wait for another reply.
    when 221 ! 421 => state := abor;
    when 226 => get_data(state);--CAN CHANGE STATE.
      case state is
        when retr ! nlst =>
          parameter := get_portnum();
          if parameter = "" then
            state := quit;
          else
            state := port;
            send_cmd(state,parameter);
          end if;
        when abor => null;
        when others => state := abor;
          @put("in process_reply.Bad state.");
      end case;
    when 250 => get_data(state);
      case state is
        when retr ! nlst => state := noop;
          send_cmd(state,"");
        when abor => null;
        when others => state := abor;
          @put("in process_reply.Bad state.");
      end case;
    when 425 ! 426 =>
      parameter := get_portnum();
      if parameter = "" then
        state := quit;
      else
        state := port;
        send_cmd(state,parameter);
      end if;
    when 450 ! 451 ! 500 ! 501 ! 550 =>
      state := noop;
      send_cmd(state,"");
    when others => null;
  end case;
--END WHEN 'RETR'.
when stor =>
  case reply is
    when 125 ! 150 => send_data(state);
      if state = stor then
state := noop;
send_cmd(state,"");
end if;
when 221 ! 421 => state := abor;
when 226 =>
    parameter := get_portnum();
    if parameter = "" then
        state := quit;
    else
        state := port;
        send_cmd(state,parameter);
    end if;
when 250 => state := noop;
send_cmd(state,"");
when 425!426 =>
    parameter := get_portnum();
    if parameter = "" then
        state := quit;
    else
        state := port;
        send_cmd(state,parameter);
    end if;
when 450!451!452!500!501 => state := noop;
send_cmd(state,"");
when 532 =>
    put("ENTER YOUR ACCOUNT NUMBER: ");
    parameter := get_line();
    state := acct;
    send_cmd(state,parameter);
    if not (state = abor) then
        state := stor;
    end if;
when 552 ! 553 => state := noop;
send_cmd(state,"");
when others => null;
end case;
--END WHEN 'STOR'.

when user =>
    case reply is
when 230 =>
    parameter := get_portnum();
    if parameter = "" then
        state := quit;
    else
        state := port;
        send_cmd(state,parameter);
    end if;
when 331 =>
    parameter := get_password();
    state := pass;
    send_cmd(state,parameter);
when 332 =>
    put("ENTER YOUR ACCOUNT NUMBER: ");
    parameter := get_line();
    state := acct;
    send_cmd(state,parameter);
when 421 => state := abor;
when 500!501 =>null;--CANNOT SEND BAD USER CMD.
when 530 =>parameter := get_parameter(user);
    if parameter = "" then
        state := abor;
    else
        state := user;
        send_cmd(state,parameter);
    end if;
when others => null;
end case;
--END WHEN USER.
when others => put("bad state in process_reply");
    state := noop;
    send_cmd(state,"");
end case;
end process_reply;
end lib1;
with funcs, asmlib, lib1, typpkg, bit, io, strlib, get_ip;
-------------------

procedure ftp is
--AUTHOR: ALEC YASINSAC APRIL 1986
--CONFIGURATION: THIS PROGRAM IS WRITTEN TO RUN ON A Z-100
-- OPERATING UNDER Z-DOS.
--DESCRIPTION: THIS PROCEDURE DRIVES THE REMOTE FILE
-- TRANSFER PROCESS ON THE NPS AEGIS LOCAL AREA NEWORK.
-- THE USER IS PROMPTED TO SELECT HIS DESIRED DESTINATION
-- AND AN FTP COMMAND CONNECTION IS ESTABLISHED. THE
-- PROCESS THEN BECOMES A CYCLE OF SENDING COMMANDS AND
-- PROCESSING REPLIES. THE CYCLE ENDS WHEN THE USER
-- ENTERS QUIT AND THE QUIT COMMAND IS SENT.

use asmlib, funcs, lib1, bit, typpkg, io, strlib, get_ip;
ip: array (1..4) of integer;
byts: array (1..4) of byte;
auxfile, host: file;
byte, org_ocwl: byte;
opt : cmd_typ;
cnt_exit: BOOLEAN;
reply, wait, ctr, tst: integer;
parameter, portnum: string;
dataline: byte array; --512 bytes.
more_replys, stopit: boolean;

begin --BEGIN PROCEDURE FTP.
inport(ocwl_reg,org_ocwl);
outport(ocwl_reg,ocwl);
clrscreen;
outport(aux_cmd_prt,clr);
put("WELCOME TO THE MULTIUSER SYTSTEM ");
put("FILE TRANSFER PROCESS (FTP). ");
new_line;
open(host,"hosts.fil",read_only);
if (ioresult = 255) then
  close(host);
  put("FILE HOSTS.FIL DOES NOT EXIST."); NEW_LINE;
else
  close(host);
  open(auxfile,"aux",read_write);
  get_addr(ip(1),ip(2),ip(3),ip(4));
    --HAVE THE USER SELECT THE REMOTE ADDR.
  stopit := true;
  for i in 1..4 loop
    byts(i) := byte(ip(i));
    if ip(i) /= 0 then
      stopit := false;
    end if;
  end loop;
  if not stopit then

268
@new_line; put("address = ");
@for i in 1..4 loop put(ip(i)); put("\n"); end loop;
-- MUST SEND AND RECEIVE CODE_FTP BEFORE PROCEEDING.
put("ATTEMPTING CONNECTION WITH CONCENTRATOR");
write(auxfile,code_fpt);
wait := 0;
loop  -- WAIT FOR CHARACTER FROM 8612.
inport(aux_stat_byt);
if tstbit(INTEGER(byt),RxRdy) THEN
  inport(aux_byt);
  case byt is
    when code_fpt => clrscrn;
      put("trying..."); new_line;
      exit;
    when code_cls => write(auxfile,code_fpt);
      put("received code_cls"); new_line;
    when others => write(auxfile,code_cls);
      put("got byte other than code_fpt");
      new_line;
  end case;
else
  if checkterm() then  -- CHECK FOR ^].
    write(auxfile,code_cls);
    return;
  end if;
  wait := wait + 1;
  if wait > 32000 then
    put("time-wait"); new_line;
    wait := 0;
    write(auxfile,code_fpt);
    end if;
  end if;
end loop;
loop
  ctr := 4;
  send_trns(byts'address, aux_data_byt, ctr);
  exit when ctr = 0;
  if checkterm() then
    write(auxfile,code_cls);
    return;
  end if;
end loop;
if not stopit then
  opt := nul;
loop  -- MAIN LOOP FOR PROCESSING FTP REQUESTS.
  ctr := 0;
  get_dataline(dataline, ctr);
  case dataline(1) is
    when code_abort =>
      put("FTP TERMINATED BY REMOTE HOST.");
      if is_open(typpkg.locfile) then
        close(typpkg.locfile);
      end if;
end if;
exit;

when code_cls =>
  if opt = quit then --ONLY SEND QUIT ONCE.
    exit;
  else
    opt := quit;
send_cmd(opt,"\"");
    put("FTP TERMINATED BY USER.");
    if is_open(typpkg.locfile) then
      close(typpkg.locfile);
    end if; new_line;
  end if;
when code_data => --WILL NOT SEND FTP CMD.
  if is_open(typpkg.locfile) then
    for j in 2..ctr loop
      write(locfile,dataline(j));
      @prntdata(dataline(j));
      --LOCFILE IS OPENED IN USER OPTIONS
      --WHEN THE RETR COMMAND IS SENT.
    end loop;
  else --DISPLAY TO SCREEN.
    for j in 2..ctr loop
      prntdata(dataline(j));
    end loop; new_line;
  end if;
when code_reply =>--WILL SEND AN FTP CMD.
  make_reply(dataline, ctr, reply, parameter);
  process_reply(reply, parameter, opt);
when code_null => --NOTHING FROM REMOTE HOST.
  case opt is --BOTH ENDS WAITING.
    when acct =>
      parameter := get_parameter(acct);
send_cmd(opt, parameter);
    when cwd ! dele ! help ! noop =>
      user_options(opt);
    when pass => parameter := get_password();
send_cmd(opt,parameter);
    when port => null; --KEEP WAITING.
    when quit => opt := abor;
    when retr ! stor =>
      put("REQUEST NOT PROCESSED.");
    user_options(opt);
    when user => parameter := get_username();
send_cmd(opt,parameter);
    when others => user_options(opt);
  end case;
  when others => null;
end case;
exit when (opt = abor);
end loop; --ENDS MAIN PROCESSING LOOP.
end if; --ENDS INNER IF NOT STOPIT.
write(auxfile,code_cls);
outport(aux_cmd_prf,dtr);  --WILL CAUSE CONCENTRATOR
        --TO TERMINATE ANY TRANSIENT PROCESSES.
    end if;  --ENDS IF NOT STOPIT.
end if;  --ENDS IF IORESULT = 255.
end ftp;

.BATCH FILE IN COMPILATION ORDER

The following is the contents of the batch (.bat or .sub)
file used to compile the preceding programs.

del a:typpkg.sym
del a:asmlib.sym
del a:func$s.sym
del a:libl.sym
del a:get_ip.sym
del a:asmlib.jrl
del a:get_ip.jrl
del a:func$s.jrl
del a:libl.jrl
del a:ftp.jrl
del a:ftp.com
janus a:typpkg.spc
janus a:asmlib.spc
janus a:func$s.spc
janus a:libl.spc
janus a:get_ip.spc
jasma86 a:asmlib
janus a:get_ip/w
janus a:func$s/w
janus a:libl/w
janus a:ftp/w
jlink a:ftp
APPENDIX I

LISTING OF Z-100 LOCAL PROGRAMS

PACKAGE global is

asciinull : CONSTANT BYTE := BYTE(0);
asciilbs  : CONSTANT BYTE := BYTE(16#23#);
asciisterisk: CONSTANT BYTE := BYTE(16#2A#);
asciiperiod : CONSTANT BYTE := BYTE(16#2E#);
asciicomma : CONSTANT BYTE := BYTE(16#2C#);
asciati : CONSTANT BYTE := BYTE(16#40#);
asciicolon : CONSTANT BYTE := BYTE(16#3A#);
asciibus : CONSTANT BYTE := BYTE(16#08#);
asciitilde : CONSTANT BYTE := BYTE(16#1F#);
asciunderln : CONSTANT BYTE := BYTE(16#60#);
asciif : CONSTANT BYTE := BYTE(16#0C#);
asciicr : CONSTANT BYTE := BYTE(16#0D#);
asciilf : CONSTANT BYTE := BYTE(16#0A#);
asciicntlR : CONSTANT BYTE := BYTE(16#12#);
asciicntlQ : CONSTANT BYTE := BYTE(16#11#);
ascibell : CONSTANT BYTE := BYTE(16#07#);
ascispaces : CONSTANT BYTE := BYTE(16#20#);
asciquest : CONSTANT BYTE := BYTE(16#3F#);
asciicntlZ : CONSTANT BYTE := BYTE(16#1A#);
ascizero : CONSTANT BYTE := BYTE(16#30#);
asciine : CONSTANT BYTE := BYTE(16#39#);
ascia : CONSTANT BYTE := BYTE(16#41#);
ascia_a : CONSTANT BYTE := BYTE(16#61#);
asciib : CONSTANT BYTE := BYTE(16#42#);
asciib_c : CONSTANT BYTE := BYTE(16#62#);
asciic : CONSTANT BYTE := BYTE(16#43#);
asciic_c : CONSTANT BYTE := BYTE(16#63#);
asciid : CONSTANT BYTE := BYTE(16#44#);
asciic_a : CONSTANT BYTE := BYTE(16#64#);
asciie : CONSTANT BYTE := BYTE(16#45#);
asciif : CONSTANT BYTE := BYTE(16#46#);
asciif_c : CONSTANT BYTE := BYTE(16#66#);
asciig : CONSTANT BYTE := BYTE(16#47#);
asciig_c : CONSTANT BYTE := BYTE(16#67#);
asciit : CONSTANT BYTE := BYTE(16#49#);
asciิi : CONSTANT BYTE := BYTE(16#69#);
asciitL : CONSTANT BYTE := BYTE(16#4C#);
asciit_l : CONSTANT BYTE := BYTE(16#6C#);
asciim : CONSTANT BYTE := BYTE(16#4D#);
asciim_l : CONSTANT BYTE := BYTE(16#6D#);
asciin : CONSTANT BYTE := BYTE(16#4E#);
ascii_n : CONSTANT BYTE := BYTE(16#6E#);
ascii0 : CONSTANT BYTE := BYTE(16#4F#);
asciiP : CONSTANT BYTE := BYTE(16#50#);
ascii_P : CONSTANT BYTE := BYTE(16#70#);
asciiQ : CONSTANT BYTE := BYTE(16#51#);
ascii_q : CONSTANT BYTE := BYTE(16#71#);
asciiR : CONSTANT BYTE := BYTE(16#52#);
ascii_r : CONSTANT BYTE := BYTE(16#72#);
asciiS : CONSTANT BYTE := BYTE(16#53#);
ascii_s : CONSTANT BYTE := BYTE(16#73#);
asciiT : CONSTANT BYTE := BYTE(16#54#);
ascii_T : CONSTANT BYTE := BYTE(16#74#);
asciiU : CONSTANT BYTE := BYTE(16#55#);
ascii_u : CONSTANT BYTE := BYTE(16#75#);
asciiV : CONSTANT BYTE := BYTE(16#56#);
ascii_V : CONSTANT BYTE := BYTE(16#76#);
asciiW : CONSTANT BYTE := BYTE(16#57#);
ascii_w : CONSTANT BYTE := BYTE(16#77#);
asciiX : CONSTANT BYTE := BYTE(16#58#);
ascii_X : CONSTANT BYTE := BYTE(16#78#);
asciiZ : CONSTANT BYTE := BYTE(16#5A#);
ascii_Z : CONSTANT BYTE := BYTE(16#7A#);
ascii_DEL : CONSTANT BYTE := BYTE(16#7F#);
term : CONSTANT BYTE := BYTE(16#1D#);
ready : CONSTANT BYTE := BYTE(0);
talk : CONSTANT BYTE := asciiT;
getfile : CONSTANT BYTE := asciiG;
sendfile : CONSTANT BYTE := asciiS;
sending : CONSTANT BYTE := ascii_s;
receiving : CONSTANT BYTE := ascii_r;
repeatsnd : CONSTANT BYTE := asciiR;
unable : CONSTANT BYTE := asciiU;
filedat : CONSTANT BYTE := asciiF;
close : CONSTANT BYTE := asciiC;
whothere : CONSTANT BYTE := asciiW;
badtrns : CONSTANT BYTE := asciiB;
acklast : CONSTANT BYTE := asciiA;
Imhere : CONSTANT BYTE := asciiI;
EOF : CONSTANT BYTE := asciiE;
wait_for_ack : CONSTANT BYTE := ascii_w;
quit : CONSTANT BYTE := asciiQ;
prt_chg : CONSTANT BYTE := asciiP;
netStat : CONSTANT BYTE := asciiN;
print : CONSTANT BYTE := ascii_p;
dir : CONSTANT BYTE := asciiD;
dir_data : CONSTANT BYTE := ascii_d;
info : CONSTANT BYTE := asciiI;
log : CONSTANT BYTE := asciiL;
broadcast : CONSTANT BYTE := BYTE(16#FF#);
hdr_len : CONSTANT INTEGER := 6;
threshld : CONSTANT INTEGER := 10000;
ocwl_reg : constant integer := (16#00f3#);
ocwl : constant byte := byte(16#aa#);
code_cls : constant byte := byte(16#c3#); -- C
code_abort : constant byte := byte(16#c1#); -- A
code_status : constant byte := byte(16#d3#); -- S
code_arlog : constant byte := byte(16#d2#); -- R
code_local : constant byte := byte(16#cc#); -- L
code_lstn : constant byte := byte(16#cf#); -- O
code_estab : constant byte := byte(16#cl#); -- A
code_reqPrt : constant byte := byte(16#f0#); -- p
code_RTS : constant byte := byte(16#25#); -- T
code_print : constant byte := byte(16#d4#); -- T
code_endprint: constant byte := byte(16#f4#); -- T
dat : constant integer := (16#ec#);
stat : constant integer := (16#ed#);
cmd : constant integer := (16#ef#);
clr : constant byte := byte(16#25#);
DSR : constant integer := 7;
DTR : constant byte := byte(16#27#);
RxRdy : constant integer := 1;
rs232_delay : constant integer := 100;
num_prts : constant integer := 23;
nullbyt : constant byte := byte(0);
max_mem_blk : CONSTANT INTEGER := 30;
num_prts : CONSTANT INTEGER := 23;
head : CONSTANT INTEGER := num_prts + 1;
TYPE array8 is ARRAY(1..8) of BYTE;
TYPE array3 is ARRAY(1..3) of BYTE;
TYPE array2 is ARRAY(1..2) of BYTE;
TYPE array4 is ARRAY(1..4) of BYTE;
datstrg : array(1..512) of byte;
TYPE array512 is ARRAY(1..512) of BYTE;

TYPE fcb_REC is RECORD
drv : BYTE;
name : array8;
ext : array3;
exnt : INTEGER;
status : INTEGER;
Fsize: array4;
date : array2;
time : array2;
resrvd: array8;
rec : BYTE;
rdm : array4;
END RECORD;

TYPE lcb_REC is RECORD
state : BYTE;
dest : BYTE;
dest_chg : BYTE;
strgSz : BYTE;
name : STRING;
link : INTEGER;
fcbA : fcb_REC;
fcbB : fcb_REC;
sndQ : INTEGER;
rsvQ : INTEGER;
filQ : INTEGER;
namQ : INTEGER;
search : BOOLEAN;
fileopen : BOOLEAN;
endFile : BOOLEAN;
cnt_remain : array4;
act : BOOLEAN;
line_cnt : INTEGER;
wait : BOOLEAN;
END RECORD;

TYPE buffer is RECORD
  dst : BYTE;
src : BYTE;
typ : BYTE;
cksum : BYTE;
len : array2;
data : array512;
frm_len : INTEGER;
END RECORD;

runfilFCB : fcb_REC;
mem_manag_tbl : ARRAY(1..max_mem_blk) OF INTEGER;
used_blk : INTEGER;
trnsQ : INTEGER;
bytaddr : BYTE;
src_prt : BYTE;
dst_prt : BYTE;
prt : INTEGER;
data : byte;
ptr : integer;
org_ocwl : byte;
mem : ARRAY(1..max_mem_blk) OF buffer;
lcb : ARRAY(0..head) OF lcb_REC;
loopcnt : INTEGER;
cksum_snt : BYTE;
bytcnt : INTEGER;
act,CTS : BOOLEAN;
byt,ch : BYTE;
free_blk : INTEGER;
quit_received : BOOLEAN;
byt_for_prt_chg : BYTE;
bell_on : BOOLEAN;
mailbox : BOOLEAN;
verbose : BOOLEAN;
runfil : BOOLEAN;
int : INTEGER;

275
found : BOOLEAN;
runfileQ : INTEGER;
estab : BOOLEAN;
logged_in : BOOLEAN;
printer : INTEGER;

END global;
with global;
PACKAGE library is
use global;

PROCEDURE activate(prt : IN INTEGER);
PROCEDURE deactivate(prt : IN INTEGER);
PROCEDURE get_memory(blk : OUT INTEGER);
FUNCTION arr_to_int(arr : IN array2) RETURN INTEGER;
PROCEDURE give_memory(blk : IN INTEGER);
PROCEDURE put_in_trnsQ(blk : IN INTEGER);
FUNCTION inc_arr(arr : IN array2) RETURN array2;
PROCEDURE prompt;
PROCEDURE add_to_Q(blk : IN INTEGER);

END library;

with bit, util, asmlib, global;
PACKAGE BODY library is
use bit, util, asmlib, global;

PROCEDURE add_to_Q(blk : IN INTEGER) is
ptr : INTEGER;
BEGIN
ptr := blk;
LOOP
EXIT WHEN mem_manag_tbl(ptr) = 0;
ptr := mem_manag_tbl(ptr);
END LOOP;
mem_manag_tbl(ptr) := blk;
END add_to_Q;

PROCEDURE prompt is
BEGIN
NEW_LINE;
IF prt = head THEN
PUT("all");
ELSE

276
PROCEDURE activate(prt : IN INTEGER) is
BEGIN
    IF NOT lcb(prt).act THEN
        lcb(prt).link := lcb(head).link;
        lcb(head).link := prt;
        lcb(prt).act := TRUE;
    END IF;
END activate;

PROCEDURE deactivate(prt : IN INTEGER) is
    ptr : INTEGER;
BEGIN
    IF lcb(prt).act THEN
        ptr := prt;
        LOOP
            EXIT WHEN lcb(ptr).link = prt;
            ptr := lcb(ptr).link;
        END LOOP;
        lcb(ptr).link := lcb(prt).link;
        lcb(prt).act := FALSE;
    END IF;
END deactivate;

PROCEDURE get_memory(blk : OUT INTEGER) is
BEGIN
    if free_blk > 0 then
        blk := free_blk;
        free_blk := mem_manag_tbl(blk);
        mem_manag_tbl(blk) := 0;
        used_blk := used_blk + 1;
    else
        blk := 0;
    end if;
END get_memory;

FUNCTION arr_to_int(arr : IN array2) RETURN INTEGER is
    int : INTEGER;
BEGIN
    poke(int'ADDRESS,arr(2));
    poke(int'ADDRESS+1,arr(1));
    RETURN int;
END arr_to_int;

PROCEDURE give_memory(blk : IN INTEGER) is
    old : INTEGER;
BEGIN

old := free_blk;
free_blk := blk;
mem_manag_tbl(blk) := old;
used_blk := used_blk - 1;

END give_memory;

PROCEDURE put_in_trnsQ(blk : IN INTEGER) is
ptr : INTEGER;
BEGIN
mem(blk).frm_len := mem(blk).frm_len + hdr_len;
IF trnsQ = 0 THEN
trnsQ := blk;
ELSE
add_to_Q(trnsQ);
END IF;
END put_in_trnsQ;

FUNCTION inc_arr(arr : IN array2) RETURN array2 is
int : INTEGER;
rslt : array2;
BEGIN
int := arr_to_int(arr);
int := int + 1;
rslt(1) := hi(int);
rslt(2) := lo(int);
RETURN rslt;
END inc_arr;

END library;

pragma warning(OFF);
pragma debug(OFF);

with util, bit, io, strlib, library, global, asmlib;
PACKAGE BODY filexfer is
use util, bit, io, strlib, library, global, asmlib;
asciiff : CONSTANT BYTE := BYTE(16#OC#);

PROCEDURE parse(blk : IN INTEGER; fcb IN OUT fcb_REC;
eol : OUT BOOLEAN) is
front,middle,rear : INTEGER;
done : BOOLEAN;
BEGIN
done := FALSE;
outer: LOOP
EXIT WHEN done;
done := TRUE;
front := 1;
eol := TRUE;
IF mem(blk).data(front) = asciicomma THEN
front := front + 1;

278
END IF;
LOOP --remove spaces, etc
   EXIT outer WHEN front > mem(blk).frm_len;
   EXIT WHEN mem(blk).data(front) > asciispace;
   front := front + 1;
END LOOP;

middle := front;
LOOP
   EXIT WHEN middle > mem(blk).frm_len
   OR mem(blk).data(middle) = asciiperiod
   OR mem(blk).data(middle) = asciicomma;
   middle := middle + 1;
END LOOP;

rear := middle;
LOOP
   EXIT WHEN rear > mem(blk).frm_len
   OR mem(blk).data(rear) = asciicomma;
   rear := rear + 1;
END LOOP;

fcb.drv := BYTE(INTEGER(current_dsk())+1); --set drive

IF mem(blk).data(front+1) = asciicolon THEN
   CASE mem(blk).data(front) is
     WHEN asciiA..asciiP =>
       fcb.drv := BYTE(INTEGER(mem(blk).data(front)) - INTEGER(asciiat));
       front := front + 2;
     WHEN ascii_a..asciip =>
       fcb.drv := BYTE(INTEGER(mem(blk).data(front)) - INTEGER(ascii_a)+1);
       front := front + 2;
     WHEN others => done := FALSE;
   END CASE;
END IF;
IF front = middle THEN
   EXIT outer;
END IF;
LOOP --remove spaces
   EXIT WHEN mem(blk).data(front) > asciispace;
   front := front + 1;
END LOOP;
ptr := 1;
inner1: LOOP --make assign
   CASE mem(blk).data(front) is
     WHEN asciiA..asciiz ! asciizero..asciinue ! asciquest ! asciounderln =>
       fcb.name(ptr) := mem(blk).data(front);
   END CASE;
279
WHEN ascii_a..ascii_z =>
  mem(blk).data(front) := capital(mem(blk).data(front));
  fcb.name(ptr) := mem(blk).data(front);
WHEN asciiasterisk =>
  inner3: LOOP
    EXIT inner1 WHEN ptr > 8;
    fcb.name(ptr) := asciiquest;
    ptr := ptr + 1;
  END LOOP inner3;
WHEN asciiBS =>
  IF ptr > 1 THEN
    ptr := ptr - 2;
  END IF;
WHEN others =>
  IF mem(blk).data(front) >= asciisspace THEN
    done := FALSE;
    EXIT inner1;
  END IF;
END CASE;
ptr := ptr + 1;
EXIT inner1 WHEN ptr > 8;
front := front + 1;
IF front = middle THEN
  LOOP
    EXIT inner1 WHEN ptr > 8;
    fcb.name(ptr) := asciisspace;
    ptr := ptr + 1;
  END LOOP;
END IF;
END LOOP inner1;
IF mem(blk).data(middle) = asciiperiod THEN
  middle := middle + 1;
END IF;
LOOP  -- remove spaces
  EXIT WHEN mem(blk).data(middle) > asciisspace
  OR middle >= rear;
  middle := middle + 1;
END LOOP;
ptr := 1;
IF middle >= rear THEN
  LOOP
    EXIT WHEN ptr > 3;
    fcb.ext(ptr) := asciisspace;
    ptr := ptr + 1;
  END LOOP;
ELSE
  inner2: LOOP  -- make assign
    CASE mem(blk).data(middle) is
      WHEN asciiA..asciiZ ! asciizero..asciinine
      asciiquest ! asciiunderln =>
        fcb.ext(ptr) := mem(blk).data(middle);
WHEN ascii_a..ascii_z =>
  mem(blk).data(middle) :=
      capital(mem(blk).data(middle));
  fcb.ext(ptr) := mem(blk).data(middle);
WHEN asciiasterisk =>
  inner4: LOOP
    EXIT inner2 WHEN ptr > 3;
    fcb.ext(ptr) := asciiquest;
    ptr := ptr + 1;
  END LOOP inner4;
WHEN asciiBS =>
  IF ptr > 1 THEN
    ptr := ptr - 2;
  END IF;
WHEN others =>
  IF mem(blk).data(middle) >= ascii_space THEN
    done := FALSE;
    EXIT inner2;
  END IF;
END CASE;
ptr := ptr + 1;
EXIT inner2 WHEN ptr > 3;
middle := middle + 1;
IF middle = rear THEN
  LOOP
    EXIT inner2 WHEN ptr > 3;
    fcb.ext(ptr) := ascii_space;
    ptr := ptr + 1;
  END LOOP;
END IF;
END LOOP inner2;
END IF;
FOR i IN rear..mem(blk).frm_len LOOP
  mem(blk).data(i+1-rear) := mem(blk).data(i);
END LOOP;
mem(blk).frm_len := mem(blk).frm_len - rear + 1;
EOL := FALSE;
IF mem(blk).frm_len < 0 THEN
  mem(blk).frm_len := 0;
END IF;
END LOOP outer;
END parse;

PROCEDURE create_FCB(blk : IN INTEGER) is
  prt : INTEGER;
  rslt : INTEGER;
BEGIN
  prt := INTEGER(mem(blk).src);
  IF lcb(prt).state = sending OR lcb(prt).state = receiving THEN
    PUT("cannot open another file for this destination");
    mem(blk).dst := mem(blk).src;
mem(blk).src := src_prt;
mem(blk).typ := unable;
mem(blk).cksum := BYTE(0);
mem(blk).len(1) := BYTE(0);
mem(blk).len(2) := BYTE(0);
mem(blk).cksum := cksum(mem(blk)'ADDRESS,hdr_len);
put_in_trnsQ(blk);
ELSE
IF verbose THEN
  PUT("receiving file ");
  FOR i IN 1..8 LOOP
    lcb(prt).FCBb.name(i) := mem(blk).data(i);
    prntdata(lcb(prt).FCBb.name(i));
  END LOOP;
  PUT('.');
  FOR i IN 1..3 LOOP
    lcb(prt).FCBb.ext(i) := mem(blk).data(i+8);
    prntdata(lcb(prt).FCBb.ext(i));
  END LOOP;
END IF;
lcb(prt).FCBb.drv := BYTE(INTEGER(current_dsk()+1)); -- set drive
create_file(lcb(prt).FCBb'ADDRESS,rslt);
IF rslt = 0 THEN
  lcb(prt).state := receiving;
  lcb(prt).fileopen := TRUE;
  lcb(prt).FCBb.extnt := 0;
  lcb(prt).FCBb.rec := BYTE(0);
  lcb(prt).line_cnt := 0;
  IF mem(blk).dst /= broadcast THEN
    mem(blk).dst := mem(blk).src;
    mem(blk).src := src_prt;
    mem(blk).typ := acklast;
    mem(blk).cksum := BYTE(0);
    mem(blk).len(1) := BYTE(0);
    mem(blk).len(2) := BYTE(0);
    mem(blk).cksum := cksum(mem(blk)'ADDRESS,hdr_len);
    put_in_trnsQ(blk);
  ELSE
    give_memory(blk);
  END IF;
ELSE
  PUT(" OUT OF DISK SPACE");
  mem(blk).dst := mem(blk).src;
  mem(blk).src := src_prt;
  mem(blk).typ := unable;
  mem(blk).cksum := BYTE(0);
  mem(blk).len(1) := BYTE(0);
  mem(blk).len(2) := BYTE(0);
  mem(blk).cksum := cksum(mem(blk)'ADDRESS,hdr_len);
  put_in_trnsQ(blk);
END IF;
PROCEDURE receive_file(blk : IN INTEGER) is
  prt : INTEGER;
  ptr : INTEGER;
  succ : BOOLEAN;
BEGIN
  prt := INTEGER(mem(blk).src);
  IF lcb(prt).fileopen AND lcb(prt).state = receiving THEN
    setDMA(mem(blk).data(1)'ADDRESS);
    lcb(prt).FCBb.Rsize := 512;
    IF mem(blk).cksum = BYTE(0) THEN --cksum OK
      write_file(lcb(prt).FCBb'ADDRESS,succ);
      IF verbose THEN
        PUT("G");
        lcb(prt).line_cnt := lcb(prt).line_cnt + 1;
        IF lcb(prt).line_cnt = 80 THEN
          NEW_LINE;
          lcb(prt).line_cnt := 0;
        END IF;
      END IF;
    END IF;
  ELSE
    IF verbose THEN
      PUT("B");
      lcb(prt).line_cnt := lcb(prt).line_cnt + 1;
      IF lcb(prt).line_cnt = 80 THEN
        NEW_LINE;
        lcb(prt).line_cnt := 0;
      END IF;
    END IF;
    succ := FALSE;
  END IF;
END IF;
END create_FCB;

IF mem(blk).dst /= broadcast THEN
  IF succ THEN
    mem(blk).dst := mem(blk).src;
    mem(blk).src := src_prtr;
    mem(blk).typ := acklast;
    mem(blk).cksum := BYTE(0);
    mem(blk).len(1) := BYTE(0);
    mem(blk).len(2) := BYTE(0);
    mem(blk).cksum := cksum(mem(blk)'ADDRESS,hdr_len);
    put_in_trnsQ(blk);
  ELSE
    mem(blk).dst := mem(blk).src;
    mem(blk).src := src_prtr;
    mem(blk).typ := badtrns;
    mem(blk).cksum := BYTE(0);
    mem(blk).len(1) := BYTE(0);
    mem(blk).len(2) := BYTE(0);
    mem(blk).cksum := cksum(mem(blk)'ADDRESS,hdr_len);
  END IF;
END IF;
put_in_trnsQ(blk);
END IF;
ELSE
    give_memory(blk);
END IF;
ELSE
    IF mem(blk).dst /= broadcast THEN
        mem(blk).dst := mem(blk).src;
        mem(blk).src := src prt;
        mem(blk).typ := unable;
        mem(blk).cksum := BYTE(0);
        mem(blk).len(1) := BYTE(0);
        mem(blk).len(2) := BYTE(0);
        mem(blk).cksum := cksum(mem(blk) 'ADDRESS, hdr_len);
        put_in_trnsQ(blk);
    ELSE
        give_memory(blk);
    END IF;
END IF;
END IF;
END receive_file;

PROCEDURE close_FCB(blk : IN INTEGER) is
    prt : INTEGER;
    rslt : INTEGER;
BEGIN
    prt := INTEGER(mem(blk).src);
    IF lcb(prt).state /= receiving THEN
        mem(blk).dst := mem(blk).src;
        mem(blk).src := src prt;
        mem(blk).typ := unable;
        mem(blk).cksum := BYTE(0);
        mem(blk).len(1) := BYTE(0);
        mem(blk).len(2) := BYTE(0);
        mem(blk).cksum := cksum(mem(blk) 'ADDRESS, hdr_len);
        put_in_trnsQ(blk);
    ELSE
        close_file(lcb(prt).FCBb'ADDRESS);
        lcb(prt).state := ready;
        lcb(prt).fileopen := FALSE;
        IF mem(blk).dst /= broadcast THEN
            mem(blk).dst := mem(blk).src;
            mem(blk).src := src prt;
            mem(blk).typ := ackLast;
            mem(blk).cksum := BYTE(0);
            mem(blk).len(1) := BYTE(0);
            mem(blk).len(2) := BYTE(0);
            mem(blk).cksum := cksum(mem(blk) 'ADDRESS, hdr_len);
            put_in_trnsQ(blk);
        ELSE
            give_memory(blk);
        END IF;
    END IF;
END IF;
END close_FCB;

PROCEDURE send_file(prt : IN INTEGER) is
  blk   : INTEGER;
  found : BOOLEAN;
  EOL   : BOOLEAN;
  rslt  : INTEGER;
BEGIN
  IF lcb(prt).name = 0 THEN
    lcb(prt).state := ready;
    RETURN;
  END IF;
  IF lcb(prt).search THEN
    IF lcb(prt).fileopen THEN
      IF lcb(prt).endFile THEN
        close_file(lcb(prt).FCBb'ADDRESS);
        IF verbose THEN
          lcb(prt).line_cnt := 0;
          prompt;
        END IF;
        get_memory(blk);
        mem(blk).dest := lcb(prt).dest;
        mem(blk).src := srcprt;
        mem(blk).typ := global.EOF;
        mem(blk).cksum := BYTE(0);
        mem(blk).len(1) := BYTE(0);
        mem(blk).len(2) := BYTE(2);
        mem(blk).data(1) := asciiCR;
        mem(blk).data(2) := asciiFF;
        mem(blk).frm_len := hdr_len + 2;
        mem(blk).cksum :=
          cksum(mem(blk)'ADDRESS,mem(blk).frm_len);
        lcb(prt).fileQ := blk;
        lcb(prt).state := repeat_snd;
        lcb(prt).fileopen := FALSE;
      ELSE
        get_memory(blk);
        mem(blk).frm_len := 512;
        setDMA(mem(blk).data(1)'ADDRESS);
        lcb(prt).FCBb.Rsize := 512;
        read_file(lcb(prt).FCBb'ADDRESS,rslt);
        CASE rslt is
          WHEN 0 => null;
          WHEN 1 =>
            lcb(prt).endfile := TRUE;
          WHEN 2 => null;
          WHEN 3 =>
            lcb(prt).endfile := TRUE;
        END CASE;
      IF rslt = 0 OR rslt = 3 THEN

mem(blk).dst := lcb(prt).dest;
mem(blk).src := src_prt;
mem(blk).typ := filedat;
mem(blk).cksum := BYTE(0);
mem(blk).len(1) := hi(mem(blk).frm_len);
mem(blk).len(2) := lo(mem(blk).frm_len);
mem(blk).frm_len := mem(blk).frm_len + hdr_len;
mem(blk).cksum := checksum(mem(blk)'ADDRESS,
mem(blk).frm_len);
lcb(prt).filQ := blk;
lcb(prt).state := repeatsnd;
ELSE
  give_memory(blk);
END IF;
ELSE
  setDMA(lcb(prt).FCBb'ADDRESS);
  search_nxt(lcb(prt).FCBa'ADDRESS, found);
  IF found THEN
    IF verbose THEN
      NEW_LINE;
      PUT("sending file ");
      FOR i IN 1..8 LOOP
        prntdata(lcb(prt).FCBb.name(i));
      END LOOP;
      PUT(".");
      FOR i IN 1..3 LOOP
        prntdata(lcb(prt).FCBb.ext(i));
      END LOOP;
      NEW_LINE;
    END IF;
    open_file(lcb(prt).FCBb'ADDRESS, found);
    lcb(prt).FCBb.Rsize := 512;
    lcb(prt).FCBb.extnt := 0;
    lcb(prt).FCBb.rec := BYTE(0);
    lcb(prt).fileopen := TRUE;
    lcb(prt).cnt_remain := lcb(prt).FCBb.Fsize;
    lcb(prt).endfile := FALSE;
    get_memory(blk);
    mem(blk).dst := lcb(prt).dest;
    mem(blk).src := src_prt;
    mem(blk).typ := sendfile;
    mem(blk).cksum := BYTE(0);
    mem(blk).len(1) := BYTE(0);
    mem(blk).len(2) := BYTE(35);
    FOR i IN 1..8 LOOP
      mem(blk).data(i) := lcb(prt).FCBb.name(i);
    END LOOP;
    FOR i IN 1..3 LOOP
      mem(blk).data(i+8) := lcb(prt).FCBb.ext(i);
    END LOOP;
    mem(blk).data(12) := asciiLF;
  END IF;
END IF;
ELSE
  setDMA(lcb(prt).FCBb'ADDRESS);
  search_nxt(lcb(prt).FCBa'ADDRESS, found);
  IF found THEN
    IF verbose THEN
      NEW_LINE;
      PUT("sending file ");
      FOR i IN 1..8 LOOP
        prntdata(lcb(prt).FCBb.name(i));
      END LOOP;
      PUT(".");
      FOR i IN 1..3 LOOP
        prntdata(lcb(prt).FCBb.ext(i));
      END LOOP;
      NEW_LINE;
    END IF;
    open_file(lcb(prt).FCBb'ADDRESS, found);
    lcb(prt).FCBb.Rsize := 512;
    lcb(prt).FCBb.extnt := 0;
    lcb(prt).FCBb.rec := BYTE(0);
    lcb(prt).fileopen := TRUE;
    lcb(prt).cnt_remain := lcb(prt).FCBb.Fsize;
    lcb(prt).endfile := FALSE;
    get_memory(blk);
    mem(blk).dst := lcb(prt).dest;
    mem(blk).src := src_prt;
    mem(blk).typ := sendfile;
    mem(blk).cksum := BYTE(0);
    mem(blk).len(1) := BYTE(0);
    mem(blk).len(2) := BYTE(35);
    FOR i IN 1..8 LOOP
      mem(blk).data(i) := lcb(prt).FCBb.name(i);
    END LOOP;
    FOR i IN 1..3 LOOP
      mem(blk).data(i+8) := lcb(prt).FCBb.ext(i);
    END LOOP;
    mem(blk).data(12) := asciiLF;
  END IF;
END IF;
mem(blk).data(13) := asciiCR;
FOR i IN 1..20 LOOP
  IF i <= LENGTH(lcb(head).name) THEN
    mem(blk).data(13+i) := conv_byt(lcb(head).name(i));
  ELSE
    mem(blk).data(13+i) := asciiispace;
  END IF;
END LOOP;
mem(blk).data(34) := asciiLF;
mem(blk).data(35) := asciiFF;
mem(blk).frm_len := 35 + hdr_len;
mem(blk).cksum :=
  cksum(mem(blk)'ADDRESS,mem(blk).frm_len);
lcb(prt).flIQ := blk;
lcb(prt).state := repeatsnd;
ELSE
  lcb(prt).search := FALSE;
  END IF;
END IF;
ELSE
parse(lcb(prt).namQ,lcb(prt).FCBa,EOL);
IF EOL THEN
  lcb(prt).state := ready;
give_memory(lcb(prt).namQ);
lcb(prt).namQ := 0;
  IF prt = printer THEN
    outport(dat,code_endprint);
    printer := 99;
  END IF;
ELSE
  setDMA(lcb(prt).FCBb'ADDRESS);
  search_first(lcb(prt).FCBa'ADDRESS,found);
  IF found THEN
    IF verbose THEN
      NEW_LINE;
      PUT("sending file ");
      FOR i IN 1..8 LOOP
        prntdata(lcb(prt).FCBb.name(i));
      END LOOP;
      PUT(".");
      FOR i IN 1..3 LOOP
        prntdata(lcb(prt).FCBb.ext(i));
      END LOOP;
      NEW_LINE;
    END IF;
    lcb(prt).search := TRUE;
    open_file(lcb(prt).FCBb'ADDRESS,found);
    lcb(prt).cnt_remain := lcb(prt).FCBb.Fsize;
    lcb(prt).fileopen := TRUE;
    lcb(prt).endfile := FALSE;
  ELSE
  END IF;
END ELSE;
END IF;
JANUS/ADA IMPLEMENTATION OF A STAR CLUSTER NETWORK OF
PERSONAL COMPUTERS W. (U) NAVAL POSTGRADUATE SCHOOL
MONTEREY CA R L HARTMAN ET AL. JUN 86
UNCLASSIFIED

AD-A173 595
mem(blk).dst := lcb(prt).dest;
mem(blk).src := src_prt;
mem(blk).typ := sendfile;
mem(blk).cksum := BYTE(0);
mem(blk).len(1) := BYTE(0);
mem(blk).len(2) := BYTE(35);
FOR i IN 1..8 LOOP
  mem(blk).data(i) := lcb(prt).FCBb.name(i);
END LOOP;
FOR i IN 1..3 LOOP
  mem(blk).data(i+8) := lcb(prt).FCBb.ext(i);
END LOOP;
mem(blk).data(12) := asciiLF;
mem(blk).data(13) := asciiCR;
FOR i IN 1..20 LOOP
  IF i <= LENGTH(lcb(head).name) THEN
    conv_byt(lcb(head).name(i));
  ELSE
    mem(blk).data(13+i) := ascii_space;
  END IF;
END LOOP;
mem(blk).data(34) := asciiLF;
mem(blk).data(35) := asciiFF;
mem(blk).frm_len := 35 + hdr_len;
mem(blk).cksum :=
  cksum(mem(blk)'ADDRESS,mem(blk).frm_len);
lcb(prt).filq := blk;
lcb(prt).state := repeatsnd;
END IF;
END IF;
END IF;
END send_file;

PROCEDURE send_dir(prt : IN INTEGER) is
  blk : INTEGER;
  found : BOOLEAN;
  EOL : BOOLEAN;
  ptr : INTEGER;
  total : INTEGER;
  line_tot: INTEGER;
BEGIN
  IF lcb(prt).namq = 0 THEN
    lcb(prt).state := ready;
    RETURN;
  END IF;
  ptr := 0;
  total := 0;
  line_tot := 0;
  blk := lcb(prt).filq;
LOOP
IF lcb(prt).search THEN
   setDMA(lcb(prt).FCBb'ADDRESS);
   search_nxt(lcb(prt).FCBa'ADDRESS,found);
   IF found THEN
      IF line_tot = 0 THEN
         ptr := ptr + 1;
         mem(blk).data(ptr) :=
            BYTE(INTEGER(lcb(prt).FCBb.drv) + 64);
      END IF;
      ptr := ptr + 1;
      mem(blk).data(ptr) := asciicolon;
      ptr := ptr + 1;
      mem(blk).data(ptr) := asciiispace;
      FOR i IN 1..8 LOOP
         ptr := ptr + 1;
         mem(blk).data(ptr) := lcb(prt).FCBb.name(i);
      END LOOP;
      ptr := ptr + 1;
      mem(blk).data(ptr) := asciiispace;
      line_tot := line_tot + 1;
      IF line_tot = 4 THEN
         line_tot := 0;
         ptr := ptr + 1;
         mem(blk).data(ptr) := asciiCR;
         ptr := ptr + 1;
         mem(blk).data(ptr) := asciiLF;
      END IF;
      total := total + 1;
      IF total = 32 THEN
         mem(blk).dst := lcb(prt).dest;
         mem(blk).src := srcprt;
         mem(blk).typ := dir_data;
         mem(blk).cksum := BYTE(0);
         mem(blk).len(1) := hi(ptr);
         mem(blk).len(2) := lo(ptr);
         mem(blk).frm_len := ptr + hdr_len;
         mem(blk).cksum :=
            cksum(mem(blk)'ADDRESS,ptr+hdr_len);
         put_in_trnsQ(blk);
         lcb(prt).filQ := 0;
      END IF;
   ELSE
      lcb(prt).search := FALSE;
   END IF;
END IF;
*
ELSE
    parse(lcb(prt).namQ,lcb(prt).FCBb,EOL);
    IF EOL THEN
        lcb(prt).state := ready;
        give_memory(lcb(prt).namQ);
        IF ptr /= 0 THEN
            mem(blk).dst := lcb(prt).dest;
            mem(blk).src := srcprt;
            mem(blk).typ := dir_data;
            mem(blk).cksum := BYTE(0);
            mem(blk).len(1) := hi(ptr);
            mem(blk).len(2) := lo(ptr);
            mem(blk).frm_len := ptr + hdr_len;
            mem(blk).cksum :=
                cksum(mem(blk)'ADDRESS,ptr+hdr_len);
            put_in_transQ(blk);
            EXIT;
        ELSE
            give_memory(blk);
            EXIT;
        END IF;
    END IF;
    setDMA(lcb(prt).FCBb'ADDRESS);
    search_fst(lcb(prt).FCBb'ADDRESS,found);
    IF found THEN
        lcb(prt).search := TRUE;
        IF line_tot = 0 THEN
            ptr := ptr + 1;
            mem(blk).data(ptr) :=
                BYTE(INTEGER(lcb(prt).FCBb.drv) + 64);
        END IF;
        ptr := ptr + 1;
        mem(blk).data(ptr) := asciicolon;
        ptr := ptr + 1;
        mem(blk).data(ptr) := asciispace;
        FOR i IN 1..8 LOOP
            ptr := ptr + 1;
            mem(blk).data(ptr) := lcb(prt).FCBb.name(i);
        END LOOP;
        ptr := ptr + 1;
        mem(blk).data(ptr) := asciinewline;
        FOR i IN 1..3 LOOP
            ptr := ptr + 1;
            mem(blk).data(ptr) := lcb(prt).FCBb.ext(i);
        END LOOP;
        ptr := ptr + 1;
        mem(blk).data(ptr) := asciispace;
        line_tot := line_tot + 1;
        IF line_tot = 4 THEN
            line_tot := 0;
            ptr := ptr + 1;
            mem(blk).data(ptr) := asciiCR;
            290
ptr := ptr + 1;
mem(blk).data(ptr) := asciilF;
END IF;
total := total + 1;
IF total = 32 THEN
mem(blk).dst := lcb(prt).dest;
mem(blk).src := src_prt;
mem(blk).typ := dir_data;
mem(blk).cksum := BYTE(0);
mem(blk).len(1) := hi(ptr);
mem(blk).len(2) := lo(ptr);
mem(blk).frm_len := ptr + hdr_len;
mem(blk).cksum :=
put_in_cksum(mem(blk)'ADDRESS,ptr+hdr_len);
ptittrnsQ(blk);
EXIT;
END IF;
END IF;
END LOOP;
END send_dir;

PROCEDURE information(prt :IN INTEGER) is
blk : INTEGER;
rslt : INTEGER;
ptr : INTEGER;
BEGIN
outer: LOOP
IF lcb(prt).fileopen THEN
blk := lcb(prt).filQ;
IF mem(blk).frm_len = 0 THEN
lcb(prt).FCBb.Rsize := 512;
setDMA(mem(blk).data(1)'ADDRESS);
read_file(lcb(prt).FCBb'ADDRESS,rslt);
IF rslt = 0 OR rslt = 3 THEN
mem(blk).frm_len := 1;
LOOP
IF mem(blk).frm_len > 512 THEN
mem(blk).frm_len := 0;
EXIT;
END IF;
IF mem(blk).data(mem(blk).frm_len) =
BYTE(16#1A#) THEN
lcb(prt).fileopen := FALSE;
EXIT outer;
END IF;
IF mem(blk).data(mem(blk).frm_len) =
BYTE(16#09#) THEN
mem(blk).frm_len :=
mem(blk).frm_len + 1;
EXIT outer;
END IF;
291
END IF;
prntdata(mem(blk).data(mem(blk).frm_len));
mem(blk).frm_len := mem(blk).frm_len + 1;
END LOOP;
ELSE
lcb(prt).fileopen := FALSE;
EXIT outer;
END IF;
ELSE
LOOP
IF mem(blk).frm_len > 512 THEN
mem(blk).frm_len := 0;
EXIT;
END IF;
IF mem(blk).data(mem(blk).frm_len) =
BYTE(16#1A#) THEN
lcb(prt).fileopen := FALSE;
EXIT outer;
END IF;
IF mem(blk).data(mem(blk).frm_len) =
BYTE(16#09#) THEN
mem(blk).frm_len := mem(blk).frm_len + 1;
EXIT outer;
END IF;
prntdata(mem(blk).data(mem(blk).frm_len));
mem(blk).frm_len := mem(blk).frm_len + 1;
END LOOP;
END IF;
ELSE
lcb(prt).state := ready;
EXIT;
END IF;
END LOOP outer;
IF lcb(prt).fileopen THEN
NEW LINE;
NEW LINE;
PUT("hit space bar to continue, 'Q' to quit");
NEW LINE;
ELSE
close_file(lcb(prt).FCBb'ADDRESS);
give_memory(blk);
lcb(prt).filq := 0;
lcb(prt).state := ready;
END IF;
END information;
END filexfer;
pragma warning(OFF);
pragma debug(OFF);

--PACKAGE: locftp.PKG

292
PROCEDURE locftp is
use filexfer, global, library, bit, asmlib, strlib, io, util;

PROCEDURE handle_kybd_input(ch : IN BYTE) is
  blk : INTEGER;
  num : INTEGER;

BEGIN
  CASE lcb(prt).state is
    WHEN ready =>
      CASE ch is
        WHEN asciiT ! ascii t =>
          lcb(prt).state := talk;
          PUT("Talk, enter text, ^Z to send");
          get_memory(blk);
          IF blk /= 0 THEN
            lcb(prt).sndQ := blk;
            mem(blk).frm_len := 0;
            NEW_LINE;
            ELSE
              PUT("out of memory");
              lcb(prt).state := ready;
              prompt;
              END IF;
        WHEN asciiS ! asciiis =>
          get_memory(blk);
          IF blk = 0 THEN
            PUT("out of memory"); NEW_LINE;
            lcb(prt).state := ready;
            prompt;
            ELSE
              lcb(prt).state := sendfile;
              PUT("Send <filename> enter text, ");
              put("^Z to send");
              NEW_LINE;
              lcb(prt).namQ := blk;
              mem(blk).frm_len := 0;
            END IF;
        WHEN asciiG ! ascii g =>
          get_memory(blk);
          IF blk = 0 THEN
            PUT("out of memory");
            lcb(prt).state := ready;
            prompt;
            ELSE
              IF prt = head THEN
                PUT("cannot 'get' from 'all'");
                prompt;
              END IF;
        END CASE;
    WHEN head =>
      prompt;
    WHEN ready =>
      lcb(prt).state := talk;
      PUT("Talk, enter text, ^Z to send");
      get_memory(blk);
      IF blk /= 0 THEN
        lcb(prt).sndQ := blk;
        mem(blk).frm_len := 0;
        NEW_LINE;
        ELSE
          PUT("out of memory");
          lcb(prt).state := ready;
          prompt;
          END IF;
    WHEN asciiS ! asciiis =>
      get_memory(blk);
      IF blk = 0 THEN
        PUT("out of memory"); NEW_LINE;
        lcb(prt).state := ready;
        prompt;
        ELSE
          lcb(prt).state := sendfile;
          PUT("Send <filename> enter text, ");
          put("^Z to send");
          NEW_LINE;
          lcb(prt).namQ := blk;
          mem(blk).frm_len := 0;
        END IF;
      WHEN asciiG ! ascii g =>
        get_memory(blk);
        IF blk = 0 THEN
          PUT("out of memory");
          lcb(prt).state := ready;
          prompt;
          ELSE
            IF prt = head THEN
              PUT("cannot 'get' from 'all'");
              prompt;
            END IF;
      END CASE;
  END CASE;
END handle_kybd_input;
ELSE
   lcb(prt).state := getfile;
   PUT("Get <filename> enter text, ");
   put("^Z to get");
   NEW_LINE;
   lcb(prt).sndQ := blk;
   mem(blk).frm_len := 0;
   END IF;
END IF;
WHEN asciiQ ! ascii_q =>
   PUT("Quit [confirm]");
lcb(prt).state := quit;
WHEN asciiQuest =>
   NEW_LINE;
   PUT("all"); NEW_LINE;
   PUT("Bell"); NEW_LINE;
   PUT("Change group"); NEW_LINE;
   PUT("Directory"); NEW_LINE;
   PUT("Get"); NEW_LINE;
   PUT("Information"); NEW_LINE;
   PUT("List"); NEW_LINE;
   PUT("Mailbox"); NEW_LINE;
   PUT("Netstat"); NEW_LINE;
   PUT("Print"); NEW_LINE;
   PUT("Quit"); NEW_LINE;
   PUT("Send"); NEW_LINE;
   PUT("Talk"); NEW_LINE;
   PUT("Verbose"); NEW_LINE;
   PUT("Who's there"); NEW_LINE;
   PUT("<destination >"); NEW_LINE;
   PUT("#"); NEW_LINE;
   PUT("?");
   prompt;
WHEN BYTE(16#30#)..BYTE(16#39#) =>
   prntdata(ch);
   lcb(prt).dest_chg := ch;
   lcb(prt).state := prt_chg;
WHEN asciiW ! ascii_w =>
   get_memory(blk);
   IF blk = 0 THEN
      PUT("out of memory");
prompt;
   ELSE
      PUT("Who's there?");
      LOOP
         EXIT WHEN lcb(head).link = head;
         deactivate(lcb(head).link);
      END LOOP;
   END IF;
   mem(blk).frm_len := 0;
   mem(blk).dst := broadcast;
   mem(blk).src := srcprt;
   mem(blk).typ := whothere;
mem(blk).len(1) := BYTE(0);
mem(blk).len(2) := BYTE(0);
mem(blk).cksum := BYTE(0);
 mem(blk).cksum :=
    cksum(mem(blk)'ADDRESS,hdr_len);
put_in_trnsQ(blk);
prompt;
END IF;
WHEN asciiN ! ascii_n =>
    PUT("Netstat");
    outport(dat,code_status);
    LOOP
        import(stat,data);
        EXIT WHEN tstbit(INTEGER(data),RxRdy);
    END LOOP;
inport(dat,data);
prompt;
WHEN asciiL ! ascii_l =>
    PUT("List"); NEW_LINE;
    ptr := lcb(head).link;
    PUT("term #, name, " );
    put("state of connection");
    NEW_LINE;
    LOOP
        EXIT WHEN ptr = head;
        put(ptr); PUT("-> ");
        IF ptr < 10 THEN
            PUT(" ");
        END IF;
        put(lcb(ptr).name);
        FOR i IN LENGTH(lcb(ptr).name) .. 20 LOOP
            PUT(" ");
        END LOOP;
        CASE lcb(ptr).state is
            WHEN ready =>
                PUT(" ready");
            WHEN sending =>
                PUT(" sending");
            WHEN getfile =>
                PUT(" getfile");
            WHEN repeatsnd =>
                PUT("repeat transmission");
            WHEN wait_for_ack =>
                PUT("wait for acknowledgement");
            WHEN receiving =>
                PUT("receiving");
            WHEN dir =>
                PUT("Directory");
            WHEN others =>
                PUT(" unknown state");
        END CASE;
    NEW_LINE;
    295
ptr := lcb(ptr).link;
END LOOP;
PUT("Your terminal number is ");
PUT(INTEGER(src_prt));
prompt;
WHEN asciiA ! ascii_a =>
  PUT("all");
  prt := head;
  dst_prt := broadcast;
prompt;
WHEN asciiI ! ascii_i =>
  lcb(prt).FCBb.name(1) := asciiI;
  lcb(prt).FCBb.name(2) := asciiN;
  lcb(prt).FCBb.name(3) := asciiF;
  lcb(prt).FCBb.name(4) := asciiO;
  open_file(lcb(prt).FCBb'ADDRESS,found);
  IF found THEN
    get_memory(blk);
    lcb(prt).state := info;
    lcb(prt).filQ := blk;
    lcb(prt).fileopen := TRUE;
    lcb(prt).FCBb.extnt := 0;
    lcb(prt).FCBb.rec := BYTE(0);
    mem(blk).frm_len := 0;
    NEW_LINE;
    information(prompt);
  ELSE
    FOR i IN 1..8 LOOP
      prntdata(lcb(prt).FCBb.name(i));
    END LOOP;
    PUT('.');
    FOR i IN 1..3 LOOP
      prntdata(lcb(prt).FCBb.ext(i));
    END LOOP;
    PUT(" not on current logged disk");
prompt;
  END IF;
WHEN asciiD ! ascii_d =>
  lcb(prt).state := dir;
  PUT("Directory, enter text, ^Z to send");
  get_memory(blk);
  IF blk /= 0 THEN
    lcb(prt).sndQ := blk;
    mem(blk).frm_len := 0;
  END IF;
NEW_LINE;
ELSE
  PUT("out of memory");
lcb(prt).state := ready;
prompt;
END IF;
WHEN asciiB ! ascii_b =>
  IF bell_on THEN
    bell_on := FALSE;
    PUT("Bell-OFF");
  ELSE
    bell_on := TRUE;
    PUT("Bell-ON");
  END IF;
prompt;
WHEN asciiM ! ascii_m =>
  IF mailbox THEN
    mailbox := FALSE;
    PUT("Mailbox-OFF");
  ELSE
    mailbox := TRUE;
    PUT("Mailbox-ON");
  END IF;
prompt;
WHEN asciipace =>
  IF lcb(prt).state = info THEN
    information(prt);
  ELSE
    prompt;
  END IF;
WHEN asciiV ! ascii_v =>
  IF verbose THEN
    PUT("Verbose-OFF");
    verbose := FALSE;
  ELSE
    PUT("Verbose-ON");
    verbose := TRUE;
  END IF;
prompt;
WHEN asciiilbs =>
  PUT("destination terminal is now ");
  put("your terminal");
  prt := INTEGER(src_prt);
  dst_prt := src_prt;
prompt;
WHEN asciiip ! ascii_i_p =>
  IF used_blk >= max_mem_blk - 1 THEN
    PUT("out of memory"); NEW_LINE;
    lcb(prt).state := ready;
prompt;
ELSE
  outport(dat,code_print);
LOOP
  inport(stat,data);
  IF tstbit(INTEGER(data),RxRdy) THEN
    inport(dat,data);
    IF data <= BYTE(num_prts) THEN
      estab := TRUE;
      printer := INTEGER(data);
      prt := printer;
      dst_prt := data;
      lcb(prt).state := sendfile;
      PUT("Print <filename> ");
      put("enter text, ^Z to print");
      NEW_LINE;
      get_memory(blk);
      lcb(prt).name := blk;
      mem(blk).frm_len := 0;
      EXIT;
    ELSE
      PUT("Printer busy");
      prompt;
      EXIT;
    END IF;
  END IF;
END LOOP;
END IF;
WHEN asciiC ! ascii_c =>
  PUT("Change group, enter destination ");
  outport(dat,code_clis);
  estab := FALSE;
  mailbox := TRUE;
WHEN asciiCR =>
  prompt;
WHEN others =>
  PUT("unrecognized command, ");
  put("type '?' for command list");
  prompt;
END CASE;
WHEN talk =>
  blk := lcb(prt).sndQ;
CASE ch is
WHEN asciicntlR =>
  NEW_LINE;
  FOR i IN 1..mem(blk).frm_len LOOP
    prntdata(mem(blk).data(i));
  END LOOP;
WHEN asciicntlZ =>
  mem(blk).typ := talk;
  mem(blk).len(1) := hi(mem(blk).frm_len);
  mem(blk).len(2) := lo(mem(blk).frm_len);
  mem(blk).dst := dst_prt;
  mem(blk).src := src_prt;
  mem(blk).cksum := BYTE(0);
mem(blk).cksum := cksum(mem(blk)'ADDRESS, mem(blk).frm_len + hdr_len);
put_in_trnsQ(blk);
lcb(prt).sndQ := 0;
lcb(prt).state := ready;
prompt;
WHEN asciicntlQ =>
    NEW_LINE;
    PUT("discarding entries");
give_memory(blk);
lcb(prt).sndQ := 0;
lcb(prt).state := ready;
prompt;
WHEN asciiBS ! asciiDEL =>
    IF mem(blk).frm-len > 0 THEN
        mem(blk).frm_len := mem(blk).frm_len - 1;
        prntdata(asciiBS);
        prntdata(asciispace);
        prntdata(asciiBS);
    END IF;
    WHEN others =>
        mem(blk).frm_len := mem(blk).frm_len + 1;
        mem(blk).data(mem(blk).frm_len) := ch;
        prntdata(ch);
        IF ch = asciiCR THEN
            mem(blk).frm_len := mem(blk).frm_len + 1;
            mem(blk).data(mem(blk).frm_len) := asciiLF;
            prntdata(asciiLF);
        END IF;
    IF mem(blk).frm_len = 512 THEN
        mem(blk).typ := talk;
        mem(blk).len(1) := hi(mem(blk).frm_len);
        mem(blk).len(2) := lo(mem(blk).frm_len);
        mem(blk).dst := dst_prt;
        mem(blk).src := src_prt;
        mem(blk).cksum := BYTE(0);
        mem(blk).cksum := cksum(mem(blk)'ADDRESS,
            mem(blk).frm_len + hdr_len);
        put_in_trnsQ(blk);
    END IF;
    ELSE
        NEW_LINE;
        PUT("out of memory");
lcb(prt).sndQ := 0;
lcb(prt).state := ready;
prompt;
    END IF;
END IF;
END CASE;
WHEN quit =>
  IF ch = asciiCR THEN
    quit_received := TRUE;
    PUT("Good-bye.");
    NEW_LINE;
  ELSE
    NEW_LINE;
    lcb(prt).state := ready;
    prompt;
  END IF;
WHEN prt_chg =>
  CASE ch is
    WHEN BYTE(16#30#)..BYTE(16#39#) =>
      prntdata(ch);
      num := Land(INTEGER(lcb(prt).dest_chg),
                 INTEGER(16#000F#));
      num := (num * 10) +
             Land(INTEGER(ch),INTEGER(16#000F#));
      IF num > num_prts THEN
        num := prt;
        PUT("port num out of range");
      END IF;
    WHEN asciiCR =>
      num := Land(INTEGER(lcb(prt).dest_chg),
                   INTEGER(16#000F#));
    WHEN others =>
      num := prt;
      PUT("bad input");
  END CASE;
  lcb(prt).state := ready;
  prt := num;
  dst_prt := BYTE(prt);
  activate(prt);
  IF NOT estab THEN
    outport(dat,code_local);
  END IF;
  prompt;

WHEN sendfile =>
  blk := lcb(prt).namQ;
  CASE ch is
    WHEN asciicntlR =>
      NEW_LINE;
      FOR i IN 1..mem(blk).frm_len LOOP
        prntdata(mem(blk).data(i));
      END LOOP;
    WHEN asciicntlZ =>
      lcb(prt).state := sending;
      lcb(prt).search := FALSE;
      lcb(prt).fileopen := FALSE;
      lcb(prt).endFile := FALSE;
      lcb(prt).line_cnt := 0;
prompt;
WHEN asciicnt1Q =>
    NEW_LINE;
    PUT("discarding entries");
    give_memory(blk);
    lcb(prt).state := ready;
    lcb(prt).namQ := 0;
    IF prt = printer THEN
        outport(dat,code_endprint);
        printer := 99;
    END IF;
    prompt;
WHEN asciIBS ! asciIDEL =>
    IF mem(blk).frm_len > 0 THEN
        mem(blk).frm_len := mem(blk).frm_len - 1;
        prntdata(asciIBS);
        prntdata(asciispace);
        prntdata(asciIBS);
    END IF;
    WHEN others =>
        mem(blk).frm_len := mem(blk).frm_len + 1;
        mem(blk).data(mem(blk).frm_len) := ch;
        prntdata(ch);
        IF ch = asciICR THEN
            mem(blk).frm_len := mem(blk).frm_len + 1;
            mem(blk).data(mem(blk).frm_len) := asciILF;
            prntdata(asciILF);
        END IF;
        IF mem(blk).frm_len = 512 THEN
            lcb(prt).state := sending;
            lcb(prt).search := FALSE;
            lcb(prt).fileopen := FALSE;
            lcb(prt).endFile := FALSE;
            prompt;
        END IF;
    END CASE;
WHEN getfile =>
    blk := lcb(prt).sndQ;
    CASE ch is
        WHEN asciicnt1R =>
            NEW_LINE;
            FOR i IN 1..mem(blk).frm_len LOOP
                prntdata(mem(blk).data(i));
            END LOOP;
        WHEN asciicnt1Z =>
            mem(blk).typ := getfile;
            mem(blk).len(1) := hi(mem(blk).frm_len);
            mem(blk).len(2) := lo(mem(blk).frm_len);
            mem(blk).dst := dstprt;
            mem(blk).src := srcprt;
            mem(blk).cksum := BYTE(0);
            mem(blk).cksum := cksum(mem(blk)'ADDRESS,"
mem(blk).frm_len + hdr_len);
put_in_trnsQ(blk);
lcb(prt).sndQ := 0;
lcb(prt).state := ready;
prompt;
WHEN asciicntlQ =>
    NEW_LINE;
    PUT("discarding entries");
give_memory(blk);
lcb(prt).sndQ := 0;
lcb(prt).state := ready;
WHEN asciiBS!asciiDEL =>
    IF mem(blk).frm_len > 0 THEN
        mem(blk).frm_len := mem(blk).frm_len - 1;
        prntdata(asciiBS);
        prntdata(asciispace);
        prntdata(asciiBS);
    END IF;
    WHEN others =>
        mem(blk).frm_len := mem(blk).frm_len + 1;
        mem(blk).data(mem(blk).frm_len) := ch;
        prntdata(ch);
        IF ch = asciiCR THEN
            mem(blk).frm_len := mem(blk).frm_len + 1;
            mem(blk).data(mem(blk).frm_len) := asciiLF;
            prntdata(asciiLF);
        END IF;
        IF mem(blk).frm_len = 512 THEN
            mem(blk).typ := getfile;
            mem(blk).len(1) := hi(mem(blk).frm_len);
            mem(blk).len(2) := lo(mem(blk).frm_len);
            mem(blk).dst := dst_prp;
            mem(blk).src := src_prp;
            mem(blk).cksum := BYTE(0);
            mem(blk).cksum := cksum(mem(blk)'ADDRESS,
                mem(blk).frm_len + hdr_len);
            put_in_trnsQ(blk);
            lcb(prt).sndQ := 0;
            lcb(prt).state := ready;
        END IF;
        END CASE;
    WHEN dir =>
        blk := lcb(prt).sndQ;
        CASE ch is
            WHEN asciicntlR =>
                NEW_LINE;
                FOR i IN 1..mem(blk).frm_len LOOP
                    prntdata(mem(blk).data(i));
                END LOOP;
            WHEN asciicntlZ =>
                mem(blk).typ := dir;

mem(blk).len(1) := hi(mem(blk).frm_len);
mem(blk).len(2) := lo(mem(blk).frm_len);
mem(blk).dst := dst_prt;
mem(blk).src := src_prt;
mem(blk).cksum := BYTE(0);
mem(blk).cksum := cksum(mem(blk)'ADDRESS,
   mem(blk).frm_len + hdr_len);
put_in_trnsQ(blk);
lcb(prt).sndQ := 0;
lcb(prt).state := ready;
prompt;
WHEN asciicntlQ =>
  NEW LINE;
give_memory(blk);
lcb(prt).sndQ := 0;
lcb(prt).state := ready;
WHEN asciiBS ! asciiDEL =>
  IF mem(blk).frm_len > 0 THEN
    mem(blk).frm_len := mem(blk).frm_len - 1;
    prntdata(asciiBS);
    prntdata(asciiisspace);
    prntdata(asciiBS);
  END IF;
WHEN others =>
  mem(blk).frm_len := mem(blk).frm_len + 1;
  mem(blk).data(mem(blk).frm_len) := ch;
  prntdata(ch);
  IF ch = asciiCR THEN
    mem(blk).frm_len := mem(blk).frm_len + 1;
    mem(blk).data(mem(blk).frm_len) := asciiLF;
    prntdata(asciiLF);
  END IF;
  IF mem(blk).frm_len = 512 THEN
    mem(blk).typ := getfile;
    mem(blk).len(1) := hi(mem(blk).frm_len);
    mem(blk).len(2) := lo(mem(blk).frm_len);
    mem(blk).dst := dst_prt;
    mem(blk).src := src_prt;
    mem(blk).cksum := BYTE(0);
    mem(blk).cksum := cksum(mem(blk)'ADDRESS,
       mem(blk).frm_len + hdr_len);
    put_in_trnsQ(blk);
lcb(prt).sndQ := 0;
lcb(prt).state := ready;
  END IF;
END CASE;
WHEN log =>
  IF ch = asciiBS OR ch = asciiDEL THEN
    IF length(lcb(head).name) > 0 THEN
      lcb(head).name := remove(lcb(head).name,
         length(lcb(head).name),1);
prntdata(asciiBS);
prntdata(asciispace);
prntdata(asciiBS);
END IF;
ELSE
prntdata(ch);
lcb(head).name :=
    insert(char_to_str(byte_to_chr(ch)),
    lcb(head).name,length(lcb(head).name)+1);
END IF;
WHEN info =>
CASE ch is
WHEN asciispace =>
    information(prt);
WHEN asciiQuest =>
    prntdata(ch); NEW_LINE;
    PUT("space bar"); NEW_LINE;
    PUT("Quit"); NEW_LINE;
WHEN asciiQ ! ascii_q =>
    close_file(lcb(prt).FCBb'ADDRESS);
    lcb(prt).fileopen := FALSE;
    lcb(prt).state := ready;
    give_memory(lcb(prt).filQ);
    prompt;
WHEN others =>
    PUT("unrecognized command");
    NEW_LINE;
END CASE;
WHEN others =>
    IF prt /= INTEGER(src_prt) THEN
        ptr := INTEGER(src_prt);
        dst_prt := src_prt;
    ELSE
        ptr := head;
        dst_prt := BYTE(head);
    END IF;
    NEW_LINE;
    PUT("Process running on this connection, ");
    put("changing destination terminal");
    prompt;
END CASE;
END handle_kybd_input;

PROCEDURE handle_incoming_packet(blk : IN INTEGER) is
BEGIN
    ptr := INTEGER(mem(blk).src);
    IF ptr <= num_prt THEN
        activate(ptr);
        CASE mem(blk).typ is
            WHEN talk =>
                IF lcb(prt).state /= ready THEN
                    IF lcb(ptr).rcvQ = 0 THEN
lcb(ptr).rcvQ := blk;
ELSE
  add_to_Q(lcb(ptr).rcvQ);
END IF;
ELSE
  NEW_LINE;
  PUT("msg fr ");
  PUT(lcb(ptr).name);
  PUT(ptr); PUT('>') ; NEW_LINE;
  FOR i IN 1..arr_to_int(mem(blk).len) LOOP
    prntdata(mem(blk).data(i));
  END LOOP;
  IF bell on THEN
    prntdata(asciibell);
  END IF;
  give_memory(blk);
prompt;
END IF;
WHEN sendfile =>
  create_FCB(blk);
  NEW_LINE;
WHEN getfile =>
  IF lcb(ptr).state /= ready THEN
    IF verbose THEN
      PUT("unable to send a file at this ");
      put("time because");
      IF lcb(ptr).state = sending THEN
        PUT(" state of terminal is sending");
      ELSE
        PUT(" state of terminal is dir data");
      END IF;
      NEW_LINE;
    END IF;
    mem(blk).dst := mem(blk).src;
    mem(blk).src := srcprt;
    mem(blk).typ := unable;
    mem(blk).cksum := BYTE(0);
    mem(blk).len(1) := BYTE(0);
    mem(blk).len(2) := BYTE(0);
    mem(blk).cksum :=
      cksum(mem(blk)'ADDRESS,hdr_len);
    put_in_trnsQ(blk);
  ELSE
    lcb(ptr).state := sending;
    lcb(ptr).search := FALSE;
    lcb(ptr).fileopen := FALSE;
    lcb(ptr).endfile := FALSE;
    lcb(ptr).namQ := blk;
    lcb(ptr).line_cnt := 0;
  END IF;
WHEN filedat =>
  receive_file(blk);
WHEN global.EOF =>

305
close_FCB(blk);
prompt;
WHEN whothere =>
  mem(blk).frm_len := 0;
  FOR i IN 1..LENGTH(lcb(head).name) LOOP
    mem(blk).data(i) := conv_byt(lcb(head).name(i));
    mem(blk).frm_len := mem(blk).frm_len + 1;
  END LOOP;
  mem(blk).dst := mem(blk).src;
  mem(blk).typ := ImHere;
  mem(blk).len(1) := hi(mem(blk).frm_len);
  mem(blk).len(2) := lo(mem(blk).frm_len);
  mem(blk).cksum := BYTE(0);
  mem(blk).cksum := cksum(mem(blk).ADDRESS,
   mem(blk).frm_len + hdr_len);
  put_in_trnsQ(blk);
  WHEN ImHere =>
    lcb(ptr).name :=
      arr_to_strg(mem(blk).len(2) 'ADDRESS);
    NEW LINE;
    PUT(ptr); PUT('>' );
    PUT(lcb(ptr).name);
    give_memory(blk);
    prompt;
  WHEN acklast =>
    IF lcb(ptr).state = wait_for_ack THEN
      lcb(ptr).state := sending;
      give_memory(lcb(ptr).filQ);
      lcb(ptr).filQ := 0;
    END IF;
    give_memory(blk);
  WHEN badtrns =>
    PUT("rec'd badtrns");
    IF lcb(ptr).state = wait_for_ack THEN
      lcb(ptr).state := repeatsnd;
    END IF;
    give_memory(blk);
    prompt;
  WHEN unable =>
    PUT("rec'd unable");
    IF lcb(ptr).state = wait_for_ack THEN
      lcb(ptr).state := ready;
      IF lcb(ptr).namQ /= 0 THEN
        give_memory(lcb(ptr).namQ);
        lcb(ptr).namQ := 0;
      END IF;
      IF lcb(ptr).filQ /= 0 THEN
        give_memory(lcb(ptr).filQ);
        lcb(ptr).filQ := 0;
      END IF;
    END IF;
  END IF;
give_memory(blk);
prompt;
WHEN dir =>
  IF 1cb(ptr).state /= receiving
  OR 1cb(ptr).state /= sending THEN
    1cb(ptr).namQ := blk;
    1cb(ptr).state := dir_data;
  END IF;
WHEN dir_data =>
  IF 1cb(ptr).state = talk OR
    1cb(ptr).state = sendfile THEN
    IF 1cb(ptr).rcvQ = 0 THEN
      1cb(ptr).rcvQ := blk;
    ELSE
      add_to_Q(1cb(ptr).rcvQ);
    END IF;
  ELSE
    PUT("directory fr ");
    PUT(ptr); PUT('>' ); NEW_LINE;
  END IF;
ELSE
  PUT("directory fr ");
  PUT(ptr); PUT('>' ); NEW_LINE;
  FOR i IN 1..arr_to_int(mem(blk).len) LOOP
    printdata(mem(blk).data(i));
  END LOOP;
give_memory(blk);
prompt;
END IF;
WHEN code_status =>
  NEW_LINE;
  PUT(" Naval Postgraduate ");
put("School AEGIS Local Area Network");
  NEW_LINE;
  PUT(" programmed by:");
  NEW_LINE;
  PUT(" Robert Hartman and ");
put("Alec Yasinsac");
  NEW_LINE;
  PUT(" advisor: Prof. U ");
put("Kodres");
  NEW_LINE;
PUT("Network Status information follows: Your ");
put("terminal No. is ");
PUT(ptr);
  NEW_LINE;
PUT("Local memory blocks in use/total is ");
PUT(used_blk);
PUT('/');
PUT(max_mem_blk);
  NEW_LINE;
PUT("term pcb state local addr tcp state ");
PUT("term pcb state local addr tcp state");
  NEW_LINE;
  FOR i IN 0..INTEGER(mem(blk).data(1)) LOOP
PUT(i);
IF i < 10 THEN
  PUT(" ");
ELSE
  PUT(" ");
END IF;
CASE mem(blk).data(2+(i*4)) is
  WHEN BYTE(0) => PUT("closed");
  WHEN BYTE(1) => PUT("t_init");
  WHEN BYTE(2) => PUT("telnet");
  WHEN BYTE(3) => PUT("f_init");
  WHEN BYTE(4) => PUT("ftp ");
  WHEN BYTE(5) => PUT("lnet ");
  WHEN BYTE(6) => PUT("1_init");
  WHEN BYTE(7) => PUT("local ");
  WHEN BYTE(8) => PUT("clsing");
  WHEN others => PUT("unkn ");
END CASE;
PUT(" ");
PUT(INTEGER(mem(blk).data(3+(i*4))));
IF (INTEGER(mem(blk).data(3+(i*4)))) < 10 THEN
  PUT(" ");
ELSE IF (INTEGER(mem(blk).data(3+(i*4)))) < 100 THEN
  PUT(" .");
ELSE PUT(" ");
END IF;
END IF;
PUT(INTEGER(mem(blk).data(4+(i*4))));
IF (INTEGER(mem(blk).data(4+(i*4)))) < 10 THEN
  PUT(" ");
ELSE IF (INTEGER(mem(blk).data(4+(i*4)))) < 100 THEN
  PUT(" ");
ELSE PUT(" ");
END IF;
END IF;
CASE mem(blk).data(5+(i*4)) is
  WHEN BYTE(1) => PUT("listen ");
  WHEN BYTE(2) => PUT("syn_snt ");
  WHEN BYTE(3) => PUT("syn_rcv ");
  WHEN BYTE(4) => PUT("estab ");
  WHEN BYTE(5) => PUT("fin_wait_1 ");
  WHEN BYTE(6) => PUT("fin_wait_2 ");
  WHEN BYTE(7) => PUT("close_wait ");
  WHEN BYTE(8) => PUT("closing ");
  WHEN BYTE(9) => PUT("last_ack ");
  WHEN BYTE(10) => PUT("time_wait ");
  WHEN others => PUT("closed ");
END CASE;
IF i rem 2 = 1 THEN
  NEW_LINE;
END IF;
END IF;
END LOOP;
IF verbose THEN
  PUT("number of used blocks/total: ");
  ptr := INTEGER(mem(blk).data(1)) * 4 + 6;
  PUT(INTEGER(mem(blk).data(ptr)));
  PUT('/');
  ptr := ptr + 1;
  PUT(INTEGER(mem(blk).data(ptr)));
  NEW_LINE;
  ptr := ptr + 1;
  PUT("TCBs in use/total: ");
  PUT(INTEGER(mem(blk).data(ptr)));
  PUT('/');
  ptr := ptr + 1;
  PUT(INTEGER(mem(blk).data(ptr)));
  NEW_LINE;
  PUT("Ethernet controller board status follows:");
  NEW_LINE;
  ptr := ptr + 4;
  PUT("Ethernet physical address is ");
  FOR i IN 1..6 LOOP
    PUT(INTEGER(mem(blk).data(ptr)));
    PUT(".");
    ptr := ptr + 2;
  END LOOP;
  NEW_LINE;
  PUT("frames received......................");
  PUT(two_bytes(mem(blk).data(ptr)'ADDRESS));
  NEW_LINE;
  ptr := ptr + 2;
  PUT("frames in receive FIFO................");
  PUT(two_bytes(mem(blk).data(ptr)'ADDRESS));
  NEW_LINE;
  ptr := ptr + 2;
  PUT("frames transmitted...................");
  PUT(two_bytes(mem(blk).data(ptr)'ADDRESS));
  NEW_LINE;
  ptr := ptr + 2;
  PUT("excess collisions....................");
  PUT(two_bytes(mem(blk).data(ptr)'ADDRESS));
  NEW_LINE;
  ptr := ptr + 2;
  PUT("collision fragments received....... ");
  PUT(two_bytes(mem(blk).data(ptr)'ADDRESS));
  NEW_LINE;
  ptr := ptr + 2;
  PUT("lost frames......................... ");
  PUT(two_bytes(mem(blk).data(ptr)'ADDRESS));
  NEW_LINE;
  ptr := ptr + 2;
  PUT("multicast frames accepted......... ");
PUT(two_bytes(mem(blk).data(ptr)"ADDRESS));
NEW_LINE;
ptr := ptr + 2;
PUT("multicast frames rejected............");
PUT(two_bytes(mem(blk).data(ptr)"ADDRESS));
NEW_LINE;
ptr := ptr + 2;
PUT("crc errors........................");
PUT(two_bytes(mem(blk).data(ptr)"ADDRESS));
NEW_LINE;
ptr := ptr + 2;
PUT("alignment errors...................");
PUT(two_bytes(mem(blk).data(ptr)"ADDRESS));
NEW_LINE;
ptr := ptr + 2;
PUT("collisions........................");
PUT(two_bytes(mem(blk).data(ptr)"ADDRESS));
NEW_LINE;
ptr := ptr + 2;
PUT("out-of-window collisions............");
PUT(two_bytes(mem(blk).data(ptr)"ADDRESS));
NEW_LINE;
ptr := ptr + 2;
END IF;
give_memory(blk);
prompt;
WHEN others =>
  IF verbose THEN
    PUT("received unknown type");
  END IF;
give_memory(blk);
prompt;
END CASE;
ELSE
  give_memory(blk);
END IF;
END handle_incoming_packet;

PROCEDURE established is
  loopthrshld : INTEGER;
  thrshld : INTEGER;
  loopcnt : INTEGER;
  local_seg_num : array2;
  foreign_seg_num : array2;
  msgcnt : INTEGER;
  no_send : INTEGER;
  no_rec : INTEGER;
  TYPE inpt is RECORD
    size : BYTE;
    ch : STRING;
  END RECORD;
npt : inpt;

blk : INTEGER;
BEGIN
    PUT("Connection Ready"); NEW_LINE;
    trnsQ := 0;
    used_blk := 0;
    lcb(head).link := prt;
    lcb(head).dest := broadcast;
    lcb(prt).link := head;
    lcb(prt).act := TRUE;
    lcb(head).act := TRUE;
    FOR i IN 1..max_mem_blk - 1 LOOP
        mem_manag_tbl(i) := i + 1;
    END LOOP;
    mem_manag_tbl(max_mem_blk) := 0;
    free_blk := 1;
    quit_received := .FALSE;
    bell_on := .FALSE;
    mailbox := .FALSE;
    verbose := TRUE;
    runfil := .FALSE;
    runfilQ := 0;
    estab := .FALSE;
    logged_in := .FALSE;
    printer := 99;
    lcb(prt).state := log;
    PUT("Login: ");
    LOOP
        -- check for control codes from concentrator
        inport(stat,data);
        IF tstbit(INTEGER(data),RxRdy) THEN
            inport(dat,data);
            CASE data is
                WHEN code_cls =>
                    IF mailbox THEN
                        outport(dat,code_lstn);
                        estab := .FALSE;
                    ELSE
                        EXIT;
                    END IF;
                ELSE
                    EXIT;
                END CASE;
            END IF;
        END LOOP;
        NEW_LINE;
        PUT("Connection Established");
        estab := TRUE;
        IF NOT logged_in THEN
            lcb(head).name := "NO-NAME";
            logged_in := TRUE;
            lcb(head).state := ready;
        END IF;
        prompt;
    END LOOP;
END;
WHEN code_local =>
    output(dat, dst_prt);
WHEN others => null;
END CASE;
END IF;

IF keypress() THEN
    --handle keyboard input
    getch(ch);  
    IF logged_in THEN
        IF estab THEN
            IF runfil = asciicntlQ THEN
                runfil := FALSE;
                IF lcb(prt).fileopen THEN
                    close_file(lcb(prt).FCBb'ADDRESS);
                    lcb(prt).fileopen := FALSE;
                END IF;
                IF lcb(prt).namQ /= 0 THEN
                    give_memory(lcb(prt).namQ);
                    lcb(prt).namQ := 0;
                END IF;
                IF lcb(prt).filQ /= 0 THEN
                    give_memory(lcb(prt).filQ);
                    lcb(prt).filQ := 0;
                END IF;
                lcb(prt).state := ready;
            END IF;
            ELSE
                handle_kybd_input(ch);
            END IF;
        ELSE
            handle_kybd_input(ch);
        END IF;
    ELSE
        CASE ch is
            WHEN asciizero..asciinine =>
                handle_kybd_inputs(ch);
            WHEN asciIP ! ascii_p =>
                handle_kybd_input(ch);
            WHEN asciiCR =>
                handle_kybd_input(ch);
            WHEN asciiQuest =>
                PUT("Information");
                NEW_LINE;
                PUT("?");
                NEW_LINE;
                PUT("Quit");
                NEW_LINE;
                PUT("Netstat");
                NEW_LINE;
                PUT("Print");
                NEW_LINE;
                PUT("<destination>");
                NEW_LINE;
        END CASE
    END IF;
END IF;

312
WHEN asciiI ! ascii_i =>
    handle_kybd_input(ch);
WHEN asciiSpace =>
    handle_kybd_input(ch);
WHEN asciiN ! ascii_n =>
    handle_kybd_input(ch);
WHEN asciiQ ! ascii_q =>
    handle_kybd_input(ch);
WHEN others =>
    PUT("not established, ");
    put("enter destination # or P");
    prompt;
END CASE;
END IF;
ELSE
    CASE ch is
        WHEN asciiA..asciiZ =>
            handle_kybd_input(ch);
        WHEN ascii_a..ascii_z =>
            ch := capital(ch);
            handle_kybd_input(ch);
        WHEN asciiQuest =>
            PUT("Enter your name followed by <CR>");
            NEW LINE;
            PUT(lcb(head).name);
        WHEN asciiBS ! asciiDEL =>
            handle_kybd_input(ch);
        WHEN asciiCR =>
            lcb(prt).state := ready;
            logged_in := TRUE;
            prompt;
        WHEN others =>
            PUT("illegal entry");
            NEW LINE;
            PUT(lcb(head).name);
    END CASE;
    IF length(lcb(head).name) = 20 THEN
        lcb(prt).state := ready;
        logged_in := TRUE;
        NEW LINE;
        PUT("maximum name length is 20");
        prompt;
    END IF;
END IF;
END IF;
EXIT WHEN quit_received;

inport(stat,data);
IF tstbit(INTEGER(data),DSR) AND used_blk /=
max_mem_blk THEN
inport(dat,data);

313
bytcnt := 518;
get_memory(blk);
gle_trns(mem(blk) ADDRESS, dat, bytcnt);
IF bytcnt > 0 THEN
  byt := mem(blk).cksum;
  mem(blk).cksum := BYTE(0);
  mem(blk).frm_len := arr_to_int(mem(blk).len);
  msgcnt := mem(blk).frm_len + hdr_len;
  IF byt /= cksum(mem(blk) ADDRESS, msgcnt) THEN
    PUT("***error in cksum***");
    NEW_LINE;
    mem(blk).cksum := BYTE(1);
  END IF;
  IF msgcnt > bytcnt THEN
    PUT("Entire msg NOT rec'd");
    PUT(" msg len = "); PUT(msgcnt);
    PUT(" byt cnt = "); PUT(bytcnt);
    NEW_LINE;
    give_memory(blk);
  ELSE
    handle_incoming_packet(blk);
  END IF;
ELSE
  give_memory(blk);
END IF;
END IF;

-- poll the LCBs
FOR i IN 0..head LOOP
  CASE lcb(i).state IS
    WHEN sending =>
      IF used_blk < 5 THEN
        send_file(i);
      END IF;
    WHEN repeatsnd =>
      blk := lcb(i).filQ;
      IF blk /= 0 THEN
        send_trns(mem(blk) ADDRESS, dat,
                  mem(blk).frm_len);
        IF mem(blk).frm_len = 0 THEN
          IF lcb(i).dest = broadcast OR
            i = printer THEN
            lcb(i).state := sending;
            give_memory(blk);
            lcb(i).filQ := 0;
          ELSE
            lcb(i).state := wait_for_ack;
          END IF;
        END IF;
      END IF;
  END CASE;
END FOR;
lcb(prt).line_cnt + 1;
IF lcb(prt).line_cnt = 80 THEN
  NEW_LINE;
  lcb(prt).line_cnt := 0;
END IF;
END IF;
ELSE
  lcb(i).state := ready;
END IF;

WHEN dir_data =>
  IF used_blk < 5 THEN
    get_memory(blk);
    lcb(i).filQ := blk;
    send_dir(i);
  END IF;
  WHEN others =>
    null;
END CASE;

IF lcb(i).rcvQ /= 0 AND lcb(prt).state /= talk AND
  lcb(prt).state /= sendfile THEN
  blk := lcb(i).rcvQ;
  lcb(i).rcvQ := mem_manag_tbl(blk);
  handle_incoming_packet(blk);
END IF;
END LOOP;

--send transmissions
IF trnsQ /= 0 THEN
  send_trns(mem(trnsQ)'ADDRESS,dat,
            mem(trnsQ).frm_len);
  IF mem(trnsQ).frm_len = 0 THEN
    blk := trnsQ;
    trnsQ := mem_manag_tbl(blk);
    give_memory(blk);
  END IF;
END IF;
END LOOP;
outport(dat,code_cls);
END established;

BEGIN
  inport(dat,data);  --clear port
  inport(ocwl_reg,org_ocwl);  --save mask 'till end
  outport(ocwl_reg,ocwl);
  outport(cmd,clr);
  clrscreen;
  FOR i IN 0..head LOOP
lcb(i).state := ready;
lcb(i).name := "";
lcb(i).act := FALSE;
lcb(i).sndQ := 0;
lcb(i).rcvQ := 0;
lcb(i).filQ := 0;
lcb(i).dest := BYTE(i);
END LOOP;
outport(dat, code_reqPrt);
LOOP
  inport(stat, data);
  IF tstbit(INTEGER(data), RxRdy) THEN
    inport(dat, data);
    IF data = code_reqPrt THEN
      LOOP
        inport(stat, data);
        IF tstbit(INTEGER(data), RxRdy) THEN
          inport(dat, data);
          EXIT;
        END IF;
      END LOOP;
      PUT("your terminal number is ");
      src prt := data;
      PUT(INTEGER(src prt)); NEW_LINE;
      EXIT;
    ELSE
      outport(dat, code_reqPrt);
      END IF;
  END IF;
END LOOP;
prt := head;
dstprt := BYTE(16#FF#);
outport(dat, code_lstn);
LOOP
  LOOP
    inport(stat, data);
    EXIT WHEN tstbit(INTEGER(data), RxRdy);
  END LOOP;
  inport(dat, data);
  EXIT WHEN data = code_lstn;
  IF data = code cls THEN
    outport(dat, code_lstn);
  ELSE
    outport(dat, code_cls);
  END IF;
END LOOP;
established;
PUT("Connection terminated"); NEW_LINE;
outport(dat, code cls);
outport(ocwl_reg, org_ocwl); --restore state
END locftp;
APPENDIX J

LISTING OF Z-100 MULTI-USE PROGRAMS

package asmlib is

  function byte_to_char (byt: in byte) return character;
  function byte_to_chr (byt: in byte) return character;
  --BYTE_TO_CHR DOES NOT CLEAR BIT SEVEN.
  procedure prntdata(byt : IN byte);
  PROCEDURE getch(chr : OUT BYTE);

  PROCEDURE delete_file(addr : IN INTEGER);
  PROCEDURE create_file(addr : IN INTEGER;
                         rslt : OUT INTEGER);
  PROCEDURE compute_cksum(addr : IN INTEGER;
                           amt : IN INTEGER;cksm : OUT BYTE);
  PROCEDURE write_file(addr : IN INTEGER;
                       succ : OUT BOOLEAN);

  PROCEDURE close_file(addr : IN INTEGER);
  PROCEDURE setDbMA(addr :IN INTEGER);
  PROCEDURE search_first(addr :IN INTEGER;
                         fnd : OUT BOOLEAN);
  PROCEDURE search_next(addr :IN INTEGER;
                       fnd : OUT BOOLEAN);
  PROCEDURE send_trns(addr,data_pr: IN INTEGER;
                       num : IN OUT INTEGER);

  PROCEDURE open_file(addr :IN INTEGER;
                      found : OUT BOOLEAN);
  PROCEDURE read_file(addr :IN INTEGER;
                      rslt : OUT INTEGER);

  FUNCTION current_dsk RETURN BYTE;
  FUNCTION capital(char :IN BYTE) RETURN BYTE;
  FUNCTION lower_case(char :IN character)
                   RETURN character;
  FUNCTION arr_to_strg(addr :IN INTEGER)RETURN string;

  FUNCTION conv_byt(char :IN CHARACTER) RETURN BYTE;
  PROCEDURE get_strg(addr :IN INTEGER);
  PROCEDURE get_trns(addr,data_pr : IN INTEGER;
                    num : IN OUT INTEGER);
  PROCEDURE prnt_buf(addr : IN INTEGER);

  FUNCTION cksum(addr, bytcnt : IN INTEGER)RETURN BYTE;
  function no_echo return byte;

317
FUNCTION two_bytes(addr : IN INTEGER) RETURN INTEGER;

procedure clrscreen;

end asmlib;

--PACKAGE NAME: ASMLIB.ASM
--AUTHOR: ALEC YASINSAC and Robert Hartman
--DATE: JAN 86
--SUBROUTINES CONTAINED: 1. POLLER

Package assembly asmlib is
jmp main  --ASM PACKAGE MUST JUMP ANY CODE NOT INTENDED
          --AS INITIALIZATION CODE.

stat    equ  0edH
cmd     equ  0efH
dat     equ  0ecH
DSR   equ  80H
DTR   equ  27H
clr    equ  25H
TxRdy  equ  1H
RxRdy  equ  2H
rs232_delay equ 400 ;833 usec/byte @ 9600 BAUD
                  ;4 usec/loop

---------------------------------------------------------------
function byte_to_char (byt: in byte) return character is

    pop   bx
    pop   ax
    push  bx
    and   al,7fh

    ret

end byte_to_char;

---------------------------------------------------------------
function byte_to_chr (byt: in byte) return character is

    pop   bx
    pop   ax
    push  bx

    ret

end byte_to_chr;

---------------------------------------------------------------
procedure prntdata(byt : IN BYTE) is

    pop   di
    pop   dx
    push  di
    and   dl,7fh
    mov   ah,02h ; SET AH REG FOR CONSOLE DISPLAY

318
int 21h ; SEND CHAR FROM PORT TO THE CONSOLE
ret
end prntdata;

------------------------------------------------------------------------

procedure getch(char : OUT BYTE) is
  POP ax ;rtn
  POP di ;char
  PUSH di
  PUSH ax
  MOV dl,0ffH
  MOV ah,6
  INT 21H
  MOV [di],al
  RET

end getch;

------------------------------------------------------------------------

PROCEDURE delete_file(addr IN INTEGER) is
  POP ax
  POP dx
  PUSH ax
  MOV ah,13H
  INT 21H
  RET

END delete_file;

------------------------------------------------------------------------

PROCEDURE create_file(addr: IN INTEGER, rslt: OUT INTEGER) is
  POP ax
  POP si
  POP dx
  PUSH dx
  PUSH si
  PUSH ax
  MOV ah,16H
  INT 21H
  MOV ah,0
  MOV [si],ax
  RET

END create_file;

------------------------------------------------------------------------

PROCEDURE compute_cksum(addr : IN INTEGER, amt : IN INTEGER,
                         cksm : OUT BYTE) is
  POP ax
  POP di
  POP cx
  POP si
  PUSH si
  PUSH cx
  PUSH di

319
```

PUSH ax
MOV dx,0
again3: MOV al,[si]
INC si
XOR dl,al
LOOP again3
MOV [di],dl
RET

END compute_cksum;

PROCEDURE write_file(addr: IN INTEGER,succ: OUT BOOLEAN) is
  POP ax ;rtn
  POP di ;succ
  POP dx ;addr
  PUSH dx
  PUSH di
  PUSH ax
  MOV ah,15H
  INT 21H
  CMP al,0
  JZ good
  MOV al,0
  MOV [di],al
  RET

  good: MOV al,1
  MOV [di],al
  RET

END write_file;

PROCEDURE close_file(addr : IN INTEGER) is
  POP ax
  POP dx
  PUSH ax
  MOV ah,10H
  INT 21H
  RET

END close_file;

PROCEDURE setDMA(addr : IN INTEGER) is
  POP ax
  POP dx
  PUSH ax
  MOV ah,1aH
  INT 21H
  RET

END setDMA;

PROCEDURE search_first(addr: IN INTEGER, fnd: OUT BOOLEAN) is
```

320
```assembly
PROCEDURE search_first;

    POP ax
    POP di
    POP dx
    PUSH dx
    PUSH di
    POP ax
    MOV ah,11H
    INT 21H
    CMP al,0ffH
    JE notfnd
    MOV al,1
    MOV [di],al
    RET

notfnd: MOV al,0
    MOV [di],al
    RET

END search_first;

PROCEDURE search_nxt(addr: IN INTEGER, fnd: OUT BOOLEAN) is

    POP ax
    POP di
    POP dx
    PUSH dx
    PUSH di
    POP ax
    MOV ah,12H
    INT 21H
    CMP al,0ffH
    JE notfndl
    MOV al,1
    MOV [di],al
    RET

notfndl: MOV al,0
    MOV [di],al
    RET

END search_nxt;

PROCEDURE send_trns(addr, Data_prt: IN INTEGER, amt: IN OUT INTEGER) is

    wait_time EQU 1000
    POP ax ;rtn
    POP di ;amt
    POP dx ;Data_prt
    POP si ;addr
    PUSH si
    PUSH dx
    PUSH di
    PUSH ax
    INC dx
    IN al,dx
```
AND al, DSR
JNZ send_trnsL5
MOV al, DTR
INC dx
INC dx
OUT dx, al
DEC dx
DEC dx
IN al, dx
AND al, DSR
JNZ send_trnsL5 ;--too soon for DSR
MOV bx, wait_time
MOV cx, [di]

send_trnsL1:
IN al, dx
AND al, DSR
JNZ send_trnsL5
DEC bx
JZ send_trnsD
JMP send_trnsL1

send_trnsL5:
NOP
IN al, dx
AND al, DSR
JZ send_trnsD ;--this was inserted due
JZ send_trnsL2

send_trnsL2:
IN al, dx
AND al, DSR
JZ send_trnsD
MOV al, [si]
DEC dx
OUT dx, al
INC si
INC dx

send_trnsL3:
IN al, dx
AND al, TxRdy
JZ send_trnsL3
LOOP send_trnsL2
LOOP send_trnsL3
MOV [di], cx ;--transmission complete
MOV cx, rs232_delay

send_trnsL4:
NOP
LOOP send_trnsL4

send_trnsD:
MOV al, clr
INC dx
INC dx
OUT dx, al
DEC dx
DEC dx
MOV cx, wait_time

322
send_trnsD1:
   IN   al,dx
   AND  al,DSR
   JZ   send_trnsD2
   LOOP send_trnsD1
send_trnsD2:
   RET
END send_trns;

PROCEDURE open_file(addr: IN INTEGER,found: OUT BOOLEAN) is
   POP   ax
   POP   di
   POP   dx
   PUSH  dx
   PUSH  di
   PUSH  ax
   MOV   ah,0fH
   INT   21H
   CMP   al,0
   JZ    open_fileD
   MOV   al,0
   MOV   [di],al
   RET
open_fileD:
   MOV   al,1
   MOV   [di],al
   RET
END open_file;

PROCEDURE read_file(addr: IN INTEGER, rslt: OUT INTEGER) is
   POP   ax
   POP   di
   POP   dx
   PUSH  dx
   PUSH  di
   PUSH  ax
   MOV   ah,14H
   INT   21H
   MOV   ah,0
   MOV   [di],ax
   RET
END read_file;

FUNCTION current_dsk RETURN BYTE is
   MOV   ah,19H
   INT   21H
   RET
END current_dsk;
FUNCTION capital(char : IN BYTE) RETURN BYTE is
  POP bx
  POP ax
  PUSH bx
  AND al,5fH
  RET
END capital;

FUNCTION lowercase(char: IN character) RETURN character is
  POP bx
  POP ax
  PUSH bx
  OR al,20H
  RET
END lowercase;

FUNCTION conv_byt(char IN CHARACTER) RETURN BYTE is
  POP bx
  POP ax
  PUSH bx
  RET
END conv_byt;

PROCEDURE get_strg(addr : IN INTEGER) is
;--addr points to a buffer whos first byte is its size
;--the second byte has byte count received from the kybd
;--the third byte begins the input string
  POP ax
  POP dx
  PUSH ax
  MOV ah,0aH
  INT 21H
  RET
END get_strg;

PROCEDURE get_trns(addr,Dprt : IN INTEGER,
amt : IN OUT INTEGER) is
  POP ax    ;--rtn
  POP si    ;--num
  POP dx    ;--data_prt
  POP di    ;--addr
  PUSH di
  PUSH dx
  PUSH si
  PUSH ax
  MOV cx,[si]
MOV bx, 0
INC dx
IN al, dx
AND al, DSR
JZ get_prt_dataD
INC dx
INC dx
MOV al, DTR
OUT dx, al
DEC dx
DEC dx
MOV ah, 255
get_prt_dataL:
IN al, dx
AND al, RxRdy
JNZ get_prt_dataL
IN al, dx
AND al, DSR
JZ get_prt_dataD
DEC ah
JNZ get_prt_dataL
get_prt_dataL1:
DEC dx
;-- getting data
IN al, dx
MOV [di], al
INC di
INC bx
INC dx
MOV ah, 255
LOOP get_prt_dataL
get_prt_dataD1:
MOV al, clr
INC dx
INC dx
OUT dx, al
get_prt_dataD:
MOV [si], bx
RET
END get_trns;

---------------------------------------------------------------------

PROCEDURE prnt_buf(addr : IN INTEGER) is
  POP ax
  POP si
  PUSH ax
  MOV cl, [si]
  MOV ch, 0
  INC si
prnt_bufL:
  MOV dl, [si]
  INC si
and    dl,7fH
mov    ah,02h
int    21h
LOOP   prnt_bufL
RET
END prnt_buf;

FUNCTION cksum(addr, bytcnt : IN INTEGER) RETURN BYTE is
  POP  ax
  POP  cx
  POP  si
  PUSH ax
  MOV  al,0
cksml:  MOV  bl,[si]
  XOR  al,bl
  INC  si
  LOOP cksml
RET
END cksum;

function no_echo return byte is
  ; PROCEDURE TO ALLOW A USER TO ENTER HIS PASSWORD
  ; WITHOUT ECHO TO THE CONSOLE.
    pop    dx
    mov    ah,8  ; SET FOR NO ECHO FUNC INTERRUPT.
    int    21h
    push   dx
    ret
end no_echo;

FUNCTION arr_to_strg(addr IN INTEGER) RETURN string is
  POP  bx
  POP  ax
  PUSH bx
RET
END arr_to_strg;

FUNCTION two_bytes(addr : IN INTEGER) RETURN INTEGER is
  POP  bx
  POP  si
  PUSH bx
  MOV  ax,[si]
RET
END two_bytes;
procedure clrscreen is
  mov ah, 02h
  mov dl, lbh
  int 21h
  mov dl, 45h
  int 21h
  ret
end clrscreen;

main: -- ANY INITIALIZATION CODE WOULD FOLLOW THIS LABEL
end asmlib;
package get_ip is
  procedure get_addr(ip1, ip2, ip3, ip4: out integer);
end get_ip;

package body get_ip is

procedure get_addr(ip1, ip2, ip3, ip4: out integer) is
  -- INPUT: HOSTS.FIL
  -- OUTPUT: INTERNET PROTOCOL ADDRESS
  -- DESCRIPTION: GET_IP PRINTS THE CONTENTS OF THE FILE 'HOSTS.FIL' AND
  -- ASSOCIATES WITH IT A SELECTOR NUMBER. THE USER IS
  -- PROMPTED TO SELECT HIS DESTINATION BY KEYING IN A NUM-
  -- BER. GET ADDR THEN INTERPRETS THE ADDR AND RETURNS
  -- THE SELECTED ADDRESS TO THE CALLING ROUTINE.
  use IO, strlib;
type iprec is array (1..40) of integer;
inaddr1, inaddr2, inaddr3, inaddr4 : iprec;
iname: string(21);
selection, ctr: integer;
infile, outfile, hosts, con: file;
badinp: boolean;
inp: string;
buf, k : integer;
begin -- begin procedure get_addr
  ctr := 0;
  new line;
  put("THE FOLLOWING IS THE LIST");
put(" OF DESTINATIONS AVAILABLE. ");
new_line;
put("0  TERMINATE PROCESS."); new_line;
open(hosts,"hosts.fil",read_only);
while not end_of_file(hosts) loop
  ctr := ctr + 1;
  get (hosts, inaddr1(ctr));
  get (hosts, inaddr2(ctr));
  get (hosts, inaddr3(ctr));
  get (hosts, inaddr4(ctr));
  inname := " ";
  k := 1;
  while (not end_of_line(hosts) and k < 21 ) loop
    read (hosts, inname(k));
    k := k + 1;
  end loop;
  skip_line(hosts);
  new_line;
  put(ctr);
  put(" ");
  put(inname);
  put(" ");
  put(inaddr1(ctr));
  put(" ");
  put(inaddr2(ctr));
  put(" ");
  put(inaddr3(ctr));
  put(" ");
  put(inaddr4(ctr));
end loop;
close (hosts);
selection := ctr + 1; new_line;
loop -- VALIDATE INPUT HERE
  put("ENTER A NUMBER BETWEEN 0 AND ");
  put(ctr); put("."); new_line;
  put("ENTER ZERO TO TERMINATE PROCESS."); new_line;
  badinp := false; new_line;
  put(">> ");
  inp := get_line();
  -- PRESUME THERE WILL NOT BE MORE THAN
  -- 99 POSSIBLE REMOTE HOSTS. MUST CHECK
  -- FOR THE POSSIBILITY OF TWO DIGITS.
  for i in 1..length(inp) loop
    if not (inp(i) in '0'..'9') then
      badinp := true;
      exit;
    end if;
  end loop; --ENDS FOR LOOP.
  if not badinp then
    selection := str_to_int(inp);
    if selection <= ctr then
      exit;
    end if;
  end if;
end loop;
if (selection = 0) then
    ip1 := 0; ip2 := 0; ip3 := 0; ip4 := 0;
    -- BY CONVENTION, THE IP ADDRESS RETURNED = ZERO
    -- INDICATES USER TERMINATION.
else
    ip1 := inaddr1(selection);
ip2 := inaddr2(selection);
ip3 := inaddr3(selection);
ip4 := inaddr4(selection);
end if;  -- selection = 0
end get_addr;
end get_ip;

-- this program is used to download a program into the
-- concentrator for the AEGIS LAN.

with bootasm, bit;
PROCEDURE boot is
use bootasm, bit;

threshold : constant integer := 10000;
ocwl_reg : constant integer := (16#00f3#);
ocwl : constant byte := byte(16#aa#);
code_download : constant byte := byte(16#c4#);
code_end : constant byte := byte(16#FF#);
dat : constant integer := (16#EC#);
stat : constant integer := (16#ED#);

-- this program is used to download a program into the
-- concentrator for the AEGIS LAN.

with bootasm, bit;
PROCEDURE boot is
use bootasm, bit;

threshold : constant integer := 10000;
ocwl_reg : constant integer := (16#00f3#);
ocwl : constant byte := byte(16#aa#);
code_download : constant byte := byte(16#c4#);
code_end : constant byte := byte(16#FF#);
dat : constant integer := (16#EC#);
stat : constant integer := (16#ED#);

-- this program is used to download a program into the
-- concentrator for the AEGIS LAN.

with bootasm, bit;
PROCEDURE boot is
use bootasm, bit;

threshold : constant integer := 10000;
ocwl_reg : constant integer := (16#00f3#);
ocwl : constant byte := byte(16#aa#);
code_download : constant byte := byte(16#c4#);
code_end : constant byte := byte(16#FF#);
dat : constant integer := (16#EC#);
stat : constant integer := (16#ED#);

-- this program is used to download a program into the
-- concentrator for the AEGIS LAN.

with bootasm, bit;
PROCEDURE boot is
use bootasm, bit;

threshold : constant integer := 10000;
ocwl_reg : constant integer := (16#00f3#);
ocwl : constant byte := byte(16#aa#);
code_download : constant byte := byte(16#c4#);
code_end : constant byte := byte(16#FF#);
dat : constant integer := (16#EC#);
stat : constant integer := (16#ED#);

-- this program is used to download a program into the
-- concentrator for the AEGIS LAN.

with bootasm, bit;
PROCEDURE boot is
use bootasm, bit;

threshold : constant integer := 10000;
ocwl_reg : constant integer := (16#00f3#);
ocwl : constant byte := byte(16#aa#);
code_download : constant byte := byte(16#c4#);
code_end : constant byte := byte(16#FF#);
dat : constant integer := (16#EC#);
stat : constant integer := (16#ED#);

-- this program is used to download a program into the
-- concentrator for the AEGIS LAN.

with bootasm, bit;
PROCEDURE boot is
use bootasm, bit;

threshold : constant integer := 10000;
ocwl_reg : constant integer := (16#00f3#);
ocwl : constant byte := byte(16#aa#);
code_download : constant byte := byte(16#c4#);
code_end : constant byte := byte(16#FF#);
dat : constant integer := (16#EC#);
stat : constant integer := (16#ED#);

-- this program is used to download a program into the
-- concentrator for the AEGIS LAN.

with bootasm, bit;
PROCEDURE boot is
use bootasm, bit;

threshold : constant integer := 10000;
ocwl_reg : constant integer := (16#00f3#);
ocwl : constant byte := byte(16#aa#);
code_download : constant byte := byte(16#c4#);
code_end : constant byte := byte(16#FF#);
dat : constant integer := (16#EC#);
stat : constant integer := (16#ED#);

-- this program is used to download a program into the
-- concentrator for the AEGIS LAN.

with bootasm, bit;
PROCEDURE boot is
use bootasm, bit;

threshold : constant integer := 10000;
ocwl_reg : constant integer := (16#00f3#);
ocwl : constant byte := byte(16#aa#);
code_download : constant byte := byte(16#c4#);
code_end : constant byte := byte(16#FF#);
dat : constant integer := (16#EC#);
stat : constant integer := (16#ED#);

-- this program is used to download a program into the
-- concentrator for the AEGIS LAN.

with bootasm, bit;
PROCEDURE boot is
use bootasm, bit;

threshold : constant integer := 10000;
ocwl_reg : constant integer := (16#00f3#);
ocwl : constant byte := byte(16#aa#);
code_download : constant byte := byte(16#c4#);
code_end : constant byte := byte(16#FF#);
dat : constant integer := (16#EC#);
stat : constant integer := (16#ED#);

-- this program is used to download a program into the
-- concentrator for the AEGIS LAN.

with bootasm, bit;
PROCEDURE boot is
use bootasm, bit;

threshold : constant integer := 10000;
ocwl_reg : constant integer := (16#00f3#);
ocwl : constant byte := byte(16#aa#);
code_download : constant byte := byte(16#c4#);
code_end : constant byte := byte(16#FF#);
dat : constant integer := (16#EC#);
stat : constant integer := (16#ED#);

-- this program is used to download a program into the
-- concentrator for the AEGIS LAN.

with bootasm, bit;
PROCEDURE boot is
use bootasm, bit;

threshold : constant integer := 10000;
ocwl_reg : constant integer := (16#00f3#);
ocwl : constant byte := byte(16#aa#);
code_download : constant byte := byte(16#c4#);
code_end : constant byte := byte(16#FF#);
dat : constant integer := (16#EC#);
stat : constant integer := (16#ED#);

-- this program is used to download a program into the
-- concentrator for the AEGIS LAN.

with bootasm, bit;
PROCEDURE boot is
use bootasm, bit;

threshold : constant integer := 10000;
ocwl_reg : constant integer := (16#00f3#);
ocwl : constant byte := byte(16#aa#);
code_download : constant byte := byte(16#c4#);
code_end : constant byte := byte(16#FF#);
dat : constant integer := (16#EC#);
stat : constant integer := (16#ED#);

-- this program is used to download a program into the
-- concentrator for the AEGIS LAN.

with bootasm, bit;
PROCEDURE boot is
use bootasm, bit;

threshold : constant integer := 10000;
ocwl_reg : constant integer := (16#00f3#);
ocwl : constant byte := byte(16#aa#);
code_download : constant byte := byte(16#c4#);
code_end : constant byte := byte(16#FF#);
dat : constant integer := (16#EC#);
stat : constant integer := (16#ED#);

-- this program is used to download a program into the
-- concentrator for the AEGIS LAN.

with bootasm, bit;
PROCEDURE boot is
use bootasm, bit;

threshold : constant integer := 10000;
ocwl_reg : constant integer := (16#00f3#);
ocwl : constant byte := byte(16#aa#);
code_download : constant byte := byte(16#c4#);
code_end : constant byte := byte(16#FF#);
dat : constant integer := (16#EC#);
stat : constant integer := (16#ED#);

-- this program is used to download a program into the
-- concentrator for the AEGIS LAN.

with bootasm, bit;
PROCEDURE boot is
use bootasm, bit;

threshold : constant integer := 10000;
ocwl_reg : constant integer := (16#00f3#);
ocwl : constant byte := byte(16#aa#);
code_download : constant byte := byte(16#c4#);
code_end : constant byte := byte(16#FF#);
dat : constant integer := (16#EC#);
stat : constant integer := (16#ED#);
BEGIN

inport (dat, data);
inport (ocwl_reg, org_ocwl);
outport (ocwl_reg, ocwl);
outport (cmd, DTR);
outport (dat, code_download);
loopcnt := 0;
PUT("Welcome to the Naval Postgraduate School's ");
PUT("Computer Science Lab");
NEW_LINE;
LOOP
  inport (stat, data);
  IF tstbit (INTEGER(data), RxRdy) THEN
    inport (dat, data);
    EXIT WHEN data = code_download;
    RETURN;
  END IF;
  loopcnt := loopcnt + 1;
  IF loopcnt = threshold THEN
    RETURN;
  END IF;
END LOOP;
PUT("Please standby ...");
NEW_LINE;
FCB.drv := BYTE(0);
FCB.name(1) := BYTE(16#43#); -- C
FCB.name(2) := BYTE(16#4F#); -- O
FCB.name(3) := BYTE(16#4E#); -- N
FCB.name(4) := BYTE(16#54#); -- T
FCB.name(5) := BYTE(16#52#); -- R
FCB.name(6) := BYTE(16#54#); -- O
FCB.name(7) := BYTE(16#4C#); -- L
FCB.name(8) := BYTE(16#20#); --
FCB.ext(1) := BYTE(16#50#); -- P
FCB.ext(2) := BYTE(16#52#); -- R
FCB.ext(3) := BYTE(16#47#); -- G

open_file (FCB'address, found);
IF found THEN
  FCB.extnt := 0;
  FCB.rec := BYTE(0);
  FCB.Rsize := 512;
  setDMA (buf'ADDRESS);
  linecnt := 0;

330
LOOP
  read_file(FCB'ADDRESS,rslt);
  IF rslt = 0 OR rslt = 3 THEN
    send(buf'ADDRESS);
    PUT('*');
    linecnt := linecnt + 1;
    IF linecnt = 80 THEN
      NEW_LINE;
    END IF;
    EXIT WHEN rslt = 3;
  ELSE
    close_file(FCB'address);
    EXIT;
  END IF;
END LOOP;
NEW_LINE;
FOR i IN 1..4 LOOP
  LOOP
    inport (stat, data);
    EXIT WHEN tstbit(INTEGER(data),TxRdy);
  END LOOP;
  outport(dat,code_end);
END LOOP;
  PUT("Download to concentrator complete"); NEW_LINE;
ELSE
  PUT("'CONTROL.PRG' not found on current drive");
  NEW_LINE;
END IF;
END boot;

Package assembly bootasm is
jmp main -------ASM PACKAGE MUST JUMP ANY CODE NOT
-------INTENDED AS INITIALIZATION CODE.

stat equ 0edH
cmd equ 0efH
dat equ 0eCH
TxRdy equ 1h
RxRdy equ 2H

-----------------------------------------------
PROCEDURE close_file(addr : IN INTEGER) is
  POP ax
  POP dx
  PUSH ax
  MOV ah,10H
  INT 21H
  RET
END close_file;

-----------------------------------------------
PROCEDURE setDMA(addr : IN INTEGER) is
```plaintext
            POP    ax
            POP    dx
            PUSH   ax
            MOV     ah,1aH
            INT    21H
            RET
END setDMA;

PROCEDURE open_file(addr: IN INTEGER, found: OUT BOOLEAN) is
    POP    ax
    POP    di
    POP    dx
    PUSH   dx
    PUSH   di
    PUSH   ax
    MOV     ah,0fH
    INT    21H
    CMP     al,0
    JZ      open_fileD
    MOV     al,0
    MOV     [di],al
    RET
open_fileD:
    MOV     al,1
    MOV     [di],al
    RET
END open_file;

PROCEDURE read_file(addr: IN INTEGER, rslt: OUT INTEGER) is
    POP    ax
    POP    di
    POP    dx
    PUSH   dx
    PUSH   di
    PUSH   ax
    MOV     ah,14H
    INT    21H
    MOV     ah,0
    MOV     [di],ax
    RET
END read_file;

PROCEDURE send(addr : IN INTEGER) is
    POP    ax
    POP    si
    PUSH   ax
    MOV     cx,512
sendL:    IN     al,stat
          AND    al,TxRdy
          JZ     sendL
RESET    sendL
```

332
MOV al,[si]
OUT dat,al
INC si
LOOP sendL
RET
END send;

main: -- ANY INITIALIZATION CODE WOULD FOLLOW THIS LABEL
end bootasm;
1. Communication

Communication is viewed as inter-process communication, even if it is to and from a terminal or printer.

2. Datagram

A datagram is a group of characters or bytes entailing a message combined with the source and destination address of the message. Datagram may also refer to a type of network service in which each message is handled as an isolated entity.

3. FTP, IP, TCP, TELNET

Each of these terms represent a documented network protocol. 'Telecommunications Control Protocol', 'Internet Protocol', 'File Transfer Protocol', and 'TELNET Protocol' each provide at least one of the ISO standard layers of protocol as described by Tannebaum [Ref. 2]. These protocols are specified in [Ref. 3].

4. Hosts

Hosts are computers connected to a network and are the originators and receivers of information as far as the networks are concerned.
5. LAN

LAN is an acronym for Local Area Network and is used to represent any network operating exclusively within a low radius region.

6. MULTIBUS

The AEGIS multi-user system is built with a Multibus frame which allows multiple SBC's to communicate directly with common memory within the frame.

7. Networks

Networks can be either local networks like ethernet or large networks like ARPANET.

8. NPS

This is an acronym for Naval Postgraduate School, Monterey, California.

9. Octet

An octet is a grouping of eight data bits.

10. Packets

Packets is a term used to mean a set of data for one transaction between a host and its network. A packet can mean just a few bytes to several thousand bytes. They are transferred over a network as a group unless fragmentation occurs which we will discuss later.

11. Ports

Ports are channels through which processes communicate. A process may have many ports or just one.
(ie. a non-sharable asset like a printer has only one port).

12. Process

Processes are active elements in a host computer (ie, a program in execution).

13. SMTP

Simple Mail Transfer Protocol (SMTP) is used to pass mail across the network (rfc821)

14. Single Board Computer (SBC)

A single board computer is a configuration of VLSI circuitry on one computer board capable of performing the functions of a computer. When this term is used in the thesis it is usually referring to the Intel 86/12A SBC which is the driving force of the AEGIS multi-user system.

15. TAC

TAC (terminal access controller) is a way of accessing a network by connecting a hard-wire or dial-up phone connection to the controller for access to a network without going through a host computer. The TAC's are positioned around the country to allow fairly short phone connections to the world wide network.

16. USART

A USART is a microprocessor that provides communication interface between computers or between a computer and a peripheral device.
17. Z-100

This is the specific model name for the microcomputers used in this network configuration. The vendor is Zenith Data Systems.
LIST OF REFERENCES

1. Reeke, D. R., Remote Terminal Login From a Microcomputer to the UNIX Operating System Using Ethernet As the Communications Medium, M. S. Thesis, Naval Postgraduate School, Monterey, California, December, 1984.


# INITIAL DISTRIBUTION LIST

<table>
<thead>
<tr>
<th>No.</th>
<th>Copies</th>
<th>Distribution Details</th>
</tr>
</thead>
</table>
| 1.  | 2      | Defense Technical Information Center  
|     |        | Cameron Station  
|     |        | Alexandria, Virginia 22314-6145 |
| 2.  | 2      | Library, Code 0142  
|     |        | Naval Postgraduate School  
|     |        | Monterey, California 93943-5000 |
| 3.  | 1      | Department Chairman, Code 52  
|     |        | Department of Computer Science  
|     |        | Naval Postgraduate School  
|     |        | Monterey, California 93943-5000 |
| 4.  | 9      | Professor Uno R. Kodres, Code 52K  
|     |        | Department of Computer Science  
|     |        | Naval Postgraduate School  
|     |        | Monterey, California 93943-5000 |
| 5.  | 4      | Capt. Alec F. Yasinsac  
|     |        | Rte. 2 Box 346  
|     |        | Sparta, N. C. 28675 |
| 6.  | 3      | Lt. Cmdr Robert L. Hartman  
|     |        | VF/A 161  
|     |        | FPO San Francisco, California 96631 |
| 7.  | 1      | Computer Technology Programs  
|     |        | Code 37  
|     |        | Naval Postgraduate School  
|     |        | Monterey, California 93943-5000 |
| 8.  | 1      | Lt. Col. John D. Reeke, USMC  
|     |        | 9547 University Avenue  
|     |        | Des Moines, Iowa 50322 |
| 9.  | 1      | Professor Bruce J. McLennan, Code 52M  
|     |        | Department of Computer Science  
|     |        | Naval Postgraduate School  
|     |        | Monterey, California 9393-50000 |
| 10. | 1      | Professor Gordon E. Latta, Code 53L  
|     |        | Department of Mathematics  
|     |        | Naval Postgraduate School  
|     |        | Monterey, California 93943-5000 |

339
11. Lt. Joann Ammann
Naval Security Group Activity
Skaggs Island, Sonoma, California  95476