MULTIPLEXING THE ETHERNET INTERFACE AMONG VME/VME-686 USERS
(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA
A K SAKELLAROPOULOS ET AL. DEC 83
NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

MULTIPLEXING THE ETHERNET INTERFACE AMONG VAX/VMS USERS

by

Antonios K. Sakellarakopoulos
and
Ioannis K. Kidoniefs

December 1983

Thesis Advisor: Uno R. Kodres

Approved for public release; distribution unlimited
This thesis focuses on the multiplexing of Ethernet interface among VAX 11/780 users. Since there is only one channel that connects VAX 11/780 system to Ethernet Local Area Network, multiplexing of the NI1010 Unibus Ethernet Communication Controller is necessary in order to service multiple VAX users concurrently via Ethernet. Described herein is a Time Division Multiplexing (Continued)
ABSTRACT (Continued)

The developed software enables those users to communicate in any combination with one or more computer systems, which are connected to Ethernet as well. Two Microcomputer Development Systems (MDS) and VAX/VPS systems were used for the implementation and testing of the project. The software is designed in such a way that those MDS's act very much like virtual VAX/VMS terminals. The whole system can easily be expanded to serve more than nine users.
Multiplexing the Ethernet Interface
Among VAX/VMS Users

by

Antonios K. Sakellaropoulos
Major, Hellenic Air Force
B.S., Hellenic Air Force Academy, 1972

and

Ionnas K. Kidoris
Lieutenant, Hellenic Navy
B.S., Hellenic Naval Academy, 1975

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE
MASTER OF SCIENCE IN ENGINEERING SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1983

Authors: ____________________________

Approved by: ____________________________

Thesis Advisor: ____________________________

Second Reader: ____________________________

Chairman, Department of Computer Science

Dean of Information and Policy Sciences
ABSTRACT

This thesis focuses on the multiplexing of Ethernet interface among VAX 11/780 users.
Since there is only one channel that connects VAX 11/780 system to Ethernet Local Area Network, multiplexing of the NI1010 Unibus Ethernet Communication Controller is necessary in order to service multiple VAX users concurrently via Ethernet.

Described herein is a Time Division Multiplexing Of NI1010 controller, which can serve up to nine (9) separate VAX users. The developed software enables those users to communicate in any combination with one or more computer systems, which are connected to Ethernet as well.

Two Microcomputer Development Systems (MDS) and VAX/VHS system were used for the implementation and testing of the project. The software is designed in such a way that those MDS's act very much like virtual VAX/VHS terminals.

The whole system can easily be expanded to serve more than nine users.
# TABLE OF CONTENTS

I. INTRODUCTION .................................................. 10
   A. DISCLAIMER .................................................. 10
   B. CONVENTIONS .............................................. 10
   C. STRUCTURE OF THE THESIS .............................. 10
   D. THESIS OBJECTIVES ..................................... 12

II. THESIS BACKGROUND ............................................. 14
   A. SYSTEM LAYOUT ........................................... 14
   B. PREVIOUS WORK ........................................... 14
      1. HSU - Ethernet Communication ..................... 14
      2. VAI - Ethernet Communication ..................... 14

III. RELEVANT INFORMATION ....................................... 16
   A. MULTIPLEXING .............................................. 16
   B. PROCESSES ................................................ 16
      1. PROCESS Definition .................................. 16
      2. JOB Definition ....................................... 21
      3. IMAGE Definition .................................... 21
      4. Types of Processes ................................... 21
   C. INTERPROCESS COMMUNICATION .......................... 22
      1. Event Flag Services ................................. 22
      2. Input/Output Services .............................. 26
      3. Time and Time Conversion Services .............. 31

IV. DESIGN CONCEPT ............................................... 35
   A. GENERAL .................................................. 35
      1. EVOLUTION OF THE DESIGN ............................ 35
      2. LANGUAGE SELECTION .................................. 35
      3. PRICE SIZE ........................................... 36
   B. HIGH LEVEL DESIGN ..................................... 37
V. DETAILED DESIGN OF THE MULTIPLEXING SYSTEM
   A. PROGRAM 'ETHERNUT' DETAILED DESIGN
      1. Variables and Data Structure
      2. Initializations
      3. Body Body
   B. 'ETHERNUT' DETAILED DESIGN
      1. Variables and Data Structure
      2. Body Body
      3. Subroutine "Initiate"
      4. Subroutine "Swap"

VI. CONCLUSIONS
   A. PRESENT DESIGN
   B. FUTURE DESIGN

APPENDIX A: SYSTEM SERVICES AND AREA-TYPE LIBRARY ROUTINES
   A. CALLING THE SYSTEM SERVICES
      1. Function Calls
      2. Function Arguments
      3. Testing System Status Codes
   B. SYSTEM SERVICES AND LIBRARY ROUTINES USED IN THE PROGRAM
      1. System Service Routines

APPENDIX B: ETHERNET LOCAL AREA NETWORK

APPENDIX C: D1010 BOARD DESCRIPTION-FEATURES

APPENDIX D: D1010 ETHERNET CONTROLLED MULTIPLEXING
   USER'S MANUAL
   A. GENERAL INFORMATION
   B. SPECIFIC INFORMATION
1. Operation of SDS Systems .......................... 111

APPENDIX B: ETHERNET .................................. 115

APPENDIX C: TEKTRONIX .................................. 127

APPENDIX D: SOFTWARE PROTOCOL IN SDS USING ETHERNET

APPENDIX E: HIGH LEVEL DESIGN OF A VIRTUAL TERMINAL

APPENDIX F: PROGRAM "LOGIC" ............................ 151

LIST OF REFERENCES ...................................... 153

INITIAL DISTRIBUTION LIST .............................. 153
LIST OF TABLES

I. Event Flag Clusters ........................................ 24
II. Summary of Event Flag Services ............................ 25
III. Baggage Services ........................................... 27
IV. Summary of I/O Services .................................... 28
V. Basic Flows and Flow Conversion Services ............... 34
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>System Layout</td>
<td>15</td>
</tr>
<tr>
<td>3.1</td>
<td>Layout of Process Virtual Address Space</td>
<td>20</td>
</tr>
<tr>
<td>3.2</td>
<td>Access Modes and AST Delivery</td>
<td>29</td>
</tr>
<tr>
<td>3.3</td>
<td>Example of an AST</td>
<td>30</td>
</tr>
<tr>
<td>3.4</td>
<td>Two processes accessing a mailbox</td>
<td>32</td>
</tr>
<tr>
<td>4.1</td>
<td>Use of Event Flags</td>
<td>40</td>
</tr>
<tr>
<td>4.2</td>
<td>Process Communication Via Buffers</td>
<td>61</td>
</tr>
<tr>
<td>4.3</td>
<td>Message Traffic inside the VAS</td>
<td>64</td>
</tr>
<tr>
<td>5.1</td>
<td>Hierarchy Diagram of Process &quot;Server1&quot;</td>
<td>64</td>
</tr>
<tr>
<td>5.2</td>
<td>Hierarchy Diagram of Process &quot;Client1&quot;</td>
<td>64</td>
</tr>
<tr>
<td>6.1</td>
<td>Layout of Status Flags (IO Registers)</td>
<td>67</td>
</tr>
<tr>
<td>6.2</td>
<td>I/O Status Flags</td>
<td>67</td>
</tr>
<tr>
<td>6.3</td>
<td>The Local Area between Solver Selection tree</td>
<td>106</td>
</tr>
<tr>
<td>6.4</td>
<td>Selecting the correct Domain</td>
<td>105</td>
</tr>
</tbody>
</table>
I. INTRODUCTION

A. 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 will be listed below, followed by the firm holding the trademark:

- Ethernet
- Netz Corporation
- Unix, VAX, VMS
- Digital Equipment Corporation
- DEC

B. CONVENTIONS

For reasons of convenience and readability, some of the widely used terms in this thesis will be referred to henceforth as follows:

- VAX/VMS will imply the VAX 11/700 system operating under Virtual Memory System (VMS)
- Ethernet will stand for Ethernet Local Area Network
- E 10/10 will stand for E 10/10 Series Ethernet Communication Controller (ECC) Ethernet Interface

C. STRUCTURE OF THE THESIS

The rest of Chapter I gives a general development history of the project and describes the objectives of this thesis.
Chapter II provides the system layout as it currently exists. It describes how the VAX and the MDS's systems are connected via Ethernet and refers to the previous relevant work which has been performed by other students.

In Chapter III information which is very useful and sometimes necessary in order to understand this thesis, is provided. These pieces of information were found in a variety of references and it would be very painful for the reader to try to locate them on his own.

The material up to and including Chapter III constitutes the background for the understanding of this thesis. In Chapter IV however, the real work starts with the high level design of the Ethernet multiplexing. In this chapter the design concept is described, and justification for the decisions that were made is given.

Chapter V contains the detailed design and implementation of the project. In reality Chapter V constitutes the documentation of the developed software. In some instances, things that have been mentioned in previous parts of this thesis are repeated, when it was thought that they are necessary for the thorough understanding of the program.

Chapter VI contains the conclusion of this work.

This thesis is also supported by several appendices.

Appendix A provides a description of VAX/VMS system services and the Real Time Library routines which were used in the program, along with information on their use.

Appendix B provides general information about the Ethernet local area network.

Appendix C gives a short description of XI010 controller which constitutes the VAX - Ethernet interface.


Appendices E and F provide the two programs which compose this project (see Chap. IV D).
Appendix I includes the modified programs of a previous work by Mark Stotzer (see Chap. II B.1) with a brief explanation of the changes.

Appendix II provides a high level design of a virtual terminal network, along with some hints which may be useful to the person who will work in this area.

Finally in Appendix I is given the program "LOGER" which creates a detached process to run the image "LOGINOUT.EXE". More information about this effort can be found in Chapter II.

1. THESIS OBJECTIVES

Once this thesis started, the objective was to create a concept of virtual terminals for the VAX/VMS system. More specifically, two DCS microcomputers which were connected to the Ethernet local area network, had to be made to act like virtual terminals of VAX/VMS which is also connected to the network.

After most of the work was done and the greatest part of the project had been designed and implemented, all that was left was the invocation of the Loginout procedure [Ref. 1] from an DCS terminal. But at that point it was discovered that under the current design of the VAX/VMS Loginout procedure, it is impossible to invoke this procedure by anything else except the physically connected VAX terminal. So the original goal was changed.

The new objective was the multiplexing of the Ethernet interface among VAX/VMS users i.e. NI1010 controller (see Appendix C). In other words, how it could be possible for different users of VAX/VMS to use concurrently the unique terminal via which VAX is connected to Ethernet.

Since the original and the final objectives were in the same general direction, a large part of the original design
was maintained. The developed software finally provides a
degree of virtual terminal service. The deviation from full
virtual terminal service is that currently the Loginout-
procedure must be executed from a VAX terminal, and the
program that is responsible for the execution of commands
entered from an MDS terminal must also run from a VAX/VMS
terminal.

Since we need to occupy one real VAX terminal, in order
to use an MDS as a virtual terminal, there is no practical
usefulness in the virtual terminal service as it currently
exists. However the Digital Equipment corporation is
working on the modification of the Loginout procedure. As
soon as this modified version of the Loginout procedure
becomes available, this thesis can be relatively easily
modified so that full virtual terminal service will be
achieved.

In Appendix H a high level design of a virtual terminal
network is provided, which may be useful to the person who
will undertake this task when the Loginout problem will be
eliminated.
II. THESIS BACKGROUND

Before the development of the present thesis, Stotzer [Ref. 2] and Metniyom [Ref. 3], had worked on the communication interface between MDS microcomputers and VAX/VMS. Specifically they worked on transferring single messages and files from VAX to MDS and vice versa.

A. SYSTEM LAYOUT

In order that communication between the VAX/VMS and the MDS systems to be achieved, both of them were connected to Ethernet local area network. Figure 2.1 depicts how the MDS's and the VAX are currently connected. This configuration existed on the fifth floor of the Spanagel Hall at Naval Postgraduate School, when this thesis was being developed.

One single density and one double density MDS are connected to the Ethernet. Each one of them is equipped with an NI3010 board [Ref. 4], which contains all the data communication controller logic required for interfacing those microcomputers to the Ethernet.

Similarly between the VAX and the Ethernet stands an NI1010A Unibus Ethernet communication controller [Ref. 5], which is also a single board that contains the required logic for interfacing the VAX to Ethernet.

Appendix B contains brief information about Ethernet. A short description of NI1010 controller is given in Appendix C.
Figure 2.1 System Layout.
B. PREVIOUS WORK

1. MDS - Ethernet Communication

Stotzer had undertaken the part which deals with the software interface between the MDS and the Ethernet. Stotzer's programs were modified in order to meet the needs of a continuous, uninterrupted, MDS-VAX communication, because in his original programs, after the transmission of a single message, one had to invoke the program again each time.

Since a part of this thesis was developed concurrently with Stotzer's thesis, several suggestions were made to him, and he redesigned his programs so that they could be used for the purposes of this thesis.

In spite of the new effort, after Stotzer's thesis was completed and he had departed, very few but crucial changes were made to his software, in order it to serve completely the objectives of this thesis. In Appendix G the modified Stotzer's programs are provided.

2. VAX - Ethernet Communication

Netniyom worked on the interface between the VAX and the Ethernet. His programs made possible the receiving and sending of messages and files from and to the NI1010 board.

According to his design, a process is continuously monitoring the NI1010 and as soon as it receives a message, it sends an acknowledge to the source of the message, and displays this message on the terminal.

This process can serve only one VAX user at a time and no other process can establish access to the NI1010 unless the previous one has terminated, freeing the unique channel of communication with the NI1010 and consequently with the Ethernet.

16
In contrast to Stotzer's programs which were used almost unchanged, Nakiiyon's software was not used. However, his work was studied most carefully by the writers of this thesis, who obtained the initial knowledge of the VAX-Ethernet communication from his document.
III. RELEVANT INFORMATION

Before proceeding with the design of the multiplexing system of the Ethernet interface among VAX - 11 users, it would be more convenient for the reader if some relevant background information was provided. In this way the reader will save the trouble to find this information on his own and thus discrepancies in assumptions will be avoided.

A. MULTIPLEXING

In general terms the word multiplexing refers to the use of a single facility to handle concurrently several similar but separate operations. The main use of multiplexing however, is in the field of data communications, where it is used for the transmission of several lower speed data streams, over a single higher speed line.

In the context of this project, multiplexing of the Ethernet interface implies a scheme under which many VAX users share the unique channel of communication between VAX and Ethernet.

Multiplexing is divided into two basic categories: 
Frequency Division Multiplexing (FDM) and Time Division Multiplexing (TDM).

In FDM the frequency spectrum is divided up among the logical channels, with each user having exclusive possession of his frequency band.

In TDM the users take turns (in a round robin fashion), each one periodically getting the entire bandwidth for a short burst at a time.

The latter scheme was used for the multiplexing of Ethernet interface among VAX users. Each user who requests
service via Ethernet, makes exclusive use of the file -
Ethernet communication channel for a short period of time.
Then another user takes control of the channel for a while,
and so on. When all the users have used the channel,
control comes back to the first user and a new cycle begins.

3. PROCESS

1. **Process Definition**

Process is the fundamental program unit in VAX/VMS
which is selected by the scheduler for execution (Ref. 1).
A process is automatically created for each user when he
logs on. The user runs programs, one at a time, in his
process. Only one program can run at a time in any process.
A process is identified by a process ID or PID.
A process is fully described by hardware context,
software context and virtual address space description.

a. Hardware Context

The hardware context consists of copies of the
general purpose registers, the four per process stack
pointers, the program counter (PC), the processor status
longword (PSL), and the process-specific processor regis-
ters including the memory management registers and the ASF
level register.

The hardware context resides in a data structure
called the hardware process control block that is used
primarily when a process is removed from or selected for
execution.

b. Software Context

Software context consists of all the data
required by various parts of the operating system, to make
scheduling and other decisions about a process.
c. Virtual Address Space Description

The virtual address space of a process is divided into two regions:

- The program region (PO), which contains the image currently being executed.
- The control region (PI), which contains the information maintained by the system on behalf of the process. It also contains the user stack, which expands toward the lower-addressed end of the control region.

The following Figure 3.1 illustrates the layout of a process's virtual address space.

![Figure 3.1 Layout of Process Virtual Address Space](image)

The initial size of a process's virtual address space depends on the size of the image being executed. More information about virtual address space can be found in Chapter 10 of VAX/VMS System Services Reference Manual (Ref. 6).
2. JOBS DEFINITION

A process may create subprocesses, and those subprocesses may create new ones and so on. The collection of the creator process, all the subprocesses created by it, and all subprocesses created by its descendants, is called a JOB.

3. IMAGE DEFINITION

The processes that execute in the context of a process are called IMAGES. Images usually reside in files that are produced by one of the VMS/CP commands.

4. TYPES OF PROCESSES

Processes are divided into two broad categories with diverse attributes. These are the DETACHED PROCESSES and the SUBPROCESSES (Ref. 5).

a. Detached Process

The detached process is a fully independent process. The creation of a detached process is primarily a function performed by VMS/CP at login time. One can also create a detached process using the DETACH system service, provided that he has the DETACH privilege.

The attributes of a detached process (Ref. 6) are the following:
- Has own resources
- Has own quotas
- May have a different user identification code (UID) from creator
- Terminates independently of creator
- Detach privilege required to create
- Cannot access creator's devices.
D. Interprocess

A process is a single sequence of the computer resource requests and state transitions before the process is completed. A new process is created by the SCL/PLC operating system. When using SCL/PLC services, it is not specified the creation of a new process, if default a new process is created.

In general, the attributes of a process are:
- start time
- start address of program
- last system call
- next system call
- parent process
- non-privileged signals
- processes from PCL/PLC
- can access devices allocated to process.

C. INTERPROCESS COMMUNICATION

During the entire design of the multiprocess system, major effort was placed on the efficiency and speed of the program execution. Instead of processes being executed sequentially, it was thought that they might be executed concurrently. In such a case the processes need to communicate among each other for reasons of synchronization and mutual exclusion. Thus the issue of interprocess communication is brought up.

VAL/BUS provides the necessary services (system routines) for achieving efficient interprocess synchronization/communication.

The integration of these services are:

- Event Flag Services.
- Input/Output Services.
- Timer and Time Conversion Services.
Further on, a brief description of each category is given:

1. **Event Flag System**

Event flags are the simplest form of interprocess communication and synchronization signals between processes. They are a series of bits (one per flag), not cleared by one process and accessed by the same or different processes. For synchronization within a process, local flag flags are used. For interprocess synchronization, user processes create event flags are used. Event flags can only be used to coordinate user processes within the same event synchronization class (ESC) group. The event flag system services are used to initialize, set, clear, disable and enable for event flags to be set.

Each event flag is associated with a unique decimal number. Event flag arguments in system services calls refer to these numbers. For example, if the flag 1 is specified in a call to the SYMLOG system service, the event flag number 1 is set using the ESC system service. To allow manipulation of groups of event flags, the flags are ordered in clusters, with 32 flags in each cluster, numbered from right to left, corresponding to bits 0 through 31 in a word. The clusters are also numbered from 0 to 3. The range of event flag numbers encompasses the flags in all clusters: Event flag 0 is the first flag in cluster 0, event flag 31 is the first flag in cluster 1 and so on.

The ranges of event flag numbers and the clusters to which they belong are summarized in the Table 1.

The event flag system services used in the program of this thesis are shown in the Table 11. In the next section, a more detailed description of these services. Information about their use can be found in the UNIX/VMS System Services Reference Manual (Ref. 6).

23
It is important to note that there are some other ways also to set event flags. The following system services, some of which will be examined more closely later, accept or require the event flag to be set after the operation is completed:

- Create I/O request (INQ and INQD forms).
- Input and output requests.
- Create lock request (LKX and LKXS forms).
- Set Process (SRTSP).
- Update security file on disk (SUPSF).
- Get Job/process information (SETPJ).
- Get System-wide information (SETPS).

### TABLE 1
Event Flag Clusters

<table>
<thead>
<tr>
<th>CLUSTER</th>
<th>CODE</th>
<th>EXPLANATION</th>
<th>RESTRICTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0-11</td>
<td>Local Event Flags</td>
<td>General use by one process.</td>
</tr>
<tr>
<td>1</td>
<td>12-63</td>
<td>. . . . .</td>
<td>. . . . .</td>
</tr>
<tr>
<td>2</td>
<td>64-95</td>
<td>Assignable requests</td>
<td>Must be assigned to event cluster before use.</td>
</tr>
<tr>
<td>3</td>
<td>96-127</td>
<td>. . . . .</td>
<td>. . . . .</td>
</tr>
</tbody>
</table>

Note that each of the above system services clears the specified event flag before it begins the requested operation.
<table>
<thead>
<tr>
<th>SERVICE</th>
<th>FUNCTION(S)</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Associate Event Flag Cluster (ASCFUNC)</td>
<td>Creates a temporary or permanent common event flag cluster.</td>
<td>TCB quota (for temporary) / privileged (for permanent).</td>
</tr>
<tr>
<td></td>
<td>Creates a common event flag cluster in memory shared by all processors.</td>
<td>Same privilege</td>
</tr>
<tr>
<td></td>
<td>Establishes association of a process with an existing common event flag cluster.</td>
<td>Group association</td>
</tr>
<tr>
<td>Get Event Flag (GESETF)</td>
<td>Turns on an event flag (local or common)</td>
<td>None (for local) / Group association (for common)</td>
</tr>
<tr>
<td>Clear Event Flag (GESCFF)</td>
<td>Turns off an event flag (local or common)</td>
<td>As for GESETF.</td>
</tr>
<tr>
<td>Read Event Flags (GERSRF)</td>
<td>Returns the states of all event flags (local or common).</td>
<td>None (if local) / Group association (if common)</td>
</tr>
<tr>
<td>Wait For Single Event Flag (GESWATF)</td>
<td>Places the current process in a wait state pending the setting of an event flag in a local or in a common event flag cluster.</td>
<td>As in GESETF</td>
</tr>
</tbody>
</table>
I/O Subsystem Services

The I/O subsystem on VAX/VMS has a three-tiered hierarchy. The top tier is VAX-11 Resource Management Services which provides access to files, unit record readers and certain foreign devices. All VAX-11 high level languages use VAX-11 RMS to perform I/O. Thus, VAX-11 statements like READ and WRITE statements cause the compiler to issue calls to VAX-11 RMS.

The second tier is the Queue I/O system services throughout the program. They perform device I/O and VAX-11 RMS generates calls to these services. A programmer uses the Queue I/O services when:

- Accessing devices not supported by RMS (real time devices as the NI1010 board).
- Performing I/O operations not supported by RMS (logical or physical I/O).
- Performing I/O operations not supported by the language's interface to RMS. (Asynchronous I/O in FORTRAN).

The bottom tier is the device driver itself. These services act as the user interface to the device driver which is never directly accessed by the application program.

To maintain a level of device independence, VAX/VMS provides internal device services which allow programs using the queue I/O services to be written without regard for the driver object or unit number of the device. In fact, if the "I" macro is used, instead of the Queue I/O services, the name of the device need not be considered by the programmer. More will lists the most important Device-Dependent I/O services.
In Table IV are listed the two basic Mailbox and Message I/O services used in this thesis to provide the adequate communication among processes. After this overview of the most important I/O services, four of which, namely $ASSIGN, $QIO, $QIOW and $CPMBX appear in the programs of this thesis, it is considered useful to carry out a brief discussion of two basic features that VAX/VMS incorporates

<table>
<thead>
<tr>
<th>Service name</th>
<th>Function(s)</th>
<th>Restriction(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Mailbox and Assign Channel ($CREMBX)</td>
<td>Creates a temporary or a permanent mailbox.</td>
<td></td>
</tr>
<tr>
<td>Delete Mailbox ($DELMBX)</td>
<td>Marks a permanent mailbox for deletion.</td>
<td></td>
</tr>
</tbody>
</table>

TABLE III
MAILBOX SERVICES

in the above services: AST's and MAILBOXes. These features, constituting the backbone of the whole design, have provided the desired efficiency and facilitated the communication between processes.

a. Asynchronous Trap Services

Some system services allow a process to request that it be interrupted when a particular event occurs. Since the interrupt occurs (out of sequence) with respect to process execution, the interrupt mechanism is called an asynchronous system trap (AST). The trap provides a transfer of control to a user-specified procedure that handles the event.
The system services that use an AST mechanism accept as argument the address of an AST service routine.

TABLE IV
SUMMARY OF I/O SERVICES

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Function(s)</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign I/O Channel ($ASSIGN)</td>
<td>Establishes a path for an I/O request or network operations.</td>
<td>None (for I/O request), NETMBOX privilege (for network operations).</td>
</tr>
<tr>
<td>Deassign I/O Channel ($DASSIGN)</td>
<td>Releases linkage for an I/O path.</td>
<td>Access mode</td>
</tr>
<tr>
<td>Deassign I/O Channel ($DASSIGN)</td>
<td>Releases a path from the network.</td>
<td></td>
</tr>
<tr>
<td>Queue I/O request ($QIO)</td>
<td>Initiates an input or output operation.</td>
<td>Access mode</td>
</tr>
<tr>
<td>Queue I/O request and Wait for event flag ($QIOW)</td>
<td>Initiates an input or output operation and causes the process to wait until it is completed before continuing execution.</td>
<td>Access mode</td>
</tr>
<tr>
<td>Allocate Device ($ALLOC)</td>
<td>Reserves a device for exclusive use by a process and its subprocesses.</td>
<td>None</td>
</tr>
<tr>
<td>Deallocate Device ($DALLOC)</td>
<td>Relinquishes exclusive use of a device.</td>
<td>Access mode</td>
</tr>
</tbody>
</table>

that is, a routine to be given control when the event occurs. These services are:

-- Declare AST ($DCLAST)
-- Enqueue Lock Request ($ENQ)
-- Get Device/Volume Information ($GETDVI)
-- Get Job/Process Information ($GETJPI)
-- Get System Wide Information ($GETSYM)
-- Queue I/O Request ($QIO)
-- Set Timer ($SETIMR)
-- Set Power Recovery AST ($SETPRA)
-- Update Section File On Disk ($UPDSEC)

Of the above, the $QIO and the $SETIMR services used in the programs, include in their arguments ASTs.

ASTs are queued for a process by access mode. An AST for a more privileged access mode always takes precedence over one for a less privileged access mode; that is, an AST will interrupt any AST service routine executing at a less privileged mode; however, the process can receive ASTs from more privileged access modes (for example a Kernel-mode AST at I/O completion).

Figure 3.2 shows a program interrupted by a user-mode AST, and the user-mode AST service routine interrupted by a Kernel-mode AST:

![Figure 3.2 Access Modes and AST Delivery.](image)

Some examples are given below, which may clarify the way ASTs work.
Example with $SETIMR$ AST

In the Set Timer ($SETIMR$) system service, one can specify the address of a routine (subroutine or function) in the main program to be executed when a time interval expires or when a particular time of day is reached. The service schedules the execution of the routine and returns. Up to this point the sequence of the program execution has not been changed. Now when the requested timer event occurs, the system "delivers" an AST by interrupting the process and calling the specified routine.

```
subroutine rec(MRpacket)

implicit integer*4 (a-z)
external dummy

#1. istat = sys$pio (%val(1), %val(nichan), %val(iosb),
      dummy, MRpacket, MRpacket,
      %val(1522),...)

  return
end

#3. subroutine dummy(MRpacket)

  return
end
```

Figure 3.3 Example of an AST.

Example with $QIO$ AST

In this example, the $QIO$ service is called. You can now specify not only the address of a routine but also the parameter to be passed to this routine. Figure 3.3 taken from the program "Ethermult" shows how an AST can be delivered.
Notes on Figure 3.3:

#1. The AST subroutine should be declared as "external".

#2. The AST subroutine name (address) and its parameter are among the arguments of the $QIO service, at the proper positions. The service is executed in the normal sequence of the program, sets the receive mode and returns. The program continues executing until the I/O is completed, i.e. until a packet is received. When this happens, the AST is "delivered", the program is interrupted, and control is transferred at the subroutine "dummy" which is executed. When control returns, the program continues executing from the point of interruption.

More about ASTs can be found in:

-- VAX/VMS Real Time User's Guide [Ref. 9].
-- VAX/VMS System Services Reference Manual [Ref. 6].

b. Mailboxes

Mailboxes are synchronous (mainly) virtual devices which may be used to transfer information among cooperating processes. The amount of information transferred via mailboxes is normally less than 512 bytes. Actual data transfer is accomplished by using VAX/11 RMS or I/O services.

When the Create Mailbox and Assign Channel ($CREMBX) service creates a mailbox, it also assigns a channel to it for use by the creating process. Other processes can then assign channels to the mailbox using either the $CREMBX or the $ASSIGN system services.

The $CREMBX system service creates the mailbox. It identifies the mailbox by a user-specified logical name and assigns it an equivalence name. This name is a physical device name in the format MBAn: where n is a unit number.
Mailboxes are either temporary or permanent. The user privileges TMPNBX and PRMNBX are required to create temporary or permanent mailboxes respectively. The temporary mailbox is deleted as soon as the image that created it ceases to exist.

The SQIO (or SQIOW) system service is used to perform I/O to the mailbox. In a usual sequence of operations involving a mailbox, $CREMBX, SQIO and SQIOW system services are used. First, the $CREMBX creates the mailbox and assigns a channel to it and then the SQIO (or SQIOW) reads or writes to it. Other processes may have access to it by means of a distinct channel assigned to the process and can use also SQIO to read or write to it. In general, mailboxes are one of the simplest facilities to use that VAX/VMS provides for interprocess communication.

![Diagram](Numbers indicate the order of events)

*Figure 3.4 Two processes accessing a mailbox.*

Figure 3.4 gives a schematic interpretation of the communication between processes via a mailbox.

More information about mailboxes can be found in:

-- VAX-11 Fortran User's Guide [Ref. 10].
-- VAX/VMS Real Time User's Guide [Ref. 9].
-- VAX/VMS System Services Reference Manual [Ref. 6].

32
3. **Time and Time Conversion Services**

Many applications require the scheduling of progress activities based on clock time. Under VAX/VMS, a process may schedule events for a specified time interval. Time services can do the following:

1. Schedule the setting of an event flag, or queue of an asynchronous system trap (AST) for the current process, or cancel a pending request that has not yet been honored.

2. Schedule a wake-up request for a hibernating process, or cancel a pending wake-up request that has not yet been honored.

3. Set or recalibrate the current system time, if the caller has the proper user privileges.

The timer services require the user to specify the time in a unique 64-bit format. To work with the time in different formats, one can use time conversion services to:

1. Obtain the current date and time in an ASCII string or in system format.

2. Convert an ASCII string into the system time format.

3. Convert the time from system format to integer values.

The following table lists the timer and time conversion services.
### TABLE V

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Function(s)</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get Time (SGETTIN)</td>
<td>Returns the date and time in system format.</td>
<td>None</td>
</tr>
<tr>
<td>Convert Binary Time to Numeric Time (SBUNITIN)</td>
<td>Converts a date and time from system format to numeric integer values.</td>
<td>None</td>
</tr>
<tr>
<td>Convert Binary Time to ASCII string (SBSTIN)</td>
<td>Converts a date and time from system format to an ASCII string.</td>
<td>None</td>
</tr>
<tr>
<td>Convert ASCII string to Binary (SBINTIN)</td>
<td>Converts a date and time in an ASCII string to the system format.</td>
<td>None</td>
</tr>
<tr>
<td>Set Timer (SSETIN)</td>
<td>Requests setting of an event flag or queuing of an ASP based on absolute or delta time value.</td>
<td>Top level quota.</td>
</tr>
<tr>
<td>Cancel Timer request (SCANTIN)</td>
<td>Cancels previously issued timer requests.</td>
<td>Access mode.</td>
</tr>
</tbody>
</table>
IV. DESIGN CONCEPT

A. GENERAL

1. EVOLUTION OF THE DESIGN

The design for multiplexing the XI1010, the Ethernet interface device, was changed several times, as a part of it was implemented and experience was obtained. One major problem with this work was that the writers of this thesis had not worked in the past with VAX/VMS and sometimes it couldn't be determined in advance what could be done and what couldn't. Such experimentation took place and the original design was modified when necessary.

After the difficulties of these details were overcome, the first working version of the project was completed. However this program was not sufficiently fast, because the multiplexing was taking place at the level of the users. That is, when a user was being served, the program was staying with him until this user was fully served. For example when a user had to get a file from VAX/VMS, no other user would be served until the entire file of the current user was displayed on the SDS screen. If the file was large enough, considerable amount of time was spent, with the entire system devoted to one user, and that made the system very inefficient. Thus, it was decided that multiplexing occur at a lower level and instead of multiplexing users, the new design was multiplexing frames of the users.

2. LANGUAGE SELECTION

The language that was selected for the implementation of the project was VAX-11 FORTRAN. The reason for this
choice was that FORTRAN is very well supported by NIX/VMS, and there are very powerful system routines and other facilities which make the job of the programmer much easier. Hence it was more time to become familiar with this language which was mostly unknown to the designers.

3. Frame Size

Recall that information is transferred via Ethernet (see Appendix A) in groups of bytes. One such group is called a frame. Data in an Ethernet frame must be greater than 60 bytes and less or equal to 1500 bytes.

One decision that had to be made was about the size of the information frame to be transmitted each time.

Although testing and analysis was not very extensive, it was obvious that a frame size of 1500 was the most efficient, considering the average length of the responses to the various commands entered from an SGS terminal. This is reasonable, because after the transmission of each frame, the transmitter should receive an acknowledge from the receiver, and other secondary operations must therefore be executed. So, the smaller the frame size, the more frequently these parasite operations are executed. Also many subroutine calls are avoided. All of these operations are time consuming and reduce the efficiency of the program. For the above reason the largest frame appears to be the most efficient.

The only case where the large frame is at a disadvantage is when the message is very short. For example, if we have a message 60 bytes long, 1500 bytes still have to be transmitted but the 1440 of them will be empty. This situation may frequently occur with the last frame of a file. However since the transfer rate of Ethernet is so high (10 megabits per second) this is not very costly as far as time is concerned. So in overall, the highest frame size is still the most efficient among fixed size frames.
Variable length frames could be used in the program to make it more efficient. However, DONUT in its program provides only for a fixed frame length of 1024 bytes. Since the frame length in real programs must be the same, it was decided that this frame length be maintained.

3. HIGH LEVEL DESIGN

As was mentioned before, the Loginsot procedure cannot be learned from a terminal other than an original VAX terminal. Because of this, a way had to be found to have commands entered from an XDB terminal to be executed in the VAX environment. The solution to this problem is the system routine "spve" (see Chap. 3), which takes an input from other things, a VAX command and executes it. Unfortunately, the only command that "spve" cannot execute is the Loginsot procedure.

It was obvious from the first steps of the design phase that each user should have a program to execute the commands that are addressed to him. Since more than one user could require use of the Ethernet interface concurrently, there had to be another program which will coordinate the users, control the message traffic and assure the interleaving of the possession of the communication channel.

In the following paragraphs, a high level design of the coordinator program and the user's program is provided. To avoid confusion, from now on the coordinator program will be called "Tilcertii" and the user's program will be called "xilcertii".

1. PROCES "Tilcertii"

The program which will perform the Ethernet multiplexing should be able to perform the following operations:
First, it should be able to establish a channel of communication with the S1000 board and sense any message that arrives there.

Second, it must check the validity of any incoming messages and determine the user to which a valid message is addressed, and then send the message to the appropriate user.

Third, it must ensure that only one message is sent to the S1000 and do it in an efficient and fast way, according to specified priorities.

It was thought that a lot of time could be saved if these three tasks are executed concurrently and not one after the other.

a. One of Concept Event Flags in "Consultant"

In order that this program have control over the various user's programs, and know the status of the network or any aspect (i.e. how many users are currently logged in, what user numbers they have etc.), a cluster of seven event flags was used. The cluster which was used contains a total of 12 flags numbered from 0 to 11. Any program in the VAX/VMS environment may access this cluster and read, set, or reset any of these flags.

Flags 0 through 11 correspond to 9 users and they represent user numbers from 1 to 9. When any one of these flags is raised, this means that the user it represents, has logged in successfully onto the network. A user number flag is set and reset by the "Consultant" program, as will be seen later.

You may have noticed already that the system can accept and serve up to nine users. However, this number may be easily expanded to a large number of users by assigning more flags to user numbers, and changing the dimensions of some data structures that are used in the program.
The limitation in user numbers is imposed due to the number of common event flags available in VAX/VMS, namely 96 flags arranged in three clusters of 32 flags each. Under the current design, the system can be expanded to serve a maximum of 47 users.

The flags from 073 through 081 are used to indicate whether the user who corresponds to each of them has an answer ready to send back to the SDS which has sent a command to him.

Figure 4.1 illustrates the use of common event flags for interprocess communication.

b. Use of Mailboxes in "Ethersuit"

The communication of the "Ethersuit" program with the "Userexit" program is achieved through mailboxes. A received message is identified and put in the mailbox of the user to whom it is addressed. The message is then picked up by the "Userexit" for further elaboration.

Each user has his own mailbox which stays there as long as the corresponding "Userexit" is running. When a "Userexit" program terminates, the corresponding mailbox disappears. Figure 4.2 depicts the interprocess communication by means of mailboxes.

c. Use of AST's in "Ethersuit"

Recall that an Asynchronous System Trap is nothing else but an interrupt which transfers control at a specified place of the program when a specific event occurs. "Ethersuit" uses "Syscall" system service to continuously listen to the UI010 board. This system service sets the "Ethersuit" into a receive mode in which the program does not wait for the event to occur, but continues normal execution. As soon as a message arrives at the UI010 board, an interrupt is triggered and control of the program is
Figure 4.2  Process Communication Via Mailboxes.
transferred to a location which can be prespecified as an input parameter of the "Systqio". In this way there is no overhead due to the need of listening to the network.

d. "Ethermult" Algorithm

A high level algorithm of program "Ethermult" is given below. This algorithm does not go deep into details, because its purpose is to help the reader realize quickly what are the main operations of the program. A detailed description of the program and explanation of the code is given in the next chapter.

At this point it should be mentioned that the program uses a message queue in which the incoming messages are stored, in a FIFO order. Also there exists a user information table which contains information pertaining to the current users.

The high level algorithm is as follows:

1. Make accessible the common event flag cluster to the program.
2. Clear flags #64 through #84
3. Establish communication channel with NI1010
4. Set the "ear" of the system to listen to the network and specify where program control should be transferred when a message arrives.
5. When a message arrives, put it in the message queue, send acknowledge to the source, and reset the "ear" of the system.
6. Take the first message from the message queue.
7. Check whether the user to whom this message is addressed, is currently authorized to use the system.
8. If he is not:
   - send a caution message to the message source
   - disregard the message
   - rearrange the message queue
Else
- rearrange the message queue
- put the message into the appropriate mail box
- if this is the first message for a VAX user:
  - update the user information table.

9. Check which users have a ready answer and send each answer to the proper destination, using a round robin scheme. According to this scheme each user is serviced for as much time as it is required for one frame to be sent, and acknowledge to be received. After this the next user is serviced.

10. Check the status of the network and update the user information table if necessary.

11. Repeat steps 5 to 11.

2. Program "Usermult"
   a. Program Tasks

A user which uses an MDS terminal to access the VAX/VMS environment, is a virtual VAX/VMS user. As a VAX user he must be first authorized in order to use the system. This authorization is granted by the Loginout VAX/VMS procedure.

Since currently the Loginout procedure can be invoked only from a VAX terminal, there is no other way for a user but to use a VAX terminal in order to log on. In addition, because he intends to use the Ethernet multiplexing as well, he has also to get a second authorization, to enter the network. So after a user has successfully logged in the VAX/VMS, he must run a program which will enable him to enter the network. This program is the "Usermult" program.

"Usermult" checks the common event flags #73 through #81, that represent the users, and finds out which
of them are not already set. Then it associates them with user numbers and prompts the user to select one of those. If the user chooses a legal number, authorization to enter the system is given and the corresponding flag is raised to notify "Ethermult" about the new status. If the user made an invalid entry, the program gives him another chance.

Commands like DIR, TYPE, PRINT etc. which are entered from an NDS terminal and pertain to a specific user, must be executed and the results should be sent back to the NDS terminal.

The distribution of the messages which arrive at the NI1040 board is performed by program "Ethermult", which places each message into the appropriate mailbox, very much like a mailman does. But this requires that there exist such a mailbox. So the first thing that "Usermult" should do, after the user has been accepted by the system, is to create this user's mailbox and establish a path of communication with it.

After the mailbox is created, "Usermult" should check constantly whether a message has arrived. As soon as a command arrives, the program calls "Spawn" system procedure which executes the command.

A more efficient way of checking the mailbox would be to set the "Usermult" in receive mode and then put it in hibernation while still in the receive mode. As soon as a message arrived, an interrupt would be triggered to wake up the process in order it to continue execution. This interrupt driven scheme is more efficient for multiuser systems because this way a time slice is saved. However, for reasons of simplicity, it was decided to implement the first method.

The output of the routine "Spawn" is placed in a file which will be used by "Ethermult", to send the answer back to the calling NDS. Then the program checks the
mailbox again for the next message. The message traffic among VAX processes is illustrated in Figure 4.3.

b. User Mutual Exclusion

It may happen that more than one user requests access to the network at the same time. In such a case two users may select the same user number. If this happens many problems and confusion will be created. So there must be a way to avoid a situation like this.

The solution to the problem was again a common event flag. Specifically the very first thing "Usermult" does is to access the event flag cluster and check whether flag #84 is set. If it is so, it means that another user is requesting network service at this moment. Thus the program should wait and keep checking flag #84.

As soon as this flag is reset, the requesting program raises it, so that no other user will interfere until a user number is obtained.

c. Protection

Now another issue is brought up. If no protection is provided, a user who has control of event flag #84, is enabled to prevent any other user from initiating a login procedure. The solution which was adopted for this problem was the use of a watchdog timer.

A timer which is set at 10 seconds is energized when the user is prompted to make his selection, and a message on the screen notifies the user that he has 10 seconds to select a user number. If he does not make a selection within 10 seconds, flag #84 will be reset, and his program will terminate.

The program also resets flag #84 and terminates execution when the user makes two consecutive wrong selections. This way he cannot prevent other users from logging
Figure 4.3  Message Traffic Inside the VAX.
in, by making indefinitely invalid selections just before the 10 seconds expire.

Next a concise algorithm for the "Usermult" program follows.

"Usermult" Algorithm
1. Access event flag cluster
2. Check flag #84
3. If set keep checking
   Else set it
4. Check flags #73 through #81
5. Put those flags which are not set in an array
6. Associate those flags with user numbers and prompt the user
7. Set the timer for 10 seconds
8. In case of wrong selection give another chance
9. If time expires without selection of user number
    reset flag #84
    stop
Else
    cancel timer
    accept user number
    reset flag #84
10. Create mailbox and establish path of communication
11. Check the mailbox
12. If empty keep checking
Else
    call "Spawn" to execute the message
    put the results in a file
    go to 11
End Usermult.
V. DETAILED DESIGN OF THE MULTIPLEXING SYSTEM

In the previous chapter the high level design of the multiplexing system was given, without details on the operations of each module.

This chapter explains in detail the operations of the main program and each subroutine. In reality it constitutes the documentation of the software. Also included are comments which explain to the reader why certain decisions were made.

A. PROGRAM 'ETHERMULT' DETAILED DESIGN

1. VARIABLES AND DATA STRUCTURES

The variables which are declared in the program are in alphabetical order the following:

Ackflag : Variable to denote whether the received information is an acknowledge or a real message.

Alpha : Array of characters with 9 elements, which contains permanently the numbers from 1 to 9. It is used to solve a technical problem in concatenation of characters. The program assigns very frequently the current user number as an index for the mailboxes and reply files. But in Fortran we are not able to concatenate characters with integers. So the trick which was used was the following: Instead of saying Mailbox//Usernumber which for example may be Mailbox3, we say Mailbox//Alpha(usernumber) that means Mailbox//Alpha(3), which is Mailbox3. The
latter form is acceptable by the Fortran compiler.

C : Number of users currently in the system. It can take values from 1 to 9.

Cos : Array of 81 bytes used to store the first 81 symbols of a received Ethernet packet. Its size is adequate to hold any incoming VAX/VMS command.

Condition: Byte used as a boolean variable to denote the existence of a condition.

Dflag : Boolean variable.

Fl : The name of the common block which contains variable ackflag.

Flag : Used as Boolean variable.

Ind1 : Name of the common block in which variable 'Notyet' is found.

Ind2 : Name of the common block which contains variable 'Times'.

Iosb : An output parameter of Sys$que$ routine. Its value determines whether the operation was successful or not.

Queue : Array 9x81 which stores the incoming commands in a FIFO fashion.

Npacket : Stands for main receive packet. It is an array of 1522 bytes which includes the incoming message plus the header.

n : Variable which denotes the first empty slot of the message queue.
Michan : Indicates the number of the channel of communication which is assigned to a device by Sysfassign system routine.

NotYet : Array of integers with dimension '1x9'. Its elements correspond to one of the nine users and they are used to indicate whether the transmission of all the frames of an answer file of a user has been completed or not.

Pack : Name of common block containing variable 'Queue'.

Row : Byte showing the row of the table in which a user number was found.

RowFlag : Byte used as boolean variable.

Slot : Name of common block which contains variable 'N'.

Table : Array of bytes with dimensions '1x9'. User information is stored in this table. The first column contains the user number. The second column becomes '1' when the user has an answer that has not been sent yet, and '0' if there is no file with the answer of this user. Columns 3 and 4 contain the source address of the last message addressed to this user.

Times : Array of integers with dimension '1x9'. Each one of the nine elements specifies a unit number in which the answer file of one user will be opened. The value of the elements of array 'Times' should be greater than 11.
Section 1: Error which occurs for each number. It can take values from 1 to 9.

In the program are also included the routines:

* MAIN:* (Library: Library Name)
This routine contains all the definitions of the functions used exclusively by the U1070 board.

* IODER:*
This is a macro routine which contains all the definitions of I/O functions (Ref. 6).

* IODER:*
Macro routine which contains all the definitions of system status functions.

2. Initialization

The program starts by initializing the number of current users, the user information table, the service queue, and arrays 'Hitgent' and 'Times'.

Next the third event flag cluster named REA, which contains the flags 804 through 809, is made visible to the program. Then the flags 804 through 806 are reset to indicate that the system is empty.

Next step is to establish a channel of communication with U1070 and start up the system. More information about the system routines that execute these tasks can be found in Appendix A.

3. Main Loop

At this point the program is ready to start the actual work. The routine IEC, which "sets the ear" of the system to the U1070 board is called. It uses the "SysEqio" system routine which enables the program to listen to U1070 without waiting there. While listening, it can proceed with the execution of other operations. As soon as a
message arrives at the shift board, control of the program
is transferred to routine "Denny".

Routine "Denny" checks whether the received message
is a real message or an acknowledge. If the 16th byte of the
received packet is "7F" hex, then the message is an acknowledge. In this case, it "sets the ear" of the system again by
calling routine SEC, sets the "Listening" and restores control
at the point where the program was before the interrupt
occurred.

If the received message was a real one, routine
"Receive" is called to read acknowledge to the source, and the
first 81 characters of the packet are transferred into an
array "Conn. The size of the array "Conn" is adequate to hold
any one of 25/400000 connections.

Then the message queue is rearranged and the routine
"Prequeue" is called, to put the message in the appropriate
place of the queue.

Finally the "ear of the system" is set and control
of the program returns where it was before the interrupt
occurred.

As the trace of the "Network" continues, the next
thing is to determine the user number by examining the 16th
byte of the message whose turn is to be processed.

Since user number is of type byte (ASCII) the number
40 is subtracted in order to make it decimal.

After that the obtained user number is checked for
validity. If the user number is 0, it is immediately charac-
terized invalid. Subroutine message is called then, to
notify the user on the SBS terminal about that. Next, the
position of the queue which contains this message is
cleared, and the queue is rearranged.

The same things happen when the obtained user number
is greater than 9, of other data type (i.e. character).
There is only a small difference in the warning message to
the SBS.
If some of the above cases occur, the number is within valid limits and it should be checked whether it already exists in the user information table or not i.e. whether the user is new or he has been already using the system. Subroutine "Search" finds that out. If the number already exists in the user information table, the row in which it is found is returned.

If the second column of the row that contains the user number is 1, this means that the user hasn't sent yet the entire file with the previous answer. So, no other command should be executed, because the new file which will be created will erase the previous one, before it is sent to the D1010. Because of that, this user is not served in this round, but the next message in the queue is selected.

If no other ready answer exists, the variable "Condition" becomes 1 in order to terminate the do while loop. Since one message is processed at a time in each round of the program, and a message to be processed has been found, there is no need to continue the while loop.

Next step is to put the message in the "Com" array, and arrange the message queue properly.

Now the message should be placed to the appropriate mailbox. Then, subroutine "Distribution" is called.

Distribution calls routine "Search" to find out whether the user number exists in the user table. If routine "Search" returns variable "UserFlag" with value 1, it indicates that the user already exists in the table. If also the second column of the row in which the user number was found has value 1, it means there is still at least a part of the previous file to be sent, so no distribution occurs and control returns to the main program.

If the user number was not found in the user information table, there are two possibilities. Either he is in the system but the table is not updated, or he is not in the system at all.
If this was not the last frame, the file stays opened so that the read index is not removed from the correct position. Then control returns to routine Export and from there back to the main program.

As a final step, the main program calls the subroutine "status check" which examines the common flags and if it finds any discrepancy with the existing status, it updates the user information table and the number of current users. Then the process repeats the same cycle indefinitely.

A hierarchy diagram of the program "Ethersuit" is given in Figure 5.1.

3. "ETHERSUIT" DETAILLED DESIGN

1. VARIABLES AND DATA STRUCTURES

The variables and the data structures which are used in program "Ethersuit" are given below:

Alpha: Array of dimension 1x9. Its elements are permanently the numbers from 1 to 9. It is used for the same reason as array Alpha in program Ethersuit.

Chanel: Array 1x9 which contains channel numbers corresponding to user numbers.

Coo: Array of 9 bytes used to store a received message (message) from the corresponding mailbox stripped off from its header.

load: Logic variable. When it is 1 it means the system is full of users.

message: User message. An integer which can take values from 1 to 9.
Figure 5.2 Hierarchy Diagram Of Program ETHERMULT
The included macro routine \texttt{\$iodesf} contains all the definitions of I/O functions (see Appendix C).

2. Main Body

The program starts by calling subroutine \texttt{Authorize}. \texttt{Authorize} has as input parameter the user number and returns the variable \texttt{Load} with value 1 if the system has already 9 users and it cannot accept any new ones at the present. Detailed description of \texttt{authorize} is given in the next subsection.

If the system has accepted the user, the program proceeds with the creation of a user mailbox and establishes a channel of communication with it, using the system routine \texttt{sys$crembx}. Then it keeps checking the mailbox for any messages. When a message arrives, it reads it and puts it in the array \texttt{Com}. Routine \texttt{Sys$qiow} is used for this task.

Next, the program makes visible to its environment the cluster which starts with flag \#64. This is needed because the program must set a certain flag when an answer is ready.

After that, the program calls routine \texttt{feedfile}. This routine places the command without its header in the file \texttt{mail.com} which will be used as input in \texttt{Lib$spawn} routine.

Subroutine \texttt{Spawn} is called next. \texttt{Spawn} has as an input parameter the user number. It executes the command and puts the results in a file which is indexed by the user number. Description of \texttt{Spawn} is found in subsection 4 of this section.

After completing this cycle, the program goes back to read the mailbox and proceed as before.
3. Subroutine "Authorize"

This routine checks the common event flags #73 through #81 to find out what user numbers are available for a new user. Then it interacts with the user in order to assign a number to him. If the system is full, it returns the variable "Load" with value 1. In such a case the program terminates execution.

The routine starts by assigning a channel to the terminal. This is done because the "Sys$giow" routine is used to read a user number from the terminal. Of course, the read statement instead of "Sys$giow" routine could be used, but in this case the use of the watchdog timer would be impossible. With the read statement, the program would wait indefinitely until a user number was entered. If the user did not enter a number, he could inhibit any other user from logging on. Now however, we are able to set a time limit and if the user does not enter a number within the available time period, the program stops execution.

Next step is to invoke "Sys$giow" routine to start up the process of I/O.

Then the proper initializations are made. "Flagarray" is an array which will hold the available user numbers in order to display them later on the screen.

Variables "Cancel", "Condition", and "Load" are used as Boolean variables.

Next, the routine acquires access to the usual common event flag cluster and determines whether another user is using the system (ie. is requesting a user number), by checking flag #84. If this flag is set, it means the system is busy and the program keeps checking until the flag is reset. The variable "Condition" is used to inhibit the program from writing the message on the screen in each loop.
If the flag #84 is reset that means the system is available. So, immediately the program sets this flag in order to notify the other potential users.

Next step is to check flag #73 through #81. Those flags represent users from 1 to 9. The numbers which are represented by the flags that are reset, are put in the "Flagarray". If after the examination of the flags the "Flagarray" is still empty, this means that the system is full. Then variable "Load" takes value 1 and the routine returns control to main program, after it has printed on the screen the message "system full".

If there are available user numbers, a message is printed on the screen prompting the user to select one of the user numbers which follow on the screen. Immediately the timer is set at 10 seconds, and the "Flagarray" is displayed on the screen.

After that, the program goes to a read state. If a number is entered within 10 seconds the timer is cancelled. If not, control of the program is transferred to routine "Abort" which resets flag #84 and stops execution of the entire program.

After a number is entered, the program checks whether it can be found in the "Flagarray". If it is not found there, the counter advances to count one misselection. The user is notified for his mistake and the program presents again the available user numbers on the screen. Then it proceeds as before. If a second misselection occurs, the program transfers control to subroutine "Abort".

If the entered number is found in the "Flagarray", it is considered a legal one. Then the flag which represents this user number is set, the flag #84 is reset leaving the system available for use by other users, a message is displayed on the screen to notify the user that the number has been accepted and control returns to the main program.
4. **Subroutine "Spawn"**

Routine "Spawn" executes the DCL command which is entered from an MDS terminal.

It starts by translating the logical name of the input file "Mail.com", into a physical name. This is necessary in order that this file be accepted as input to the "Lib$spawn" procedure.

Then the file is opened and read. Two special commands are examined separately. The "Edit" and "Logout" command. It was found that the "Edit" command in order to function properly must be put explicitly as input parameter in "Lib$spawn". So, the first letters of the file "Mail.com" are examined and if they form the word "Edit", the "Lib$spawn" is called with an input parameter the string "Edit" explicitly and not the file "Mail.com" which contains this string.

The case of "Logout" is examined in order that the termination of the corresponding "Usermult" program occur, and an update of flags is effected. This will enable the main program to update the user information table and the number of current users in the system.

The rest of the commands are executed normally and the results are placed in a file called "Reply" concatenated with a user number. After that control is transferred to the main program.

Figure 5.2 illustrates the hierarchy of "Usermult" program.
Figure 5.2  Hierarchy Diagram of Program "Usermult".

61
VI. CONCLUSIONS

A. PRESENT DESIGN

The principal goals of this thesis were met. The multiplexing of the NI1010 board is considered to be efficient and though the developed software can be currently tested with only two users (working on the two MDS terminals), it should perform equally well with all nine users. Of course, more than nine users can be accommodated, if minor changes are introduced in the programs.

At present, the programs "ETHERMULT" and "USERMULT" are available in VAX/VHS public user account under user name "INTERLAN" with password "VHS". The NI1010 Ethernet Controller Multiplexing User's Manual is also available in the file NINUX.DAT. The content of this file is exactly the same as the content in Appendix D of this thesis. Users who want to have a feeling of how this multiplexing system works, can get a hard copy of this file by simply logging into the VAX/VHS under user name and password mentioned above and printing the files. Then the steps in the manual must be followed.

The files containing the software of this thesis and residing in the public user account are:

ETHERMULT.FOR (source code)
ETHERMULT.EXE (executable code)

This is the main program which does the multiplexing of the NI1010 board. When it is run, it puts the NI1010 board in a receive mode and starts a loop until a message (currently originating from an MDS terminal), is received. If this message is properly encoded and contains a legiti-
mate frame, the system responds by transmitting via the
HI1010 an answer to the originator. If this is a bad frame
it discards it.

The answer transmitted may be:
(a). One message (frame) to the originator stating
either that he has not the authority to enter the system or
that his message has not the proper form.
(b). A series of one or more frames containing the
answer to his message.

**USERHULT.FOR** (source code)
**USERHULT.EXE** (executable code)

This is the program which must be running for each user
in the system. It takes care of the user's requests, by
producing an answer file, which then is transmitted by the
"ETHERHULT" via Ethernet to the originator of the request.
It also contains the code that provides authorization for
each new user. It can be terminated by the remote user when
the command "LOGOUT" or "LO" is sent.

3. FUTURE DESIGN

The above programs, provide the possibility for several
remote users to take advantage of the majority of the
VAX/VMS facilities. These SDS terminals, executing VMS
commands, act like virtual terminals of VAX/VMS. To achieve
a full virtual terminal performance though, a user should be
able to:

1. Use the SCS/PROC system service to create a **Attached**
   process that will execute the LOGIN/OUT procedure. As "input"
   and "output" parameters of this process should be defined,
   instead of a physical VAX terminal, two files, one for the
   input and the other for the output of the LOGIN/OUT. Thus,
it will be possible for the LOGINOUT procedure to be invoked from I/O devices other than a VAX terminal (e.g., MD5 terminal). This should be feasible with the re-exec version of VAX/VHS. The program which has been worked out by the authors of this thesis during the time period they were trying to achieve this goal, is presented in Appendix I.

2. Execute the LOGINOUT.EXE file, for logging in the VAX/VHS system, from his terminal without having to run the "Usersult" program from a VHS terminal.

3. Change the "Usersult" program by deleting the "SPAWN" subroutine since, if he succeeds in the above two goals he will have direct access to the command language interpreter. Then, every VHS command would be executed as if it was typed from a real VAX/VHS terminal. The only difference would be that the input and output of the LOGINOUT procedure would not be a terminal but a file.

4. Special privileges must be granted to "Usersult" program in order to have access to the various directories where "REPL.DAT" files are generated. In cooperation with VAX-11 professional staff, it can be determined which privileges are necessary.

5. Introduce a variable length frame for the network communications, for the sake of efficiency. This would imply changes to both this and Stotzer's thesis' programs. For this thesis the change visualized is:

Instead of declaring a frame size of 1500 bytes as appears in the present form of the program, a common variable denoting the frame size would have to be introduced. The subroutine that reads a file into packets ("sendmsg") should be partly revised so that it "tailors" a frame's size according to the available data in the file (as to 1500 Bytes per frame). For example, in a 2000-bytes file the
First frame should be 1500 bytes since there are 1500 bytes of data available at that time. The second frame though should be filled-up with 500 bytes only since there are some in left. The "Endfile" condition in Pivetta will be helpful at that stage, when the "end of file" is reached, the control can be transferred to correct point of the program. At that point, a count of 500 bytes should be performed and the value of that count, namely 503, should be passed to the correct variable of the frame size. If such a scheme, a relative flexibility is achieved and the required size of the network is improved.

These are the most important changes that, in the authors’ opinion, are needed to support a virtual terminal design.
ASSEMBLE A
SYSTEM SERVICES AND NON-TIME LIBRARY ROUTINES

A. CALLING THE SYSTEM SERVICES

This section provides an overview of the calling mechanism for the system services and library routines. This mechanism varies from one language to another. Since the FORTRAN is the language used for this manual, we will deal only with the FORTRAN's calling mechanism.

1. FORTRAN CALLS

(a) All subprogram calls, including system services and library routines use a CALL instruction.

(b) System services can be called as functions of an subroutine.

(c) Subroutines do not return a status value, therefore they are rarely appropriate.

Example: call system_call(name,arg1,arg2,...,argn)

(d) Functions return a status value as the function result. The function and the variable to contain the return function value must be declared as INTEGER.

Example: integer system_call, ret_val

ret_val = system_call (arg1,arg2,...,argn)

(e) All arguments are positional and must be present. Even if the arguments are optional, their position must be denoted as empty.

Example: ret_val = system_call('arg1','arg2',...,'argn')
The sample FORTRAN calls follow:

-- Calling services as subroutines:
IEPU = 1
CALL SYSSCLEEF(EVAL(IEPU))

-- Calling services as functions:
INTEGER*4 SYSSCLEEF,STAT

IEPU = 1
STAT = SYSSCLEEF(EVAL(IEPU))

2. PASSING ARGUMENTS

There are three ways to pass arguments to system services:

(a). By Immediate Value. The argument is the actual value to be passed (a number or a symbolic representation of a symbolic value).

-- Example: IEPU = 1

CALL SYSSCLEEF(EVAL(IEPU))

(b). By Reference. The address of the argument is passed in the argument list.

-- Example: INTEGER*4 INCHAN

CALL SYSSCLEEF('IELA', INCHAN, )

OR: ISTAT = SYSASSIGN('IELA', IEPU, CALL(SYSCHAN), )

D. SYSTEM SERVICES AND LIBRARY ROUTINES USED IN THE PROGRAM

This section is not intended to examine each of the above services/routines in great detail since such information can be found in the VAX/VMS manuals. Instead, a brief description of the main features is given, along with particular points of attention for their efficient use.

In the VAX/VMS high level languages the system services appear in the form: SYS$service_name; eg, the service SESSION is written as SYS$SESSION. Also, the Sun-Times Library Procedures have the form LIB$procedure_name; eg, the procedure SUBPAGE is written as LIB$SUBPAGE.

Almost all of the above services/procedures are accompanied by arguments, either optional or mandatory, which represent the necessary information they need to carry-out the required task. Optional arguments are denoted with their names in square brackets, mandatory ones are denoted without brackets.

1. SYSTEM SERVICE ROUTINES

ASSOC -- Associate Unless Event Flag Cluster

The associate Unless Event Flag Cluster system service causes a named unless event flag cluster to be associated with a process for the execution of the current range and assigns a process-local cluster number for use with other event flag services. If the named cluster does not exist but the process has suitable privileges, the service creates the cluster.
High Level Language Format:

\[ \text{SYSFASCFFG} (\text{efn, name, <prot>, <perm>}) \]

\textbf{efn}

Number of any event flags in the common cluster to be associated. The flag number must be in the range of 64 through 95 for cluster 2 and 96 through 127 for cluster 3.

\textbf{name}

Address of a character string descriptor pointing to the text name string for the cluster. Section 3.7.1 of the System Services Reference Manual (Ref. 6) explains the format of this string. The names of event flag clusters are unique to UIC groups.

\textbf{prot}

Protection indicator controlling group access to the common event flag cluster. A value of 0 (default) indicates that any process in the creator's group may access the cluster. A value of 1 indicates that access is restricted to processes executing with the creator's UIC.

\textbf{perm}

Permanent indicator. If it is 1, the common event cluster is marked permanent. If it is 0 the cluster is temporary (this is the default value).

\textbf{Privilege Restrictions}

To create a permanent common event flag cluster, the user privilege \texttt{PMCCF} is required. To create a common event flag cluster in memory shared by multiple processors, the user privilege \texttt{SMES} is required.
Resources required/returned:

Creation of temporary common event flag clusters uses the process' quota (TQELN); the creation of a permanent cluster does not affect the quota.

Notes

(1). Temporary clusters are automatically deleted when the image that created them, exits.
(2). Since this service automatically creates the common event flag cluster if it does not already exist, cooperating processes need not be concerned with which process executes first to create the cluster. The first process to call $ASCEFC creates the cluster and the others associate with it regardless of the order in which they call the service.
(3). The initial state of all event flags in a newly created common event flag cluster is 0.
(4). If a process has already associated a cluster number with a named common event flag cluster and then issues another call to $ASCEFC with the same cluster number, the service disassociates the number from its first assignment before associating it with its second.

ASSIGN — Assign I/O channel

The Assign I/O Channel system service provides a process with an I/O channel so that input/output operations can be performed on a device, or establishes a logical link with a remote node on a network.
High level language format:

```
SYS$ASSIGN (devnam, chan, <acmode>, <mbxnam>)
```

devnam
Address of a character string descriptor pointing to the device name string. The string may be either a physical device name or a logical name. If the device name contains a colon, the colon and the characters that follow it, are ignored. If the first character in a string is an underscore (_), the name is considered a physical device name. Otherwise the name is considered a logical name and logical name translation is performed until either a physical device name is found or the system default number of translations has been performed.

chan
Address of a word to receive the channel number assigned.

acmode
Access mode to be associated with the channel. The most privileged access mode used is the access mode of the caller. I/O operations on the channel can only be performed from equal and more privileged access modes.

mbxname
Address of a character string descriptor pointing to the logical name string for the mailbox to be associated with the device, if any. The mailbox receives status information from the device driver.

Privilege restrictions

The NETRNX privilege is required to perform network operations.
Notes

(1) Only the owner of a device can associate a mailbox with the device, and only one mailbox can be associated with the device at a time.

(2) Channels remain assigned until they are explicitly deassigned with the Deassign I/O channel ($DASSGW) system service, or, if they are user mode channels, until the image that assigned the channel is terminated.

**SBINTIM -- Convert ASCII String to Binary Time**

The Convert ASCII String to Binary Time converts an ASCII string to an absolute or delta time value in the system 64-bit time format suitable for input to the Set Timer ($SSETIMR) or Schedule Wakeup ($SCHDWK) system services.

**High level language format:**

```
SYS$SBINTIM (timbuf,timadr)
```

**timbuf**
Address of a character string descriptor pointing to the buffer containing the absolute or delta time to be converted. The required formats of the ASCII strings and syntax rules along with several examples, are described in the VAX/VMS System Services Reference Manual [Ref. 6].

**Notes**

(1) The $SBINTIM service executes at the access mode of the caller and does not check whether address arguments
are accessible before it executes. Therefore, an access violation causes an exception condition if the input buffer descriptor cannot be read or the output buffer cannot be written.

(2). This service does not check the length of the argument list, and therefore cannot return the SSS_INSFRG (insufficient arguments) error status code. If the service does not receive enough arguments (for example you omit required commas in the call), you might not get the desired result.

**SCANTIN** -- Cancel Timer

The Cancel Timer Request system service cancels all or a selected subset of the Set Timer ($SETIMR) requests previously issued by the current image executing in a process. Cancellation is based on the request identification specified in the $SETIMR system service. If more than one timer request was given the same request identification, they are all canceled.

High Level Language format:

```plaintext
SISSCANTIN (<requidt>,<acnode>)
```

**requidt**

Request identification of the timer request(s) to be canceled. A value of 0 (the default) indicates that all timer requests are to be canceled.
ACMEDE

Access mode of the request(s) to be canceled. The most privileged access mode used is the access mode of the caller. Only those timer requests issued from an access mode equal to or less privileged than the resultant access mode are canceled.

Privilege Restrictions

Timer requests can be canceled only from access mode equal or more privileged than the access mode from which the requests were issued.

Resources Required/Returned

Cancelled timer requests are restored to the process' quota for timer queue entries (TQELB quota). Outstanding timer requests are automatically cancelled at image exit.

$CLEAR -- Clear Event Flag

The Clear Event Flag system service sets an event flag in a local or common event flag cluster to 0.

High Level Language Format

$CLEAR (efn)

efn

Event number of the event flag to be cleared.
**SCRECH** -- Create Mailbox and Assign Channel

The Create Mailbox and Assign Channel system service creates a virtual mailbox device named SBAN: and assigns an I/O channel to it. The system provides the unit number, n, when it creates the mailbox. If a mailbox with the specified name exists, the SCRECH service assigns a channel to the existing mailbox.

**High Level Language Format:**

```
SCRECH (<preflag>, chan, <maxmsg>, <bufquo>,
<promsk>, <acmode>, <loqmac>)
```

**preflag**
Permanent indicator. A value of 1 indicates that a permanent mailbox is to be created. The logical name, if specified, is entered in the system logical name table. A value of 0 (the default) indicates a temporary mailbox.

**chan**
Address of a word to receive the channel number assigned.

**maxmsg**
Number indicating the maximum number of messages that can be sent to the mailbox. If not specified, or if specified as 0, the system provides a default value.

**bufquo**
Number of bytes of system dynamic memory that can be used to buffer messages sent to the mailbox. If not specified, or if specified as 0, the system provides a default value.
Numeric value representing the protection mask for the mailbox. The mask contains four 8-bit fields. Bits are read from right to left in each field. If not specified, or specified as 0, read and write privilege is granted to all users.

Access node to be associated with the channel to which the mailbox is assigned. The most privileged access node is the node of the caller.

Address of a character string descriptor pointing to the logical case string for the mailbox. The equivalence mask for the mailbox is IBM. The first character in the equivalence mask string is an underscore character (_). One or more underscore and the mailbox is assigned other I/O channels to the mailbox.

Privilege Restrictions

The user privileges TSBHES and PBHES are required to create temporary and permanent mailboxes, respectively.

The user privilege ESHE is required to create a mailbox in memory shared by multiple processors.

Resources Required/Returned

(1) System dynamic memory is required for the allocation of a device data base for the mailbox and for an entry in the logical name table, if a logical name is specified.

(2) When a temporary mailbox is created, the process' buffered I/O byte count quota (BYPCTRL) is reduced by
the amount specified in the EUPQQO argument. The size of the
mailbox unit control block and the logical name (if one is
specified), are also subtracted from the quota. The quota
is returned to the process when the mailbox is deleted.

Rules
(1) After a mailbox is created, the creating
process and other processes can assign additional channels
to it by calling the Assign I/O Channel (SASSIGN) or Create
Mailbox (CREMBX) system services. The system maintains a
reference count of a number of channels assigned to a
mailbox; the count is decreased whenever a channel is deas-
signed with the Deassign I/O Channel (DASSIGN) system
service or when the image that assigned the channel termi-
nates. If it is a temporary mailbox, it is deleted when
there are no more channels assigned to it.

(2) A mailbox is treated as a shareable device; it
cannot, however, be counted or allocated. In other words, it
cannot be reserved for exclusive use (allocated) or cannot
be linked with a volume and a process (counted).

(3) CREMBX merely assigns a channel if the mailbox
already exists in order to remove the need for cooperating
processes to consider which process must execute first to
create the mailbox. If a temporary mailbox is being created,
CREMBX implicitly qualifies the mailbox name with the group
number to check whether the mailbox already exists. In
other words, there can be only one mailbox per group with
the same name. For permanent mailboxes, there can be only
one mailbox with a particular name. However, a permanent
mailbox and group of mailboxes can have the same name.
**EXECL** -- Create Process

The Create Process system service allows a process to create another process. The created process can be either a subprocess or a detached process.

A detached process is a fully independent process. For example, the process that the system creates when a user logs in is a detached process. A subprocess, on the other hand, is related to its creator in a tree-like structure: it receives a portion of the creating process’ states and must terminate before the creating process. The specification of the DET argument controls whether the created process is a subprocess or a detached process.

**High Level Language Format:**

```
EXECL (pidaddr, image, <input>, <output>,
<error>, <privadr>, <quota>, <priena>, <haspri>, <sic>,
<absext>, <utemp>)
```

**pidaddr**

Address of a langword to receive the process identification number assigned to the created process.

**image**

Address of a character string descriptor pointing to the file specification of the image to be activated in the created process. The image name can have a maximum of 63 characters. If the image name contains a logical name, the equivalence name must be in a logical name table that can be accessed by the created process.

**input**

Address of a character string descriptor pointing to the equivalence name string to be associated with the
logical name SYSINPUT is the logical name table for the created process. The equivalence name string can have a maximum of 63 characters.

**NAME**

Address of a character string descriptor pointing to the equivalence name SYSINPUT in the logical name table for the created process. The equivalence name string can have a maximum of 63 characters.

**KEY**

Address of a character string descriptor pointing to the equivalence name string to be associated with the logical name SYSERROR in the logical name table for the created process. The equivalence name string can have a maximum of 63 characters.

**MASK**

Address of an 64-bit mask defining privileges for the created process. The mask is formed by setting the bits corresponding to specific privileges. The SPRVDEF macro defines the symbolic names for the bit offsets. For more information see the VAX/VMS System Services Reference Manual (Ref. 6).

**ADDR**

Address of a list of values assigning resource quotas to the created process. If no address is specified, or the address is specified as 0, the system supplies default values for the resource quotas.

**CMD**

Address of a character string descriptor pointing to a 7- to 15-character process name string to be assigned to the created process. The process name is implicitly qualified by the group number of the caller, if a subprocess is
This report of a subject is intended to provide a detailed account of the current process in progress. If not specified in the original report, the process shall be
(1) The amount of subprocesses that a process can create is controlled by the subprocess quota (PRCLM); the quota account is returned when a subprocess is deleted.

(2) The create process system service requires system dynamic security.
(4) Each subprocess is isolated, the value of any variable may be referenced from the main value the variable has at creation and when the subprocess is initiated. The current memory of any subprocess that is added back to the main available to the system, any period time value is shared by the system and all the subprocesses.

(5) Some other conditions are not satisfied until the created process executes. These conditions include a model of execution issue, initial secondary, system, or system logical super dimension, and not the system of insufficient privilege to execute the requested issue.

(6) If a limit will be specified, the address is not used unless the created process actually terminates. At the time of initial system service is issued for the address is the status of the terminating process and an executing message is sent to the address. If the address no longer exists, cannot be restored, or is full, the error is treated as if no address had been specified.

(7) All subprocesses created by a process must terminate before the creating process can be deleted. If subprocesses exist when their creator is deleted, they are automatically deleted.

(8) A detached process cannot run an image containing a call to the non-cline library procedure LIBSEQ_UNLINK; this restriction exists because no CII is defined when the new process is created.

90
SOIO- QUEUE I/O REQUEST

The Queue I/O Request system service initiates an input or output operation by queuing a request to a channel associated with a specific device. Control returns immediately to the issuing process, which can synchronize I/O completion in one of the three ways:

1. Specify the address of an ASR routine that is to execute when the I/O completes.

2. Wait for a specified event flag to be set.

3. Pull the specified I/O status block for a completion status.

When the service is invoked, the event flag is cleared (event flag 0, if not specified); if the IOSS argument is specified, the I/O status block is cleared.

High-level Language Format

SOIL910(<cfn>,<chan>,<func>,<iose>,<astadr>,<astprn>,
       <p1>,<p2>,<p3>,<p4>,<p5>,<p6>)

<cfn>   Number of the event flag that is to be set at requested completion. If not specified, it defaults to 0.

<chan>  Number of the I/O channel assigned to the device to which the request is directed.

<func>  Function code and modifier bits that specify the operation to be performed. The code is expressed symbolically. For reference purposes, the function codes are listed...
in VAX/VMS System Service Manual Appendix A, Section A.2. Complete details on valid I/O function codes and parameters required by each are documented in the VAX/VMS I/O User's Guide.

**IAEB**

Address of a quadword I/O status block that is to receive final completion status.

**ASTAD**

Address of the entry mask of an AST service routine to be executed when the I/O completes. If specified, the AST routine executes at the access mode from which the SQIO service was requested.

**ASTPAR**

AST parameter to be passed to the AST service routine.

**DI TO DE**

Optional device- and function-specific I/O request parameters.

**Privilege Restrictions**

The Queue I/O Request system service can be performed only on assigned I/O channels and only from access modes that are equal to or more privileged than the access mode from which the original channel assignment was made.

**Resources Required/Returned**

(1). Queued I/O requests use the process's quota for buffered I/O (BIOLN) or direct I/O (DIOLN); the process's
buffered I/O byte count (BYTLM) quota; and, if an AST routine is specified, the process's AST limit quota (ASTLHM).

(2). System dynamic memory is required to construct a data base to queue the I/O request. Additional memory may be required on a device-dependent basis.

Notes

(1). The specified event flag is set if the service terminates without queuing an I/O request.

Figure A.2 I/O Status Block.

(2). The I/O status block has the following format:

status

Completion status of the I/O request.

byte count
Number of bytes actually transferred. Note that for some devices this contains only the low-order word of the count. For information on specific devices, see the VAX/VMS I/O User's Guide.

Device- and function-dependent information varies according to the device and operation being performed. The information returned for each device and function code is documented in the VAX/VMS System Services, I/O User's Guide [Ref. 11].

(3). Many services return character string data and write the length of the data returned in a word provided by the caller. Function codes for the SQIO system service (and the LENGTH argument of the $OUTPUT system service) require length specifications in longwords. If lengths returned by other services are to be used as input parameters for SQIO requests, a longword should be reserved to ensure that no error occurs when SQIO reads the length.

**SQIOH—Queue I/O Request and Wait for Event Flag**

The Queue I/O Request and Wait for Event Flag system service combines the SQIO and $WAITPR (Wait for Single Event Flag) system services. It can be used when a program must wait for I/O completion.

**High-level Language Format**

```
SYSSQIOH (<efn>, <chan>, <func>, <iosb>, <astadr>, <astprm>, <p1>, <p2>, <p3>, <p4>, <p5>, <p6>)
```

**efn**
Number of the event flag that is to be set at request completion. If not specified, it defaults to 0.

**chan**

Number of the I/O channel assigned to the device to which the request is directed.

**func**

Function code and modifier bits that specify the operation to be performed. The code is expressed symbolically.

**ioab**

Address of a quadword I/O status block that is to receive final completion status.

**mask**

Address of the entry mask of an AST service routine to be executed when the I/O completes. If specified, the AST routine executes at the access mode from which the SQIO service was requested.

**astPar**

AST parameter to be passed to the AST completion routine.
Optional device-and-function-specific I/O request parameters.

The first parameter may be specified as P1 or PIV, depending on whether the function code requires an address or a value, respectively. If the keyword is not used, P1 is the default; that is, the argument is considered an address. P2 through Pn are always interpreted as values.

Privilege Restrictions

See the description of the SQIO system service for details.

Resources Required/Returned

See the description of the SQIO system service for details.

Notes

See the description of the SQIO system service for details.

SREADER- Read Event Flags

The Read Event Flags system service returns the current status of all 32 flags in a local or common event flag cluster.

High-level Language Format

SREADER(ofn, state)

ofn

Number of any event flag within the cluster to be read. A flag number of 0 through 31 specifies cluster 0, 32 through 63 specifies cluster 1, and so forth.
Address of a longword to receive the current status of all event flags in the cluster.

**SETAFF** = **SET EVENT FLAG**

The Set Event Flag system service sets an event flag in a local or common event flag cluster to 1. Any processes waiting for the event flag resume execution.

**High-level Language Format**

```
SETAFF(efn)
```

*efn* Number of the event flag to be set.

**SETTINTER** = **SET TIMER**

The Set Timer system service allows a process to schedule the setting of an event flag and/or the queuing of an ATG at some future time. The time for the event can be specified as an absolute time or as a delta time.

When the service is invoked, the event flag is cleared (event flag 0 is cleared, if none is specified).

**High-level Language Format**

```
SETTINTER(efn, daytim, astadr, requidt)
```
Event flag number of the event flag to set when the time interval expires. If not specified, it defaults to 0.

EXTIME
Address of the quadword expiration time. A positive time value indicates an absolute time at which the timer is to expire. A negative time value indicates an offset (delta time) from the current time.

ENTRY
Address of the entry mask of an AST service routine to be called when the time interval expires. If not specified, it defaults to 0, indicating no AST is to be queued.

REQID
Number indicating a request identification. If not specified, it defaults to 0. A unique request identification can be specified in each set timer request, or the same identification can be given to related timer requests. The identification can be used later to cancel the timer request(s). If an AST service routine is specified, the identification is passed as the AST parameter.

Resources Required/Returned
(1) The Set Timer system service requires dynamic memory.

(2) The Set Timer system service uses the process's quota for timer queue entries (TQELS) and, if an AST service routine is specified, the process's AST limit quota (ASTLN).
The access node of the caller is the access node of the request end of the EST.

(2) If a specified absolute time value has already passed, the timer expires at the next system clock cycle (that is, within 10 milliseconds).

(3) The Convert ASCII String to Binary Time (SDBINTT) system service converts a specified ASCII string to the quadword time format required as input to the SSETIM service.

**STBLLOG—Translate Logical Name**

The Translate Logical Name system service searches the logical name tables for a specified logical name and returns an equivalence name string. The process, group, and system logical name tables are searched in that order.

The first string match returns the equivalence string into a user-specified buffer; the search is not iterative.

**High-level Language Format**

`STBLLOG(logname, <rslen>,` `rsbuf, <table>, <acnode>, <dsbask>)`

`logname`

Address of a character string descriptor pointing to the logical name string.

93
Address of a word to receive the length of the translated equivalence name string.

Address of a character string descriptor pointing to the buffer that is to receive the resultant equivalence name string.

Address of a byte to receive the number of the logical name table in which the match was found. A return value of 0 indicates that the logical name was found in the system logical name table; 1 indicates the group table, and 2 indicates the process table.

Address of a byte to receive the access mode from which the logical name table entry was made. Data received in this byte is valid only if the logical name match was found in the process logical name table.

Mask in which bits set to 1 disable the search of particular logical name tables. If bit 0 is set, the system logical name table is not searched; if bit 1 is set, the group logical name table is not searched; if bit 2 is set, the process logical name table is not searched.
If no mask is specified or is specified as 0 (the default), all three logical name tables are searched.

**Note**

If the first character of a specified logical name is an underscore character (_), no translation is performed. However, the underscore character is removed from the string and the modified string is returned in the output buffer.

**SYNWAIT** - Wait for Single Event Flag

The Wait for Single Event Flag system service tests a specific event flag and returns immediately if the flag is set. Otherwise, the process is placed in a wait state until the event flag is set.

**High-level Language Format**

`SYNWAIT (efn)`

*efn*

Number of the event flag for which to wait.

**Note**

The wait state caused by this service can be interrupted by an asynchronous system trap (AST) if (1) the access mode at which the AST executes is more privileged than or equal in privilege to the access from which the wait was issued and (2) the process is enabled for ASTs at that access mode.
When the AST service routine completes execution, the system repeats the $WAITPR request. If the event flag has been set, the process resumes execution.

**LIB$SPAWN** - Spawn a Subprocess

LIB$SPAWN requests the calling process's Command Language Interpreter (CLI) to spawn a subprocess for executing CLI commands. LIB$SPAWN provides the same function as the DCL SPAWN command. The subprocess inherits the following from the caller's environment:

- Process logical names
- Global and local CLI symbols
- Default device and directory
- Process privileges
- Process nondeductible quotas
- Current command verification setting

For more information see the VAX/VMS Command Language User's Guide.

**Format**

```
ret-status = LIB$SPAWN(<command-string.rt.dx> ',
<input-file.rt.dx> <output-file.rt.dx>', <flags.rlu.r>',
<process-name.rt.dx> ', <process-id.wlu.r>',
<completion-status.wlc.r>', <completion-efn.rbu.r>',
<completion-astadr.szeu.r>', <completion-astprm.rz.v>>>)
```
**command-string**

A CLI command to be executed by the spawned subprocess. If omitted, commands are taken from the file specified by `input-file`. See notes below for additional information. Passed by descriptor.

**input-file**

An equivalence name to be associated with the logical name `SYS$INPUT` in the logical name table for the subprocess. If omitted, the default is the caller's `SYS$INPUT`. See notes below for additional information. Passed by descriptor.

**output-file**

An equivalence name to be associated with the logical names `SYS$OUTPUT` and `SYS$ERROR` in the logical name table for the subprocess. If omitted, the default is the caller's `SYS$OUTPUT`. Passed by descriptor.

**flags**

A longword of flag bits designating optional behavior. If omitted, the default is that all flags are clear. Passed by reference. The flags defined are:

**Bits 0 NOWAIT**

If set, the calling process continues executing in parallel with the subprocess. If clear, the calling process hibernates until the subprocess completes.

**Bit 1 NOCLISYN**

If set, the spawned subprocess does not inherit CLI symbols from its caller. If clear, the subprocess inherits all currently defined process logical names. You may want to specify `NOLOGNAM` to help prevent commands redefined by logical name assignments from affecting the spawned commands.
Bits 3 through 31 are reserved for future expansion and must be zero.

**process-name**

The name desired for the subprocess. If omitted, a unique process name will be generated. Passed by descriptor.

**process-id**

The longword to receive the process identification of the spawned subprocess. This value is only meaningful if the NOWAIT flags bit is set. Passed by reference.

**completion-status**

The longword to receive the subprocess' final completion status. If the NOWAIT flags bit is set, this value is not stored until the subprocess completes; use the completion-efn or completion-astadr parameters to determine when the subprocess has completed. Passed by reference.

**completion-efn**

The number of a local event flag to be set when the spawned subprocess completes. If omitted, no event flag is set. Specifying this parameter is only meaningful if the NOWAIT flags bit is set. Passed by reference.

**completion-astadr**

The entry mask of a procedure to be called by means of an AST when the subprocess completes. Specifying this parameter is only meaningful if the NOWAIT flags bit is set and if completion-astadr has been specified.

**completion-astparm**

A value to be passed to the procedure specified by completion-astadr as an AST routine parameter. Typically, this would be the address of a block of storage to be read or written by the AST procedure. Specifying this parameter...
is only meaningful if the NOWAIT flags bit is set and in Completion-addr has been specified.

Notes

If neither command-string nor input-file is present, command input will be taken form the parent terminal. If both command-string and input-file are present, the subprocess will first execute command-string and then read from input-file. If only command-string is specified, the command will be executed and the subprocess will be terminated. If input-file is specified, the subprocess will be terminated either by a LOGOUT command or an end-of-file.

The subprocess does not inherit process-permanent files, nor procedure or image context. No LOGIN.COM file is executed.

Unless the NOWAIT flags bit is set, the caller's process is put into hibernation until the subprocess completes. Because the caller's process hibernates in supervisor mode, any user-mode ASTs queued for delivery to the caller will not be delivered until the caller reawakes. Control can also be restored to the caller by means of an ATTACH command or a suitable call to LIB$ATTACH from the subprocess.

This procedure is supported for use with the DCL command language interpreter. If used when the current CLI is MCR, the error status LIB$NOCLI will be returned.

If an image is run directly as a subprocess or as a detached process, there is no CLI present to perform this function. In such cases, the error status LIB$NOCLI is returned.

LIB$STOP- Stop Execution via Signaling

LIB$STOP is called whenever your program must indicate an exception condition or output a message because it
is impossible to continue execution or return a status code to the calling program. LIB$STOP scans the stack frame by frame, starting with the most recent frame, calling each established handler (see the VAX-11 Run-Time Library User's Guide). LIB$STOP guarantees that control will not return to the caller.

**Format**

CALL LIB$STOP (condition-value.rl.c.r<,parameters.rl.v,...>)

**condition-value**

A standard signal for a VAX-11 32-bit condition value. Passed by immediate value.

**parameters**


**Notes**

The argument list is copied to the signal argument list vector, and the PC and PSL of the caller are appended to the signal vector.

The severity of condition-value is forced to SEVERE before each call to a handler.

If any handler attempts to continue by returning a success completion code, the error message ATTEMPT TO CONTINUE FROM STOP is printed and your program exists.

If a handler calls SYS$UNWIND, control will not return to the caller, thus changing the program flow. A handler can also modify the saved copy of R0/R1 in the mechanism vector.
The only way a handler can prevent the image from exiting after a call to LIB$STOP is to unwind the stack using the SYS$UNWIND system service.
APPENDIX B
ETHERNET LOCAL AREA NETWORK

A convenient method of connecting computers over short distances is the Ethernet local area computer network. In fact, Ethernet has now been recognized by more than a dozen manufacturers as the de facto standard for local area computer communications.

The 10 Mbit per second, packet switching network is designed to interconnect hundreds of high-function computers or workstations within 2.5 kilometers of each other. Ethernet uses a passive, equitable, highly efficient statistical method known as carrier-sense multiple-access with collision detection (CSMA/CD) that enables stations on the network to share access to a 50-ohm coaxial cable transmission medium. A cable segment can be up to 500 m long and connect up to 100 stations. Each station attaches to a coaxial cable via a transceiver system, through a cable that connects the transceiver to the station and can not exceed 50 m in length.

Messages are formatted into standard frames made up of bytes. Framing consists of a destination portion (6 bytes), a source portion (6 bytes), the message type (2 bytes), data (46 to 1500 bytes), and a frame-check sequence (4 bytes). Messages can be addressed to a single station, to all stations (broadcast), or to a number of selected stations. Signals are transmitted using Manchester encoding, a means of combining separate data and clock signals into a single, self-synchronizable data stream suitable for transmission on a serial channel.

The CSMA/CD approach can be summarized as follows:
Carrier-sense means that each station "listens" to the cable before transmitting a packet; if some other station is already transmitting, the first station senses the presence of a carrier and defers transmitting its own packet until the cable is quiescent.

Multiple-access means that all stations tap into and share the same coaxial cable. Every transmitted packet is "heard" by all stations on the Ethernet. The intended recipients detect incoming packets by recognizing their addresses embedded in the packets; other packets are discarded.

If two or more stations transmit packets at the same time, their signals will be intermixed on the coaxial cable. This is known as a collision. By listening while transmitting and comparing what is heard on the cable with the data being transmitted, each station can detect collisions and back off by waiting a random time interval before attempting to retransmit the packet. The efficiency of the network remains high even under conditions of heavy load, because the mean of the random back off interval increases each time a collision occurs.

Related Information

In the Figure B.1 is depicted in steps the decision of selecting which medium to pick for a local network. A user can move from left to right across the selection "tree", checking the distances, bandwidth, and applications supported by twisted pair, baseband, and broadband medium classes. Optical fiber currently seems best only for point-to-point communications.

Like the physical medium, choices are available for the access method. Figure B.2 is a selection tree to determine the optimal access method for a specific operating environment. So, a prospective user can again can move from left to right to see the transmission characteristics supplied by the three most popular access techniques:
Figure B.1  The Local Area Network medium selection tree.
Figure B.2 Selecting the access method.
- Slotted time division multiple access
- Token passing
- Carrier sense multiple access with collision detection
APPENDIX C
NI1010 BOARD DESCRIPTION-FEATURES

DESCRIPTION

The NI1010 UNIBUS Ethernet Communications Controller is a single hex-height board that contains all the data communications controller logic required for interfacing DEC's family VAX-11 and PDP-11 minicomputers to an Ethernet local area network. The controller board complies in full with the Xerox/Intel/Digital Ethernet V1.0 specification by performing the specified CSMA/CD data link and physical channel functions. Also, when attached to a transceiver unit, provides the host UNIBUS system a complete connection onto the Ethernet baseband coaxial cable local area network, permitting 10 Mbit per second transmissions over distances up to 2500 meters.

FEATURES

Implements Ethernet Data Link Layer Functions

The NI1010 formats frames and performs the CSMA/CD transmit link management functions required to successfully deliver frames onto the network. When not transmitting a frame, the NI1010 continuously listens to the network for frame traffic intended for it. Only frames with a matching address are accepted by the controller for subsequent transfer to the host UNIBUS system. The controller performs Physical, Multicast-Group (up to 63), and Broadcast address recognition. CRC generation and checking is also performed.

Implements Ethernet Physical Channel Functions The NI1010 transmits and receives 10 Mbits per second bit streams with
electrical and timing specifications compatible for direct connection to an Ethernet transceiver unit. The controller performs the required frame synchronization functions, and Manchester encoding/decoding of the bit streams.

**Supports High Station Performance**

The NI1010 has being designed to offer high network performance while minimizing the service load placed upon the host UNIBUS system. Serving to buffer the host from the unpredictable interarrival times characteristic of network traffic, the board has a receive FIFO (first-in, first-out) memory which can store up to 13.5 Kbytes of received frames. For transmit buffering, the NI1010 has a 1.5 Kbyte FIFO from which all frame retransmissions are made. All data transfers between the NI1010 and host UNIBUS memory are performed by the NI1010's DMA controller. The DMA controller may be preloaded by the host with up to 15 receive buffer descriptors.

**Extensive Diagnostic Features:**

The NI1010A controller offers comprehensive network and board-level diagnostic capabilities. LED indicators mounted on the edge of the board provide a visual indication of whether or not the host is communicating onto the network. For a comprehensive station diagnosis, the NI1010A may be operated in three different types of data loopback. On power-up, or by host command, the controller performs a confidence test on itself. A LED indicator shows the pass/fail operational state of the board.

**Collects Network Statistics:**

The NI1010A tallies statistical values on various network traffic and error conditions observed.
One Hex-Height UNIBUS Board:

The NI1010A fits into one UNIBUS SPC slot. The controller is mechanically, electrically, and architecturally compatible with Digital Equipment Corporation's UNIBUS specifications.

SPECIFICATIONS

- 10 million bit per second data transmission rate
- Coaxial cable segments up to 500 meters (1640 feet) in length.
- Up to 100 transceiver connections per coaxial cable segment.
- Up to 2 repeaters in path between any two stations.
- Up to 1500 meters (4920 feet) of coaxial cable between any two sections.
- Up to 50 meters (165 feet) of transceiver cable between an NI1010A controller and its transceiver.
- Up to 2500 meters (1.55 miles) maximum station separation.
- Point-to-point links up to 1000 meters (3280 feet) in length.
- Up to 1024 stations per network.
APPENDIX D
NI1010 ETHERNET CONTROLLER MULTIPLEXING USER'S MANUAL

A. GENERAL INFORMATION

"Ethermult" and "Usermult" are two programs that provide the means for the multiplexing of the NI1010 Ethernet Communications Controller Board which takes the role of the interface between VAX-11/780 and Ethernet. In their present form, they enable nine users to access the VAX/VMS facilities from an MDS terminal provided that:

1. The program "Ethermult" is running in a VAX/VMS terminal.
2. Each user has his own "Usermult" program running in a VAX/VMS terminal.

In this scheme, each user can execute VMS commands from his terminal as if he had a real VAX/VMS terminal to do his job.

B. SPECIFIC INFORMATION

Both programs reside on the VAX/VMS under the public user account with user name "INTERLAN" and password "VMS".

First thing that a person willing to work with the multiplexing should have, is an account in VAX/VMS. This can be easily arranged through Mrs Olive M. Paek of the VAX-11 professional staff (Rm 525B).

Next he should login in a VAX/VMS terminal and type the following commands:

$COPY <CR>
$FROM: _DRA1: INTERLAN ETHERMUL.EXE <CR>
$TO: * <CR>
The same commands should be repeated for the "USERMULT.EXE" file.

Now, the "Ethermult" should be executed from a VMS terminal and after that, the "Userault" program should also be executed in as many VMS terminals as many users are required. This can be done by typing:

$ R ETHERMULT <CR>
$ R USERNULT <CR>

All programs, as they are set up now, must be executed from one directory i.e. the login procedure at each VMS terminal should be done using the same user name and password. This happens since the main program ("Etherult") can only have access in the answer files that reside on its directory. That is, the answer file of the user, say, 9 (REPL9.DAT) should be in the same directory the ETHERMULT.EXE is executing. Otherwise, when the program tries to find it in order to send it to MDS it will fail since they reside in different directories.

1. Operation on MDS Systems

After the required number of "USERMULT" programs along with the "ETHERMULT" have been executed in different VMS terminals, the modified "ETHERNET" program (see Appendix E) must be executed in as many MDS terminals (currently only two are available) as the number of users is.

There are two diskettes with the same "ETHERNET" program, one for the single and one for the double density MDS. The procedure in the MDS side, same for either of them, is the following:

When the system is booted up with the corresponding diskette, execute ETHERNET.COM by typing:

A> ETHERNET <CR>
Now, a series of prompts appears on the MDS screen:

ETHERNET COMMUNICATION PROGRAM—VERSION 5.0
ALLOWS THIS HOST TO CONNECT TO THE NET.
CNTL-H=BACKSPACE FOR TEXT ENTRIES:

********** MAIN MENU **********
WRITE RECEIVED FILES TO DISK NO:
DEFAULT DRIVE(A) = 1
DISK DRIVE A = 2
DISK DRIVE P = 3

ENTER DRIVE NUMBER=>

The user can enter the drive number he wishes without affecting the program, since no data is going to be transferred to the MDS diskettes. After typing in one of the three numbers (1, 2 or 3) another set of prompts appears:

ETHERNET FRAME DATA BLOCK SIZE:
SELECT 128 FOR ALL FILE OPERATIONS
AND VAX COMMUNICATIONS.
128 BYTES = 1
256 BYTES = 2
512 BYTES = 3
1024 BYTES = 4
1500 BYTES = 5

ENTER SELECTION=>
Here also any selection will not affect the program since the frame size is fixed to 1530 bytes when the virtual terminal mode (nr. 3) is selected in the next set of prompts:

OPERATING MODES:

<table>
<thead>
<tr>
<th>Mode Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECEIVE WAIT LOOP</td>
<td>1</td>
</tr>
<tr>
<td>TRANSMIT FILE OR MESSAGE</td>
<td>2</td>
</tr>
<tr>
<td>VIRTUAL TERMINAL OF VAX</td>
<td>3</td>
</tr>
<tr>
<td>VAX COMMAND MODE</td>
<td>4</td>
</tr>
<tr>
<td>DISCONNECT FROM NET</td>
<td>5</td>
</tr>
</tbody>
</table>

ENTER SELECTION=>

Now the "VIRTUAL TERMINAL OF VAX" (nr. 3) must be selected.

After this, a "V>" appears and the user is ready to type his own commands. They should be the usual VMS commands, preceded by the characteristic number of the user, selected when the "Usermult" program was executed.

Example: V>1dir <CR> (for the user nr 1) or V>3show time <CR> (for the user nr 3)

When the answer to the command appears on the screen, a new command can be typed in after the "V>" readback.

If the user wants to finish his session, he can type a "." <CR> and the sets of selection prompts appear again. If he wants to logout of the VAX/VMS multiplexing system he should type "LO" and the "Usermult" corresponding to him will exit. From then on, this user cannot enter the
multiplexing system except if he runs again the "Usermult" program from a VMS terminal.
APPENDIX E

AUTHORS: MAJOR ANTONIOS SAKELLAROPoulos
HELlenic Air Force

LIEUTENANT IOA'VITIS KIDONIEFS
HELlenic Navy

THESIS ADVISOR: PROF. UVO KODRES

Naval Postgraduate School, December 1983

This program performs the multiplexing of Ethernet Interface among VAX users. It should run in conjunction with program "Usermult". Detailed description of both programs can be found in the thesis of the authors, under the title "Multiplexing the Ethernet Interface Among VAX users".

program ethermult

variables' declaration:

imlicit integer*4(a-z)
integer*2 iosb(2), c, condition
integer*4 nchan, sys$diow, sys$sassign
include 'dra0$(nossys, lanassign)ndef.for'
include '$(ioedef)'
include '$(ssdef)'
byte MRpacket(1522), MRtoacket(1508), usnum,
1
dflag, flag, com(81),
2
row,rowflag
character alpha(9)'/1', '2', '3', '4', '5', '6', '7', '8', '9'/,
1
mailbox$7'/usrmail'/,
2
msq$27'/Invalid user #. Msq ignored'/,
3
msgl$27'/Missing user #. Msq ignored'/
external dummy
common nchan/oak$queue(9,81)/f1/ackflag/slot/n,
1
/ind1/notyet(9)/ind2/times(9)/filunit/unitnr

Initializations:

c=0
unitnr=0
do i=1,9
  do j=1,4
    table(i,j)=0
  enddo
  do l=1,81
    mqueue(i,l)=0
  enddo
  notyet(l)=0
enddo
m=11
do i = 1,9
  times(i) = m
m = m + 1
end do

C Associate the common event flag cluster \( \text{FET} \):
status = systa$asc(\text{fuc}(\text{val}(o4),'net'),
if (.not.status) call lib$stop(\text{val}(status))

C Initialize common event flags \# 24 to \# 34 to zero:
do i=24,34
  status = sys$sclref(\text{val}(i))
  if (.not.status) call lib$stop(\text{val}(status))
end do

C Assign a channel to 'NIA0:
  istat$sysassign('NIA0',nichan,)
  if (.not.istat) tvoe *, 'Assign error!'

C Start up and go on line:
  istat = sys$close(\text{val}(nichan),
  1 \text{val}(io$s+setmode .or. io$s=startup),
  2 \text{iosb},
  if (.not.istat) tvoe *, 'Istat start up error!'
  if (iosb(1).lt.0) tvoe *, 'Start up error!'

C Initial setting of the receive-mode:
call rec(M$packet) ! Receive the command with the user number.

10 i = 1
  condition = 0
  do while ((condition.eq.0).and.(i.le.9).and.(mqueue(i,7).ne.0))
    usrnun = mqueue(i,19)
    usrnun = usrnun - 48
    ! "Convert" to decimal.
  c Check the user number:
  if (usrnum.eq.0) then ! Invalid user number.
    call message(msg,mqueue(i,15),mqueue(i,16))
    do j = 1,81
      mqueue(i,j) = 0
    ! Clear this slot.
    enddo
    call arrqueue
    go to 10
  else if (usrnum.gt.9) then ! Missing user number.
    call message(msg1,mqueue(i,15),mqueue(i,16))
    do j = 1,81
      mqueue(i,j) = 0
    enddo
    call arrqueue
    go to 10
  end if
  end do
  call search(usrnum,table,c,rowflag,row)
  if (table(row,2).eq.1) then ! Reply file for previous command
    ! of this user hasn't been sent yet.
    i = i + 1
    ! serve next user.
  else
    condition = 1
  end if

C Extract the command and out it in a buffer:
do j=1,81
  com(j)=mqueue(i,j) ! Load the buffer with the command.
  mqueue(i,j)=0  ! Zero the command buffer.
endo
call arrqueue
c Write the command to the user's mailbox:
call distribution(com,usrnun,dflag,table,msq,c)
    if (dflag.eq.1) go to 10 ! No such user number finally.
    end if
end do

c Send the command to the user:
call export(c,table,'MROacket')

c Check the status of the users and update user table and number of current users, if necessary.
call status=check(c,table)
go to 10
20 end

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

subroutine SENDMSG(outfile,ans,usrnun,'MROacket,c,table)
   c ACTUAL SENDING OF MESSAGE.
   c WAIT FOR ACKNOWLEDGE.
   c IF NOT ACKNOWLEDGE IN 5 SEC, RETRANSMIT.
   c IF NOT ACKNOWLEDGE IN 10 SEC, TRANSMISSION.
   c
   integer*4(a-z)
   integer*2 iosp(2),endfil,nchars,last /0/,c
   integer*4 nchan,sys$io$, sys$assign
   include 'e-tn: (ncssys,interlanindef.for'
   include '(/io$def)'
   include '(%sys$def)'
   byte Temobuff(80),ans(2),usrnun,'MROacket(1522)
   byte SToacket(150A),SRoacket(1522),row,
   1 table(9,4),rowflag
   character outfile*9,del*6/'$ del1 '/' vers*2//'**/',
   1 delfile*17,
   2 alpha(9)/'1', '2', '3', '4', '5', '6', '7', '8', '9'/
   common nchan/fi/ackflag/ind1/notyet(9)/ind2/times(9),
   1 /filunit/unitnr

delfile=del1//outfile//vers

   c Assign destination address:
   SToacket(1)='02'x
   SToacket(2)='07'x
   SToacket(3)='01'x
   SToacket(4)='00'x
   SToacket(5)=ans(1)
   SToacket(6)=ans(2)
   SToacket(7)='00'x
   SToacket(8)='00'x

delfil=0
20 do i=9,1538
S\text{t}\text{o}\text{a}c\text{k}e\text{t}(i) = '00'x

\text{enddo}

j = 9
\text{do while}(j, i.e., 1425) ! store no more than 1500 characters in S\text{t}\text{o}\text{a}c\text{k}e\text{t}
\text{read}(t\text{i}m\text{e}s(\text{unit}n\text{r}), 21, \text{end}=40) \text{nchars}, (\text{T}e\text{m}\text{o}b\text{u}ff(i), i=1, 80)

\text{format}(a, 30x) ! count the nr of characters in the line
\text{do } n=1, \text{nchars}
S\text{t}\text{o}c\text{k}e\text{t}(j) = \text{T}e\text{m}\text{o}b\text{u}ff(n)
\text{ } j = j + 1
\text{enddo}
S\text{t}\text{o}c\text{k}e\text{t}(j) = '00'x ! carriage return at end of line
S\text{t}\text{o}c\text{k}e\text{t}(j+1) = '0A'x ! line feed
\text{j} = j + 2
\text{enddo}

\text{call tranceive(S}t\text{o}c\text{k}e\text{t}) ! send the packet
\text{status} = \text{sys}w\text{ai}t\text{fr}(\text{val}(2))
\text{if } (\text{.not. status}) \text{ call } \text{lib}st\text{oo}(\text{val}(\text{status}))

\text{Check if an acknowledge was sent:}
\text{if } ((\text{\#Packet}(18).eq.'FF'x) .or. (\text{ackflag}.eq. \text{.ss$\&wasset})) \text{ then}
\text{status} = \text{sys}sc\text{lr}ef(\text{val}(2))
\text{if } (\text{.not. status}) \text{ call } \text{lib}st\text{oo}(\text{val}(\text{status}))
\text{if } (\text{endfil}.ne.-1) \text{ GO TO 50 } ! \text{ Send only one frame of this file.}
\text{go to 47}
\text{end if}

\text{close(unit} = \text{times(\text{unit}n\text{r}))}
S\text{t}\text{o}c\text{k}e\text{t}(8) = '0F'x ! For last packet recognition from M\text{D}S.
\text{endfil} = -1 ! Flag raised at end-of-file.
\text{go to 28} ! Transmit the contents of the last packet.

\text{status} = \text{lib}sp\text{awn}(\text{del}file)
\text{if } (\text{.not. status}) \text{ call } \text{lib}st\text{oo}(\text{val}(\text{status}))
\text{status} = \text{sys}sc\text{lr}ef(\text{val}(63+\text{usernum}))
\text{if } (\text{.not. status}) \text{ call } \text{lib}st\text{oo}(\text{val}(\text{status}))
\text{call search(usernum, table, c, rowflag, row)}
\text{if } (\text{rowflag}.eq.1) \text{ then}
\text{table(row, 2)} = 0 ! User free to enter distribution if needs so.
\text{end if}

\text{if } (\text{endfil}.ne.-1) \text{ then}
\text{notyet(usernum)} = 1
\text{else}
\text{notyet(usernum)} = 0
\text{end if}
\text{return}
\text{end}

\text{subroutine tranceive(T}o\text{ack)}
implicit integer*4(a-z)
integer*2 ios0(2)
integer*4 nihan
include 'dra0,(nossys.interlan)nidef.for'
include '(3iojef)'
include '(3issdef)'
byte Toack(1508)
common nihan

Load transmit data and send:

stat=sys$giow(,val(nihan),
1 val(in+1rds),
2 ios0,Toack,val(1508),...,)

if(iosb(1).lt.0) call lib$stop(val(iosb(1)))
if(iosb(2).ne.0) call lib$stop(val(iosb(2)))

return
end

subroutine distribution(com,usernum,dflag,table,msq,c)

This routine searches the user information table to find out
whether a given number exists as a valid user number. If it is so
it puts the message into a mailbox that is indexed with the number
of the user to whom the msg was addressed.
If the given number was not a valid user number, the message is
ignored. In this case the "dflag" is returned with value "1".

implicit integer*4(a-z)
include 'dra0,(nossys.interlan)nidef.for'
include '(3iojef)'
include '(3issdef)'
byte dflag,com(81),table(9,4),usernum,row, 1 rowflag
character aloha(9)'/1','2','3','4','5','6','7','8','9/', 1
mailbox*7'/usrmial'/:msq=27
character outfile*4,'/real.','fil*4/./dat'/
integer*2 c,channel(9)

call search(usernum,table,c,rowflag,row)

IF ((rowflag.eq.1).AND.(table(row,2).eq.1)) then ! User number exists
in user table and he has answer in process.

return
ELSE

i = 1
dflag = 0
outfile=outfile//aloha(usernum)//fil

Check if the user number exists in the user info table:

if(c.at.0) then
  do while ((i.le.c).AND.(table(i,1).ne.usernum))
    i = i+1
  end do
if (i.or.c) then ! The user is addressed to an unidentified user
status = sys$readf(zval(72+usernum),usr) ! Check if the
   corresponding flag is set
   if (.not.status) call lin$stop(zval(status))
   if (status.ne.ss$wasset) then ! Not a valid user finally.
      dflag = 1
      call message(msa,com(15),com(16))
      return
   else ! Valid user, update user table
      c = c + 1
      table(c,1) = usernum
      table(c,2) = 1 ! This user has passed through distribution
      table(c,3) = com(15) ! Associate addresses
      table(c,4) = com(16)
   end if
   else ! This user already is authorized to use the system.
      table(row,2) = 1 ! Indicate pass through distribution.
      table(row,3) = com(15) ! Associate addresses.
      table(row,4) = com(16)
   end if
else
   go to b
end if

create mailbox and assign a channel to it:
status=sys$cremx(channel(usernum),....,mailbox//alpha(usernum))
   if (.not.status) then
      type '*', 'Error in creating user mailbox'
      call lib$stop(zval(status))
   end if
write the command to user mailbox:
status=sys$sqow(,zval(channel(usernum)),zval(io+1tds),....
   if (.not.status) then
      type '*', 'Error in writing user's mailbox'
      call lib$stop(zval(status))
   end if
END IF

return
end

subroutine export(c,table, "Roacker")

This routine finds out which user has priority to send his
rely to NI110 controller and sends one frame (1500 bytes) of
it. Then proceeds to the next ready answer, sends one frame and
so on, until all users with ready answer have send one frame.

implicit integer*4(a-z)
inquire C '(*$def)'
integer*2 c,i

120
if (c.eq.0) return

do i = 1, c
  usernum = table(i,1)
  outfile = out//aloha(usernum)//fil
  m = 1
  do while(outfile(5:5).ne.aloha(m))
    m = m + 1
  end do
  unitnr = m
  status = sys$readf(val(63+usernum),usr)
  if (.not.status) call lib$tosts%val(status)
  if ((status.eq.ss$wasset).and.(notyet(usernum),eq.0)) then
    openunit=times(unitnr),file=outfile,status='old')
  end if
  if (status.eq.ss$wasset) then ! There is an answer.
    addr(1) = table(i,3) ! Form the address of
    addr(2) = table(i,4) ! the current user.
    outfile = out//aloha(usernum)//fil
    call sendmsg(outfile,addr, usernum, 150acket, c, table)
  end if
  end do

return
end

subroutine message(msg,al,a2)

implicit integer*4(a-z)
character*27 msg
byte al,a2,Toack(1508)

Toack(1) = '02'x
Toack(2) = '07'x
Toack(3) = '01'x
Toack(4) = '00'x
Toack(5) = a1
Toack(6) = a2
Toack(7) = '00'x
Toack(5) = '0F'x

k=9
do i=1,27
  Toack(k)=char(msg(i:i))
  k=k+1
enddo
Toack(k+1)= '00'x
Toack(k+2)= '0A'x

Toack(4+3)='20'x
Toack(4+4)='20'x

call tranceive(Toack) ! send msg and receive acknowledge

return
end

subroutine xmit(Toack, MRoacket)

This subroutine transmits an already formed packet

implicit integer*4(a-z)
integer*2 ioso(2)
integer*4 nican
include '+dra0: nosys_interlanlndef.for'
include '($iodef)'
byte Toack(1508), MRoacket(1522)
common nican/bak/mqueue(9, 81)

Toack(1)= '02'x
Toack(2)='07'x
Toack(3)='01'x
Toack(4)=MRoacket(14)
Toack(5)=MRoacket(15)
Toack(6)=MRoacket(16)
Toack(7)='00'x
Toack(8)='FF'x
Toack(9)='60'x

c
Load transmit data and send:
istat=sys$tiow(2,%val(nican),%val(ios+1nds),
1 ioso,,,Toack,%val(1508),,)

if (iosb(1),lt,0) then
  type *, 'Ether xmit error!!'
call lib$stop(%val(iosb(1)))
else if(iosb(2),ne,0) then
  type *, 'Controller xmit error!!'
call lib$stop(%val(iosb(2)))
else
  i=1 ! dummy
endif

return
end

subroutine dummy(MRoacket)
This is the routine in which control of the program is transferred when a message arrives at the NI1010 board and an AST occurs.

```
implicit integer*4(a-z)
include '[(io.h)]'
include '[(ssdef)]'
byte Mpacket(1522), Toack(1503), com(81)
common/obak/mqueue(9, M1)/fl/ackflag/slot/n

if (Mpacket(18).eq.'FF') then ! Acknowledge packet received.
call rec(Mpacket) ! Reset the receive mode.
ackflag=ackflag+(%val(2)) ! Set the acknowledge flag.
if(.not.ackflag) call lib$stop(%val(ackflag))
else
    call xmit(Toack, Mpacket) ! Command packet received.
call rec(Mpacket) ! Send acknowledge to MDS.
end if
```

```
Extract the command from the received packet:
i=19
do while (Mpacket(i).ne.' ') i=i+1 enddo
k=1
do j=1, i
com(k)=Mpacket(j)
k=k+1
enddo
call arrqueue ! Arrange the command queue.
call formqueue(com) ! Put the command at the first empty slot of the queue.
call rec(Mpacket) ! Reset the receive mode.
end if
return
```

subroutine rec(Mpacket)

This subroutine receives a packet from MDS. This packet can be either a command packet (18th byte = 00) or an acknowledge packet (18th byte = FF). When it is called, it sets up the NI1010 receive mode and exits. Then, as soon as a packet arrives at the NI1010 it interrupts the current flow of the program (AST is "triggered") and calls the AST subroutine DUMMY.

```
implicit integer*4(a-z)
integer*2 iosb(2)
integer*4 nichan
include '[(dra01.inossys.interlan.inidef.for)]'
include '[(ssdef)]'
byte Mpacket(1522), Toack(1503)
common nichan/obak/mqueue(9, M1)
external dummy
```
status=sys$clref(%val(1))
if (not status) call lib$stop(%val(status))

c

type 'Ready to receive',nichan
istat=sys$io(%val(1),%val(nichan),%val(io$readline),
1
iosb, dummy, $Roacket, $Roacket, %val(1522), ...)

if (iosb(1).lt.0) then
  type 'Ether error in reception of msg in VAX/VMS!!'
call lib$stop(%val(iosb(1)))
else if (iosb(2).ne.0) then
  type 'Rcv error in VAX/VMS Ethernet controller!!'
call lib$stop(%val(iosb(2)))
endif
return
end

subroutine formqueue(com)

c
This subroutine outs the new command into the first empty slot
of the command queue.

imlicit integer*4(a-z)
byte com(81), pack(1522)

common/oak/mqueue(9,81)/slot/n

do i=1,81
  mqueue(n,i)=com(i) ! Fill up the first empty slot of the queue.
enddo
return
end

subroutine arqueue

imlicit integer*4(a-z)
common/oak/mqueue(9,81)/slot/n

c
Arrange the queue : 
m=1
do while (m.le.9)
  if (mqueue(m,7).eq.0) and (mqueue((m+1),7).eq.1) then
    do i =1,81
      mqueue(m,i)= mqueue((m+1),i)
      mqueue((m+1),i)= 0
    enddo
  end if
  m = m + 1
enddo

c
Locate the first empty slot in the queue :
n=10
do i=9,1,-1
  if (mqueue(i,7).eq.0) then
n=n-1
end if
end do
if(n.eq.10) type **,*** Queue is full. Command not queued !! ***
return
end

Subroutine status4-check(c,table)
This routine reads the flags of current users to check if they
are still in the system. If a flag was found reset, that means the
corresponding user has logged out. Then the user table and number
c of users in the system (c) are updated. The user table is then
rearranged.

implicit integer*4(a-z)
integer*2 change,i,c,k
byte table(9,4),user
include '(ss$def)'
include '(ss$def)'

change = 0
k = c
if (k.gt.0) then
  do i = 1,k
    user = table(i,1)
    status = sys$readef(%val(72+user),usr)
    if (.not.status) call lib$stoo(%val(status))
    if (status.ne.ss$wasset) then ! This user has logged out.
      change = 1 ! At least one change has occurred.
      c = c - 1
      table(i,1) = 0
    end if
  end do
  if(change.eq.1) call arrange(k,table)
end if
return
end

Subroutine arrange(k,table)
This routine rearranges the user table after at least one user has
left the system, so that there are no empty lines between full ones.

integer*2 m,isk,i
byte table(9,4),temp(9,4)
m = 1
i = 1
do while(i.le.k)
if (table(i,1).ne.0) then
  do j = 1,4
    temp(m,j) = table(i,j)
    table(i,j) = 0
  end do
  i = i + 1
  m = m + 1
else
  i = i + 1
end if
end do

do i = 1, m-1
  do j = 1,4
    table(i,j) = temp(i,j)
  end do
end do

return

end

subroutine search(usernum, table, rowflag, row)
  ! This routine searches the user table to find a specific usernum
  ! which is given. If it finds it there it returns the "rowflag" with
  ! value 1, and the "row" of the table in which this usernum was found.
  ! If the usernum was not found there, the rowflag is returned with
  ! value 0.

  integer*2 c,i
  byte usernum, row, rowflag,
  1       table(9,4)

  rowflag = 0
  row = 0
  i = 1

  do while((i .le. c).and.(usernum.ne.table(i,1)))
    i = i + 1
  end do
  if (i .le. c) then
    row = table(i,1)
    rowflag = 1
  end if

  return
end
APPENDIX F

AUTHORS: MAJOR ANTONIOS SAKELLAROPoulos
HELLENIC AIR FORCE

LIEUTENANT IOANNIS KIDONIEFS
HELLENIC NAVY

THESIS ADVISOR: PROF. UNO KODES

NAVAL POSTGRADUATE SCHOOL, DECEMBER 1983

This program should run together with program "ETHERMULT" in order to achieve the Ethernet Interface multiplexing among VAX users. Detailed description of the program is found in the thesis of the authors under the same title.

program usermult

implicit integer*4(a-z)
integer*2 channel(9),iosb(2),endfil,doneflag,
1 load
byte com(81),ans(2),username
include 'drao:nsys.interlanindef.for'
include '(Siodef)'
character mail*7/'usrmail'/,
2 alpha(9)'1','2','3','4','5','6','7','8','9'/
common

call authorize(username,load) ! Get the username.
if (load.eq.1) go to 101 ! System cannot accept new users.

10 status=sys$crempx(,channel(username),,,mail//alpha(username))
if (.not.status) then
  type '*error in user mailbox creation'
call lib$stop(%val(status))
endif

read the mailbox:
status=sys$ziow(,%val(channel(username)),%val(io$+readlblk),,,
1 %ref(com),%val(81),,,)
if (.not.status) then
  type '*read-user-mailbox error'
call lib$stop(%val(status))
endif

associate common event flag cluster #2 with the name "NET":
status=sys$ascelf($val(64),'net',)
if (.not.status) call lib$stop (%val(status))

call feedfile(com)
call spawn(username)
status = sys$setef($val(63+username))
if (.not.status) call lib$stop(%val(status))
go to 10
subroutine authorize(usernum, load)

This subroutine checks the common event flags 73 to 81 to determine what user numbers are available for a new user. Then it interacts with the user and accepts or rejects an entered user number. If it accepts, the proper common event flag is set. If there are no available user numbers it returns the variable "load" with value "1".

implicit integer*(a-z)
integer*2 load, i, cond,
1 iso(2)
integer*4 flagarray(9), j
byte usernum
include '($iodef)'
include '($ssdef)'
include 'dra0 irresistible for'
external abort
common/cl/cancel

Assign a channel to terminal:
status = sys$assign('tt:', termchan,)
if (.not.status) call lib$stop(%val(status))

Start up and go on line:
status = sys$giow(:, val(termchan),
1 %val(io$+setmode.or.io$+startup),
2 iosb, , , , ,
if (.not.status) call lib$stop(%val(status))

Initializations:
do j=1, 9
   flagarray(j) = 0
end do
counter = 0
cancel = 0
load = 0
cond = 0
j = 1

Associate common event flag cluster #2 under the name "NET":
status = sys$ascefc(%val(64), 'net',)
if (.not.status) call lib$stop(%val(status))

Check if flag #84 is set (system occupied):
01 status = sys$readef(%val(84), state)
if (.not.status) call lib$stop(%val(status))
IF (status.eq.ss$wasset) then ! System occupied.
   if (cond.eq.0) then
      type *, ' Please wait, system occupied'
      cond = 1
      go to 01
   else
      go to 01
      128
else ! System available.
status = sys$setef(%val(P4),'net',') ! Set flag #84
if (.not.status) call lib$stop(%val(status))

C Check what event flags from 73 to 81 are set and out the remaining
ones in flagarray buffer:
do i = 1,9
status = sys$readef(%val(72+i),usr)
if (.not.status) call lib$stop(%val(status))
if (status.ne.sys$-wasset) then ! This user number is available
flagarray(j) = i
j = j+1
end if
end do

j = j-1
if (j.eq.0) then
  type *, 'System full!! No new users at the moment.'
  load = 1
  return
else
  type *, 'You may choose one of the following'
  type *, 'available user numbers.'
do i = 1,j
  write(b,04)flagarray(i)
end do
  end if
04 format(i4)

C Set the timer for 10 seconds:
call sys$bintrim('0 ::10.0','svstime') ! Convert 10 sec. to sys fmt.
call sys$setimr('svstime','systime','') ! Start the timer when

C Get the new user number:
  type *, 'You have 10 sec to enter the user number:
C Read the user's number:
status = sys$giow(',%val(termchan),'%val(ios$readlblk),1
  iosb,'%val(iosb),'usernum,'%val(1),',
if (.not.status) call lib$stop(%val(status))
usernum = usernum - 48

C Cancel the timer:
call sys$cantrim()

C Check if the new user number is acceptable:
i = 1
do while((i.le.j).and.(flagarray(i).ne.usernum))
i = i+1
end do
if ((i.eq.j+1).and.(counter.eq.0)) then ! Wrong number entered.
counter = counter + 1
  type *, 'You have entered an illegal user number!!'
go to 9
else if ((i.eq.j+1).and.(counter.eq.1)) then
  cancel = 1
call abort
else ! Valid user number. Set the proper flag.
status = sys$setef(%val(72+usernum),'net',')
if (.not.status) call lib$stop(%val(status))
status = sys$clref(%val(84)) ! Reset flag #1.
}129
if (.not.status) call lib$stoo(%val(status))
end if

end

SUBROUTINE abort
implicit integer*4(a-z)
common/c1/cancel

c Associate common event flag cluster #2 with the name "NET":
status = sys$ascefc(%val(b4),'net',)
if (.not.status) call lib$stoo(%val(status))

c Clear flag #84 :
status = sys$clref(%val(84))
if (.not.status) call lib$stoo(%val(status))

if (cancel.eq.1) then
  write(6,15)
else
  write(6,16)
end if

15 format(' Second time illegal user number. Program aborted!!')
16 format(' Time has expired. Restart the program.')

call exit
end

SUBROUTINE feedfile(com)

character*23 msgl/'Received successfully.'/
integer*2 iosb(2),first
integer*4 nicher, sys$giow, sys$assign
include '+dra0:inossys.interlan ifndef.for'
include '($ifdef)'
byte RGosocket(1522),TGoacket(1508),ans(2),com(81)

i=20
do while (com(i).eq.ichar(''))
  i = i+1
end do
open (unit=1,file='mail.com',status='old')
write(6,11)(com(j),j=20,i=1)
130
subroutine spawn(usernum)

This program spawns a subprocess for executing CLI commands. The commands that are going to be executed are contained in the file 'Mail.com' which is the input of the runtime routine Lib$spawn. The results of the execution are written in a file called 'Reola.dat'.

implicit integer*4(a-z)
character file*15 /'mail.com'/, esc$null*2
byte esc$null+num(2) /'ib', '00'/, usernum
integer*2 file$len /8/, dflag
equivalence (esc$null, esc$null+num)
external ss$-notran
character outfile*9, string*40, out*4//'reol'//, file*4//, dat'/,
I
character*15 fidite
equivalence (fileedit, string(7:20))

last = 0
outfile=outfile//alpha(usernum)///fil
dflag=0
do while(status.ne.%loc(ss$-notran))
    status=sys$trimloc(file(1:file$len), file+len, file,..)
enddo

if (file(1:2).eq. esc$null) then
    file(1:file$len) = file(5:file$len)
    file$len = file+len - 4
endif

open (unit=1, file='mail.com', status='old')
read(1,0,end=7) string
format(a)
close(unit=1)
if((string(2:5).eq.'edit') or (string(2:5).eq.'EDIT')) then
    status = lib$spawn('$edit'//fileedit, outfile)
    if(.not.status) call lib$stop(%val(status))
do to 9
end if

if ((string(2:3).eq.'LO') or (string(2:3).eq.'lo')) then
    status = sys$c!ref(%val(72+usernum))
    if (.not.status) call lib$stop(%val(status))
    last = 1
end if
status = lib$spawn('file:file+len', outfile)
if (.not., status) call lib$stop(%val(status))

9 type 10
10 format('command done')
if (last, eq, 1) call exit ! The last command was logout.
  return
11 end
APPENDIX G
SOFTWARE PROTOCOL IN MDS USING ETHERNET LAN

The following programs, developed by Mark Stotzer [Ref. 2], provide the means of accessing the Ethernet via the NI3010 Ethernet Communications Controller. The same programs work in both MDS's presently available in NPS Spanagel Hall, rooms 523 and 525.

Two modifications were introduced in these programs in order to improve the efficiency and speed of VAX-MDS communication:

1. In the subroutine "Sendmsg" was added a new assignment, namely "TXBUFF(8) = 0" in two places in order to denote that the frame that is sent is a command and not an acknowledge. This was necessary to be done since, with the previous set up, the MDS was transmitting an acknowledge frame with the type field (bytes 8 and 9) of a command frame. So when the VAX was receiving an acknowledge, it was interpreted as a command frame causing communication problems.

2. In the subroutine "Conmsg" was done a transposition of the call statements to the subroutines "Emptbuf" and "Trmsg". This way when the MDS receives a frame it sends first the acknowledge frame to VAX and after that dumps it to console resulting in much faster exchange of frames between MDS and VAX.
ETHERNET: /*MAIN MODULE-APPLICATION LAYER-ISO LEVEL 7*/

PROCEDURE OPTIONS ('MAIN');

DECLARE
/* LOCAL VARIABLES */
COUNT7 FIXED BINARY(7),
COUNT?A FIXED BINARY(7),
COUNT?B FIXED BINARY(7),
COUNT?C FIXED BINARY(7),
DSKNO CHARACTER(1),
FRAMD CHARACTER(1),
SELECT CHARACTER(1),
/* GLOBAL VARIABLES */
REC7IL FIXED BINARY(7) EXTERNAL,
FRISIZE FIXED BINARY(15) EXTERNAL,
VTERM FIXED BINARY(7) EXTERNAL,
TRMODE FIXED BINARY(7) EXTERNAL,
/* GLOBAL DATA STRUCTURES */
TXBUFF(1508) FIXED BINARY(7) EXTERNAL,
RXBUFF(1522) FIXED BINARY(7) EXTERNAL,
TXTBUFF(128) FIXED BINARY(7) EXTERNAL,
1 RXFCB EXTERNAL,
2 DSK FIXED BINARY(7),
2 FNAME CHARACTER(8),
2 FTYPE CHARACTER(3),
2 RFCB(24) FIXED BINARY(7),
1 TXFCB EXTERNAL,
2 DISK FIXED BINARY(7),
2 FNAME CHARACTER(8),
2 FTYPE CHARACTER(3),
2 TFCB(24) FIXED BINARY(7),
/* EXTERNAL MODULES */
INIT ENTRY,
SENDATA ENTRY,
RECEIVE ENTRY;

/*LAST REVISION: 09/15/83-09/00 ORIGINAL PROGRAM: 07/29/83 */
/*AUTHOR: CAPT. MARK D. STOTZER-USMC-AEGIS GROUP */
/*THESIS ADVISOR: PROFESSOR R. KODRES-COMP. SCIENCE */

PUT SKIP LIST('***
**
*****
**
****');

PUT SKIP LIST('ETHERNET COMMUNICATION PROGRAM-VERSION 5.0');

PUT SKIP LIST('ALLOWS THIS HOST TO CONNECT TO THE NET.');

PUT SKIP LIST('CNTL-E=BACKSPACE FOR TEXT ENTRIES:');

PUT SKIP('2');

RECFIL=47;
COUNT?=-1;
DO WHILE (COUNT?=1);
COUNT?A=1;
DO WHILE (COUNT?A=1);
PUT SKIP('2');

134
PUT SKIP LIST('*************** MAIN MENU ***************');
PUT SKIP LIST('WRITE RECEIVED FILES TO DISK NO:');
PUT SKIP LIST('DEFAULT DRIVE(A) = 1');
PUT SKIP LIST('DISK DRIVE A = 2');
PUT SKIP LIST('DISK DRIVE 3 = 3');
PUT SKIP LIST('*************** MAIN MENU ***************');
GET LIST(DSNOS);
PUT SKIP(2);
IF DSNOS='1' THEN DO;
  RXFB3.DISK=2;
  COUNT7A=2;
END;
ELSE IF DSNOS='2' THEN DO;
  RXFB3.DISK=1;
  COUNT7A=2;
END;
ELSE IF DSNOS='3' THEN DO;
  RXFB3.DISK=2;
  COUNT7A=2;
END;
ELSE PUT SKIP LIST('INVALID DRIVE NUMBER-REENTER:');
END;/*DC LOOP*/
COUNT7=1;
DO WHILE (COUNT7B=1);
PUT SKIP LIST('ETHERNET FRAME DATA BLOCK SIZE:');
PUT SKIP LIST('SELECT 128 FOR ALL FILE OPERATIONS');
PUT SKIP LIST('AND VAX COMMUNICATIONS.');
PUT SKIP LIST(' 128 BYTES = 1');
PUT SKIP LIST(' 256 BYTES = 2');
PUT SKIP LIST(' 512 BYTES = 3');
PUT SKIP LIST(' 1024 BYTES = 4');
PUT SKIP LIST(' 1520 BYTES = 5');
PUT SKIP LIST('*************** MAIN MENU ***************');
PUT SKIP LIST('ENTER SELECTION==>');
GET LIST(FRAMDS);
PUT SKIP(2);
IF FRAMDS='1' THEN DO;
  FRSIZE=128;
  COUNT7B=2;
END;
ELSE IF FRAMDS='2' THEN DO;
  FRSIZE=256;
  COUNT7B=2;
END;
ELSE
IF FRAMD='3' THEN DO;
    FR SIZE = 512;
    COUNT7E = 2;
    END;
ELSE IF FR AMD='4' THEN DO;
    FR SIZE = 1024;
    COUNT7E = 2;
    END;
ELSE IF FR AMD='5' THEN DO;
    FR SIZE = 1500;
    COUNT7E = 2;
    END;
ELSE PUT SKIP LIST('INCORRECT CHOICE-REENTER:');
END;/* DO LOOP */
VTERM = 0;
TR MODE = 0;
CALL INIT;
PUT SKIP LIST('OPERATING MODES:');
PUT SKIP LIST(**********************); PUT SKIP LIST('RECEIVE WAIT LOOP = 1');
PUT SKIP LIST('TRANSMIT FILE CR MESSAGE = 2');
PUT SKIP LIST('VIRTUAL TERMINAL OF VAX = 3');
PUT SKIP LIST('VAX COMMAND MODE = 4');
PUT SKIP LIST('DISCONNECT FROM NET = 5');
PUT SKIP LIST(**************); PUT SKIP LIST('ENTER SELECTION ==>');
GET LIST(SELECT);
PUT SKIP(2);
IF SELECT='1' THEN DO;
    TXBUFF(1)=5;
    TXBUFF(2)=7;
    TXBUFF(3)=1;
    PUT SKIP LIST('IN RECEIVE WAIT LOOP-TO RETURN TO');
    PUT SKIP LIST('MAIN MENU: ENTER <CR> ==>');
    PUT SKIP LIST(***************);
    CALL RECEIVE;
END;
ELSE IF SELECT='2' THEN CALL TRANS2;
ELSE IF SELECT='3' THEN DO;
    VTERM=1;
    FR SIZE=1500;
    PUT SKIP LIST('****** VAX TERMINAL MODE *****');
    PUT SKIP(1);
PUT LIST ('VAX TERMINAL SERVICE:');
PUT LIST ('DATA BLOCK SIZE PER FRAME=');
PUT LIST ('FRSIZE:---------------------------------');
PUT LIST ('TERMINAL ENTRY BY LINE OF TEXT:');
PUT LIST ('BEGIN AFTER INITIAL V PROMPT: "V">');
PUT LIST ('ENTER: TEXT LINE<CR>');
PUT LIST ('PROMPT WILL AUTOMATICALLY REAPPEAR');
PUT LIST ('OF THE NEXT LINE YOU BEGIN.');
PUT LIST ('END TERMINAL SESSION:');
PUT LIST ('ENTER: <CR> AFTER "V">');

---

TXBUFF(1)=2;
TXBUFF(2)=7;
TXBUFF(?)=1;
TXUFF(4)=3;
TXUFF(?)=7;
TXUFF(6)=127;
TXUFF(?)=0;
TXUFF(8)=0;
COUNT7C=1;

PUT LIST ('V>');</n
DO WHILE (COUNT7C=1);
    CALL SENDATA;
    PUT LIST ('V>');</n
    IF VTERM=0 THEN /*END TERMINAL SESSION*/
    DO:
        PUT LIST ('*** END TERMINAL SESSION ***');</n        COUNT7C=2;
    END;

    ELSE
    DO;
        CALL INIT;
        CALL RECEIVE;
        PUT LIST ('E'EV>');</n
    END;

    ELSE /* DO LOOP */
    END;

    IF SELECT = '4' THEN
    DO:
        PUT LIST ('*** VAX COMMAND INSTRUCTIONS ***');</n        PUT LIST ('TO DOWNLOAD A FILE FROM THE VAX:');</n        PUT LIST ('ENTER THE MESSAGE:');</n        PUT LIST ('FILENAME(VAX) FTYPE(VAX)/XXX ""');</n        PUT LIST ('WHERE XXX=EXE FOR NON-TEXT FILES');</n        PUT LIST ('AND XXX=TXT FOR TEXT FILES');</n        PUT LIST ('FILE WILL THEN BE IMMEDIATELY SENT');</n        PUT LIST ('TO THIS HOST.');</n        PUT LIST ('TO UPLOAD A FILE TO THE VAX:');</n
    END;

137
PUT SKIP LIST('1. ENTER THE MESSAGE:');
PUT SKIP LIST('GNAME(VAX).FTYPE(VAX)/XXX''');
PUT SKIP LIST('TO OPEN A VAX FILE BY THE ABOVE NAME');
PUT SKIP LIST('2. THEN:');
PUT SKIP LIST('SEND THE FILE TO THE VAX ADDRESS USING');
PUT SKIP LIST('THE NORMAL FILE Sending SELECTIONS.');
PUT SKIP LIST('THE MESSAGE: ');
PUT SKIP 1;
TRMODE=1; /*SET VAX CMD MODE FLAG TO TRUE*/
FRSIZE=128;
TXBUFF(1)=2;
TXBUFF(2)=7;
TXBUFF(3)=1;
TXBUFF(4)=0;
TXBUFF(5)=7;
TXBUFF(6)=127;
TXBUFF(7)=0;
TXBUFF(8)=0;
CALL SENDATA;
CALL INIT;
RXBUFF(17)=255;
CALL RECEIVE;
END;
ELSE
IF SELECT='5' THEN
COUNT7=2;
ELSE
PUT SKIP LIST('INCORRECT OFMODE SELECTION-REENTER: ');
END; /* DO LOOP */
PUT SKIP LIST('DISCONNECTING FROM NET-RETURNING TO CP/M.');
TRANS2:
PROCEDURE;
DECLARE /* LOCAL VARIABLES */
COUNT6 FIXED BINARY(7),
COUNT6A FIXED BINARY(7),
COUNT6B FIXED BINARY(7),
COUNT6C FIXED BINARY(7),
SENDTYPE CHARACTER(1),
FTYP CHARACTER(1),
DN0 CHARACTER(1),
/* FILE DATA ENTRY DCLS */
I FIXED,
FN CHARACTER(20),
LOWER CHARACTER(26) STATIC INITIAL
('abcdefghijklmnopqrstuvwxyz'),
UPPER CHARACTER(25) STATIC INITIAL
('ABCDEFGHIJKLMNOPQRSTUVWXYZ'),
/* GLOBAL VARIABLES */
FILTYP FIXED BINARY(7) EXTERNAL,
YNOP FIXED BINARY(7) EXTERNAL,
/* GLOBAL DATA STRUCTURES */
TXBUFF'1509) FIXED EINARY 7; EXTERNAL.
1 TXXCF EXTERNAL.
2 DISK FIXED EINARY(7),
2 FNAME CHARACTER(8),
2 FTYPE CHARACTER(3),
2 FTCP(24) FIXED EINARY(7).
/* EXTERNAL MODULES */
SENDATA ENTRY;

COUNT6 = 1;
DO WHILE(COUNT6 = 1);
   PUT SKIP LIST('TRANSMISSION OPTIONS:');
   PUT SKIP LIST('SEND A MESSAGE = 1');
   PUT SKIP LIST('SEND A DISK FILE = 2');
   PUT SKIP LIST('**********');
   PUT SKIP LIST('ENTER SELECTION ==>');
GET LIST(SENDTYPE);
PUT SKIP(2);
TXBUFF(8) = 1;
IF SENDTYPE = '1' THEN
   DO;
      TXBUFF(7) = 3;
      CALL SENDATA;
      COUNT6 = 2;
   END;
ELSE
   IF SENDTYPE = '2' THEN
      DO;
         TXBUFF(7) = 15;
         COUNT6A = 1;
      DO WHILE(COUNT6A = 1);
         PUT SKIP LIST('NATURE OF FILE TO SEND:');
         PUT SKIP LIST('TEXT (ASCII) FILE = 1');
         PUT SKIP LIST('MACHINE CODE (COM) FILE = 2');
         PUT SKIP LIST('**********');
         PUT SKIP LIST('ENTER TYPE OF FILE CHOICE ==>');
GET LIST(F'TYP);
PUT SKIP(2);
IF F'TYP = '1' THEN
   DO;
      FILTYP = 1;
      COUNT6A = 2;
   END;
ELSE
   IF F'TYP = '2' THEN
      DO;
         FILTYP = 2;
         COUNT6A = 2;
      END;
ELSE
   PUT SKIP LIST('INCORRECT CHOICE-REENTER:');
END; /* DO LOOP */
COUNT6 = 1;
DO WHILE(COUNT6 = 1);
COUNT6C = 1;
139
DO WHILE (COUNT6C=1):
  PUT SKIP LIST ('SPECIFY FILE TO SEND:');
  PUT SKIP LIST ('FILE LOCATED ON:');
  PUT SKIP LIST (' DRIVE A = 1');
  PUT SKIP LIST (' DRIVE B = 2');
  PUT SKIP LIST ('***************');
  PUT SKIP LIST ('ENTER DRIVE NUMBER==>');
  GET LIST (DRNC);
  PUT SKIP (2);
  IF DRNO='1' THEN DO;
    TXFCB.DISK=1;
    COUNT6C=2;
  END;
  ELSE IF DRNO='2' THEN DO;
    TXFCB.DISK=2;
    COUNT6C=2;
  END;
  ELSE PUT SKIP LIST ('INVALID DRIVE-REENTER:');
END;/* DO LOOP */
PUT SKIP LIST ('ENTER: "FILENAME.FILETYPE"==>');
GET LIST (FN);
PUT SKIP (2);
FN=TRANSLATE (FN, UPER, LOWER);
I=INDEX (FN, '.');
IF I=0 THEN DO;
  TXFC3.FNAME=FN;
  TXFC3.FTYPE='';
END;
ELSE DO;
  TXFC3.FNAME=SUESTR(FN, 1, I-1);
  TXFC3.FTYPE=SUESTR(FN, I+1);
END;
TXFC3.TFCB(1)=0;
TXFC3.TFCB(4)=0;
TXFC3.TFCB(21)=0;
CALL SENDATA;
IF FNOP = 1 THEN COUNT6B=2;
END;/* DO LOOP */
COUNT6=2;
ELSE PUT SKIP LIST ('INCORRECT TRANSMIT MODE-REENTER:');
END;/* DO LOOP */
END TRANS2;

END ETHERNET;/* ISO LAYER 7 MODULE */
SENDATA: /* PRESENTATION LAYER MODULE-ISO LEVEL 6 */
PROCEDURE;
DECLARE
/* LOCAL VARIABLES */
COUNT5A FIXED BINARY(7),
VAXMODE CHARACTER(1),
DESTADDR CHARACTER(1),
/* GLOBAL VARIABLES */
TRMODE FIXED BINARY(?) EXTERNAL,
VTERM FIXED BINARY(?) EXTERNAL,
FRSIZE FIXED BINARY(?) EXTERNAL,
/* GLOBAL DATA STRUCTURES */
TXBUFF(1508) FIXED BINARY(?) EXTERNAL;

/* AUTHOR: CAPT. MARK D. STOTZER-USMC-AEGIS GROUP */
/* ORIGINAL PROGRAM: 07/29/83 */
/* LAST REVISION: 11/21/83-2200 BY IOANNIS KIDONIIFS */
/* AND ANTHONY SAKILLAROFOUNOS */
/* THESIS ADVISOR: PROF. JNC R. KODES-COMPUTER SCIENCE */

IF VTEPM = 1 THEN /* TERMINAL MODE */
DO;
CALL SENDMSG;
RETURN;
END;
IF TRMODE = 1 THEN /* VAX COMMAND MODE */
DO;
CALL SENDMSG;
RETURN;
END;
COUNT5A = 1;
DO WHILE (COUNT5A = 1);
PUT SKIP LIST ('ADDRESS IS ON THIS NETWORK: ');
PUT SKIP LIST ('00-03-BA: MDS SYSTEM = 1');
PUT SKIP LIST ('00-04-0A: MDS SYSTEM = 2');
PUT SKIP LIST ('00-02-07: VAX 11/760 = 3');
PUT SKIP LIST ('*********************************');
PUT SKIP LIST ('ENTER SELECTION ==>');
GET LIST (DESTADDR);
PUT SKIP (2);
TXBUFF(1) = 2;
TXBUFF(2) = 7;
TXBUFF(3) = 1;
TXBUFF(4) = 0;
IF DESTADDR = '1' THEN
DO;
TXBUFF(5) = 3;
TXBUFF(6) = 234;
IF TXBUFF(7) = 0 THEN
CALL SENDMSG;
ELSE
CALL SENDFILE;
COUNT5A = 2;
END;
ELSE IF DESTADDR='2' THEN
  DO;
    TXBUFF(5)=4;
    TXBUFF(6)=10;
    IF TXBUFF(7)=0 THEN
      CALL SENDMSG;
    ELSE
      CALL SENDFILE;
    COUNT5A=2;
  END;
ELSE IF DESTADDR='3' THEN
  DO;
    TXBUFF(5)=7;
    TXBUFF(6)=127;
    TRMODE=0;
    IF TXBUFF(7)=0 THEN
      CALL SENDMSG;
    ELSE
      CALL SENDFILE;
    COUNT5A=2;
  END;
ELSE
  PUT SKIP LIST('INVALID NET ADDRESS SELECTED-REENTER:');
END; /* DO LCOP */

SENDMSG: /* MESSAGE SENDING MODULE */
PROCEDURE;
DECLARE /* LOCAL VARIABLES */
  /* GLOBAL VARIABLES */
  FRSIZE FIXED BINARY(15) EXTERNAL,
  TRMCDE FIXED BINARY(7) EXTERNAL,
  VTERM FIXED BINARY(7) EXTERNAL,
  /* GLOBAL DATA STRUCTURES */
  TXBUFF(1502) FIXED BINARY(7) EXTERNAL,
  PXBUFF(1522) FIXED BINARY(7) EXTERNAL,
  /* EXTERNAL MODULES */
  FILBUF ENTRY,
  SENDFRAM ENTRY;
IF VTERM=1 THEN /* VIRTUAL TERMINAL MODE */
DO;
  CALL FILBUF;
  TXBUFF(9)=8;
  IF TXBUFF(9)=96 THEN
    RETURN;
  IF TXBUFF(9) 48 & TXBUFF(10)=96 THEN /*END SESSION*/
    VTERM=0; /*END TERMINAL SESSION*/
  ELSE
    TXBUFF(8)=0;
    CALL SENDFRAM;
  END; /* DO */
}
ELSE DO:
''MESSAGE SENDER:'': PUT SKIP LIST('MAXIMUM NUMBER OF CHARACTERS=');
PUT LIST(FRSIZE);
''ENTER MESSAGE AFTER PROMPT: >'');
PUT SKIP LIST('END MESSAGE WITH ACCENT: ');
PUT SKIP LIST('’); CALL FILBUF; /*FILL TRANSMIT BUFFER FROM CONSOLE*/
CALL SENDFRAM; /* SEND THE MESSAGE */
END;
END SENDMSG;

SENDFILE: /* FILE SENDING MODULE*/

PROCEDURE /* LOCAL VARIABLES */
DECLARE
''COUNT4 FIXED BINARY(7),
/* GLOBAL VARIABLES */
''FILYP FIXED BINARY(7) EXTERNAL,
''FNOP FIXED BINARY(7) EXTERNAL,
''LFRM FIXED BINARY(7) EXTERNAL,
/* GLOBAL DATA STRUCTURES */
''TXBUFF(1528) FIXED BINARY(7) EXTERNAL,
/* EXTERNAL MODULES */
''VAXTXT ENTRY,
''TRNDMA ENTRY,
''OPENDF ENTRY,
''RDISK ENTRY,
''SENDFRAM ENTRY;

TXBUFF(7)=15;
TXBUFF(8)=0;
CALL OPENDF;
IF FNOP= THEN /*FILE NOT ON DISK*/
DO:
''FILE NOT ON DISK-REENTER DATA:'';
PUT SKIP(2);
RETURN;
END;
IF TXBUFF(6)=127 & FILYP= THEN
CALL VAXTXT; /*VAX TEXT FILE FORMAT CONVERTER*/
ELSE DO;
CALL TRNDMA; /* SET DISK DMA ADDRESS*/
''****** FILE TRANSFER BEGINS ******'';
PUT SKIP(2);
COUNT4=1;
DO WHILE(COUNT4=1);
CALL RDISK; /*READ A DISK FILE RECORD*/
IF LFRM =1 THEN
DO;
''SENDTRANSmitter;
TXBUFF(8)=1;
ELSE

COUNT4=2;
END; /* DO LOOP */
TXBUFF(3)=255;
CALL SENDFRAM;
PUT SKIP LIST('***** FILE TRANSFER ENDS *****');
PUT SKIP(2);
RETURN;
END:
END SENDFILA;

END SENDATA; /* ISO LAYER 6 TRANSMIT MODULE */
RECDATA: /* ISO LAYER 6 RECEIVE MODULE */

PROCEDURE;

DECLARE /* GLOBAL DATA STRUCTURES */
RXBUFF(1522) FIXED BINARY(7) EXTERNAL;

/*LAST REVISION: 11/31/83-2000 BY IOANNIS KIDONIETS*/
/* AND ANTHONY SAXELLAROPOULOS*/

/*ORIGINAL PROGRAM:08/17/83*/
/*AUTHOR: CAPT MARK D. STOTZER-USMC-AEGIS GROUP*/
/*THESIS ADVISOR: PROF. UNO R. YODRES-COMPUTER SCIENCE*/

IF RXBUFF(17)= 0 THEN /* MESSAGE FRAME */
CALL CONMSG;
ELSE
IF RXBUFF(17)= 15 THEN /* FILE FRAME */
CALL FILER;
ELSE
PUT SKIP LIST('RECEIVED IMPROPERLY ENCODED FRAME');

CONMSG: /* MESSAGE RECEIPT MODULE */

PROCEDURE;

DECLARE /* GLOBAL VARIABLES */
TRMODE FIXED BINARY(7) EXTERNAL,
FRSIZE FIXED BINARY(15) EXTERNAL,
VTERM FIXED BINARY(7) EXTERNAL,
/* GLOBAL DATA STRUCTURES */
RXBUFF(1522) FIXED BINARY(7) EXTERNAL,
/*EXTERNAL MODULES */
TRMSG ENTRY,
EMTBUF ENTRY;

IF VTERM=1 THEN /* NOT IN VIRTUAL TERMINAL MODE*/
DO;
PUT SKIP LIST('***** RECEIVED MESSAGE IS: ');
PUT SKIP(2);
END;
CALL TRMSG; /* SEND THE ACK FRAME */
CALL EMTBUF; /* DUMP THE RECDY FRAME DATA TO CONSOLE */
IF VTERM =1 THEN
DO;
PUT SKIP(2);
PUT SKIP LIST('***** END OF MESSAGE TEXT. ');
PUT SKIP(2);
PUT SKIP LIST('BACK IN WAIT LOOP-ENTER<CR> TO EXIT=>');
PUT SKIP LIST('***********');
PUT SKIP(2);
END;
ELSE

145
IF RXBUFF(18)=15 THEN /* END OF TERMINAL REPLY */
    PUT SKIP LIST("V>"),
END CONMSG;

FILER: /* FILE FRAME RECEIPT MODULE*/

PROCEDURE;

DECLARE /* GLOBAL VARIABLES */
    TRMODE FIXED EINARY(7) EXTERNAL,
    RXEUFF FIXED EINARY(7) EXTERNAL,
    VTERM FIXED EINARY(7) EXTERNAL,
    /* GLOBAL DATA STRUCTURES */
    1 RXFCS EXTERNAL,
    2 DISK FIXED EINARY(7),
    2 FNAME CHARACTER(8),
    2 FTYPE CHARACTER(8),
    2 TFCB(24) FIXED EINARY(7),
    RXBUFF(1522) FIXED EINARY(7) EXTERNAL,
    /* EXTERNAL MODULES */
    RCVDMA ENTRY,
    DELEDF ENTRY,
    MAKEDF ENTRY,
    WRDISK ENTRY,
    TRMSG ENTRY,
    CLOSDF ENTRY;

CALL RCVDMA;
IF RXBUFF(18)=0 THEN /* FIRST FILE FRAME */
    DO;
        PUT SKIP LIST("****** FILE RECEIPT BEGINS *******");
        PUT SKIP(2);
        RXFCS.FNAME="RECFROM ";
        RXFCS.FTYPE="NET";
        RXFCS.TFCB(1)=0; /*CURRENT EXTENT FIELD*/
        RXFCS.TFCB(4)=0;
        RXFCS.TFCB(21)=0;
        CALL DELEDF; /*DELETE OLD FILE OF THIS FN.FT*/
        CALL MAKEDF; /*CREATE A NEW ONE*/
        CALL WRDISK; /*WRITE FIRST RECORD(126 BYTES) TO DISK*/
        CALL TRMSG; /*SEND THE FIRST ACK FRAME */
    END;
ELSE IF RXBUFF(18)=1 THEN /*INTERMEDIATE FILE FRAME*/
    DO;
        CALL WRDISK; /*WRITE NEXT RECORD TO DISK*/
        CALL TRMSG; /*SEND THE ACK FRAME */
    END;
ELSE IF RXBUFF(18)=255 THEN /*LAST(DUMMY) FILE FRAME*/
    DO;
        CALL CLOSDF; /*CLOSE THE DISK FILE*/
        PUT SKIP LIST("****** END FILE RECEIPT *******");
    END;
PUT SKIP LIST('SEE FILE(S);RECFROM_.NET');
PUT SKIP(2);
CALL TERMS; /*SEND THE LAST AG;*/
PUT SKIP LIST('__________________________( )');
PUT SKIP LIST('IF RECEIVED FILE IS A TEXT FILE FROM');
PUT SKIP LIST('THE VAX THEN REFORMAT USING: ');
PUT SKIP LIST('"PIP FNAME. PTYPE=RECFROM_.NET[DEC]"');
PUT SKIP LIST('WHERE FNAME. PTYPE IS YOUR CHOICE');
PUT SKIP LIST('__________________________( )');
PUT SKIP(2);
IF TERM=1 THEN
  DO;
    PUT SKIP LIST('STILL IN VAX TERMINAL MODE: ');
    PUT SKIP LIST('Y');
  END;
ELSE
  DO;
    PUT SKIP LIST('IN WAIT LOOP-ENTER<CR> TO EXIT');
    PUT SKIP LIST('**************************');
    PUT SKIP(2);
  END;
ELSE
  PUT SKIP LIST('FRAME TYPE FIELD BYTE 2 INVALID CODE');
END FILER;
END RECEIVED; /* ISO LAYER 6 RECEIVE MODULE */
APPENDIX H
HIGH LEVEL DESIGN OF A VIRTUAL TERMINAL NETWORK

This Appendix might be useful to the person who may undertake the design of a virtual terminal network. It contains a high level design of a network in which several MDS's will act as VAX/VMS virtual terminals.

The multiplexing of Ethernet interface is the backbone of a design like this. Many routines of the software of the present thesis can be used exactly as they are now, assuming that the system will include no more than nine virtual VAX terminals.

The present configuration of the Ethernet Interface Multiplexing requires that the program "Ethermult" which performs this task will run in a VAX terminal. Since this is undesirable in a virtual terminal network, the program which will perform the coordination of the users, should be able to start execution automatically when a message arrives at the NI1010 board. Also it should be able to supervise any user, regardless of his privileges. In other words it should be able to have access to any user's Virtual Memory Space.

A solution which will fulfill those requirements would be the installation of the coordinating program inside the VMS operating system.

The program could use the "Sys$gio" system routine to listen to the NI1010 board. This routine is interrupt-driven and executes prespecified operations when an I/O event occurs. So, the program could "set the ear" of the system and then for reasons of efficiency go to hibernation. The sequence of operations in the program could be as follows:
As soon as a message arrives at the NI1010, an AST wakes the coordinating program up in order that it will undertake normal operation.

The same procedures which are executed on the real VAX terminal could be followed. So, if the first received message is a carriage return, the "Loginout" procedure is called by the coordinating program to interact with the user for identification and authorization. Naturally, routine "Sendmsg" or a similar one will be used to send the name and password requests to the MDS.

If authorization is successful the user name and the address of the corresponding virtual terminal is put in a table and the number of current users is updated. Then a mailbox for this user should be created. This mailbox will be the input port for the "Loginout" procedure (this is what cannot be achieved currently), and the output port will be a file.

As soon as a command enters a mailbox it is immediately executed in the same manner as if this command had been entered from a VAX terminal. This happens because "Loginout" maps the DCL commands to P0 and P1 spaces of the process that it creates.

Messages other than the initial carriage return for each user are queued and distributed to the appropriate mailboxes in a similar way as in "Ethermult".

An "export" routine will pick up the ready answers and send them to the NI1010.

The use of common event flags will be restricted to the denotation of ready answers. There is no need to use a flag to denote the presence of a user in the system, because the program will know that as soon as successful log in has been achieved.

Since the commands which are entered a mailbox are executed immediately the existence of programs like "Usersmult" is not required.
Finally, when a received message is the "Logout" command the user to whom this command is addressed is removed from the system, and the user information table, as well as the number of current users is updated. If all users have exited the system, the program "sets the ear" to the NI1010 and goes to hibernation again.

The software for a virtual terminal network, as visualized by the authors of this thesis, is not much different from program "Ethermult". A good understanding of this program and a thorough knowledge of VAX/VMS facilities should make the accomplishment of this task a relatively easy thing to do.
APPENDIX I

Program logger

c this program creates the detached process 'LOGER' which
runs the image 'LUGINOUT.EXE'.

implicit integer*4(a-z)
integer*4 uic/'0069000E'x,mbx=iosb(2),iachn,
1 stflad/'00000000'x/

character*6 user(2).'/SAKELL','SAKELL'/
character*12 name/. 'SAKELL'/
character*32 pass/ 'SAKELL'/
character mine(2).'/sakell'/
character inout(2).'/sakell'/

create detached process to run the LOGINOUT image,
and set as inout the file 'INPUT.DAT' and output the terminal.

status = sys$creorc(id,'sys$system:loginout','input.dat',
1 'etbl1', 'error.dat',
2 'LOGER',4,val(4),'68A1294',)
if(.not.,status) type *,'ooppa! ',status
if(status) type *,'loginout image executed'

c execute a 'show system' command to see if the detached
process 'LOGER' has been created.

status=lib$spawn('$show system')
type *,pid
end
LIST OF REFERENCES


<table>
<thead>
<tr>
<th>No.</th>
<th>Copies</th>
<th>Initial Distribution List</th>
</tr>
</thead>
</table>
| 1.  |        | Defense Technical Information Center  
Cameron Station  
Alexandria, Virginia 22314 |
| 2.  |        | Library, Code 0142  
Naval Postgraduate School  
Monterey, California 93943 |
| 3.  |        | Department Chairman, Code 52  
Department of Computer Science  
Naval Postgraduate School  
Monterey, California 93943 |
| 4.  |        | Computer Technology Curricular Office  
Code 37  
Naval Postgraduate School  
Monterey, California 93943 |
| 5.  |        | LCDR R. W. Modes, Code 52Mf  
Department of Computer Science  
Naval Postgraduate School  
Monterey, California 93943 |
| 6.  |        | Professor Uno R. Kodres, Code 52Kr  
Department of Computer Science  
Naval Postgraduate School  
Monterey, California 93943 |
| 7.  |        | Daniel Green (Code N20E)  
Naval Surface Warfare Center  
Dahlgren, Virginia 22449 |
| 8.  |        | Lieutenant Ioannis Kidoniefs  
Hellenic Navy General Staff  
Holargos, Athens - Greece |
| 9.  |        | Major Antonios Sakellaropoulos  
Hellenic Air Force General Staff  
Holargos, Athens - Greece |
| 10. |        | Dana Small  
Code 8242  
NOSC, San Diego, California 92152 |